

# **Reinforcement Learning and Differentiable Simulations for Autonomous Tuning and Control of Linear Particle Accelerators**

Vom Promotionsausschuss der  
Technischen Universität Hamburg  
zur Erlangung des akademischen Grades

Doktor-Ingenieur (Dr.-Ing.)

genehmigte Dissertation (Monografie)

von  
**Jan Kaiser**

aus  
Köln

2025

## **1. Gutachterin:**

Prof. Dr.-Ing. Annika Eichler  
Maschine Strahlkontrollen  
Deutsches Elektronen-Synchrotron DESY  
Institut für Regelungstechnik  
Technische Universität Hamburg

## **2. Gutachter:**

Prof. Dr.-Ing. Görschwin Fey  
Institut für Eingebettete Systeme  
Technische Universität Hamburg


## **Vorsitzender des Prüfungsausschusses:**

Prof. Dr.-Ing. Gerhard Bauch  
Institut für Nachrichtentechnik  
Technische Universität Hamburg

Tag der mündlichen Prüfung: 4. Juni 2025

doi:10.15480/882.16060

ISBN: 978-3-8439-5683-3

 <https://orcid.org/0000-0003-3445-0678>

This work is licensed under a Creative Commons “Attribution 4.0 International” licence.



*Any sufficiently advanced technology  
is indistinguishable from magic.*

ARTHUR C. CLARKE'S THIRD LAW



## Acknowledgements

I would like to take this page to express my gratitude to those who have supported me on this PhD journey and enabled the successful completion of this work. It is only through their help, encouragement and inspiration that I was able to reach this milestone.

First and foremost, I would like to thank my supervisor, Annika. Her unwavering support, encouragement and inspiring mentorship have been invaluable throughout the past years, and I am deeply grateful for her enabling me to have the best possible PhD experience. I truly appreciate her patience, insightful feedback, the freedom she afforded me to explore my own ideas, and her constant belief in my abilities. She also enabled me to travel a lot to present my work, which I appreciate immensely. I could not have wished for a better supervisor, and I can say without a doubt that I have grown through her guidance.

Furthermore, I would like to thank Görschwin, who has been a constant source of support and advice. When I was finishing my Master's degree, it was him, who encouraged me to pursue a PhD, who took the time to answer all my questions, and who ultimately led me to join Annika at DESY.

In this light, I would also like to thank Arne. Besides treading on much the same path as me, from studying computer science at TUHH to joining DESY for our PhDs, it was him allowing me to (unintentionally) take a Master's thesis offer from him, that set both of us on this journey in the first place.

I would also like to thank all my colleagues and friends at DESY for making this ride so enjoyable, and for all the fruitful discussions, collaborations and fun times we had together. Particular thanks go to everyone in the IPC group: Annika, Arne, Christian, Jannis, Max and others; with whom I had many fun and insightful times in the office. Furthermore, I would like to thank the wider MSK group, especially Holger for providing such a great working environment and fertile ground for research. I am also grateful to Bianca, for all the fun Wednesday lunch breaks and insightful discussions. A great deal of thanks also go to Oliver, who is responsible for much of my understanding of accelerator physics. I further thank the ARES team: Florian, Willi, Frank, Max, Thomas, Sonja and others; for allowing me so much beam time, and for all the help, support and fun shifts we had together.

Additionally, I would like to thank my collaborators at SLAC, KIT, CERN and others. Their support and collaboration have been invaluable to my research. A special mention goes to all Cheetah collaborators, who have made that project a truly inspiring success. Special thanks go to Andrea, Chenran, Luca and Bianca, who have quite literally travelled from one end of the world to another with me – from Chicago to Seoul.

I am deeply grateful to my friends and family, who have supported me throughout this journey. Thank you to Bianca, Frank, Lars, Arne, Andrea, Christian and my dad for proofreading this dissertation and providing valuable feedback. Thank you also to my mum and dad, my sister, and my grandparents for their unconditional love and support, but most of all for being a constant source of inspiration and encouragement, even though I am pretty sure they still don't really understand what it is that I do. It is childhood experiences, like programming sessions with my dad or flipping through the old red maths book with my grandpa, that sparked my interest in science and technology in the first place. I cherish those memories dearly; and the old red maths book remains on my shelf to this day, even though it is barely held together by tape at this point. Last but certainly not least, I would like to thank my sister's dog Livvi, who was a much-needed source of cuddles and probably a few too many face licks during the final writing phase of this dissertation.

Thank you all for being part of this journey with me!



## Summary

This dissertation is about the development of reinforcement learning (RL) methods and differentiable simulations for autonomous particle accelerator tuning and control.

It focusses primarily on methods to solve the problem of training RL agents for complex tasks, which is especially difficult with accelerators, where beam time is scarce and safety is a concern. As such, this dissertation introduces domain randomisation to the training process. This technique opens the door to training RL policies on a simulation of the accelerator. By randomising over model discrepancies, the trained policy can be deployed to the real accelerator zero-shot, reducing the wall time required from years to less than one hour and eliminating the need for beam time altogether. Extensive comparative studies of RL policies are conducted on a transverse beam tuning task at the Accelerator Research Experiment at SINBAD (ARES) linear accelerator facility at Deutsches Elektronen-Synchrotron (DESY). They show that RL-trained policies outperform both expert human operators and the state of the art in black-box optimisation methods used for accelerator tuning, namely Bayesian optimisation (BO). Despite the success of the proposed additions to RL training, a main difficulty is identified in the engineering challenge that continues to hinder the deployment to day-to-day operations. This dissertation introduces and explores retraining using the same training configuration, including algorithm, hyperparameters, policy architecture and reward function, to transfer to novel similarly characterised transverse beam tuning tasks using other lattice geometries at ARES. A method that goes one step further by taking advantage of domain randomisation to train lattice-agnostic policies which can be transferred zero-shot or with minimal fine-tuning is also proposed and found to reduce the required amount of training and engineering.

As part of the effort to make RL training for accelerators more tractable, a high-speed differentiable beam dynamics simulator with various speed optimisations, called *Cheetah*, is developed based on the *PyTorch* machine learning library. Cheetah is found to be orders of magnitude faster than existing conventional simulators, for example reducing simulated training times on the transverse beam tuning task from weeks to under one hour. In addition to supporting graphics processing unit (GPU) acceleration for further speed gains, Cheetah natively supports automatic differentiation, unlocking the application of novel methods, such as gradient-based accelerator tuning and system identification, as well as differentiable physics-informed priors for BO, to particle accelerators. Cheetah also integrates natively with the wider machine learning ecosystem, enabling AI/ML coupling of programmatic physics models with data-driven modular neural network surrogates for fast and fully differentiable modelling of complex dynamics.

The RL training setup is then applied to the more complex and high-impact task of maximising the Free-Electron Laser (FEL) pulse intensity at the Linac Coherent Light Source (LCLS) at SLAC National Accelerator Laboratory (SLAC), where trained policies are shown to achieve near-optimal pulse intensities after just a few tuning steps. This demonstrates the ability to scale RL training to complex tasks, but also highlights that training times increase significantly with the complexity of the task. A novel gradient-based RL training method based on Cheetah is proposed to address this challenge, reducing the number of training samples required for FEL intensity tuning by nearly two orders of magnitude through the use of the true gradient of the return over the policy parameters rather than a noisy estimate of the policy gradient.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	Machine Learning . . . . .	5
2.1.1	Neural Networks . . . . .	5
2.1.2	Automatic Differentiation and Backpropagation . . . . .	8
2.1.3	Supervised Learning . . . . .	10
2.1.4	Reinforcement Learning . . . . .	11
2.1.5	Bayesian Optimisation . . . . .	19
2.2	Particle Accelerators . . . . .	20
2.2.1	Overview . . . . .	20
2.2.2	ARES . . . . .	24
2.2.3	European XFEL . . . . .	28
2.2.4	LCLS and LCLS-II . . . . .	28
2.3	Related Work . . . . .	32
2.3.1	Machine Learning for Particle Accelerators . . . . .	32
2.3.2	Differentiable Simulations . . . . .	36
2.3.3	Autonomous Accelerator Optimisation and Tuning . . . . .	37
2.3.4	Deep Reinforcement Learning in General . . . . .	40
2.3.5	Reinforcement Learning for Particle Accelerators . . . . .	44
<b>3</b>	<b>High-Speed Differentiable Beam Dynamics Simulations</b>	<b>51</b>
3.1	Start-to-End Neural Network Surrogates . . . . .	52
3.1.1	Longitudinal Phase Space Diagnostics at European XFEL . . . . .	53
3.1.2	Combined Scalar- and Spectral Virtual Diagnostics . . . . .	55
3.1.3	Achieving High Resolution at Any Scale . . . . .	55
3.1.4	Results . . . . .	57
3.2	Physics-based High-Speed Differentiable Simulations . . . . .	58
3.2.1	Fast Differentiable Tracking in Python . . . . .	60
3.2.2	Use Case Examples . . . . .	67
3.3	Coupling Physics-based Models and Modular Neural Network Surrogates . . . . .	73
3.3.1	Microsecond Space Charge Delta Modelling . . . . .	74
3.3.2	Results . . . . .	76
3.4	Summary . . . . .	78
<b>4</b>	<b>Transverse Beam Parameter Tuning</b>	<b>79</b>
4.1	Zero-Shot Reinforcement Learning-Trained Optimisation . . . . .	80
4.1.1	Problem Formulation . . . . .	80
4.1.2	Training . . . . .	83

4.2	Evaluation . . . . .	87
4.2.1	Alternative Algorithms . . . . .	88
4.2.2	Performance of Reinforcement Learning-trained Optimisation . . . . .	90
4.2.3	Comparing to Alternative Algorithms . . . . .	98
4.3	Zero-Shot Feedback for Drift and Fault Compensation . . . . .	109
4.3.1	Running as a Feedback . . . . .	109
4.3.2	Robustness to Actuator Failure . . . . .	113
4.4	Transfer to Novel Lattices . . . . .	114
4.4.1	Retraining from Scratch . . . . .	115
4.4.2	Domain Randomisation for Lattice-Agnostic Policies . . . . .	123
4.5	Natural Language-Driven Particle Accelerator Tuning . . . . .	128
4.5.1	Optimisation Scheme and Prompt Design . . . . .	129
4.5.2	Evaluation . . . . .	137
4.6	Summary . . . . .	147
<b>5</b>	<b>FEL Tuning</b>	<b>149</b>
5.1	Modelling FEL at LCLS . . . . .	149
5.2	Gradient-free Reinforcement Learning . . . . .	151
5.3	Gradient-based Reinforcement Learning . . . . .	157
5.4	Summary . . . . .	160
<b>6</b>	<b>Conclusion</b>	<b>163</b>
6.1	Summary . . . . .	163
6.2	Discussion . . . . .	166
6.3	Outlook . . . . .	167
<b>A</b>	<b>Transfer Maps</b>	<b>169</b>
A.1	Drift Space . . . . .	169
A.2	Steering Dipole Magnet . . . . .	170
A.3	Quadrupole Magnet . . . . .	170
<b>B</b>	<b>Alternate Tuning Result Over Speed Distributions</b>	<b>173</b>
	<b>Symbols and Abbreviations</b>	<b>179</b>
	<b>Bibliography</b>	<b>187</b>

# Chapter 1

## Introduction

Particle accelerators are some of the most sophisticated, complex and advanced machines in operation today. Their primary purpose is the acceleration of charged subatomic particles, such as electrons or protons, to high speeds, typically close to the speed of light. The accelerated particles are then used in a wide range of applications instrumental in facilitating scientific and technological progress, such as fundamental research in physics, cancer therapy, the development of vaccines and drugs, as well as the development and production of novel materials enabling, for example, carbon-neutral transportation.

However, the operation of particle accelerators is a complex and challenging task. State-of-the-art applications place strict requirements on the properties of the beams delivered by the accelerator. Attaining these conditions requires extensive tuning of the accelerator's subsystems, and maintaining them requires advanced control schemes. Because of how complex accelerator tuning tasks often are, they typically continue to be performed manually by experienced human operators. However, manual tuning is a time-consuming process, with over 2000 hours a year spent on tuning at some facilities [62], and the quality of the machine setup after tuning depending substantially on the operators' experience, limiting the reproducibility of results. Effective automation of accelerator tuning has the potential to allow for faster and more reproducible tuning, while enabling working points that are currently unattainable. Consequently, this would push the limits of accelerator operability, enabling novel experiments and applications, increase the beam time available for experiments, and improve the quality of the results obtained, ultimately facilitating advances in all the fields served by particle accelerators.

Over the years, various methods have been developed to automate accelerator tuning and control. Tuning methods range from simple grid search and random search to more advanced methods from the field of numerical optimisation [79], such as Nelder-Mead simplex optimisation [4, 79, 303], genetic algorithms [16, 21, 136, 229] and in most-recent developments, Bayesian optimisation (BO) [271]. These methods have achieved promising results on a variety of accelerator tuning tasks, but they have limitations, including a demand for large amounts of samples, their sensitivity to noise and an inability to handle non-linear, non-convex objective functions. Most critically, even the most capable algorithms like BO eventually fall victim to the so-called *curse of dimensionality* [19], where their performance drops exponentially as the number of tuning dimensions increases and the objective function becomes more complex. High-dimensional and non-linear objective functions, however, are common with complex plants such as particle accelerators. Similarly, conventional control schemes, such as proportional-integral-derivative (PID) controllers, are widely deployed to accelerator systems and crucial to achieving the high

levels of performance already attained by these machines today. However, conventional controllers, too, have their limitations, such as difficulties in handling non-linear dynamics, the requirement for a general understanding of the effects of different actuators on the system and the need for manual tuning of the controller parameters, including complex arrangements like gain scheduling to cover the many different operating conditions encountered in accelerator operations.

Machine learning methods for function approximation and automation have emerged in the past decade as promising solutions to numerous complex tasks across a wide range of fields that were previously thought beyond the ability of computational methods. Reinforcement learning (RL) in particular has been shown to be capable of solving problems at super-human performance levels [12, 305], and outperform classical control methods [59] and numerical optimisation algorithms [51, 190, 191]. This suggests that RL could be a potent method for solving complex accelerator tuning and control tasks, where the complexity of the tasks exceeds the capabilities of classical methods.

However, at the start of this work in 2020, the application of RL to accelerators and other complex real-world systems was very much in its infancy. RL had been shown to be capable of solving complex tasks at super-human performance levels, but these results mostly focussed on benchmark problems for the study of RL algorithms, such as *Atari* games [223], the board game *Go* [305], or video games like *Dota 2* [240] and *Starcraft II* [339]. Some work had shown that RL could be applied to small yet complex real-world systems, such as the control of a robotic hand [239], but other work at the time suggested that the complexities of state-of-the-art RL continue to hinder its application to real-world tasks [69, 145].

In the accelerator domain, RL had been shown to be capable of solving simple accelerator tuning tasks [26, 45, 130, 243, 311], such as the optimisation of the beam trajectory in a linear accelerator [153]. However, the explored tasks were comparatively simple-to-solve, low-dimensional and linear in nature, serving primarily as a proof-of-principle. What is more, studies were focussed on applications in simulation environments, with the deployment of RL to real-world accelerators still a mostly open challenge. Key difficulties with deploying RL to complex real-world accelerator tuning tasks include the high demand for training samples inherent to RL. The limited availability and high cost of beam time at accelerator facilities make the collection of training samples for RL training prohibitively expensive. Moreover, the safety-critical nature of accelerator operation makes the exploration required for obtaining informative training samples dangerous and requires that safety is incorporated into the training process.

As such, the central contributions of this dissertation focus on overcoming the challenges of deploying RL to complex real-world accelerator tuning tasks, addressing the challenges, such as high sample cost and safety requirements, inhibiting training on real accelerators. To this end, the following contributions are made:

- A scheme for training RL policies zero-shot in simulation, without the need for beam time and without the risk of violating the safety of the facility. The scheme includes methods to overcome the sim-to-real gap through the use of domain randomisation and an appropriate problem formulation.
- A high-speed differentiable beam dynamics simulation, both data-driven and physics-based, to enable fast data collection for simulated RL training in feasible time, which can also be used for gradient-based accelerator tuning and system identification as well as to improve the performance of BO on accelerator tuning tasks, and enable the

coupling of data-driven and physics-based models for improved model performance and accuracy. In this context, the applicability of high-speed differentiable models is studied not only in the context of RL but also their significant impact on other machine learning efforts for accelerators and adjacent activities.

- An approach to training RL policies once, which can then be reused on a variety of tasks, forgoing the need to train new policies for new tasks, ultimately reducing the number of required training samples and engineering effort.
- A gradient-based RL policy optimisation method that leverages the developed differentiable beam dynamics simulation to reduce the sample complexity of RL training.
- Trained RL-based accelerator tuning policies for optimising the transverse beam parameters and maximising the Free-Electron Laser (FEL) pulse intensity.
- An extensive evaluation of RL-trained tuning policies against other state-of-the-art autonomous tuning methods, among them BO, on a variety of accelerator tuning tasks, including transverse beam parameter tuning and FEL pulse intensity maximisation.
- A proposal for and exploration of a novel large language model (LLM)-based approach for tuning accelerators and for numerical optimisation, removing the need for training and algorithm definition entirely, requiring only a natural language description of the task and goal at hand.

Ultimately, these contributions make training and deploying RL-trained policies to particle accelerators and other complex real-world systems more feasible. This advancement unlocks RL as a powerful tool to drive a paradigm shift in accelerator operation – from actuator-driven control, where human operators adjust settings to achieve optimal measurements, to specification-driven operation, where operators specify desired experimental conditions, and autonomous agents ensure these conditions are met. As such, autonomous particle accelerator tuning methods promise to not only improve the performance of accelerators on existing applications but also open up new possibilities in scientific research and industrial applications, marking a transformative advance in the field of particle acceleration.

This dissertation is structured as follows: The background of the presented work is introduced in Chapter 2, including relevant concepts and methods of machine learning, RL and particle accelerators, as well as a review of related work. In Chapter 3, the development of high-speed differentiable simulations of particle accelerators is presented as a key prerequisite for the research into deploying RL. Chapter 4 then introduces the central methods of this dissertation to overcome the challenges of deploying RL to particle accelerators, including comprehensive studies of their performance, practical challenges and comparisons to state-of-the-art optimisation methods like BO. In addition, this chapter introduces methods to address the engineering and training complexities inherent to RL, including lattice-agnostic policies and LLM-based optimisation. The studies in this chapter are performed at the example of a transverse beam tuning task that is representative of a wide range of common accelerator tuning tasks, while being complex enough to challenge the capabilities of state-of-the-art optimisation methods and simple enough to allow for a comprehensive study of the methods' performance. The developed methods are then extended to the more complex high-impact task of maximising the FEL pulse intensity in Chapter 5. Here, it is additionally explored how the developed differentiable simulations

can be employed to further reduce the effort required for RL training in the context of accelerators by enabling the use of the true policy gradient by policy optimisation methods. Finally, this dissertation concludes with a summary of the results, a discussion of their implications and an outlook on future research in Chapter 6.

# Chapter 2

## Background

This chapter introduces the background of the research presented in this dissertation. Machine learning with a focus on neural networks, automatic differentiation, supervised learning, BO and most importantly RL, are introduced in Section 2.1. A brief introduction to particle accelerators is provided in Section 2.2, including specific facilities and tuning tasks relevant to this work. Related work is reviewed in Section 2.3, covering general research on differentiable simulations and RL, along with accelerator-specific work on autonomous tuning, and the application of machine learning and RL to accelerators.

### 2.1 Machine Learning

Machine learning in the most general sense is a broad term describing a large collection of statistical methods and algorithms that enable machines to learn from data to make predictions or decisions without being explicitly programmed to do so. These methods usually involve the use of a model with free parameters that are adjusted following some training algorithm to optimise a loss or reward function measuring the model's performance.

Three main categories of machine learning are commonly distinguished: supervised learning, where models are trained on labelled data to make predictions or classifications; unsupervised learning, where models identify patterns in unlabelled data; and reinforcement learning, where models learn to make decisions by interacting with an environment and receiving rewards or penalties based on their actions. In addition, the range of models that can be used for machine learning is vast, including linear models, decision trees, support vector machines, neural networks and many more.

This section introduces the machine learning concepts and methods that are relevant to the work presented in this dissertation, introducing neural networks in Section 2.1.1, automatic differentiation (an important prerequisite for many neural network training algorithms) in Section 2.1.2, supervised learning in Section 2.1.3 and reinforcement learning in Section 2.1.4. In addition, this section includes a brief introduction to BO, a data-driven optimisation algorithm that is adjacent to machine learning and increasingly relevant for state-of-the-art particle accelerator optimisation.

#### 2.1.1 Neural Networks

Neural networks are a class of machine learning models inspired by the operation of biological neurons in the brains of animals and humans [111]. First conceptualised in 1943 by McCulloch and Pitts [214], and first implemented by Rosenblatt in 1957 as the

*Perceptron* [262, 263], neural networks have become arguably the most significant class of models in the field of machine learning since the success of *AlexNet* [181, 182] on the *ImageNet* challenge in 2012. This success is in part due to the universal approximation theorem, which states that neural networks can approximate any continuous function to arbitrary precision given a sufficiently large number of neurons in a single hidden layer [137]. This ability to fit a large variety of functions in combination with their differentiability allowing for ease of training with backpropagation [273] and stochastic gradient descent (SGD) [259] have made neural networks the go-to model for many machine learning tasks today, giving rise to a field of research known as *deep learning*.

The smallest unit of a neural network is the *neuron*. It is inspired by the biological neuron. A neuron takes a vector of input values  $\mathbf{x}$ , multiplies them with a vector of weights  $\mathbf{w}$  and takes the sum of the results. Then, a bias term  $b_j$  is added, and the result is passed through a non-linear function  $\varphi$  called the *activation function* to produce an output  $y_j$ . For a single neuron with index  $j$  and number of inputs  $n$ , where  $x_i$  is the  $i$ -th component of the input vector  $\mathbf{x}$ ,  $w_{ij}$  is the weight connecting the  $i$ -th input to the  $j$ -th neuron,  $b_j$  is the bias of the  $j$ -th neuron and  $y_j$  is the output of the  $j$ -th neuron, this process can be written as

$$y_j = \varphi \left( \sum_{i=1}^n w_{ij} x_i + b_j \right). \quad (2.1)$$

It is also illustrated in relation to a biological neuron in Fig. 2.1. Multiple neurons can be combined into a *layer*, otherwise known as a *perceptron*. A perceptron can be neatly expressed as a matrix-vector multiplication and addition

$$\mathbf{y} = \varphi (W \cdot \mathbf{x} + \mathbf{b}), \quad (2.2)$$

where  $W$  is a matrix of weights connecting the inputs to the neurons,  $\mathbf{b}$  is a vector of biases for each neuron and  $\mathbf{y}$  is the output vector of the layer. Note that it is only through the inclusion of a non-linear activation function that a perceptron can approximate non-linear functions. Commonly chosen activation functions include the sigmoid, hyperbolic tangent ( $\tanh$ ), rectified linear unit (ReLU) [225] and Gaussian error linear unit (GELU) [125] functions. They are plotted in Fig. 2.2. Stacking multiple layers of perceptrons sequentially gives rise to multilayer perceptrons (MLPs), the simplest form of neural network commonly used in practice today. A full MLP contextualised in a flowchart of its inputs, outputs and high-level training process is shown in Fig. 2.3.

Various extensions and modifications to the MLP have been proposed over the years. Some of the most notable ones include convolutional neural networks (CNNs) [111, 187], autoencoders [33, 111, 129, 186], recurrent neural networks (RNNs) [111, 273] and transformers [333].

CNNs [111, 187] convolve the input data with a set of kernels to extract features from the data. This makes them mostly invariant to translations of features in the input and particularly well-suited for tasks like image recognition, as for example in *AlexNet* [181], but they can also be employed on other types of data such as time series.

Autoencoders [33, 111, 129, 186] are a type of neural network architecture usually trained in an unsupervised setting to learn a compressed representation of the input data. They consist of an encoder that maps the input data to a lower-dimensional latent space and a decoder that maps the latent space back to the input space. Autoencoders are used for tasks like data denoising, dimensionality reduction and anomaly detection.

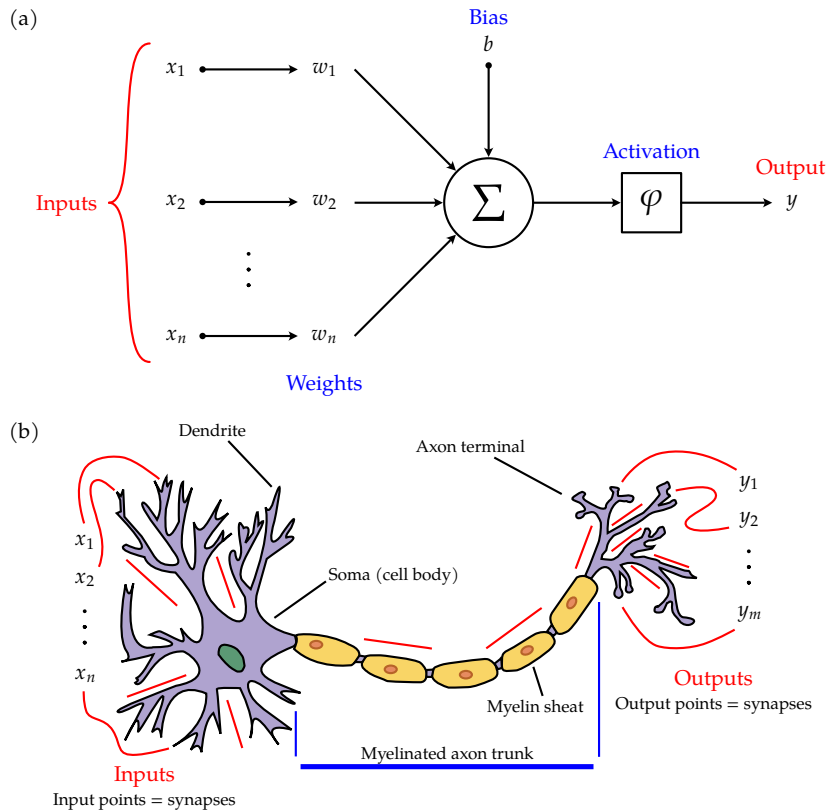


Figure 2.1: (a) Flowchart of an artificial neuron as found in an artificial neural network function approximator [155] according to Eq. (2.1). (b) Illustration of a biological neuron for comparison [77].

RNNs [111, 273] are a type of neural network architecture that is particularly well-suited for operating on sequence data, such as time series or natural language. They have a recurrent connection, adding their output from the previous time step to their input at the current time step, allowing them to maintain a memory of past inputs. The most common variant of RNNs is the long short-term memory (LSTM) cell, which has a more sophisticated gating mechanism that allows it to maintain a longer memory of past inputs [134]. RNNs are particularly effective on tasks like time series forecasting, machine translation, speech recognition and sentiment analysis.

A quite recent development in the field of neural network architectures are transformers [333]. Like RNNs, transformers are especially suitable for operating on sequence data. However, they do not have a recurrent connection, but instead rely on an attention mechanism to learn the relationships between different parts of the input sequence. This makes them more parallelisable and faster to train than RNNs, while also allowing them to maintain a longer memory of past inputs than RNN architectures. Transformers have allowed the field of natural language processing (NLP) to make significant progress in recent years, enabling so-called large language models (LLMs) that can function as intelligent conversational agents, summarisers, translators and coding assistants. Other variants have also found applications to time series forecasting [194] and image recognition [67].

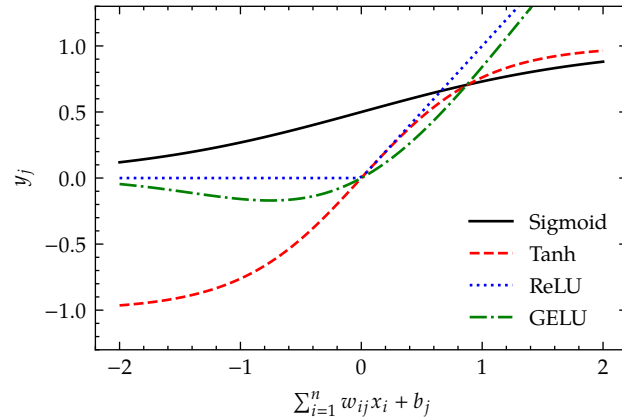


Figure 2.2: Sigmoid, tanh, ReLU and GELU activation functions commonly used in neural networks plotted over the result of adding the bias to the weighted sum of the inputs.

### 2.1.2 Automatic Differentiation and Backpropagation

An important prerequisite for training neural networks and for gradient-based optimisation in general is the ability to compute the gradient of a function with respect to its input. Two well-known methods for computing gradients are numerical differentiation, which approximates the numerical value of the gradient by evaluating the function at multiple points using the method of finite differences, and symbolic differentiation, which computes an exact expression for the gradient by applying the rules of calculus. However, both methods have their drawbacks: Both suffer from high computational complexity. Numerical differentiation further suffers from numerical errors, such as truncation and rounding errors, while symbolic differentiation struggles with expression swell, where the derivative expressions quickly grow intractably large, and requires closed-form expressions, which are not always available for real-world functions.

Automatic differentiation largely overcomes these issues by taking advantage of the chain rule, which states that the derivative of a composition of functions can be computed as the product of the derivatives of the individual functions, e.g.

$$\frac{\partial}{\partial x} f(g(x)) = f'(g(x)) \cdot g'(x). \quad (2.3)$$

This allows for the computation of the gradient of a complex function or program by breaking it down into a sequence of elementary operations, for which the derivative expressions are known. These can then be used to compute the numerical gradient of these small expressions early, and combine them using the chain rule to obtain the numerical value of the gradient of the entire complex function.

Two different kinds of automatic differentiation are generally distinguished: forward mode and reverse mode.

In forward mode automatic differentiation, the derivative of a function is computed starting from the input following the data flow of the function as it is evaluated. That is, given the input's derivative

$$\frac{\partial x}{\partial x} = 1, \quad (2.4)$$

the derivative of  $g(x)$  is computed as

$$\frac{\partial g}{\partial x} = g'(x) \cdot 1, \quad (2.5)$$

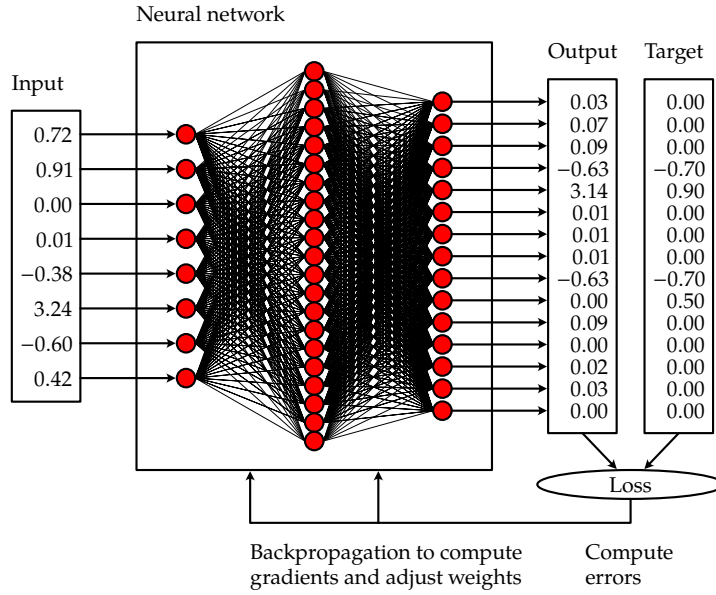


Figure 2.3: Illustration of an MLP in a supervised learning context. The model takes input data  $x$  and produces an output  $\hat{y}$  by passing the input through multiple layers of neurons, where neurons are drawn in red, and the connections between them are drawn as black lines. The output is then compared to the target output  $y$  using an error-computing loss function. The backpropagation algorithm is used to compute the gradient of the loss function with respect to the model parameters, i.e. the neural network's weights and biases, which are then updated in the direction that minimises the loss function using a variant of SGD. This illustration is derived from [171].

followed by the derivative of  $f(g(x))$  as

$$\frac{\partial f}{\partial x} = f'(g(x)) \cdot \frac{\partial g}{\partial x}. \quad (2.6)$$

In reverse mode (sometimes also called backward mode) automatic differentiation, the derivative of a function is computed starting from its output after the function has been evaluated. To this end, the function is first evaluated in the forward pass, storing intermediate values and operations in a computational graph. Then, the derivative of the function is computed in a backward pass by applying the chain rule in reverse order, e.g. the derivative of  $f(g(x))$  is computed as

$$\frac{\partial f}{\partial x} = f'(g(x)) \cdot \frac{\partial g}{\partial x}, \quad (2.7)$$

with the derivative of  $g(x)$  computed as

$$\frac{\partial g}{\partial x} = g'(x) \cdot 1. \quad (2.8)$$

Forward mode differentiation generally requires less memory than reverse mode differentiation, because it does not need to store intermediate values and operations, and is more computationally efficient for functions where the number of inputs is smaller than the number of outputs. Reverse mode differentiation, on the other hand, is more memory-intensive, but more computationally efficient for functions where the number of

outputs is significantly smaller than the number of inputs, as is often the case with machine learning models [15].

A specific use case of reverse mode automatic differentiation is the *backpropagation* algorithm, which is commonly used to train neural networks in modern machine learning. Backpropagation computes the gradient of the loss function with respect to the model's free parameters, that is the weights  $W$  and biases  $\mathbf{b}$  of each layer in the case of MLPs, by applying the chain rule in reverse order. Considering a two-layer MLP with a loss function  $\mathcal{L}$  and parameters  $\theta = \{W, \mathbf{b}\}$ , the gradient of the loss with respect to the weights and biases can be computed as

$$\frac{\partial \mathcal{L}}{\partial \theta} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_n} \frac{\partial \mathbf{y}_n}{\partial \theta} = \frac{\partial \mathbf{y}_n}{\partial \mathbf{y}_{n-1}} \frac{\partial \mathbf{y}_{n-1}}{\partial \mathbf{y}_{n-2}} \cdots \frac{\partial \mathbf{y}_1}{\partial \theta}, \quad (2.9)$$

where  $\mathbf{y}_i$  is the output of the  $i$ -th layer of the MLP, with  $\mathbf{y}_1$  being the input to the first layer,  $\mathbf{y}_n$  being the output of the last layer. Based on the computed gradients, the MLP's parameters are then updated in the direction that minimises the loss function, for example using SGD or one of its variants, such as Adam [173].

### 2.1.3 Supervised Learning

Supervised learning is one of the most common machine learning paradigms. It assumes the presence of a dataset of input-output pairs

$$\mathcal{D} = \{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^N, \quad (2.10)$$

where  $N$  is the number of samples in the dataset. The goal of supervised learning is to train a model to accurately approximate the mapping from the input  $\mathbf{x}$  to the output  $\mathbf{y}$  by minimising a loss function  $\mathcal{L}$  that measures the difference between the predicted output  $\hat{\mathbf{y}}$  and the true output  $\mathbf{y}$ . This means finding the model parameters  $\theta$  that minimise the loss function, i.e.

$$\theta^* = \operatorname{argmin}_{\theta} \sum_{i=1}^N \mathcal{L}(\mathbf{y}_i, \hat{\mathbf{y}}_i), \quad (2.11)$$

where the predicted output  $\hat{\mathbf{y}}_i$  is a function of the input  $\mathbf{x}_i$  and the model parameters  $\theta$ , defined as

$$\hat{\mathbf{y}}_i = f(\mathbf{x}_i | \theta). \quad (2.12)$$

Many different models and loss functions have been developed and used over the years, and the choice depends on the specific task at hand. Commonly, differentiable models and loss functions, such as neural networks and the mean-squared error (MSE) loss function, are used. This enables the efficient computation of the gradient of the loss function with respect to the model parameters using automatic differentiation and backpropagation. The gradient can then be used with SGD or a derivative algorithm to update the model parameters in the direction that minimises the loss function.

The guiding principle of SGD [259, 310] is to compute the gradients of the loss function with respect to the model parameters on a mini-batch  $\mathcal{B}$ , that is a randomly selected small subset of the training data, average the computed gradients over the mini-batch, and update the model parameters in the direction that minimises the loss function. The stochasticity in the selection of the mini-batch that gives SGD its name means that the parameters are never adjusted exactly in the direction of the true gradient, but rather in the direction of

an approximation of it. This helps to avoid local minima and saddle points in the loss landscape, which can be problematic for optimisation algorithms. Following SGD, the model parameters are updated as

$$\theta_{k+1} = \theta_k - \alpha \frac{1}{|\mathcal{B}|} \sum_{i=1}^{|\mathcal{B}|} \nabla_{\theta} \mathcal{L}(y_i, \hat{y}_i), \quad (2.13)$$

where  $\alpha$  is the learning rate, a hyperparameter that controls the size of the step taken in the direction of the gradient,  $|\mathcal{B}|$  is the number of samples in the mini-batch,  $\theta_k$  denotes the model parameters  $\theta$  at training iteration  $k$  and  $\hat{y}_i$  depends on the model parameters  $\theta$  according to Eq. (2.12).

The supervised learning process over an MLP in a regression context is illustrated in Fig. 2.3.

### 2.1.4 Reinforcement Learning

Reinforcement learning (RL) [319] is a machine learning paradigm, where instead of learning from a dataset of input-output pairs, function approximations are learned by the principle of trial and error. The goal of RL is to find an optimal policy function  $\pi^*$  that interacts with an environment by taking actions  $\mathbf{a}_t$  based on the state  $\mathbf{s}_t$  of the environment or observations  $\mathbf{o}_t$  thereof, such that the sum of rewards  $r_t$ , i.e. the (discounted) cumulative reward

$$R(\tau) = \sum_{t=0}^T \gamma^t r_t \quad (2.14)$$

of a trajectory of states and actions

$$\tau = \{\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \dots, \mathbf{s}_T\} \quad (2.15)$$

is maximised, where  $t$  denotes a discrete time step, and  $T$  denotes a finite time horizon, i.e. the length of the trajectory. The goal is formalised as

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E}_{\tau \sim \pi} [R(\tau)]. \quad (2.16)$$

Here, the policy  $\pi$  is a set of rules or a function that maps states or observations to actions, the agent is the entity that interacts with the environment and learns the policy, and the environment is the system that the agent interacts with, i.e. the task that the agent is expected to solve. States  $\mathbf{s}_t$  capture the entire information about the environment at a given time step, while observations  $\mathbf{o}_t$  are (partial) information about the environment that the agent receives. Actions  $\mathbf{a}_t$  are the decisions that the agent can take to influence the environment, and rewards  $r_t$  are scalar values that the environment emits in response to the agent's actions, indicating how well the agent is performing. The interaction loop of an RL agent with its environment is illustrated in Fig. 2.4.

In RL, the environment is typically formalised as a Markov decision process (MDP)

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \rho_0), \quad (2.17)$$

where  $\mathcal{S}$  is the set of valid states,  $\mathcal{A}$  is the set of valid actions,  $\mathcal{R}$  is the reward function

$$r_t = \mathcal{R}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}). \quad (2.18)$$

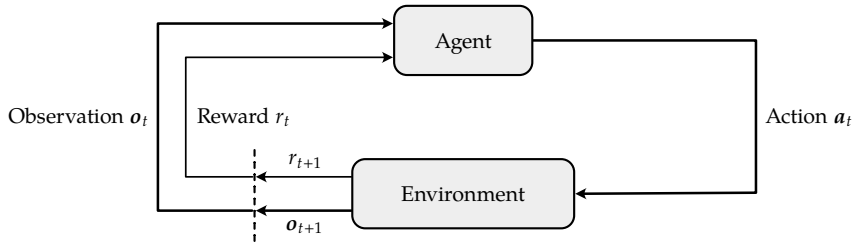


Figure 2.4: Flowchart of the RL loop [158]. Based on the observation  $o_t$  of the environment, the agent chooses an action  $a_t$ . As a result of the action, the environment transitions to a new internal state  $s_{t+1}$ , and emits the next observation  $o_{t+1}$  and reward  $r_{t+1}$ .

$P$  is the state transition probability function

$$P(s_{t+1} | s_t, a_t), \quad (2.19)$$

giving the probability of transitioning to state  $s_{t+1}$  given that action  $a_t$  is taken in state  $s_t$ , and  $\rho_0$  is the initial state distribution. The MDP formalism resembles an automaton modelled as a directed graph, where the states are the nodes and the actions are the edges. The task starts in some initial state  $s_0$  sampled from the initial state distribution  $\rho_0$ . The agent then takes an action  $a_0$ , which transitions the environment to a new state. If the MDP is deterministic, the environment transitions to a single state  $s_1$ , while if the MDP is stochastic, the environment can transition to multiple states with some probability. The environment then emits a reward  $r_0$  and the agent receives an observation  $o_0$  of the new state. This process is then repeated. In an episodic MDP, which is what is assumed throughout this dissertation, a terminal state is eventually reached, after which the task is reset and the agent starts a new episode. A simple MDP is illustrated in Fig. 2.5.

The MDP formalism assumes that the environment is fully observable. This means that the agent has access to the full state, such that the observation is a sufficient statistic of the history of states and actions, and the probability of transitioning to a new state only depends on the current observation and action. This condition is known as the Markov property. In practice, the Markov property is often violated, and the environment is only partially observable. The corresponding formalism is the partially observable Markov decision process (POMDP), which extends the MDP formalism by including a set of observations  $\mathcal{O}$  and an observation function  $\Omega(s_t, a_t)$  that gives the probability of observing an observation  $o_{t+1}$  given that the environment is in state  $s_t$  and action  $a_t$  is taken. When solving RL problems it is generally desirable that the Markov property holds. Techniques, such as temporal integration, i.e. providing a history of observations and actions to the agent, can be used to make the environment at least approximately Markovian.

Various algorithms exist to solve RL problems. These can be broadly categorised according to different taxonomies. A common taxonomy distinguishes between model-based and model-free methods. The RL methods mostly used in this dissertation are model-free methods, which learn a policy or value function directly from the environment without learning a model of the environment. Model-based methods, on the other hand, use either a handcrafted or a learned model of the environment, i.e. the state transition probability function  $P$ , and then employ the learned model for planning.

Another common taxonomy distinguishes between policy-based methods and value-based methods. Policy-based methods directly learn the policy  $\pi$  that maps states to actions. That is, in a simple tabular setting with discrete states and actions, one would perform a table lookup to find the best action for a given state. The entries in the table

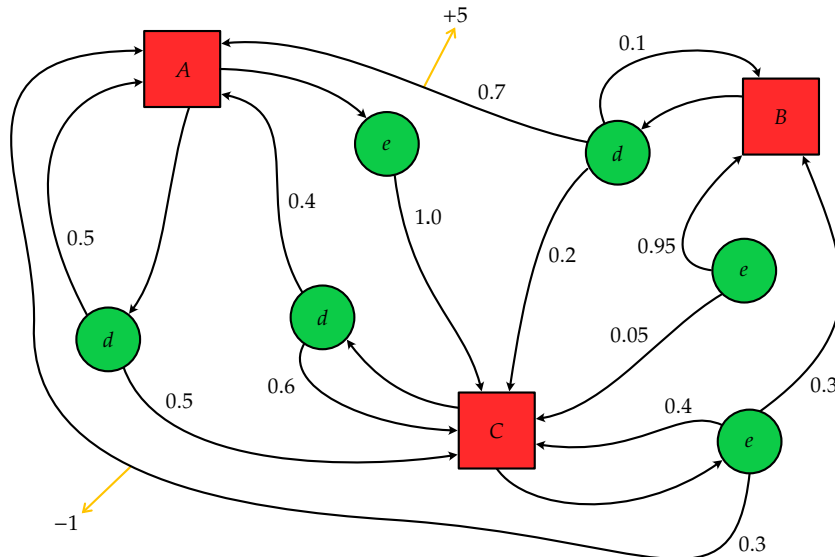


Figure 2.5: Illustration of a simple discrete MDP with  $\mathcal{S} = \{A, B, C\}$ , denoted as red squares, and  $\mathcal{A} = \{d, e\}$ , denoted as green circles. Connections from a state to an action indicate that the action can be taken when being in the state. Connections from actions to states indicate state transitions if the action is taken in the state that points into the action. These transitions are stochastic and labelled with their respective probabilities. Yellow arrows pointing away from a state transition indicate rewards that are received when the transition is taken and are labelled with the reward value. Figure adapted from [341].

are then updated based on the rewards collected by performing Monte Carlo rollouts, i.e. performing each action in each state and observing the rewards received to determine the best action for each state. On the other end of the spectrum, there are value-based methods like Q-learning. These methods learn the value function

$$V_{\pi}(s_t) = \mathbb{E}_{\tau \sim \pi} [R(\tau) \mid s_t], \quad (2.20)$$

which gives the expected cumulative reward when following a policy  $\pi$  from a given state, or the action-value function

$$Q_{\pi}(s_t, a_t) = \mathbb{E}_{\tau \sim \pi} [R(\tau) \mid s_t, a_t], \quad (2.21)$$

which gives the expected cumulative reward when taking action  $a_t$  in state  $s_t$  and then following the policy  $\pi$ . In the case of Q-learning in a discrete tabular setting, the Q-values are updated based on the rewards received when performing Monte Carlo rollouts. The policy is then derived from the Q-values by selecting the action with the highest Q-value in each state.

RL algorithms can also be categorised as on-policy or off-policy. The key difference here is that on-policy methods learn the policy that is used in the rollouts for data collection, while off-policy methods learn a policy that is different from the one used in the rollouts. This is achieved by off-policy methods assuming a fixed policy in the updates of the policy or value function, e.g. the fixed policy of always selecting the action with the highest Q-value in Q-learning. Off-policy algorithms are generally more sample-efficient than on-policy algorithms, because they can reuse and learn from data collected at an earlier stage of training or outside the training process altogether. However, off-policy methods also tend to be less stable than on-policy methods as a result of the mismatch between the policy used for data collection and the policy used for learning.

In practice, the lines between these categories are often blurred, and many algorithms combine elements of policy-based and value-based methods, as well as model-based and model-free methods, and on-policy and off-policy methods. What is more, Monte Carlo rollouts can be impractical for many real-world problems, because they require the agent to interact with the environment for a large number of samples to determine the best policy or value function. Instead, temporal difference learning, which updates the policy or value function based on the rewards received at each time step, is often used. Tabular methods are also not practical for many real-world problems, because the state and action spaces are too large to be stored in a table, especially when state and action spaces are continuous, where the number of possible states and actions is infinite. Modern RL therefore often relies on function approximation, where the policy and value functions are approximated by function approximators, such as neural networks. The class of RL algorithms employing neural networks is sometimes referred to as deep reinforcement learning.

In the following, different deep reinforcement learning algorithms relevant to the work presented in this dissertation are introduced.

### Proximal Policy Optimisation

Proximal Policy Optimisation (PPO) [292] is a policy gradient method and a descendant of the original policy gradient algorithm, also known as *REINFORCE* [319]. Policy gradient methods are a class of RL algorithms that aim to find the optimal parameters  $\theta^*$  of a parameterised stochastic policy  $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$  that maximise the expected cumulative reward

$$\max_{\theta} J(\theta) = \mathbb{E}[R(\boldsymbol{\tau})]. \quad (2.22)$$

To this end, they compute an estimate of the policy gradient, which can then be used to adjust the policy parameters through gradient ascend in the direction that makes actions leading to higher returns more likely. The most commonly used policy gradient estimator is

$$\nabla_{\theta} J(\theta) = \hat{\mathbb{E}}_t [\nabla_{\theta} \log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) \cdot \hat{A}_t], \quad (2.23)$$

where  $\hat{\mathbb{E}}_t$  denotes the empirical average over a batch of trajectories and  $\hat{A}_t$  is an estimate of the advantage  $A_t$ . The advantage  $A_t$  is a measure of how much better an action is than was expected, given the state and the policy. It is used here instead of the return  $R(\boldsymbol{\tau})$  and defined as the difference between the observed return and an estimate of the value function  $\hat{V}_{\pi_\theta}(\mathbf{s}_t)$ . That is

$$A_t \approx \hat{A}_t = R(\boldsymbol{\tau}) - \hat{V}_{\pi_\theta}(\mathbf{s}_t). \quad (2.24)$$

The advantage estimate is attained through the use of a function approximator, such as a neural network. The use of the advantage instead of the return is motivated by the fact that the advantage function reduces the variance of the policy gradient estimator, which often leads to faster and more stable learning.

To compute the policy gradient estimate with automatic differentiation, PPO like other policy gradient methods, defines a surrogate loss function to be differentiated. Note that, unlike in supervised learning, the loss operates not as a measure of policy performance, but merely as a function that has a gradient approximating the policy gradient. A commonly used policy gradient loss function is

$$\mathcal{L}^{\text{PG}}(\theta) = \hat{\mathbb{E}}_t [\log \pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t) \hat{A}_t]. \quad (2.25)$$

However, the large variance of the value estimate  $\hat{V}_t$  and the resulting variance of the advantage estimate  $\hat{A}_t$  can lead to unreasonably large policy updates in potentially wrong directions when multiple gradient steps are taken on this loss function. This causes highly unstable learning, and it is the problem that algorithms like Trust Region Policy Optimisation (TRPO) [291] and PPO [292] aim to solve by constraining the magnitude of the policy updates.

Firstly, PPO employs a modified conservative policy iteration (CPI) policy gradient loss function [166], by defining a shorthand

$$\eta_t(\theta) = \frac{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)}{\pi_{\theta_{\text{old}}}(\mathbf{a}_t | \mathbf{s}_t)} \quad (2.26)$$

for the ratio of how much more likely the action is under the new policy than under the old policy. This is also known as the *TRPO ratio* for its original use in the TRPO algorithm. Note that Schulman et al. denote the TRPO ratio as  $r_t(\theta)$ , but to avoid confusion with the reward, it is denoted as  $\eta_t(\theta)$  here. The log-likelihood in the original policy gradient loss function is replaced by the TRPO ratio  $\eta_t(\theta)$  to obtain the CPI loss function

$$\mathcal{L}_t^{\text{CPI}}(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)}{\pi_{\theta_{\text{old}}}(\mathbf{a}_t | \mathbf{s}_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t [\eta_t(\theta) \hat{A}_t]. \quad (2.27)$$

Updates to the policy parameters are then constrained in the final PPO policy gradient loss function

$$\mathcal{L}_t^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[ \min \left( \eta_t(\theta) \hat{A}_t, \text{clip}(\eta_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right] \quad (2.28)$$

by clipping the probability ratio  $\eta_t(\theta)$  to lie within a narrow interval around 1. This constrains the policy updates to be small, which stabilises learning and prevents the policy from diverging. Note that this is actually the policy gradient loss function for the most commonly used variant of PPO, called *PPO-Clip*. The original authors also propose an alternative policy gradient loss function using a penalty term on the Kullback-Leibler (KL) divergence between the old and new policy, called *PPO-KL*, but find that PPO-Clip generally performs better in practice [292].

The final PPO loss function also includes terms for the loss of the value function estimator  $\hat{V}_{\pi_\theta}(\mathbf{s}_t)$ , defined as

$$\mathcal{L}_t^{\text{VF}}(\theta) = \hat{\mathbb{E}}_t \left[ \left( \hat{V}_{\pi_\theta}(\mathbf{s}_t) - R(\boldsymbol{\tau}) \right)^2 \right] \quad (2.29)$$

to minimise the error between the estimated values  $\hat{V}_{\pi_\theta}(\mathbf{s}_t)$  and the observed returns  $R(\boldsymbol{\tau})$  for trajectories starting in state  $\mathbf{s}_t$ , as well as an entropy term  $S[\pi_\theta]$  to encourage exploration. The resulting PPO loss function with hyperparameters  $c_1$  and  $c_2$  for weighting the value function loss and the entropy term is then defined as

$$\mathcal{L}_t^{\text{CLIP+VF+S}}(\theta) = \hat{\mathbb{E}}_t \left[ \mathcal{L}_t^{\text{CLIP}}(\theta) - c_1 \mathcal{L}_t^{\text{VF}}(\theta) + c_2 S[\pi_\theta](\mathbf{s}_t) \right]. \quad (2.30)$$

The PPO algorithm alternates between collecting new trajectories from  $N$  parallel environments, and updating the policy and value function parameters to minimise the loss function, as is defined in Algorithm 1.

---

**Algorithm 1** Actor-critic style PPO algorithm [292].

---

```

for iteration = 1, 2, ... do
  for actor = 1, 2, ..., N do
    Run policy  $\pi_{\theta_{\text{old}}}$  in environment for  $T$  time steps
    Compute advantage estimates  $\hat{A}_1, \dots, \hat{A}_T$ 
  end for
  Optimise surrogate  $\mathcal{L}$  w.r.t.  $\theta$ , with  $K$  epochs and mini-batch size  $M \leq NT$ 
   $\theta_{\text{old}} \leftarrow \theta$ 
end for

```

---

## Deep Deterministic Policy Gradients

Deep deterministic policy gradients (DDPG) [2, 197] is an actor-critic-style off-policy value-based method, which learns a deterministic policy  $\pi_{\theta}(s_t)$  that maps states to actions. To learn this policy, DDPG has two components: a Q-learning component and a policy learning component.

The Q-learning component in DDPG is closely related to the deep Q-network (DQN) algorithm [223]. It relies on the idea that, if the optimal action-value function  $Q^*(s_t, a_t)$  is known, the optimal policy can be obtained by selecting the action that maximises the action-value function, i.e.

$$\pi^*(s_t) = \underset{a_t}{\operatorname{argmax}} Q^*(s_t, a_t). \quad (2.31)$$

The Q-learning component in DDPG approximates the optimal action-value function  $Q^*(s_t, a_t)$  using a neural network function approximator  $Q_{\phi}(s_t, a_t)$ . To learn the parameters  $\phi$  of the Q-network, DDPG relies on the Bellman equation for the optimal action-value function

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim P} \left[ \mathcal{R}(s_t, a_t) + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \right]. \quad (2.32)$$

Given a mini-batch  $\mathcal{B}$  of transitions  $(s_t, a_t, r_t, s_{t+1}, d)$ , where  $d = 1$  if the next state is terminal and  $d = 0$  otherwise, the Q-learning component in DDPG minimises a mean-squared Bellman error (MSBE) loss

$$\mathcal{L}(\phi, \mathcal{D}) = \mathbb{E}_{(s_t, a_t, r_t, s_{t+1}, d) \sim \mathcal{D}} \left[ \left( Q_{\phi}(s_t, a_t) - \left( r_t + \gamma (1 - d) \max_{a_{t+1}} Q_{\phi}(s_{t+1}, a_{t+1}) \right) \right)^2 \right], \quad (2.33)$$

which measures how well  $Q_{\phi}$  satisfies the Bellman equation.

Prior to the introduction of DQN, using neural networks to approximate the Q-function was found to be unstable. This is solved by two main techniques introduced in the original DQN paper: experience replay and target networks [223].

Experience replay is a technique that stores transitions  $(s_t, a_t, r_t, s_{t+1}, d)$  in a replay buffer  $\mathcal{D}$  and samples mini-batches  $\mathcal{B}$  of transitions from the buffer to update the Q-network. This is found to stabilise learning by breaking the temporal correlations between consecutive transitions and reducing the variance of the updates. Note that the replay buffer must be large enough to store a diverse set of transitions, and it is specifically intended that this experience was collected by policies at different stages of learning. Unlike with on-policy methods, like PPO, this is allowed as a result of the Q-learning component assuming a fixed policy that acts greedily, making DDPG an off-policy method.

Target networks are a technique that uses two copies of the Q-network, the target network  $Q_{\phi^-}$  and the online network  $Q_{\phi}$ . The problem is that the second term in the loss function, the target

$$r_t + \gamma (1 - d) \max_{\mathbf{a}_{t+1}} Q_{\phi}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \quad (2.34)$$

estimated by the Q-network, depends on the parameters  $\phi$  of the Q-network itself and changes as the Q-network is updated. This makes the loss minimisation unstable. The solution is to use a target network  $Q_{\phi^-}$ , which is a delayed copy of the online network  $Q_{\phi}$ , in the loss function. Note that the same is done for the policy network, with the policy being replaced by the target policy in the MSBE loss. The revised loss is given as

$$\mathcal{L}(\phi, \mathcal{D}) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1}, d) \sim \mathcal{D}} \left[ \left( Q_{\phi}(\mathbf{s}_t, \mathbf{a}_t) - (r_t + \gamma (1 - d) Q_{\phi^-}(\mathbf{s}_{t+1}, \pi_{\theta^-}(\mathbf{s}_{t+1}))) \right)^2 \right]. \quad (2.35)$$

In the DQN algorithm that DDPG is derived from, the target networks are updated periodically by copying the parameters of the online networks every constant number of steps [223]. The DDPG algorithm, however, uses soft so-called Polyak updates to update the target networks' parameters in every step following the rules

$$\phi^- \leftarrow \tau_p \phi + (1 - \tau_p) \phi^- \quad \text{and} \quad \theta^- \leftarrow \tau_p \theta + (1 - \tau_p) \theta^-, \quad (2.36)$$

where  $\tau_p$  is the Polyak update rate, which is typically set to a small value and controls the rate at which the target networks are updated.

The policy learning component in DDPG solves a limitation of the original DQN algorithm, where the policy in DQN is defined as greedily choosing the action that maximises the Q-value. This is fine in discrete action spaces, where the number of possible actions is finite and can easily be maximised through a brute force search over all actions. With continuous action spaces, where the number of possible actions is infinite, however, finding the action that is guaranteed to maximise the Q-value is impossible, and performing an optimisation over the action space every time an action is selected is computationally intractable.

Instead, DDPG learns the parameters  $\theta$  of a deterministic policy network  $\pi_{\theta}(\mathbf{s}_t)$ , such that the latter approximates the action that maximises the Q-value. Because the Q-value is itself approximated by a neural network, this is relatively straightforward by performing gradient ascent with respect to the policy parameters to solve the optimisation problem

$$\max_{\theta} \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[ Q_{\phi}(\mathbf{s}_t, \pi_{\theta}(\mathbf{s}_t)) \right]. \quad (2.37)$$

Because the policy in DDPG is deterministic, exploration during rollouts for data collection is achieved by adding noise in the form of a temporally correlated Ornstein-Uhlenbeck process to the action selected by the policy. In some more recent implementations, this is sometimes replaced by uncorrelated zero-mean Gaussian noise, which is simpler to implement and has been found to perform similarly well in practice.

The complete DDPG algorithm is shown in Algorithm 2. In every iteration, the agent collects experience by interacting with the environment and stores the transitions in the replay buffer. Every  $N$  steps, the agent performs a number of updates. For each update, the agent samples a mini-batch of transitions from the replay buffer, and updates the Q-network and the policy network using the loss functions defined above. At the end of each update, the target networks are updated using the Polyak update rule.

---

**Algorithm 2** DDPG algorithm [2, 197].

---

**Require:** Initial policy parameters  $\theta$ , Q-network parameters  $\phi$ , empty replay buffer  $\mathcal{D}$   
 Set target parameters equal to the online parameters  $\theta^- \leftarrow \theta, \phi^- \leftarrow \phi$

**repeat**

Observe state  $s_t$  and select action  $a_t = \text{clip}(\pi_\theta(s_t) + \epsilon, a_{\text{low}}, a_{\text{high}})$ , where  $\epsilon \sim \mathcal{N}$

Execute action  $a_t$  in the environment

Observe next state  $s_{t+1}$ , reward  $r_{t+1}$  and done signal  $d_{t+1}$

Store  $(s_t, a_t, r_{t+1}, s_{t+1}, d_{t+1})$  in replay buffer  $\mathcal{D}$

**if**  $s_{t+1}$  is terminal **then**

Reset environment state

**end if**

**if** it is time to update **then**

**for** number of updates **do**

Randomly sample mini-batch of transitions  $\mathcal{B} = \{(s_t, a_t, r_t, s_{t+1}, d)\}$  from  $\mathcal{D}$

Compute targets

$$y_i = r_t + \gamma (1 - d_i) Q_{\phi^-}(s_{t+1,i}, \pi_{\theta^-}(s_{t+1,i}))$$

Update Q-network by one step of gradient descent using

$$\nabla_{\phi} \frac{1}{|\mathcal{B}|} \sum_i (Q_{\phi}(s_{t,i}, a_{t,i}) - y_i)^2$$

Update policy by one step of gradient ascent using

$$\nabla_{\theta} \frac{1}{|\mathcal{B}|} \sum_i Q_{\phi}(s_{t,i}, \pi_{\theta}(s_{t,i}))$$

Update target networks using Polyak update rule

$$\phi^- \leftarrow \tau_p \phi + (1 - \tau_p) \phi^-$$

$$\theta^- \leftarrow \tau_p \theta + (1 - \tau_p) \theta^-$$

**end for**

**end if**

**until** convergence

---

### Twin Delayed Deep Deterministic Policy Gradients

Twin delayed deep deterministic policy gradients (TD3) [2, 100] is an extension of the DDPG algorithm that aims to improve the stability and performance of the original algorithm. The key issue with DDPG is that the Q-learning component can overestimate the Q-values, which can lead to policies that exploit these overestimations rather than the true Q-values. To address this issue, TD3 introduces three main improvements to the DDPG algorithm.

Firstly, TD3 performs *target policy smoothing*, which adds noise to the target policy during the Q-value estimation for the purpose of regularisation. That is, the inference of the target policy  $\pi_{\theta^-}(s_{t+1})$  in the MSBE loss is replaced by a clipped and noisy version of the action selected by the target policy

$$\mathbf{a}_{t+1} = \text{clip}(\pi_{\theta^-}(s_{t+1}) + \text{clip}(\epsilon, -c, c), \mathbf{a}_{\text{low}}, \mathbf{a}_{\text{high}}), \quad (2.38)$$

where  $\epsilon \sim \mathcal{N}(0, \sigma)$  is a noise term sampled from a Gaussian distribution with mean 0 and standard deviation  $\sigma$ ,  $c$  is a hyperparameter that controls the magnitude of the noise, and  $\mathbf{a}_{\text{low}}$  and  $\mathbf{a}_{\text{high}}$  are the lower and upper bounds of the legal action space. Smoothing the action selected by the target policy in this way prevents the policy from learning to exploit sharp peaks incorrectly estimated by the Q-network.

Secondly, TD3 introduces *clipped double-Q learning*, i.e. a pair of Q-networks,  $Q_{\phi_1}$  and  $Q_{\phi_2}$ , to estimate the Q-values. In the target term of the MSBE loss, the prediction from the single Q-network in DDPG is replaced by the minimum of the two Q-values

$$y(r_t, \mathbf{s}_{t+1}, d) = r_t + \gamma(1-d) \min_{i=1,2} Q_{\phi_i^-}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}). \quad (2.39)$$

The result is a revised loss formulation

$$\mathcal{L}(\phi_i, \mathcal{D}) = \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1}, d) \sim \mathcal{D}} \left[ (Q_{\phi_i}(\mathbf{s}_t, \mathbf{a}_t) - y(r_t, \mathbf{s}_{t+1}, d))^2 \right] \quad (2.40)$$

for the Q-networks, which prevents the overestimation of the Q-values by always selecting the lower of the two estimates.

Thirdly, TD3 introduces *delayed policy updates*, which update the policy and the target networks less frequently than the Q-networks. This is intended to reduce the variance of the policy updates that results from the effect a changing policy has on the target term in the MSBE loss.

Aside from these three main improvements, TD3 typically only uses uncorrelated zero-mean Gaussian noise for exploration, as opposed to the temporally correlated Ornstein-Uhlenbeck process sometimes used in DDPG.

### 2.1.5 Bayesian Optimisation

Bayesian optimisation (BO) [105] is a model-based optimisation technique that is used to find the global minimum of expensive-to-evaluate black-box functions. Due to its data-driven nature, BO is sometimes considered as a technique that lies in the intersection of traditional optimisation methods and machine learning.

Inspired by Bayesian statistics, BO maintains a probabilistic model of the objective function, which in modern implementations is typically expressed as a Gaussian process (GP). BO starts with a prior distribution over the objective function, which represents

prior beliefs about the function. This prior can be an informed prior, but is often set to be uninformative, such as a zero mean and a large variance. The prior is then updated as new data points are collected, which in BO are the evaluations of the objective function. The updated distribution is called the posterior distribution. Based on the posterior distribution, BO computes an acquisition function, which is a utility function that quantifies how useful it would be to evaluate the objective function at a given point. New points are then selected by finding the maximum of the acquisition function, which is typically designed to balance exploration and exploitation.

This general framework leaves open a number of design choices. Informed priors, for example in the form of expert knowledge or previous evaluations of the objective function, can be used to speed up the optimisation process. The choice of the GP kernel and the hyperparameters of the GP can also have a significant impact on the performance of BO. The acquisition function is another design choice that impacts the exploration-exploitation trade-off and can be tailored to the specific properties of the objective function.

A simple pseudocode implementation of BO is provided in Algorithm 3.

---

**Algorithm 3** BO algorithm.

---

**Require:** Objective function, prior mean function, initial set of data points, acquisition function  
**repeat**  
    Fit model to the set of data points with the prior mean function  
    Compute the acquisition function based on the model  
    Select new point by maximising acquisition function  
    Evaluate the objective function at the new point  
    Add new data point to the set of data points  
**until** convergence

---

## 2.2 Particle Accelerators

Particle accelerators are machines that accelerate charged particles to high speeds and energies [350]. They are used in a wide range of scientific and industrial applications, including fundamental research in particle physics, medical applications and industrial processes. Accelerators are typically composed of five main components: a particle source, an acceleration system, a beam transport system, a beam diagnostics system and an experimental area or target that uses the accelerated particles. In the following these components, their basic principles as well as some common applications of accelerators and some facilities are introduced. A simplified overview of an accelerator and its components is shown in Fig. 2.6. Photos of some relevant accelerator components are shown in Fig. 2.7. Note that the following is by no means an exhaustive overview of particle accelerators, but rather a brief introduction to the topic, focussing on the aspects most relevant to the work presented in this dissertation.

### 2.2.1 Overview

The particle source is responsible for producing the particles that are to be accelerated. This can be done in a number of ways, depending on the type of particles. In electron

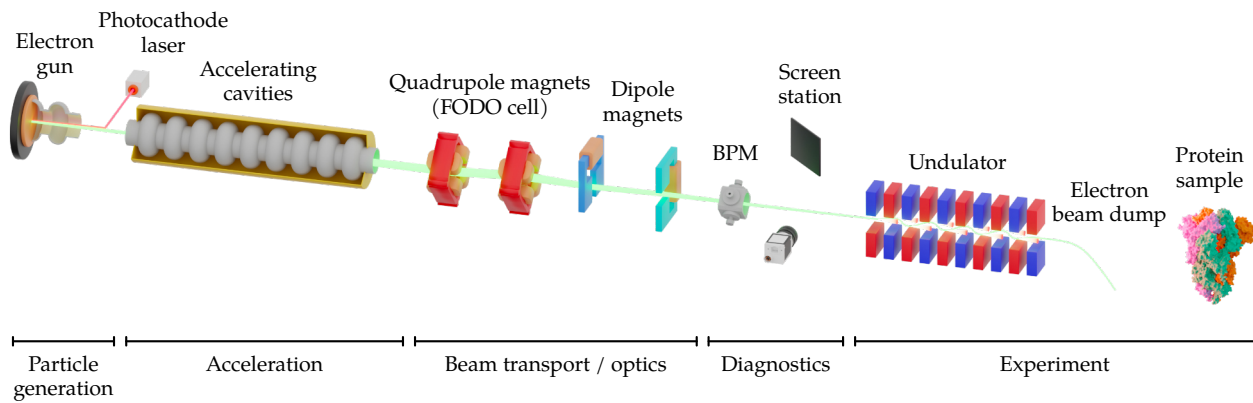


Figure 2.6: Simplified example of a particle accelerator with its main subsystems and components at the example of a linear accelerator used as an FEL. The gun with the photocathode and cathode laser are shown on the left, followed by the acceleration section with radio frequency (RF) cavities, a beam transport system consisting of two quadrupoles arranged as a commonly used FODO cell and two dipoles for steering, a beam diagnostics section with a beam position monitor (BPM) and a screen station, followed finally by an undulator producing the FEL radiation, which is in this example used to image a SARS-CoV-2 spike glycoprotein (PDB ID: 6VXX) [342, 343].

accelerators, like the ones mainly considered in this dissertation, the particles are produced by a so-called *laser cathode* that emits electrons when illuminated by a laser. This laser is pulsed, such that the electrons are emitted in short bursts, resulting in electron clouds, often referred to as *bunches*. The cathode is made from materials with a low work function, such as copper or caesium telluride, which allows the electrons to be emitted with a low energy input.

Once produced, the particles are then accelerated by an acceleration system. This system typically consists of a series of RF cavities that are used to accelerate the particles by applying an oscillating electric field driven by a high-frequency RF source. The frequency of the RF field is tuned to match the resonant frequency of the cavities, which allows the particles to gain energy as they pass through each cavity. The energy gain is determined by the phase and amplitude of the RF field. An image of an RF cavity is shown in Fig. 2.7.

The accelerated particles are then transported through the beam transport system, which is responsible for guiding the particles to the experimental area and ensuring that the particle beam has the desired properties when it arrives. The beam transport system is composed of various magnets that are used to steer and focus the beam of particles with the help of magnetic fields affecting the charged particles as a result of the Lorentz force. Dipole magnets are used to steer the beam in either the horizontal or vertical plane, ensuring that the particles remain on the desired trajectory. Some of these dipole magnets are placed along bends in the beamline to ensure that the beam follows the same path as the beamline, while others are used to correct for small deviations in the beam's trajectory. The former are often referred to as *bending magnets*, while the latter are called *steering magnets*. The strength of the magnetic field in the dipole magnets determines the amount of deflection the beam experiences. This is often measured in either the current applied to the magnet, the strength of the magnetic field the particles experience or the angle of deflection exerted on the beam. Quadrupole magnets act on the beam in a way comparable to the effect an optical lens has on light. Unlike an optical lens, a quadrupole magnet always focuses the beam in one plane and defocuses it in the other. Arranging multiple

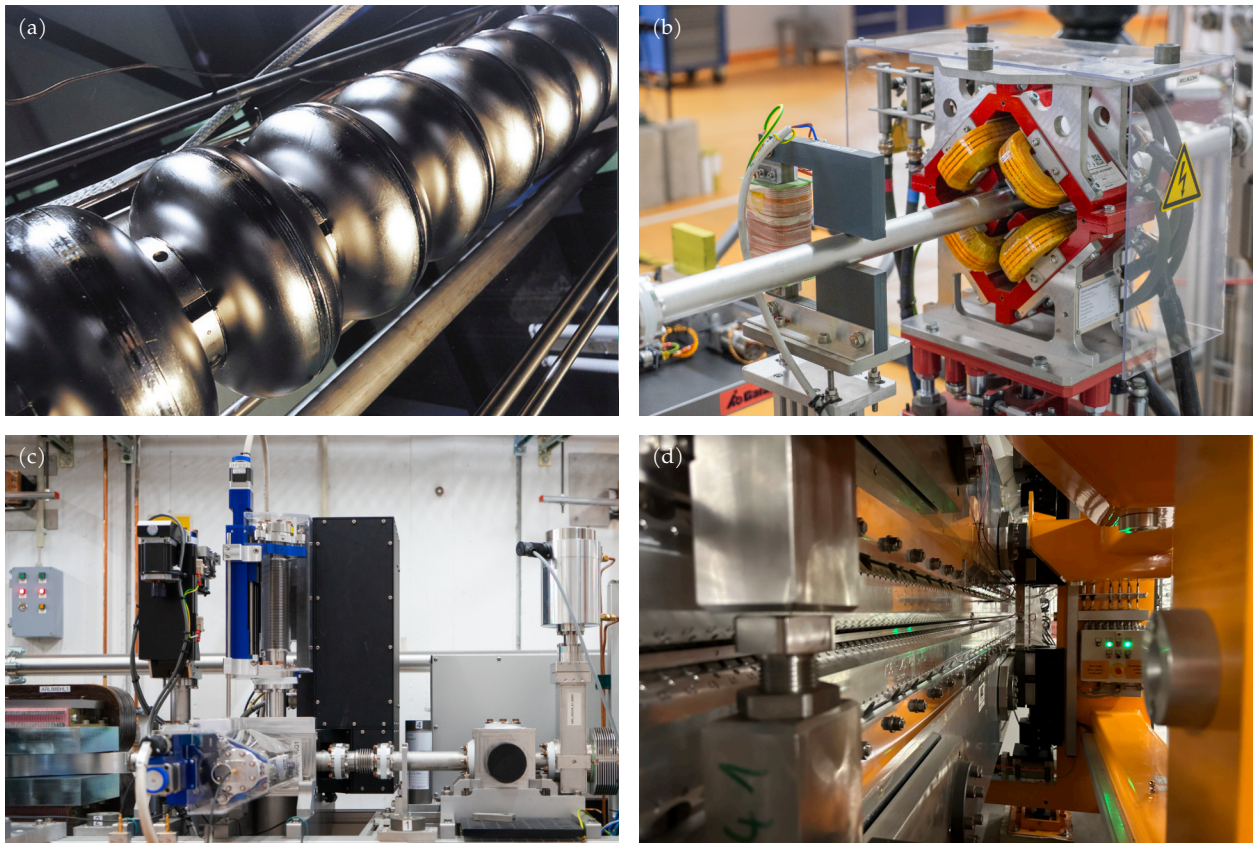


Figure 2.7: Photos of some relevant accelerator components: (a) Niobium accelerating cavities of the type used in superconducting linear accelerators like Free-Electron Laser in Hamburg (FLASH) and European X-Ray Free-Electron Laser (European XFEL) [60]. (b) A horizontal dipole steering magnet and a quadrupole magnet at the Accelerator Research Experiment at SINBAD (ARES) linac. (c) A diagnostic screen station at the ARES linac with the beam pipe oriented from right to left in the bottom third, the blue vertical stage that is used to move the screen in and out of the beamline, and the black optics box with the camera in it to its right. (d) A section of an undulator beamline at European XFEL, looking in upstream direction.

quadrupole magnets along the beamline allows for the beam to be focussed in both planes. Both a dipole steering magnet and a quadrupole magnet are pictured in Fig. 2.7. Other magnets are sometimes used in accelerators to correct for higher-order effects in the beam. Sextupole magnets, for example, are used to correct the effects of higher-order aberrations, which can cause the beam to become unstable and lose focus. However, these are not relevant for the work presented in this dissertation.

In order to act on the beam and ensure that it fulfils the experimental requirements, various beam diagnostic systems are used. The most common ones include beam position monitors (BPMs), beam loss monitors (BLMs) and diagnostic screens. BPMs use four electrodes inserted into the beam pipe to measure the electromagnetic fields generated by a passing charged particle beam. This measurement can be used to determine the transverse position of the beam as well as its current, and is an important feedback signal of the beam's trajectory. BLMs are typically placed around the beam pipe and used to measure the radiation generated by the beam. This radiation can be used to detect beam losses, that is particles that have significantly deviated from the desired trajectory and are

hitting the walls of the beam pipe. Beam loss information is important for troubleshooting when the beam is not behaving as expected, but also constitutes an important part of the safety system of an accelerator, as large beam losses can cause damage to the accelerator components and pose a radiation hazard. Diagnostic screens are used to visualise the beam in the transverse plane, allowing operators to understand its position, size and shape. Unlike BPMs and BLMs, diagnostic screens are typically not used for real-time feedback, because they are a destructive measurement that blocks the beam from reaching the experiment or another diagnostic downstream. Diagnostic screens, often referred to as just screens, are typically used for commissioning and tuning the accelerator, as well as for diagnosing beam issues that are not easily detected by other diagnostic systems. They work by inserting a small sheet of scintillating material into the beam pipe, which emits light when hit by a charged particle. A camera is then used to capture an image of the lit-up screen, which can be shown to the operators or analysed computationally to determine the beam's properties. A key advantage of diagnostic screens is that they also provide information about the beam's substructure, such as the presence of beam halos or tails, which can be important for understanding the beam's behaviour. A photo of a diagnostic screen station can be seen in Fig. 2.7. Occasionally, more complex diagnostics are designed based on these simpler diagnostics. A common one is used to measure the longitudinal phase space (LPS) of the beam, which provides information about the longitudinal distribution of the particles in space and their energy spread. In a single, so-called transverse deflecting cavity, the phase of the RF field is configured such that the bunch is rotated  $90^\circ$  around the transverse horizontal axis. This allows the bunch to be imaged longitudinally on downstream diagnostic screens.

Finally, the accelerated particles are used in the experiment. What the experiment looks like can vary greatly and depends on the intended purpose of the accelerator. In fundamental physics research, for example, accelerators are often used as colliders, where two beams of particles are collided head-on to study the properties of the particles and the fundamental forces that govern their interactions. Colliders can either be straight, where two linear accelerators are used to accelerate the particles in opposite directions, or circular, where the particles are accelerated in a ring and then collided at specific interaction points along the ring. Examples of colliders include the Large Hadron Collider (LHC) at Conseil Européen pour la Recherche Nucléaire (CERN), the Relativistic Heavy Ion Collider (RHIC) at Brookhaven National Laboratory (BNL) and Hadron-Elektron-Ringanlage (HERA) at Deutsches Elektronen-Synchrotron (DESY). Another type of accelerator are so-called light sources, which produce intense beams of light that are used for a wide range of applications, including materials science, biology and chemistry. Light sources exist in two main types: synchrotron radiation sources, which are typically circular accelerators that produce spreads of light in the form of synchrotron radiation by bending the beam of particles around a ring, and FELs, which are typically linear accelerators that produce light by passing the beam of particles through a series of alternating permanent magnets, known as undulators. The magnets in an undulator deflect the particle beam onto a sinusoidal path, causing it to emit a narrow beam of light in the form of synchrotron radiation. Examples of synchrotron radiation sources include the European Synchrotron Radiation Facility (ESRF) and the Positron-Elektron-Tandem-Ringanlage (PETRA) III at DESY. FELs in operation around the world include FLASH and European XFEL at DESY, the Linac Coherent Light Source (LCLS) at SLAC National Accelerator Laboratory (SLAC) and the Swiss Free Electron Laser (SwissFEL) at Paul Scherrer Institute (PSI). Some accelerators also do not have an experiment as such, but rather supply beams of particles to other facilities. Circular

accelerators, for example, require a smaller often linear pre-accelerator that is used to inject particles into the ring. These are typically combined into elaborate accelerator complexes, where particles are accelerated through multiple stages before being used in experiments. One such example was the HERA accelerator at DESY, which was supplied by a sequence of accelerators including the original PETRA and DESY rings, and a linear accelerator called LINAC. Similarly, the LHC at CERN is supplied by a sequence of accelerators including the Super Proton Synchrotron (SPS), Proton Synchrotron (PS) and another linear accelerator called LINAC. Most of the aforementioned facilities are user facilities, meaning that they are open to researchers from around the world to conduct experiments, and have relatively fixed and elaborate experimental setups. Some (usually smaller) accelerators like ARES, however, are used for accelerator research and development, typically providing a number of often-reconfigured experimental areas for testing new accelerator components and concepts.

In the following, three particle accelerator facilities relevant to the work presented in this dissertation are introduced. For two of these accelerators, ARES and LCLS, tuning tasks are introduced that are used as case studies for developing autonomous tuning algorithms with RL in this dissertation.

### 2.2.2 ARES

The Accelerator Research Experiment at SINBAD (ARES) particle accelerator [47, 246], located at DESY in Hamburg, Germany, is an S-band radio frequency linac with a photoinjector and two independently driven travelling wave accelerating structures operating at energies up to 155 MeV. It started operations in 2021 and is about 45 m long. The primary research focus of ARES is to produce and study sub-femtosecond electron bunches at relativistic energies. The ability to generate such short bunches is of great interest for applications such as radiation generation by FELs. ARES is also used for accelerator component research and development, such as the development of dielectric structures for beam acceleration and manipulation in the Accelerator-on-a-Chip (ACHIP) program, as well as medical applications, such as very-high-energy-electron treatment of tumours and electron computer tomography. An overview of ARES and the sections of it considered in this work is shown in Fig. 2.8. The location of ARES on the DESY campus in Hamburg is shown in Fig. 2.9. Photos from inside the ARES tunnel are shown in Fig. 2.10.

#### Transverse Beam Tuning

In Chapter 4 of this dissertation, a recurring transverse beam tuning task at ARES is considered as a case study for developing autonomous tuning algorithms. It was originally considered in [78, 158, 161, 164]. This kind of task is ubiquitous across linear particle accelerators, frequently performed during start-up and operation mode changes, and time-consuming to perform manually. What is more, it is a well-defined and well-understood task in the accelerator community, such that discovered solutions can be easily understood, evaluated and validated. At the same time, the task is complex and non-linear enough to challenge the capabilities of optimisation algorithms and provide a meaningful benchmark for the performance of autonomous tuning algorithms. Solving it would provide a valuable streamlining of accelerator operations because similar transverse tuning tasks can be found at most accelerator facilities and have to be regularly performed during everyday operations.

Specifically, there are four sections at the ARES particle accelerator where the transverse beam tuning task is commonly performed: the Experimental Area (EA), Bunch Compressor

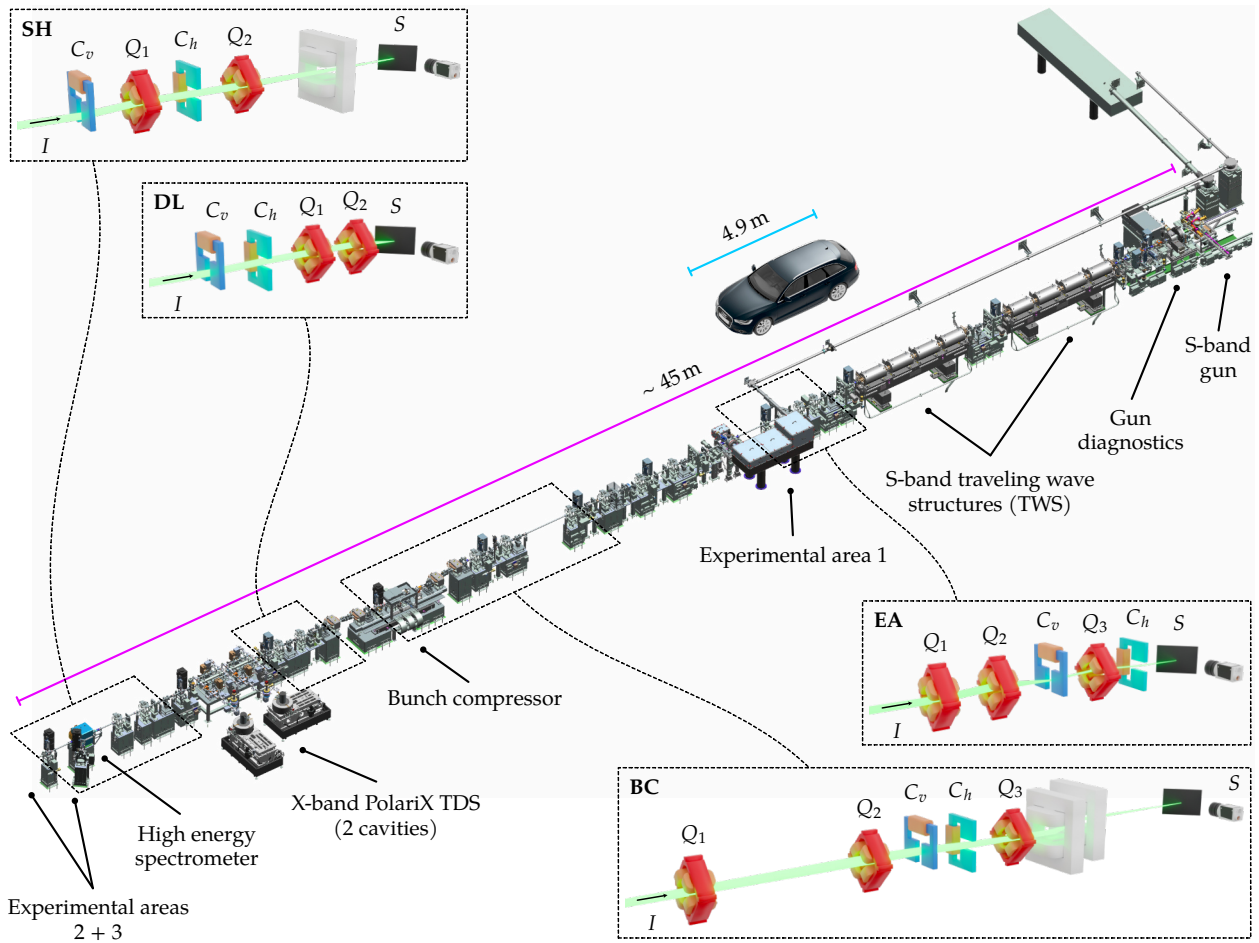


Figure 2.8: CAD-based to-scale overview of the ARES particle accelerator. The Experimental Area (EA), Bunch Compressor (BC), Diagnostic Line (DL) and Spectrometer High Energy (SH) sections considered for the transverse beam tuning tasks are marked, and conceptual not-to-scale overviews of the important components of their lattices are shown. In the latter, quadrupole magnets marked  $Q$  are indicated in red, vertical steering magnets marked  $C_v$  are indicated in blue, horizontal steering magnets marked  $C_h$  are indicated in turquoise, dipole bending magnets are indicated in white, and diagnostic screens are marked  $S$ . Other important parts of the ARES accelerator are annotated, and a car is shown for scale. CAD drawing courtesy of Frank Mayet.

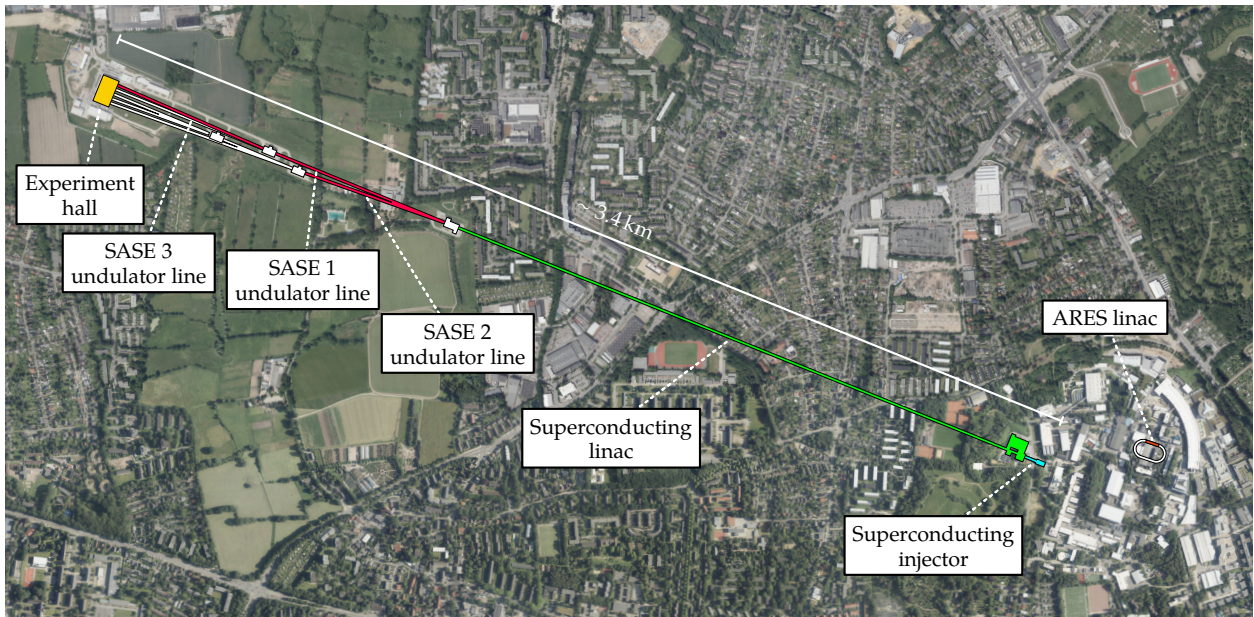


Figure 2.9: Areal view of the ARES and European XFEL facilities in Hamburg, Germany. ARES is highlighted in orange inside the former DORIS tunnel on the DESY campus in the south-easterly corner of the pictured area. The different sections, the injector (turquoise), linac (green), undulator lines (red) and experiment hall (yellow) of European XFEL are highlighted and annotated as well. Orthoimagery from [99].

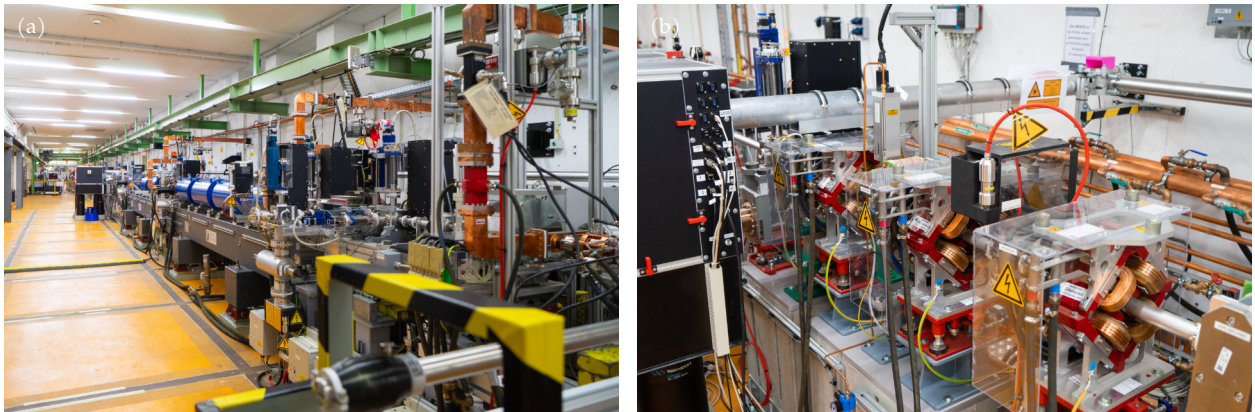


Figure 2.10: Photos from inside the ARES tunnel (former DORIS tunnel). (a) View of ARES looking in downstream direction from the gun. Eight blue solenoid magnets can be seen placed around the accelerating travelling wave structures, with three RF wave guides coupled into the gun and the two travelling wave structures. The black laser table of the Experimental Area 1 is visible in the background. (b) View of the EA section, starting from the red quadrupole magnet in the bottom right, and ending at the vertical black optics box and blue mover stage of the diagnostic screen station in the upper left next to the laser table.

(BC), Diagnostic Line (DL) and Spectrometer High Energy (SH). In each of these sections, there is a sequence of quadrupole magnets  $Q$ , which are used to focus the beam, and vertical and horizontal dipole steering magnets  $C_v$  and  $C_h$ , which are used to bend the beam in the horizontal and vertical direction, followed by a diagnostic screen station  $S$  capturing a transverse image of the electron beam. The lattices of the relevant elements in these sections are in downstream order excluding elements that are not quadrupoles, dipoles or diagnostic screen stations:

$$\begin{aligned}
 \text{EA} : & \quad (Q_1, Q_2, C_v, Q_3, C_h, S), \\
 \text{BC} : & \quad (Q_1, Q_2, C_v, C_h, Q_3, S), \\
 \text{DL} : & \quad (C_v, C_h, Q_1, Q_2, S), \\
 \text{SH} : & \quad (C_v, Q_1, C_h, Q_2, S),
 \end{aligned} \tag{2.41}$$

and illustrated in Fig. 2.8. A photo of the EA section is shown in Fig. 2.10.

In this work, quadrupole magnets are controlled through their focussing strength  $k$  in  $\text{m}^{-2}$  and dipole magnets through the steering angle  $\alpha$  in mrad by which the particles are deflected. The power supplies of all magnets can be switched in polarity. The quadrupole magnets can be actuated up to a focussing strength of  $72 \text{ m}^{-2}$ . The limit of the steering magnets is  $6.2 \text{ mrad}$ . Note that turning up the strength of a quadrupole magnet will focus the beam in the horizontal plane and defocus it in the vertical plane, while turning down the strength will have the opposite effect. Increasing the steering angle of the vertical steering magnet will steer the beam upwards, while decreasing the angle will steer the beam downwards. The horizontal steering magnet works similarly, steering the beam to the right when the angle is increased and to the left when the angle is decreased. What is more, if the beam is off-centre as it passes through the quadrupole magnet, the beam will additionally experience an angular deflection as it would with a dipole magnet. Any tuning task involving quadrupoles is therefore complex. The transverse beam parameters of beam position  $(\mu_x, \mu_y)$  and beam size  $(\sigma_x, \sigma_y)$  are computed from the screen images by fitting a 2-dimensional Gaussian distribution. The camera observes an area of about  $8 \text{ mm}$  by  $5 \text{ mm}$  at a resolution of  $2448$  by  $2040$  pixels. The effective resolution of the scintillating screen is ca.  $20 \mu\text{m}$ .

The goal of the transverse tuning task is to adjust the quadrupole magnets' focussing strengths  $k$  and steering magnets' steering angles  $\alpha$  to minimise the difference between the measured transverse beam parameters  $\mathbf{b}$  and the target transverse beam parameters  $\mathbf{b}'$  chosen by a human operator. The actuators, i.e. the magnet settings to be changed during tuning, are denoted as  $\mathbf{u}$ , where for example in the EA section  $\mathbf{u} = (k_{Q_1}, k_{Q_2}, \alpha_{C_v}, k_{Q_3}, \alpha_{C_h})$ . The tuning optimisation problem can be formalised as minimising the objective

$$\min_{\mathbf{u}} O(\mathbf{u} \mid \mathcal{B}_{\text{in}}, \mathbf{m}), \tag{2.42}$$

where

$$O(\mathbf{u} \mid \mathcal{B}_{\text{in}}, \mathbf{m}) = \mathcal{D}(\mathbf{b}, \mathbf{b}'), \tag{2.43}$$

with  $\mathcal{D}$  a distance metric between the target beam parameters  $\mathbf{b}'$  and the observed beam parameters  $\mathbf{b}$ . The observed beam parameters  $\mathbf{b}$  are a result of the beam dynamics, which in turn depend on the actuators  $\mathbf{u}$ , and environmental factors, such as the misalignments  $\mathbf{m}$  of the quadrupole magnets and the diagnostic screen, as well as the incoming beam  $\mathcal{B}_{\text{in}}$  to the accelerator section. Together with the target beam parameters  $\mathbf{b}'$ , these define the *state* of the environment. To this end, it is assumed that there exists a function

$$\mathbf{b} = f_b(\mathbf{u} \mid \mathcal{B}_{\text{in}}, \mathbf{m}) \tag{2.44}$$

that maps the actuator values  $\mathbf{u}$  currently set on the magnets to beam parameters  $\mathbf{b}$  measured on the diagnostic screen. This function is stochastically affected by random noise and other unknown variables. Two unknown variables are considered in the modelling of  $f_b$ : the incoming beam  $\mathcal{B}_{\text{in}}$ , and the misalignments  $\mathbf{m}$  of the quadrupole magnets and the diagnostic screen. In the real world, neither the incoming beam  $\mathcal{B}_{\text{in}}$  nor the misalignments  $\mathbf{m}$  are known. Other disturbances affecting  $f_b$  are present in the real world, but neither known nor modelled. As a result  $f_b$  is considered a stochastic black-box function. With most real-world tuning tasks, the state is only *partially observable*. In the case of the benchmark task, the magnet misalignments and the incoming beam cannot be easily measured or controlled, and are therefore part of the environment's hidden state.

As a measure of difference between the observed beam parameters  $\mathbf{b}$  and the target beam parameters  $\mathbf{b}'$ , the mean absolute error (MAE) is used and defined as

$$\mathcal{D}_{\text{mae}}(\mathbf{b}, \mathbf{b}') = \frac{1}{4} \sum_{i=1}^4 |b_i - b'_i|, \quad (2.45)$$

i.e. the mean of the absolute value of the beam parameter differences over all four beam parameters, where  $b_i$  denotes the  $i$ -th component of  $\mathbf{b}$ .

### 2.2.3 European XFEL

The European X-Ray Free-Electron Laser (European XFEL) [58] is an FEL located at DESY in Hamburg, Germany. Originally conceptualised as part of the never-built Tera-Electronvolt Energy Superconducting Linear Accelerator (TESLA) facility, the European XFEL began operation in 2017, becoming the world's brightest X-ray light source at the time. The tunnel of European XFEL has a total length of 3.4 km, starting at the DESY campus in Hamburg and extending to the town of Schenefeld in Schleswig-Holstein. European XFEL is based on a superconducting linear accelerator operating at 1.3 GHz and at temperatures of 2 K, accelerating electrons to energies of up to 17.5 GeV before they are sent through a pair of two undulator lines, 175 m and 105 m in length, which produce X-ray pulses at wavelengths from 4.7 nm to below 0.05 nm and a repetition rate as fast as 27 kHz to several experimental stations. An areal view of the European XFEL facility is shown in Fig. 2.9 and views from inside the tunnel are shown in Fig. 2.11. European XFEL is subject of work on start-to-end neural network surrogate modelling for virtual diagnostics in Chapter 3 of this dissertation. An overview of the components relevant for this work is shown in Fig. 3.1.

### 2.2.4 LCLS and LCLS-II

The Linac Coherent Light Source (LCLS) [83] is an FEL located at the SLAC National Accelerator Laboratory in Menlo Park, California, USA. Operating since 2009, LCLS used a new high-brightness electron injector and the final kilometre of the original 3 km-long SLAC copper linac from 1966 to accelerate electrons to energies between 3.5 GeV and 15 GeV, before sending them via a 350 m-long transfer line to a 132 m-long undulator line delivering X-ray pulses at wavelengths of 2.2 nm to 0.12 nm and a repetition rate of 120 Hz to several experimental stations.

In 2023, first lasing was achieved at an upgrade of the facility, called LCLS-II [34, 102]. The upgrade retains the LCLS copper linac and transfer line, while adding a 4 GeV superconducting linac in the first 700 m of the SLAC linac tunnel, using slightly modified



Figure 2.11: Inside the European XFEL tunnel. (a) The yellow cryomodules can be seen suspended from the ceiling [61]. They contain the superconducting RF cavities used to accelerate the electrons at a temperature of 2 K. A klystron for amplifying the 1.3 GHz RF signal is seen in blue underneath the cryomodules. (b) View of a bunch compressor made up of four large blue dipole magnets. Some quadrupole magnets in red and dipole magnets in blue can be seen upstream of the bunch compressor. (c) Undulator gallery of a self-amplified spontaneous emission (SASE) undulator beamline. The yellow undulator frames can be seen in the windows of the protective wall on the right.

versions of the cryomodules and 1.3 GHz niobium cavities used at European XFEL. In addition, the original LCLS undulators were replaced with two new undulator lines. A soft X-ray line of 21 conventionally designed undulators is connected to the superconducting linac via a bypass line past the Facility for Advanced Accelerator Experimental Tests (FACET) and LCLS copper linacs. The original LCLS undulator connected to the copper linac is replaced with a hard X-ray line of 32 horizontal gap vertical polarisation undulators (HGVPUs). A crossover line allows the superconducting linac to also feed the hard X-ray line. These upgrades allow the facility to produce X-ray pulses with a repetition rate of 1 MHz, while serving two X-ray stations simultaneously, effectively doubling the number of experiments that can be conducted. An overview of the parts of the LCLS-II facility relevant for the work in Chapter 5 is shown in Fig. 2.12, with an areal view of the full LCLS-II facility shown in Fig. 2.13.

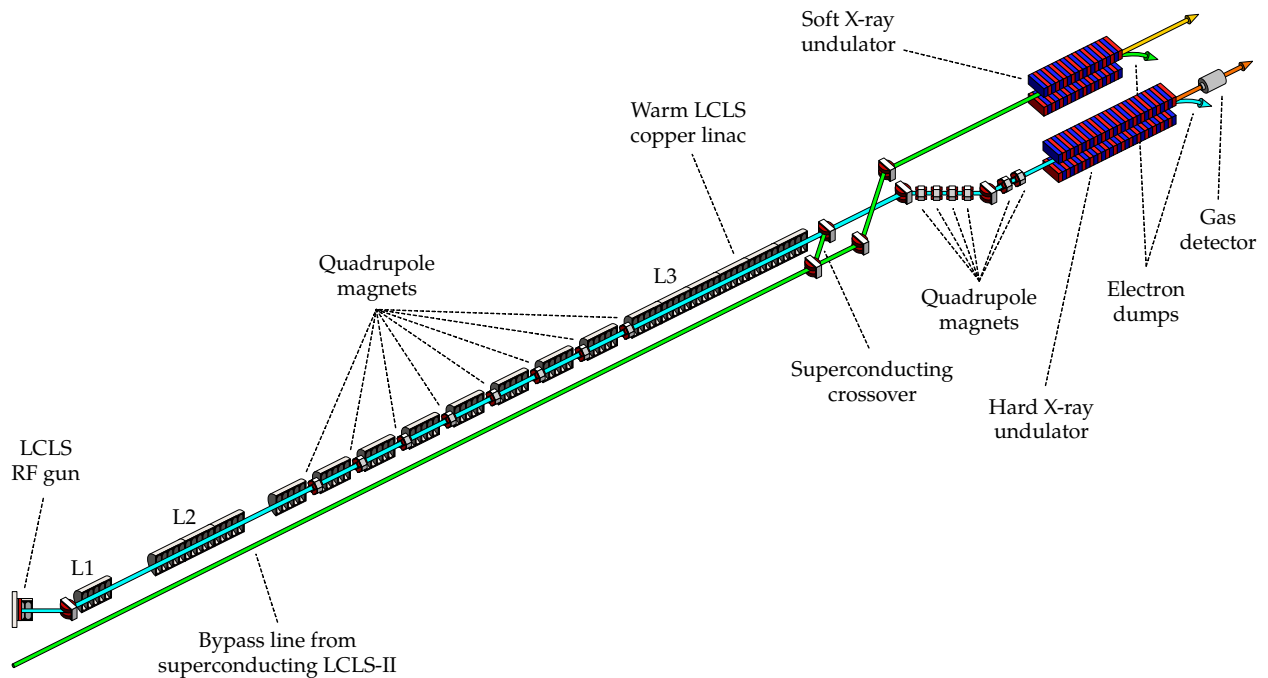


Figure 2.12: Overview of the LCLS(-II) facility focussing on the parts relevant to the FEL tuning task. The original warm LCLS copper linac starting from the LCLS RF gun and connected to the hard X-ray undulator line is shown in turquoise. This is where the FEL tuning task is performed, with the actuated quadrupole magnets being annotated as such. The gas detector providing measurements of the pulse intensity is also shown and annotated. The beam coming from the approximately 700 m long LCLS-II superconducting linac via a more than 2 km long bypass line is shown in green. The beam from LCLS-II can be transported to either the soft X-ray undulator line or the hard X-ray undulator line via a crossover line, while the beam from the copper linac can only be transported to the hard X-ray undulator line. Note that this schematic is not to scale and only shows the components relevant to the presented work.

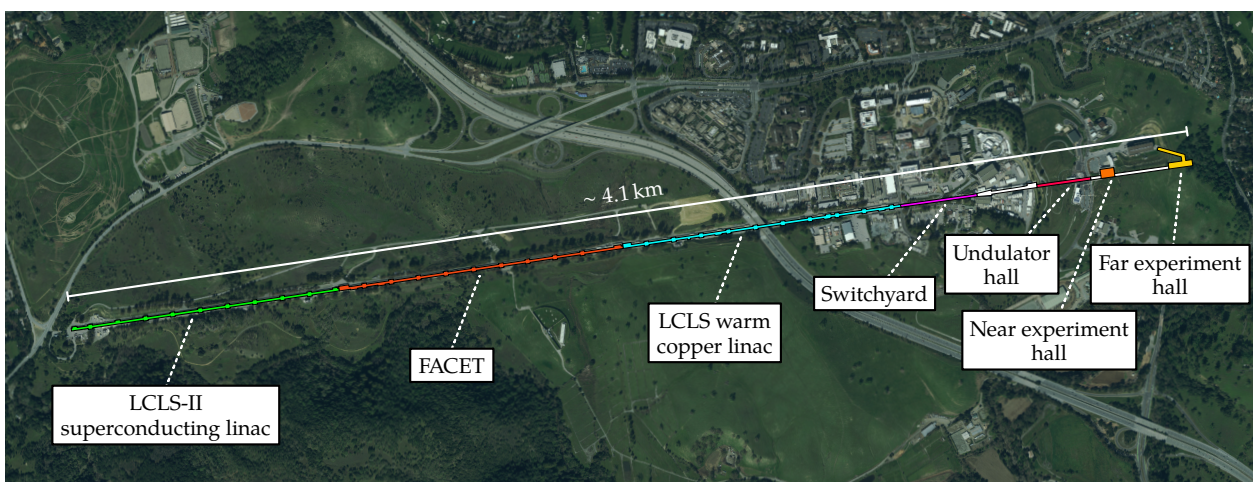


Figure 2.13: Areal view of the LCLS facility at SLAC in Menlo Park, California, USA. Orthoimagery from [331].

## FEL Tuning

A key metric for the performance of any FEL is the pulse intensity, which is a measure of the number of photons in the generated light pulse. The pulse intensity is influenced by the electron beam properties, such as the beam charge, energy, and transverse beam properties, as well as the undulator settings, such as the undulator gap and phase. To facilitate the various different experiments conducted at LCLS, the facility has to switch between different energies and phase space shapes of the electron beam frequently. Every time the working point of the FEL is changed, the pulse intensity has to be optimised for the new working point. Autonomous online tuning of the pulse intensity can facilitate faster fine-tuning and switching of working points, ultimately increasing available experiment time and improving repeatability.

The FEL pulse intensity depends on a number of factors, including the Twiss parameters of the electron beam at the undulator entrance

$$\mathbf{t} = (\beta_x, \alpha_x, \beta_y, \alpha_y), \quad (2.46)$$

as well as constant properties of the undulators, such as its length, and variable properties, such as the undulator gap. The Twiss parameters describe the beam's transverse size as  $\beta$  and divergence as  $\alpha$  in both the horizontal and vertical direction. For this work, the pulse intensity is considered to be a function of the Twiss parameters  $\mathbf{t}$  at the undulator entrance

$$I = f_I(\mathbf{t} \mid \mathbf{v}_{\text{und}}) \quad (2.47)$$

that also depends on the undulators' variable settings  $\mathbf{v}_{\text{und}}$ . The undulators' fixed properties are known and constant, and therefore omitted from this formulation. As part of this dissertation, the FEL tuning task is considered along the LCLS-I beamline of the LCLS copper linac and the hard X-ray undulator line. Here, the Twiss parameters at the undulator entrance  $\mathbf{t}$  are considered a function of the focussing strengths  $\mathbf{u} = (k_{Q_1}, \dots, k_{Q_{14}})$  of 14 quadrupole magnets in  $\text{m}^{-2}$ , which are the control parameters of the tuning task, the settings of other actuators along the linac  $\mathbf{v}_{\text{linac}}$ , and the Twiss parameters  $\mathbf{t}_0$  upstream of the first quadrupole magnet. It results

$$\mathbf{t} = f_{\mathbf{t}}(\mathbf{u} \mid \mathbf{t}_0, \mathbf{v}_{\text{linac}}). \quad (2.48)$$

Note that, while there are known design values for  $\mathbf{t}_0$ , the actual values present in the real beamline are unknown and may deviate. The remaining settings of the machine, i.e.  $\mathbf{v}_{\text{linac}}$  and  $\mathbf{v}_{\text{und}}$  should be at or near their design values during operations and can be considered constant as well as known. The goal of the tuning task is

$$\max_{\mathbf{u}} I(\mathbf{u}) \quad (2.49)$$

to find the actuator settings  $\mathbf{u}$  that maximise the pulse intensity  $I$  at the end of the undulator line measured by a gas detector (also known as GDET), where

$$I = f_I(f_{\mathbf{t}}(\mathbf{u} \mid \mathbf{t}_0, \mathbf{v}_{\text{linac}}) \mid \mathbf{v}_{\text{und}}). \quad (2.50)$$

An overview of the relevant lattice elements and the beamline considered for the FEL tuning task at LCLS is shown in Fig. 2.12. Photos of the LCLS I linac and the undulator hall with the hard X-ray undulator line are shown in Fig. 2.14.

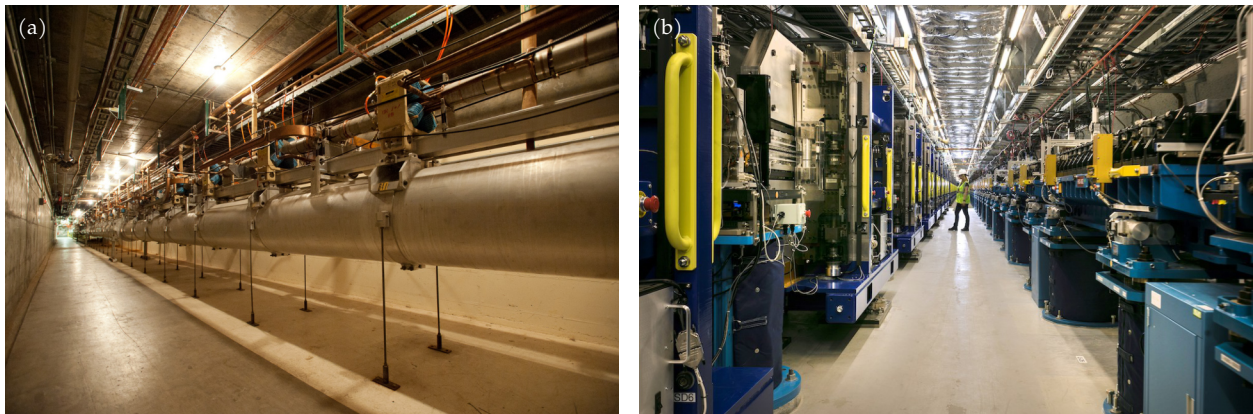


Figure 2.14: Photos from inside the LCLS tunnel. (a) Original LCLS warm copper linac [308]. (b) Undulator hall with the new undulators installed as part of the LCLS-II upgrade, the soft X-ray undulator line on the left and the hard X-ray undulator line on the right [309].

Note that there are additional quadrupole magnets installed along the beamline, which are not considered here. The 14 quadrupole magnets considered in this dissertation are chosen as a balance of complexity and tunability, while representing a starting point that can eventually be extended to 30 or more quadrupole magnets as well as to the lattices of other FEL facilities, such as European XFEL and FLASH, where pulse intensity tuning is performed in a similar manner. Autonomous solutions for tuning the pulse intensity of the LCLS FEL with RL are investigated in Chapter 5.

## 2.3 Related Work

This section reviews the state of the art in machine learning for particle accelerators, focusing on their autonomous optimisation and the use of RL for accelerator tuning tasks. The section is structured as follows: First, the use of machine learning in the context of particle accelerators in general is discussed in Section 2.3.1. Then, work on differentiable modelling and programming in general, and for particle accelerators in particular, is reviewed in Section 2.3.2. Next, existing work on autonomous accelerator tuning with methods other than RL is reviewed in Section 2.3.3, before an overview of RL work covering algorithmic developments on toy applications as well as real-world use cases in Section 2.3.4, followed by a detailed review of the use of RL for accelerator tuning and control in Section 2.3.5.

### 2.3.1 Machine Learning for Particle Accelerators

Some of the earliest work applying machine learning techniques to particle accelerators was done in the late 1980s, where linear regression was used to build surrogate models for orbit correction [127]. Neural networks started to be used in the context of particle accelerators in the early 1990s. Some of the earliest work in 1990 focussed on developing neural network-based surrogate models for ion source waveform characteristics and FEL wigglers [138], which was later used in an iterative optimisation process resembling Q-learning and BO [216, 217]. Other early work from 1991 presents an approach similar to gradient-based RL, where a neural network surrogate is trained to predict the trajectory of the beam from past measurements and magnet settings, with a second neural network

being trained to control the beam's trajectory by back-propagating errors through the surrogate model [230]. While most of the machine learning work pertaining to accelerators in the 1990s and early 2000s focussed on inverted surrogate modelling for tuning tasks, often at the example of orbit correction [35, 36, 92, 172, 285], neural networks were also applied to accelerator fault detection and identification as early as 1992 [171]. The success of these early studies was often mixed, with some studies showing promising results, while others found that the neural networks were outperformed by other methods [172], in part due to the limited computational resources available at the time.

In the early 2010s, the success of *AlexNet* [181] in image recognition renewed interest in machine learning and neural networks in particular. The following period is sometimes referred to as the *AI spring*, and characterised by a broadened acceptance of deep learning methods as valid scientific contributions [182]. This newfound interest reached the accelerator community by the late 2010s and early 2020s. In 2018, Edelen et al. outlined how machine learning can help advance the field of accelerator physics and solve the challenges that will be encountered with future facilities [75]. Since then, the amount of machine learning research in the field of particle accelerators has been growing rapidly, with novel methods being developed for surrogate modelling, virtual diagnostics, tuning, control, anomaly detection and various other applications. This section reviews some of the most relevant works in this area, focussing on surrogate modelling, virtual diagnostics, anomaly detection and NLP. Machine learning-based methods for accelerator tuning and control are reviewed in Sections 2.3.3 and 2.3.5.

Surrogate models are machine learning models designed to approximate the behaviour of complex systems. Their potential applications are manifold. In accelerator physics, they are for example often used to provide non-invasive virtual diagnostics of measurements that could otherwise only be obtained by destructive methods and time-consuming measurement campaigns.

An extensively researched application of surrogate models is a virtual diagnostic of the LPS, which would otherwise require intractable start-to-end simulations. Neural networks have been used to learn both so-called *scalar* [80, 81, 370] and *spectral* [124, 369] virtual diagnostics for 1-dimensional and 2-dimensional LPS reconstruction, using scalar inputs, such as RF settings, or spectral inputs, such as THz radiation spectra, to infer the LPS. This work was later extended to a combined scalar and spectral virtual diagnostic with adaptive resolution [160], with the goal of controlling the undulator linear taper at European XFEL [326]. The neural network-based surrogate model developed in [160, 326] is introduced as part of this dissertation in Section 3.1. A generative diffusion-based approach conditioned with chirp scalar data and BPM readings was most recently demonstrated in early results for the same task, additionally enabling the inference of megapixel phase space images and physical interpolation between different LPSs by traversing the learned latent space [279].

In general, the use of neural networks as fast surrogates for otherwise expensive-to-simulate beam dynamics, where physical models also are not always very precise, has been explored in various contexts. A neural network surrogate model has been demonstrated for the inference of the space charge field in a vacuum chamber cross-section using a physics-informed loss function including a partial differential equation with the Lorentz factor, elliptical bi-Gaussian charge density and boundary condition [101]. The transverse emittance of low energy beams, which are strongly affected by space charge effects, has also been predicted using neural networks at ARES [213]. A start-to-end surrogate model of the low-energy section of the Ferninfrarot Linac- und Test-Experiment (FLUTE) accelerator

is trained by Xu et al., predicting bunch properties, like bunch length and relative energy spread, from inputs, such as charge, gun phase and gradient, and solenoid settings [355]. Li et al. train a neural network surrogate model to predict the wavefront phase and energy spectrum of the photon beam at X-ray free-electron lasers (XFELs) from properties of the electron beam and the undulators [192].

In preparation to ultimately use the surrogate model as a basis for anomaly detection, Wielgosz et al. train an LSTM model to predict future values of the resistive voltage in superconducting magnets at LHC from past measurements [349]. What is more, there has been work by Fol on using autoencoder neural networks to predict turn-by-turn phase advance measurements as well as reconstruct readings from BPMs, which were previously identified as faulty and not providing usable data [95].

Neural network surrogate models have further been explored by Veglia et al. as an alternative to an existing look-up table-based approach of predicting closed orbit distortions from insertion device gaps as input to a singular value decomposition (SVD)-based approach for calculating the orbit response matrix at PETRA III [334, 336]. The proposed approach has the advantage that it can be continuously and parasitically fitted to changing machine conditions without the requirement for dedicated measurements.

Advanced methods for designing and training surrogate models have been demonstrated by Gupta et al. at the example of the LCLS II injector [118]. Their surrogate model is capable of predicting the output distribution of the injector, including measured diagnostics in the form of the laser distribution in addition to scalar settings in the input. They further demonstrate a transfer learning approach, enabling the bulk of the training to be done on a simulation while delivering good performance on the real machine.

Models other than neural networks have also been applied to surrogate modelling in accelerator physics. Random forests regression is used to predict quadrupolar integrated field errors, longitudinal misalignments of quadrupoles and sextupole transverse misalignments from phase advance deviations, horizontal and vertical  $\beta$  at the main interaction points, and the deviation of the normalised horizontal dispersion. The prediction can then be used for optics corrections in circular accelerators such as the LHC [95]. An interpretable random forest surrogate model is trained by Fol et al. for the purpose of speeding up optimisations of the design of the final cooling system for a future muon collider [97]. The authors further demonstrate the utility of an inverted variant of their surrogate model, which proposes promising initial input parameters, thereby reducing the number of samples required by the optimisation.

In general, surrogate models have been widely explored to assist optimisation tasks in accelerator physics. Inverted surrogate models have been used to infer the steerer inputs required for desired orbits [35, 36, 92, 172, 285]. The use of a U-Net inverted surrogate model has further been demonstrated in early results in place of the Gerchberg-Saxton (GS) algorithm to predict the inputs to a spatial light modulator (SLM) from a desired transverse laser profile [354]. In an effort to improve the speed of classical optimisers on particle accelerators, other works explore the use of machine learning to aid the optimisation by fitting surrogate models to data from the real accelerator and performing optimisation on the latter [76, 146].

Anomaly detection and predictive maintenance are important tasks in the operation of particle accelerators that have seen the application of machine learning techniques. Isolation Forests have successfully been used to improve the detection of faulty BPM readings and in doing so improve the quality of the optics measurements at LHC [95, 96].

Further, random forest classification is used to identify tripping superconducting RF

cavities and the type of fault causing the trip [321]. In a continuation of this work, CNNs and LSTM-based neural networks are explored on the same task in an effort to forgo manual feature extraction and speed up the classification process to the point where corrective action can be taken before the fault occurs [338]. In parallel work, a two-stage quench detection method has been developed, employing an analytical model of cavity dynamics for residual-based fault detection, followed by the  $k$ -medoids clustering algorithm to classify faults in order to distinguish quenches from other types of trips [32]. The authors demonstrate the ability of their approach to outperform the currently deployed system at European XFEL.

Martino et al. and Grünhagen et al. develop and compare the performance of various semi-supervised anomaly detection methods, such as  $k$ -means clustering, Gaussian mixture models (GMMs), Isolation Forests and support vector machines (SVMs), on the low-level radio frequency (LLRF) system at European XFEL with artificially injected faults [114, 209]. A similar study was performed as follow-up work on the laser-based synchronisation system at European XFEL [115]. Here, the authors compare a large number of feature extraction methods in conjunction with machine learning algorithms for the purpose of detecting and classifying faults from the output signal of a proportional-integral (PI) controller acting on the laser oscillator cavity length to lock the laser oscillator repetition rate. They focus in particular on classical machine learning algorithms, such as clustering methods,  $k$ -nearest neighbours (KNN), random forests and SVMs, but also highlight the advantages of using an autoencoder neural network for feature extraction to improve performance.

LLMs have found a lot of attention across various fields with the release of models, such as *GPT 4* [242], the *Llama* series [327] and the *Mistral/Mixtral* series [150, 151]. Consequently, there are also efforts to take advantage of the NLP abilities of LLMs for various purposes in the field of particle accelerator research. Sulc et al. demonstrate the fine-tuning of an open-source LLM to be a particle accelerator domain expert using open access scientific literature as training data, augmented by another LLM to generate question-answer pairs from research papers. The fine-tuned model, called *PACuna*, is shown to be more proficient than off-the-shelf LLMs in answering questions related to particle accelerators. Mayet demonstrates how off-the-shelf LLMs can be used as a general AI assistant for intelligent accelerator operations (GAIA), employing the *ReAct* prompting scheme to enable the LLM to intelligently trigger accelerator operation routines, automatically contact experts when needed, research questions in the facility's logbook, provide the correct control system addresses for actuators and sensors of the accelerator, and write weekly shift reports. Following work that demonstrated the ability of LLMs to perform prompt optimisation [361], optimisation of complex algorithms [260], linear regression [361] and game playing tasks [244], LLMs have also successfully been applied to solve complex non-linear numerical optimisation tasks at the example of transverse beam parameter tuning at ARES, demonstrating first steps towards enabling autonomous particle accelerator tuning through natural language [165]. As part of this dissertation, the latter work is introduced in Section 4.5.

An inter-institutional effort is also underway to improve the findability, accessibility, interoperability and reusability (FAIR)-ness of data contained within particle accelerator facility logbooks, which contain a wealth of useful and complex information in difficult-to-search unstructured formats, by using LLMs in combination with retrieval augmented generation (RAG) such that rich and comprehensive information can be extracted easily and efficiently, simply by asking questions in natural language [318]. Parallel efforts are aimed

at improving the accessibility of multi-modal data contained within logbooks by using optical character recognition (OCR) to extract textual information from commonly found images of plots and control system panels, as well as topical classification and clustering of logbook entries to enable easy search and retrieval of suitable information [203]. In a similar effort, CERN is developing a service, called *AccGPT*, for hosting fine-tuned and off-the-shelf LLMs for fast and intuitive information retrieval from their vast array of knowledge bases, such as wiki pages and document servers, with future goals of integrating logbooks and expanding the service to additional features like coding assistance [256].

### 2.3.2 Differentiable Simulations

Programmatic beam dynamics modelling is a mature field with various well-established simulation codes having been developed over the years, including *ASTRA* [94], *Bmad* [275], *Elegant* [30], *MAD-X* [49], *Ocelot* [3] and *Xsuite* [143]. These codes, however, were developed with the primary goal of accurately modelling a large number of physical effects in particle accelerators. Differentiability was not a meaningful concern in their development, and these codes are not designed to support automatic differentiation.

Neural networks are well known for their differentiability, which is a key feature that enables the training of neural networks with gradient-based optimisation algorithms. While neural networks have been widely used to build surrogate models for accelerator systems [80, 97, 118, 160, 192, 213, 355], as is discussed in detail in Section 2.3.1, these surrogate models are typically used for their speed rather than their differentiability. A few works exist, that use neural networks to model accelerator systems in a differentiable manner, such as to train a neural network to output steerer settings for trajectory correction [230], to train a neural network policy in a gradient-based RL setup to optimise the beam parameters at an FEL's undulator entrance [71, 73, 74] and for identifying optimal detector parameters at LHC [5, 109].

However, the frameworks used to implement neural networks, such as *TensorFlow* [1], *PyTorch* [248] and *JAX* [37], allow also for the convenient implementation of models other than neural networks with support for automatic differentiation. Because of this, the field of scientific computing has recently seen a downright explosion of simulators implemented in these frameworks with the goals of faster computations and differentiability. *JAX* in particular has been used to implement benchmark RL environments for research, however mostly with an emphasis on speed for scalable RL training rather than on differentiability of the simulation itself with tools like *gymnax* [184], *Jumanji* [29], *Pgx* [178] and *JaxMARL* [274]. A *JAX*-powered implementation of the *Minigrid* [54] RL research environment with explicit support for automatic differentiation is provided by the *NAVIX* project [249]. A differentiable physics engine for robotics, human perception and materials science is implemented in *Brax* [98]. Other research fields have also seen the development of differentiable simulators, such as *TORAX* for Tokamak transport simulation [56], *dynamics* for quantum systems simulation [117], *jax-cosmo* for cosmology [185], *JAX, M.D.* for molecular dynamics [288], *∂PV* for photovoltaic simulation [206], *ExoJAX* for spectrum modelling of exoplanets and brown dwarfs [170], *j-Wave* for acoustics simulations [312] as well as *JAX-CFD* [68, 177] and *JAX-Fluids* [22, 23] for computational fluid dynamics.

The accelerator community has also started to build specialised programmatic physics-based differentiable models on a per-application basis. An overview of opportunities for differentiable programming in particle physics instruments is given by Dorigo et al. [66]. A handcrafted differentiable physics model is used as the discriminator in a generative

adversarial network (GAN) setup to train a neural network to reconstruct time-domain measurements of X-ray pulses without labelled data by Ratner et al. [255]. Roussel et al. fit the hysteron density function of a Preisach model to accurately model hysteresis from experimental data [265, 268]. A differentiable beam dynamics simulation of a tomographic beamline has also been used to reconstruct phase space distributions from experimental screen images in a generative machine learning setup [265, 270]. Simultaneous calibration of all detector parameters of a liquid argon time projection chamber using a differentiable simulation of the latter is performed by Gasiorowski et al. [106]. In work by Qiang, a differentiable self-consistent space charge simulation model based on truncated power series algebra (TPSA) is developed to speed up the simulated optimisation of accelerator design parameters under consideration of space charge induced effects [253].

Three efforts have been made to develop general-purpose differentiable beam dynamics simulation codes for particle accelerators. Gonzalez-Aguilera et al. introduce *Bmad-X* [110], a library-agnostic differentiable particle tracking code written in Python based on Bmad. They demonstrate the application of Bmad-X on examples of beamline optimisation, model calibration and phase-space reconstruction. A similar effort is made with *Cheetah* [162, 163, 314], a differentiable beam dynamics simulation code primarily based on a PyTorch backend and with a stronger focus on fast computations, bridging the gap between fast hand-crafted and data-driven particle accelerator models, and support for a larger number of lattice elements and conversions from other simulation codes. Cheetah is part of the work presented in this dissertation and introduced in detail in Section 3.2. At the time of writing, Bmad-X is being merged into Cheetah to combine the advantages of both codes. A third code, *JuTrack* [344], is currently in early development, offering forward mode automatic differentiation for symplectic 6-dimensional particle tracking in the Julia programming language.

### 2.3.3 Autonomous Accelerator Optimisation and Tuning

In order to reduce the time required for the tuning of particle accelerators, to improve the reliability of the tuning process and to extend the operational limits of these machines, autonomous accelerator tuning solutions using various algorithms have been developed. Grid search and random search are still commonly used in accelerator design as well as day-to-day operations, delivering reliable and reproducible results, but due to the so-called *curse of dimensionality* [19] they require a large number of samples, which can quickly become impractical as the number of tunable parameters increases and when the cost per sample is prohibitively high. In response, dedicated and more sample-efficient optimisation algorithms have been developed, and applied to accelerator optimisation and tuning tasks.

Nelder-Mead simplex optimisation [227] has found wide-spread use in accelerator tuning, such as for injection tune-up, coupling minimisation, automatic phasing [79], optimising the dynamic aperture of storage ring designs [303], FEL pulse energy optimisation [4] and transverse beam parameter tuning [161, 164]. The latter is part of the work presented in this dissertation and elaborated on in Section 4.2.3. A modified variant of the Nelder-Mead simplex algorithm, called *RSimplex*, has been designed for robustness against measurement noise encountered in accelerator tuning, and was validated on kicker bump matching and coupling minimisation [139]. However, the Nelder-Mead simplex algorithm is known to be sensitive to noise and the choice of initial simplex, making it best suited for low-dimensional and mostly convex optimisation tasks.

Other work has focussed on robust conjugate direction search (RCDS) [140] for better

robustness against noise, which was explored on storage ring tuning applications, such as minimisation of the vertical emittance, optimisation of injection kicker bump matching, optimisation of injection beam steering [140], and improving the acceptance and beam lifetime [236]. A variant of RCDS, called safe robust conjugate direction search (RCDS-S), has been developed to ensure that the machine performance remains within a designated safety envelope during tuning and drift compensation [365].

As a potentially more sample-efficient alternative, the bound optimisation by quadratic approximation (BOBYQA) algorithm [251] has also been applied to accelerator tuning, such as for the optimisation of the beam emittance in a photoinjector [228], and beam loss minimisation and focussing [8].

Genetic algorithms have found application in accelerator optimisation as well. While sample-inefficient, they can easily be parallelised and are well-suited for high-dimensional and non-convex optimisation tasks like accelerator design optimisation. The strength Pareto evolutionary algorithm 2 (SPEA2) [371] and nondominated sorting genetic algorithm II (NSGA-II) [57] were applied to multi-objective optimisation of the design of a high brightness DC gun photoinjector [16]. The SPEA2 algorithm was also explored to optimise various aspects of accelerator design, demonstrating significant improvements in the dynamic aperture, working point selection and beam optics decoupling for the Continuous Electron Beam Accelerator Facility (CEBAF) and the proposed Medium-energy Electron-Ion Collider at Jefferson Lab [136]. A scalable simulation-based multi-objective optimisation framework incorporating the NSGA-II algorithm to address complex optimisation problems in the domain of electron beam dynamics is demonstrated through application to the Argonne Wakefield Accelerator photoinjector in [229]. The use of dimension-reduction techniques is demonstrated to speed up the convergence of genetic algorithms on high-dimensional accelerator tuning tasks at the example of tuning the vertical emittance as well as minimising beam size and orbit errors [21].

Extremum seeking (ES), which is especially appropriate for high-dimensional mostly convex dynamic optimisation tasks, has been applied to both feedback and tuning problems in accelerators, such as tuning quadrupole magnets and RF buncher cavities under constant drift [280]. Multi-objective optimisation of the beam emittance and trajectory has been demonstrated using ES [282]. A bounded variant of ES was explored for beam loss and neutron flux optimisation [283]. The same bounded ES algorithm was further applied to Touschek lifetime optimisation and vertical emittance minimisation at the ESRF-EBS storage ring [335].

Data-driven models have also been combined with classical control algorithms for particle accelerator tuning and control. In this line of work, PI controllers have been coupled with neural network predictors to achieve control over a wider range of conditions [219], neural network models have been integrated as the system model with model predictive control (MPC) [72] and the performance of ES was improved on high-dimensional control tasks through the use of a neural network predictor providing an initial guess close to the real optimum from where ES can efficiently perform the rest of the tuning [281]. In addition to their random forest-based forward surrogate model, Fol et al. demonstrate the utility of an inverted variant of their surrogate model for the purpose of proposing promising initial input parameters, thereby reducing the number of samples required by other optimisation algorithms [97].

Inverted surrogate models, have further been used to tune and control accelerators by predicting required actuator settings from desired measurements. Such models have been used for orbit correction, mapping from desired BPM readings and orbit distortions to the

steerer settings required to achieve them [35, 92, 172, 285]. The use of a U-Net [261] inverted surrogate model has been demonstrated in early results in place of the GS algorithm to predict the inputs to an SLM from a desired transverse laser profile [354].

A data-driven optimisation algorithm that has seen wide-spread use in accelerator tuning is BO [271, 301]. With BO, the number of samples required for a successful optimisation is reduced by learning a probabilistic surrogate model of the objective function at the time of optimisation and using this surrogate model to determine where to sample the objective function next. BO has been applied to FEL tuning [215]. The use of a physics-informed kernel in BO has been demonstrated at the example of vertical emittance minimisation in a storage ring [123], FEL pulse intensity tuning [70] and injection into a storage ring [357]. Laser-plasma accelerators are also tuned using BO, for example tuning laser and plasma parameters to optimise the electron beam charge [302], and to flatten the longitudinal phase space of an ionisation-injected electron bunch via optimal beam loading [149]. The use of multi-task BO is demonstrated to yield significant speed-ups in a multi-fidelity setting, where data from a fast low-fidelity model is used to reduce the number of samples required from the expensive function under optimisation in laser-plasma accelerators [90]. Various works have also investigated the incorporation of safety constraints into the BO optimisation process at particle accelerators [174, 175, 176], such as for tuning PI controllers in a laser-based synchronisation system [200]. A safe multi-task BO approach deriving from earlier work on safe BO [200] and multi-task BO [90] has also been demonstrated for controller optimisation in the same laser-based synchronisation system [201]. What is more, multi-objective BO has successfully been applied finding the full Pareto front in online accelerator tuning [267]. The use of a non-zero neural network prior mean with BO has been demonstrated to improve the sample efficiency of the optimisation process at examples of tuning the beam coming from an electron photoinjector and transmission through a heavy-ion beamline [28]. Other variants of BO, such as Bayesian Algorithm Execution (BAX) [226] and trust region Bayesian optimisation (TurBO) [84], have been developed and deployed to particle accelerators with the goal of further improving the sample efficiency of these methods. An extensive review of BO and its application to particle accelerators is provided by Roussel et al. [271].

Machine learning methods for accelerator tuning not directly driven by data related to accelerators have also been explored. In light of recent developments in NLP, LLMs have been shown to be capable of tuning particle accelerators when prompted correctly through natural language at the example of transverse beam parameter tuning using quadrupole and dipole magnets at ARES [165]. This work is part of this dissertation and details are presented in Section 4.5.

A number of software frameworks have been developed to facilitate and streamline the application of different optimisation algorithms to particle accelerators in regular operations. *sddsoptimize* [79] integrated general purpose optimisation into the *SDDS EPICS toolkit*. *Ocelot Optimizer* [4, 324] is a Python-based optimisation framework providing a graphical user interface (GUI) interface to quickly setup accelerator tuning configurations using different optimisation algorithms. Derived from *Ocelot Optimizer*, *Badger* [364] was developed as a more modular framework and GUI application for configuring tuning runs in the control room. It interfaces directly with *Xopt* [269], a Python package that allows for straightforward connection of optimisation algorithms to accelerator tuning tasks, providing implementations of various optimisation algorithms, such as BO and ES. A framework, called *Optimas* [91, 141], has been developed for highly scalable parallel optimisation at accelerator laboratories.

### 2.3.4 Deep Reinforcement Learning in General

Reinforcement learning has been successfully applied to a wide range of problems. Early work focussed on demonstrating the ability of RL to solve difficult problems and improving RL algorithms at the example of games. As early as 1995, it was shown that a Q-learning approach using an MLP to approximate the value function can be used to train *TD-Gammon*, a backgammon-playing agent that outperforms human players [322].

Interest in RL increased significantly with the introduction of the DQN algorithm in 2015 [223], early results of which were presented as early as 2013 [222]. DQN was able to learn to play 49 different Atari 2600 games directly from raw pixel data using a CNN as a Q-function approximator, enabling RL to solve problems with high-dimensional input spaces, where human-engineered feature extraction is infeasible. Most impressively, DQN was able to match or exceed the performance of professional human game testers on all games using the same algorithm, neural network architecture and hyperparameters, demonstrating the generality of the approach and negating the need for expensive hyperparameter tuning for each new task.

This work was closely followed by *AlphaGo* [305], which was able to defeat the world champion in the game of Go 5 games to 0, an achievement previously thought to be beyond the reach of artificial intelligence (AI) approaches because of its much larger state space compared to games like chess. AlphaGo used a model-based RL approach combining Monte Carlo simulation with value and policy neural networks, having learned through supervised learning from human expert games and RL from self-play. This approach was then improved upon with *AlphaGo Zero*, which managed to defeat AlphaGo 100 games to 0, while training entirely through RL from self-play without any human expert data [306]. The same approach was then extended further to generalise to the board games of chess, shogi and Go entirely from self-play while using (mostly) the same hyperparameters and neural network architecture for all three without game-specific tuning, resulting in *AlphaZero* [307]. AlphaZero achieved superhuman performance, outperforming the then best chess engine *Stockfish*, the then best shogi engine *Elmo* and its own predecessor AlphaGo Zero. The final form of this model-based line of research is *MuZero* [289], which is able to match the performance of AlphaZero on the games of chess, shogi and Go, while also achieving then state-of-the-art performance on the Arcade Learning Environment (ALE) benchmark of 57 Atari 2600 games [18], where it beat human-level performance on 51 out of 57 games. Unlike its predecessors, MuZero does not have access to a hard-coded model of the environment and its rules for planning, but learns this model instead.

Parallel efforts focussed on model-free RL to play Atari games from the ALE benchmark [18] and the *DMLab-30* benchmark [17], the latter of which provides 30 navigation and puzzle-solving tasks in a 3D environment that share the same observation and action spaces. Using a distributed RL architecture to gather experience from many independent actors, and RNN architectures, various advancements were made in playing these games by methods like *IMPALA* [85] and *R2D2* [168]. This line of research culminated in *Agent57* [12], which first beat human-level performance on all 57 games in the ALE benchmark with a single set of hyperparameters by introducing solutions to the long-term credit assignment problem and the exploration-exploitation dilemma.

RL has also been applied to more complex games that provide only imperfect information, while requiring planning over long time horizons and having complex high-dimensional state and action spaces. In *MineRL* [119], RL agents can be trained to play the game of *Minecraft*, with the goal of obtaining a diamond, which requires agents to achieve a number of prerequisite goals, making it effectively impossible to find the dia-

mond resource through random exploration. At the example of the game of *Dota 2*, a system called *OpenAI Five* [240] was trained through distributed self-play RL to ultimately defeat the world champion. Similarly, *AlphaStar* [339] was trained to play the game of *StarCraft II*, where it was able to rank above 99.8% of ranked human players. Both of these systems also demonstrated the scalability of RL approaches. Concurrent work has explored multi-agent co-adaptation at the example of two games, where multiple agents are trained together in the same environment, both through self-play and cooperative play. Work on the first-person multiplayer game *Quake III Arena*, developing a system called *For The Win*, demonstrated that agents can learn to play the game of Capture the Flag in a population-based training (PBT) setup only from pixels and a sparse reward signal indicating whether the game was won at its very end [148], with agents using a two-timescale RNN architecture and outperforming human players. Similarly, using the same system used in concurrent work by the same authors [240, 241], multiple agents based on a transformer architecture with attention over objects instead of time were trained in co-adaptation and self-play to play the game of hide-and-seek, where the agents learned to use tools and exploit the environment to win the game all the while discovering unexpected emergent behaviours [13].

The application of RL to complex real-world tasks has however proven challenging [69, 145] because of their overall complexity and the high cost of obtaining samples. Nevertheless, practical applications of RL have been demonstrated successfully in a number of domains, such as robotics, nuclear fusion and algorithmic discovery.

RL has been applied to train domain-specific optimisation algorithms by allowing the policy to select the next input delta for the objective function, enabling it to learn an optimal strategy for balancing exploration and exploitation [51, 190, 191]. This work falls into a broader research field known as *optimiser learning* [7], and can be particularly useful when objective functions encountered in the future are known to have some similar topological properties to the ones encountered during training.

Robotics are a natural application domain for RL. Levine et al. demonstrated that RL can be used to train a robotic system without hand-engineered feature extraction directly from raw pixel observations using a CNN-based policy to perform complex manipulation tasks with a *PR2* robot, such as inserting a block into a shape sorting cube, screwing a cap onto a bottle, fitting the claw of a toy hammer under a nail and placing a coat hanger on a rack [189]. RL in combination with domain randomisation and vision state estimators based on the training setup from OpenAI Five is used to train a robotic hand on dexterous manipulation tasks, demonstrating for the first time that RL can be used to train a robotic system to perform complex tasks in the real world, if the right randomisations, sensing modalities that can be modelled in simulation, and large-scale distributed training are employed [241]. This work is extended through the use of automatic domain randomisation (ADR) and memory-augmented models to train a robotic hand to solve a Rubik's cube, where ADR is found to require less manual tuning while achieving better performance than the manual domain randomisation baseline, and the models trained on an ADR-generated distribution of environments exhibit signs of emergent meta-learning at test time [239]. First-person view (FPV) drone racing using an RL-trained policy called *Swift* has been demonstrated by Kaufmann et al. [169]. Here, the policy consists of a visual-inertial perception system made from a Kalman filter and a CNN that estimates the drone's position and orientation from the onboard camera feed and data from the onboard inertial measurement unit (IMU), as well as an MLP controller that maps the estimated state to motor commands. The policy is trained on a simulator augmented with noise models built from real-world data using GP

models and  $k$ -nearest neighbour regression, ultimately achieving a level of performance that matches human champions, who themselves have had one week to practice the track, while Swift put down the fastest lap time over all. In work by Haarnoja et al., a two stage setup is used to successfully train policies for cheap off-the-shelf *Robotis OP3* humanoid robots to play the game of football in a 1-v-1 setting [120]. Here, the training is split into two phases, first training the policy to get up and score a goal against an untrained opponent, before moving to the full 1-v-1 football task. What is more, training is performed in simulation, using domain randomisation, small shaped rewards and light pushes to ensure a safe zero-shot transfer to the real robots.

Chemical reactions have also been optimised using RL, where a policy is trained to select the optimal reaction conditions, such as temperature, solvent composition, pH, catalyst and time, to optimise the outcome of the reaction [367]. The presented approach is shown to outperform the existing state-of-the-art black-box optimisation algorithms by 71 %, leading to a better understanding of the factors that control microdroplet reactions.

RL has further been applied to the control of heating, ventilation and air conditioning (HVAC) systems in buildings. For example, multi-zone HVAC control can be performed using model-based RL with an ensemble of environment-conditioned neural network models and a Model Predictive Path Integral (MPPI) control method to optimise the thermal comfort of occupants while minimising energy consumption [65]. The use of and challenges involved with using RL to control grid-interactive buildings, which intelligently manage their energy consumption based on factors like the current electricity demand in order to provide the grid with improved stability and flexibility, are explored by Nweye et al. [233].

A number of works have demonstrated the use of RL to improve the performance of telescopes. Piston misalignments in terrestrial telescopes using segmented mirrors can be corrected using RL-trained policies to maximise the Strehl ratio and reduce phase shift errors between the mirror segments only by taking diffraction images as input [116]. In work by Nousiainen et al., a model- and gradient-based RL approach is used to train convolutional policies for significantly improved adaptive optics correction by learning a differentiable model of the environment and reward function, ultimately improving the coronagraphic contrast by factors of 3 to 5 [231]. The developed approach is expected to help efforts towards direct imaging of potentially habitable exoplanets [352, 353, 362].

RL is playing an increasingly important role enabling nuclear fusion research on its path towards sustainable energy production. Seo et al. trained an RL agent on an LSTM surrogate model based on experimental data to control the discharge scenario in order to achieve target values for the normalised beta  $\beta_N$ , which behaves directly proportional to the fusion gain [299]. The trained agent is then successfully validated at the Korea Superconducting Tokamak Advanced Research (KSTAR) tokamak, where it is able to achieve the target  $\beta_N$  values with a high success rate. This is built upon in later work, where the RL agent is trained to achieve given targets for multiple parameters, including the poloidal beta, the safety factor at the 95 % flux surface and the internal inductance in an effort to constrain the plasma state in terms of stability or magnetic structure and reduce magneto-hydrodynamics (MHD) instabilities [300]. Parallel work by Degraeve et al. demonstrated the use of RL to train control policies at the Tokamak à Configuration Variable (TCV) for real-time magnetic coil control to form diverse set of plasma configurations [59], demonstrating zero-shot transfer through domain randomisation of policies trained on a simulator to the real-world system, while constraining the policy to be able to run in real-time and overcoming the computational inefficiency of the simulator through the

use of the Maximum a Posteriori Policy Optimisation (MPO) algorithm, which supports a large number of parallel environments for experience acquisition and is known to be sample-efficient.

RL has also found successful application accelerating the process of algorithmic discovery. The MuZero algorithm has been used to train rate control policies for encoding videos to the VP9 codec by selecting quantisation parameters in the open-source *libvpx* to maximise the quality of the encoded video while satisfying a target bitrate constraint [204]. The trained policy is found to achieve a 6.28 % reduction in file size compared to *libvpx*'s two-pass VBR rate control algorithm, while maintaining the same quality. This marks a significant improvement in a field that has seen large investments in hand-crafted algorithms and is widely considered to have reached the limits of potential improvements. *AlphaTensor*, a derivative of AlphaZero, was able to discover faster matrix multiplication algorithms by phrasing the task as a game, where the objective is to find tensor decompositions within a finite factor space, having the agent select how to combine different entries of the matrices [89]. The discovered algorithms are found to outperform the best known algorithms, including Strassen's two-level algorithm [315], which stood as the fastest known algorithm for over 50 years. An RL system called *AlphaDev* has further been used to automatically discover faster sorting algorithms [205]. Using an extension of AlphaZero [307], the authors phrase the design of sorting algorithms as a game, where the RL agent selects consecutive assembly instructions based on representations of the preceding instructions and the memory state as encoded by a transformer neural network and an MLP, respectively. The discovered algorithms are found to yield a speed increase of 1.7 % to 70 % over the best known sorting algorithms, and have been integrated into the *LLVM C++ sort* library, which is used by many software projects and estimated to be called over a trillion times a day.

An assistant for football tactics called *TacticAI* was trained using a combination of RL and geometric deep learning [39] from raw corner kick data recorded from Premier League matches between 2020 and 2023 [346]. *TacticAI* is able to predict the first receiver of a corner kick and the probability of a shot on goal as a direct result of the corner kick. In addition, it suggests tactical variations that meaningfully affect the outcome of the corner kick. The proposed tactics have been shown to be favoured by experts over real tactics 90 % of the time.

With the emergence of LLMs, RL has found wide-spread application in their training as well. Specifically, reinforcement learning from human feedback (RLHF) [55], a method for learning a reward model from human-annotated examples when an explicit reward function is not available, has been used as a fine-tuning step in the training of various LLMs [242, 327].

Work has also been done to simplify the processes of developing RL algorithms and solutions. The *OpenAI Gym* [38] Python package and its successor *Gymnasium* [88] provide a number of benchmark environments as well as a standardised interface for developing new environments. *Gymnasium* was developed by *The Farama Foundation* after *OpenAI* stopped actively developing new features for *Gym*. *Gym* and *Gymnasium* have become the de facto standard for developing RL environments across fields. Benchmark environments and a standardised environment interface are also provided by the *DeepMind Control Suite* [320, 329], which is specifically aimed at continuous control tasks.

Various libraries have been developed to provide well-tested implementations of RL algorithms and clear interfaces to use them. Popular ones are *Stable Baselines3* [254] and *Ray RLLib* [196], which provide benchmarked and well-documented implementations of

various RL algorithms. The former was developed as a successor to *Stable Baselines* [128], which itself was developed from *OpenAI Baselines* [64]. Other libraries used include *Keras-RL* [250], which is no longer in development, and *OpenAI Spinning Up*, which provides clean easy-to-understand implementations of algorithms that are not meant for performant production use, but rather for educational purposes. The *Acme* library [135] provides implementations of a variety of state-of-the-art algorithms developed and used by *Google DeepMind*, but comes with only minimal documentation.

### 2.3.5 Reinforcement Learning for Particle Accelerators

RL is actively investigated for accelerators since the late 2010s. Earlier work in the 1990s resembles RL, for example a Q-learning-like iterative optimisation process using neural networks as surrogate models for ion source waveform characteristics and taking the maximum of the fitted model in each step to select the input for the next sample [138, 216, 217]. Another early work from 1991 [230] presents an approach similar to gradient-based RL, where a neural network surrogate is trained to predict the trajectory of the beam from past measurements and magnet settings, with a second neural network being trained to control the beam's trajectory by back-propagating errors through the surrogate model. While these works somewhat resemble RL but do not refer to it as such, the authors of other work from 1994 [35] refer to their method as *reinforced learning* but actually employ supervised-learning, using a widespread misnomer from the 1990s [319], before the deep learning terminology solidified in the 2000s. Since then, the primary way RL has been applied to particle accelerators is as a method to train optimisers or controllers for accelerator tuning and control tasks.

The first publication on applying RL to particle accelerators appeared in 2012 [220], where an actor-critic method was explored in simulation to perform orbit correction on a storage ring of three steering magnets and three BPMs. However, the authors approach can only be applied to problems with the same number of observations and actions, and the authors do not yet use the term *reinforcement learning*.

More extensive studies on applying RL [73, 74] appeared in 2017, with both publications describing the same work. In their work, the authors train a neural network policy to optimise the beam properties at the entrance of the undulators in a simulation of a compact THz FEL at the example of the lattice of the Twente/Eindhoven University FEL (TEU-FEL). They demonstrate that by using a differentiable neural network surrogate model trained on simulated data that allows for direct access to the policy gradient, they can train a policy that can suggest quadrupole settings, RF power and phase, and solenoid strength to achieve beam parameters within 10% of the target with just one inference of the policy, where conventional numerical optimisers required hundreds of steps to perform the same task. This work was later finalised in 2020 and reported in more detail as part of the doctoral dissertation of Edelen [71].

While not directly particle accelerator control, a non-deep, i.e. not using neural networks, RL approach was proposed in 2019 for optimising network traffic in the control system of the RHIC [103] particle accelerator. Evaluating on a simulation of the RHIC control system, the authors aim to reroute network traffic to optimise the network's performance and prevent servers from crashing when they reach their maximum capacity. They employ a regret-based learning procedure and find that adding memory to the algorithm improves the performance significantly. They further look into more efficient server resource utilisation when loads are low, and explore how the objectives can be augmented to include, for

example, real-world time constraints.

Another scheme to use RL for particle accelerator control was proposed in 2019 [27]. Here, the goal is to train a feedback controller for controlling the micro-bunching instability at the Karlsruhe Research Accelerator (KARA), ultimately increasing the intensity of and stabilising the coherent synchrotron radiation (CSR) power signal. The authors postulate that their scheme is not restricted to the micro-bunching instability but might also be used for different longitudinal instabilities at storage rings. In 2021, a follow-up publication [345] continues this work and demonstrates the feasibility of the approach in simulation, assuming a more conventional RL setup with the typical challenge of RL that the environment is not differentiable, necessitating either the estimation of the policy gradients or use of a Q-learning method. In addition, this work explores the implementation of the RL controller and the training on a field-programmable gate array (FPGA) board. The doctoral dissertation of Boltz [26] reports further on this work, with multiple state-of-the-art model-free RL algorithms being studied for controlling the micro-bunching instability. Because of the tasks throughput and latency constraints, this line of work had so far only been a proof-of-concept demonstration in simulation. Hence, it was continued by Scomparin et al. in 2022 with the development of *KINGFISHER*, a framework built around the AMD-Xilinx Versal family of heterogeneous computing devices, which combine an ARM central processing unit (CPU) with an FPGA and with AI engines designed for neural network inference [295]. In their preliminary results, the authors demonstrate that their framework is capable of running the policy inference pipeline in real-time, i.e. on a tens of  $\mu\text{s}$  timescale. In later preliminary work from 2023 [296], they provide a proof-of-principle for the feasibility of training and running an RL controller using their platform by controlling the induced transversal oscillations of the electron beam in a turn-by-turn feedback at KARA. Most recently, in 2024 the authors presented preliminary results of employing the *KINGFISHER* platform for controlling the micro-bunching instability at KARA on the real machine with only online training prior to deployment [297]. More extensive results on the use of *KINGFISHER* to train neural network policies with RL to dampen betatron oscillations at KARA were presented in [298]. The authors demonstrate their implementation of a real-time experience accumulator, training-time reward definition and real-time policy inference, showing that their system can outperform the state-of-the-art commercial controller and provides a feasible method of training RL policies for real-time control of accelerators. Moreover, they show that, when an accelerator control problem runs at such high repetition rates, online acquisition of the large number of training samples required for RL becomes feasible.

Another project on FEL optimisation [43, 44] was also first reported with early results in 2019. Using a Q-learning method with linear function approximation, the authors achieve autonomous optimisation of the seed laser alignment to the electron beam at the Free Electron Laser for Multidisciplinary Investigations (FERMI) at Elettra Sincrotrone. This work also marks the first time the feasibility of using RL for accelerators was demonstrated on a real machine. A later peer-reviewed publication [45] in 2020 extends on this work, where RL is used more directly as an optimisation algorithm and in situ to achieve performance recovery of the seed-laser alignment after drifts. Policy gradient methods have also been used to directly optimise FERMI on three separate tasks [243]. These include maximisation of the amount of THz radiation produced using two steering magnets and three quadrupole magnets, maintenance of a single stage High-Gain Harmonic Generation (HG) FEL through recovery of the seed laser alignment, and maximisation of the HG FEL signal through adjustment of the dispersive strength and seed laser delay.

In a follow-up comparative study [130], early results from a first application of model-based RL to training tuning policies for accelerators on the same tuning task are reported. In particular, *DYNA*-style algorithms with uncertainty-aware models were successfully shown to work on the laser alignment problem and compared to a model-free Q-learning approach using neural networks as function approximators. The authors find that their model-based approach has better representational power and sample-efficiency during training, while the model-free approach has better asymptotic performance. All these studies are concluded in the doctoral dissertation of Bruchon [42] in 2021.

In 2020, interest in RL within the accelerator community started to increase rapidly. Under involvement from industry in the technology sector in the form of *Apple Inc.*, a policy is trained to optimise transmission through a simulation of the Los Alamos Neutron Science Center Drift Tube Linac (LANSCE DTL) [245] by tuning 3- and 5-dimensional continuous action spaces of cavity field amplitudes and phases, achieving better-than-human level performance of 85 % transmission in the 3-dimensional case and 21 % in the 5-dimensional case after up to 700 steps in simulation.

At CERN, alternating steerers and BPMs in both the Advanced Proton Driven Plasma Wakefield Acceleration Experiment (AWAKE) and Linear accelerator 4 (Linac4) were used for trajectory correction in 1 to 2 steps [153] by 2020. This work by Kain et al. aimed specifically to show that using sample-efficient model-free RL algorithms, it is possible to train directly on the real machine in feasible time. Furthermore, the authors demonstrated that, at least in the case of their considered problem, a successful sim-to-real transfer, i.e. training on a simulation and then deploying the trained policy to the real machine, is possible. Expanding on these findings, the authors also demonstrated that a trained policy can be reused months after its original training without requiring retraining, i.e. the policy generalises well enough to still perform well even after drifts. However, most of these results stem from the problem's relative simplicity, meaning very few samples are generally needed to learn a functional policy, and the authors present no investigation on the strength of long-term drift effects at the considered facility, meaning that instead of the policy being robust, drifts may simply be negligible instead. This work was extended in 2021 [154] with preliminary results from a test of a model-based *DYNA*-style RL algorithm on the AWAKE trajectory steering problem, showing that the number of required training samples can be reduced significantly. In this work, the model used by the *DYNA*-style algorithm is kept simple and not uncertainty-aware. Both these works were also reported on as part of the doctoral dissertation of Bruchon [42]. Subsequent results on the AWAKE beamline were presented in 2023 by Velotti et al. [337]. Here, an RL policy was trained on the real accelerator to optimise a matching triplet and solenoid magnets in an effort to maximise the beam brightness on a monitoring screen at the entrance of the plasma cell. In efforts to improve the sample-efficiency and make online training feasible, early results on using GP-based MPC [131] and meta-RL [132] for accelerator tuning at AWAKE were presented by Hirlaender et al. in 2023 and 2024, respectively. An advanced approach using a hybrid quantum actor-critic algorithm based on DDPG, where the critic is replaced by a clamped quantum Boltzmann machine (QBM) and trained with free energy-based reinforcement learning (FERL) was developed on the AWAKE electron steering task by Schenk et al. in 2024 [284]. The authors find that using QBMs with FERL can improve the sample efficiency of training RL policies for accelerator control tasks, however with varying results in terms of the magnitude of the improvement.

In 2021, a very different use case of RL for accelerators is studied [147]. Instead of using the RL policy for control, a simulation study is conducted, where the policy is used to

account for model uncertainties like magnet misalignments when using physics-informed neural network surrogate models for offline optimisation of the beam transmission in the control room.

A neural network policy is trained to regulate the gradient magnet power supply (GMPS) at the Fermilab Booster by St. John et al. in 2021 [311]. The authors propose a first approach of using a neural network surrogate model trained from real data, specifically an LSTM model, to train the policy. A DQN approach is used in combination with discretisation of the continuous 1-dimensional action space to train the policy, but then only evaluated on the surrogate model, where the authors demonstrate that the trained policy performs better than an existing PID controller. In addition, the authors demonstrate how such an RL-trained neural network controller can be compiled to execute on FPGAs.

The doctoral dissertation of Grech [112] in 2021 includes an explorative study applying RL to the tune and orbit feedbacks of the Beam-based Feedback System (BBFS) at the LHC in simulation. Multiple state-of-the-art RL algorithms are trained on the tune feedback and their performance in corner cases, such as environment noise, actuator failure and incorrect tune estimation, is compared to that of the PI controller deployed as part of the renovated BBFS. Multiple policies are also trained for the orbit feedback, demonstrating that applying RL to the latter is significantly harder and requires many more samples for training. An article was later published concluding the tune feedback work in 2022 [113].

An approach for focussing and centring the beam on a diagnostic screen using a series of quadrupole and steering magnets at the ARES accelerator at DESY was first proposed along with results from initial studies in simulation as well as on the real machine in 2021 [78]. By 2022, the proposed RL setup was then extended to be capable of transverse beam parameter tuning to any beam parameters chosen online by the operators and extensively evaluated [158]. Here, the authors demonstrate for the first time that a sim-to-real transfer with a large domain gap between simulations and the real world, as is usually the case with accelerator-related applications, can be achieved using *domain randomisation* and *differential actions*. They show that despite using a simplified simulation optimised for computational speed, a zero-shot transfer to the real accelerator is possible. Building on these results, work from 2023 [360] presents an initial study showing that the domain randomisation approach can be extended to train a lattice-agnostic policy capable of transverse beam parameter optimisation for any set of the same magnets, regardless of the lattice geometry. This reduces the engineering effort of deploying RL agents to multiple subsections of the same accelerator, or even different accelerators. The method was evaluated on ARES as well as on the FLUTE accelerator at Karlsruhe Institute of Technology (KIT). The same authors later conducted an extensive comparative study of RL and BO for accelerator tuning [161, 164]. They find that, while BO requires less engineering effort to deploy, when a tuning method is reused often, the effort to develop an RL tuning solution pays off, as RL is capable of converging faster to a better result. Furthermore, they demonstrate the added capability of RL to deal with actuator failures and upstream changes of the beam. Details of this line of work are the main contribution of this dissertation and presented in Chapter 4.

In 2022, preliminary studies were presented on using RL to adjust various e-cooler and Linac3 parameters to maximise the intensity at the end of the injection plateau [202]. A variational autoencoder (VAE) is used to provide latent representations of Schottky spectra as input to the policy, for the first time in the accelerator field making use of latent observations encoded by the autoencoder as well as a surrogate model for training.

In work first published in 2022 [218], the authors use RL to train a policy to focus

and centre the beam on a viewscreen downstream of a superconducting RF photoelectron injector in one step by controlling solenoid angles and positions. Similar to prior work [71, 73, 74], the policy is trained and evaluated on a differentiable neural network surrogate model trained on simulated data. To bridge the sim-to-real gap, the authors propose an *offset finding* method that finds the offset of the real machine to the simulation in 200 000 iterations, and they evaluate it using random offsets added to the simulated data. Furthermore, the authors conduct a small explainability study on their trained policy to determine changes in which observation component lead to changes in each of the action components.

Another method of achieving zero-shot sim-to-real transfer is explored by Chen et al. in 2023 [52]. Here, an orbit correction task at the China Accelerator Facility for Superheavy Elements (CAFe II) similar to [153] is considered, where, however, unobserved quadrupoles are present in between steerers and BPMs. The authors propose to incorporate *trends*, i.e. the changes of BPM readings, into the observation and the reward to obtain a mapping from observations to actions that is invariant to potential real-world errors. They demonstrate the successful zero-shot sim-to-real transfer but do not further investigate the original size of the domain gap. The authors also develop a column-based CNN architecture for their policy model, but in their experiments this appears to perform worse than a conventional MLP. In a later publication [53], the same authors present preliminary results on exploring a trend-based version of a different algorithm on two different beam tuning tasks, using the CNN architecture from their previous work. Firstly, they train an RL agent to perform orbit correction in three cavity modules of CAFe II separately. In contrast to their previous work, they integrate errors of the real world into the simulation. Furthermore, they appear to use domain randomisation on the transverse misalignments of the solenoids, though not naming it as such. Secondly, the authors apply the same algorithm for transmission efficiency optimisation using solenoid magnets at the light particle injector (LPI). Here, a neural network surrogate model fitted to simulated data is used for training. With both applications, a successful zero-shot sim-to-real transfer is demonstrated.

A number of preliminary, non-peer-reviewed studies on RL applications for accelerators have been presented at various workshops and conferences. These works primarily involve the application of standard RL methods, with limited novel additions, to various accelerator tuning and control tasks. For instance, DQN has been used to optimise RF cavity lengths when designing a new beamline [304], while other studies include a simple 2-dimensional steering task [122], and the control of RF settings to adjust bunch distance and longitudinal emittance in the CERN PS [351]. At the ADIGE beamline of Acceleratore Lineare Per Ioni (ALPI), high-voltage multipole set points have been optimised for beam emittance, using a CNN to observe the  $x, x'$  beam emittance graph image [207]. Similarly, RL has been tested for maximising transmission at the Argonne Tandem Linear Accelerator System (ATLAS) by adjusting electrostatic quadrupole magnets and steerers [208], and at the Shanghai Soft X-Ray Free Electron Laser (SXFEL) for maximising FEL power using steerers along the undulators [48]. Further studies include the maximisation of the transmission efficiency with solenoid magnets at the Light Particle Injection Platform of the Institute of Modern Physics [316], optimisation of the beam injection at the Cooler Synchrotron (COSY) by aligning transverse beam parameters with the injection acceptance via control of quadrupole magnets and steerers [9, 10, 11], tuning the configuration of a non-linear kicker at BESSY II to enhance injection efficiency [290], and compensation for drifts in kicker settings to manage batch spacing during injection into the SPS [258].

There have also been some still preliminary works first applying novel RL methods to accelerator tuning, specifically using a neuralised PID policy to control the spill and thereby the proton beam intensity at the Muon-to-Electron Conversion Experiment (Mu2e) at Fermilab [359], using inverse reinforcement learning (IRL) to learn a reward function to train a policy for re-merging bunches before extraction from the Brookhaven Alternating Gradient Synchrotron (AGS) and injection into RHIC by controlling the RF parameters [104], as well as the development of a human-in-the-loop multi-agent RL system for data quality monitoring (DQM) to inspect data collected from physics experiments, such as those conducted at LHC, and detect anomalies [247].

Efforts to align the work of the accelerator community and foster collaboration between researchers applying RL have been made in the form of the Collaboration for Reinforcement Learning for Autonomous Accelerators (RL4AA), hosting annual workshops, providing a platform for sharing knowledge and experiences, and contributing tutorials to other accelerator venues [277].

A tabular overview of the most notable lines of research in RL for accelerators, the tasks they address, facilities they are applied to and the algorithms used is given in Table 2.1.

Table 2.1: Overview of lines of the most important aspects of different lines of research in RL for accelerators

Task	Authors	Activity	Facility	Algorithm
Orbit correction in a storage ring	Meier et al.	2012	Simulation	Actor-critic method
Tune beam properties at undulator entrance	Edelen et al.	2017 - 2020	Neural network surrogate model of TEU-FEL	Gradient-based policy optimisation
Optimise control system network traffic	Gao et al.	2019	Simulation of RHIC control system	Regret-based learning
Control micro-bunching instability	Boltz et al.	2019 - 2021	Simulation of KARA	DDPG, TD3, SAC, PPO
Dampen betatron oscillations	Scomparin et al.	2022 - present	KARA	Hybrid PPO on a PC and an FPGA
FEL seed laser alignment	Bruchon et al.	2019 - 2021	FERMI	Q-learning with linear function approximation, REINFORCE, DYNA-style
Transmission optimisation in a DTL	Pang et al.	2020	Simulation of LANSCE DTL	A3C
Trajectory steering	Kain et al., Hirilaender et al. and Schenk et al.	2020 - present	AWAKE and simulation of AWAKE	PER-NAF, (PPO, TD3,) DYNA-style, GP-MPC, MAML, DDPG with FERL-trained clamped QBM critic
Maximise beam brightness on a screen	Velotti et al.	2023	AWAKE	TD3 with autoencoder for observations
Regulate GMPS	St. John et al.	2021	Neural network surrogate of Fermilab Booster GMPS	DQN (trained policy compiled to FPGA)
Tune and orbit feedback	Grech et al.	2021 - 2022	Simulation of LHC	PPO, TD3, SAC, NAF2, AF-DYNA with SAC
Transverse beam parameter tuning to choosable beam parameters	Kaiser et al.	2021 - present	ARES (training on simulation)	TD3, SAC, PPO (with domain randomisation for sim-to-real transfer and latte-agnostic policies)
focussing and centring the beam on a screen	Meier et al.	2022	Neural network surrogate	Unspecified policy-gradient approach (with offset finding for sim-to-real transfer)
Orbit correction and transmission efficiency optimisation	Chen et al.	2023	CAFe II and LPI (training on simulation)	SAC (incorporating trends in observation and reward for sim-to-real transfer)

# Chapter 3

## High-Speed Differentiable Beam Dynamics Simulations

Beam dynamics simulations are a crucial part of the design and operation of particle accelerators. They also play an important role in the development of autonomous tuning methods. RL training in particular requires a large amount of data in the form of environment interactions in order to yield a well-performing policy. For example, 6 million samples are needed to successfully train a policy on a transverse beam parameter tuning task in Chapter 4, and 50 million samples are needed to train a policy on the FEL intensity tuning task in Chapter 5. Outside of accelerator tuning, some complex tasks have even been found to require orders of magnitude larger numbers of training samples [59, 339].

The general scarcity of beam time makes collecting experimental data for machine learning methods such as RL a significant bottleneck. Gathering at least partial datasets in simulation can alleviate this problem to some degree, but existing accelerator simulation codes have mostly been developed with a focus on the design phase of accelerators, where high-fidelity and physical correctness are critical, while the number of required simulations has traditionally been limited. As a result, existing simulation codes, such as ASTRA [94], Bmad [275], Elegant [30], MAD-X [49] and Ocelot [3], can often take in the order of minutes to several hours to complete a single simulation, making data collection with existing simulation codes an impractical proposition to fulfil the growing demand for large datasets needed to train machine learning models. As such the development of high-speed beam dynamics simulations to enable the fast collection of large datasets is an indispensable prerequisite for the development of autonomous tuning methods for particle accelerators.

Moreover, a key challenge when designing the observations passed to RL policies is the availability of diagnostics that can provide the necessary information about the state of the accelerator. The most useful diagnostics are often not available, would require time-consuming setup and measurements, or they are invasive, making them unsuitable for use during beam delivery to experiments. Fast-computing beam dynamics models can provide a solution to this problem by enabling the development of real-time *virtual diagnostics* delivering otherwise unattainable insights into the accelerator's state at any time. The information provided by virtual diagnostics can also give useful data about experiments, help operators tune the accelerator as well as jump-start the development of intelligent controllers for autonomous tuning [158, 357].

Additionally, differentiability is a highly desired feature for beam dynamics models. At the moment, numerical optimisation is an important tool for accelerator design, tuning and model calibration [325, 364]. Advanced numerical optimisation methods like BO

have been used to achieve impressive results [271], but demands to solve optimisation problems of increasing dimensionality are growing, and BO may struggle to efficiently optimise objective functions with more than a few dozen degrees of freedom [271]. In the field of machine learning, gradient-based optimisation techniques have been used to effectively train large neural networks with billions of parameters [107, 327]. Coupled with automatic differentiation, which is key to enabling tractable computation of gradients for complex models, differentiable models therefore make optimisations over the large parameter spaces of accelerator facilities more feasible. They can be used to great effect for a wide range of applications, including gradient-based accelerator tuning, gradient-based system identification to infer hidden parameters for virtual diagnostics, as prior mean models in BO, and as differentiable RL environments for gradient-based RL training through the true policy gradient.

In the following, two high-speed differentiable beam dynamics modelling methods are introduced, the first using a start-to-end trained neural network, and the second using a handcrafted differentiable simulation code based on PyTorch and optimised for speed. The latter plays a key role in the training of RL policies in Chapters 4 and 5. For example, in Sections 4.1 and 4.4, the use of the fast-executing simulation code introduced in Section 3.2 is shown to reduce the training time of an RL policy from over 12 days to roughly 1 hour. The application of these models outside the use case of RL is demonstrated on multiple examples in this chapter, including gradient-based tuning, gradient- and inference-based system identification for virtual diagnostics, and physics-based BO priors. Moreover, in Section 3.3, the integration of the handcrafted high-speed differentiable simulation code with modular neural network surrogates is demonstrated, showing how both approaches can be combined to take advantage of each of their respective strengths. This chapter is then summarised in Section 3.4.

### 3.1 Start-to-End Neural Network Surrogates

An effective approach for constructing a computationally efficient and differentiable model is to leverage the power of neural networks. As general-purpose function approximators, neural networks can in theory approximate any function to a high degree of accuracy and have, in recent years, been shown to be capable of achieving incredible results in numerous domains [46, 59, 195, 333]. Neural networks are also differentiable, and neural network inference is highly parallelisable, which means that it can be accelerated using dedicated hardware such as graphics processing units (GPUs). As a result, the particle accelerator community has also started to adopt neural networks in a number of different capacities, including surrogate modelling [160, 355].

To create a differentiable model using neural networks, one must first collect a large dataset of input-output pairs that represent the function to be approximated. In the case of modelling particle accelerators, such a dataset can be collected in either one of two ways: by running many simulations with a high-fidelity simulation code, or by collecting data from real-world accelerator operations.

The advantage of training a neural network surrogate is that the resulting model can be highly accurate, while still being fast to execute, with very little regard to the computational complexity of the underlying function. Neural network surrogates are especially useful when collective effects are involved and the underlying physics are complex and expensive to evaluate. What is more, the engineering effort required to make such a model operational can be relatively low, as the software ecosystem for neural network training is very mature,

and data may already be available from prior simulations or data acquisition (DAQ) systems.

The disadvantage of training neural network surrogate models is that neural network training typically requires comparatively large and varied datasets. While DAQs can provide a large amount of data, normal accelerator operations are characterised by a high degree of repetition and long periods of constant settings, which means that parasitically collected data may not be varied enough for training a neural network model and that dedicated measurement campaigns may be required. Neural networks are also a mostly black-box model, which means that they may not be as interpretable as physics-based models, and that they may not generalise well to inputs that are outside the range of the training data, though some methods such as physics-based neural networks have been developed to try and address this [146]. Moreover, the start-to-end nature of these models means that their application is limited to the very specific task a model was trained on, and that it may not be easily adapted to other tasks, requiring a new model to be trained from scratch for each new task.

In this section, the training and application of a start-to-end neural network model is shown at the example application of virtual diagnostics for longitudinal phase space (LPS) reconstruction at the European XFEL. The described approach was originally presented in [160] and [326].

### 3.1.1 Longitudinal Phase Space Diagnostics at European XFEL

Knowledge and control of the shot-to-shot LPS of an electron beam is critical to meet the performance requirements of modern particle accelerators. For example, the resonance properties of the FEL amplification process are controlled by the linear taper of the undulators, which in turn is determined by the wakefield energy losses in the undulators' vacuum chamber. Estimating wakefield energy losses requires knowledge of the electron beam's current profile, which can be obtained from the LPS [326].

A direct measurement of the LPS can be achieved using transverse deflecting structures (TDSs). These structures facilitate the mapping of the longitudinal bunch distribution onto a transverse plane, enabling the observation of the LPS on a diagnostic screen [82]. However, conventional methods, like using a TDS to measure the LPS, are invasive and constrained by the limitations of the diagnostic screen, such as its resolution and saturation limits. These measurements are also time-consuming, often requiring extensive setup and calibration before accurate data can be taken. As a result, measurements of the LPS cannot be undertaken during beam delivery to experiments, limiting their usefulness to online accelerator operation. Similarly, taking LPS measurements during the setup of the accelerator significantly increases the time required for accelerator tuning, thereby reducing the time available for useful beam operations. In practice, all of this means that the LPS is often not measured, and phase space information is not taken advantage of, leading to suboptimal beam quality and time-consuming guesswork during accelerator tuning and working point switching.

Methods have been developed to reconstruct the LPS from other non-invasive measurements. At the European XFEL, a Fourier-based method employing spectroscopy of the coherent radiation emitted by the electron bunch [199] is deployed. This method takes advantage of the fact that the coherent radiation spectrum and the current profile of the electron bunch are linked via the Fourier transform [183]. Coherent diffraction radiation (CDR) is generated by the electron bunch passing through an aperture inside an aluminium

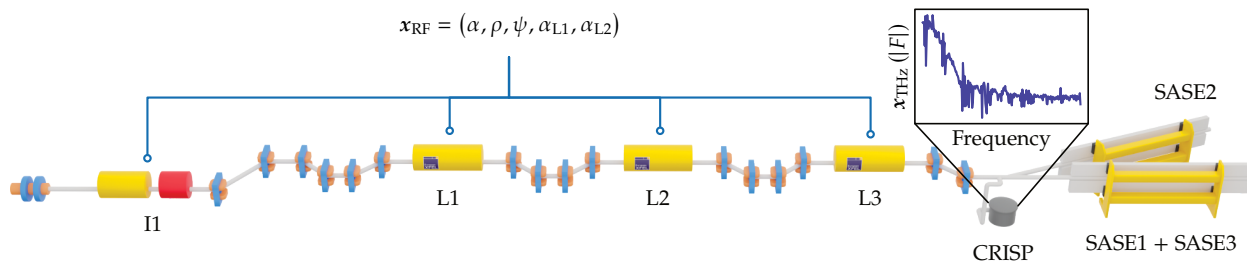


Figure 3.1: Simplified layout schematic of the European XFEL facility with the inputs to the neural network surrogate model highlighted.

screen just upstream of the kicker-septum distribution system. While the electron bunches proceed undisturbed to the SASE beamlines, the THz and infrared spectrum of the CDR is detected by a multistage grating spectrometer called *CRISP* [348], the location of which is highlighted in Fig. 3.1. *CRISP* at the European XFEL is operated at the MHz electron bunch repetition rates and is used as a noninvasive current profile diagnostic [198]. While this Fourier-based reconstruction method has been shown to have good accuracy in operation [198], the spectroscopic measurements lack phase information, which is required for a direct identification of the current profile. Instead, the phase function is obtained through a combination of analytical and iterative phase retrieval methods as described in [287]. This can lead to various faults in the reconstructed current profile. For example, the head and tail of the beam cannot be distinguished [286], which can cause flips in the longitudinal dimension. While this limitation does not pose a problem for operation or compression setups, it can impede the accurate estimation of wake losses in the lasing beam core if its position within the bunch is not known, especially when dealing with an asymmetrical current profile. Furthermore, this method is expensive to compute. With up to 2700 bunches per train and 10 Hz operation, this makes real-time bunch-by-bunch reconstruction of the entire bunch train infeasible. Unlike other methods mentioned in the following, this method also cannot reconstruct the full 2-dimensional LPS image, but only the 1-dimensional current profile.

The use of start-to-end neural network models for building virtual diagnostics of the LPS has been explored at other facilities as so-called *spectral* virtual diagnostics [124, 369] using the THz spectrum of the CDR as input to the neural network, and as *scalar* virtual diagnostics [80, 81, 370] using RF settings as input to the neural network. Both neural scalar and spectral virtual diagnostics improve on the Fourier-based method in that they are more efficient to compute, especially on hardware accelerators like GPUs that significantly speed up the inference of neural networks. Furthermore, they allow for the reconstruction of the full 2-dimensional phase space image instead of just the 1-dimensional current profile. Neural scalar and spectral virtual diagnostics, however, each have disadvantages. Spectral virtual diagnostics, just like the Fourier-based reconstruction, do not have access to phase information. Instead, the neural network must learn statistics over the training dataset to estimate the phase information. As a result, neural network-based spectral virtual diagnostics are likely to fail for bunches with uncommon phase space distributions. Scalar virtual diagnostics, on the other hand, have access to phase information. They are however susceptible to machine drifts and model calibration errors like false RF setting readbacks [80, 81, 124], leading to significant errors in the reconstructed LPSs.

### 3.1.2 Combined Scalar- and Spectral Virtual Diagnostics

By combining scalar and spectral inputs, it is possible to create a virtual LPS diagnostic that has access to phase information, while taking advantage of the THz spectrum in order to be robust against model calibration and RF readback errors. To this end, a start-to-end neural network model is proposed. This model receives two inputs: a 5-dimensional vector of RF settings

$$\mathbf{x}_{\text{RF}} = (\alpha, \rho, \psi, \alpha_{\text{L1}}, \alpha_{\text{L2}}) \quad (3.1)$$

and a 240-dimensional THz form factor  $\mathbf{x}_{\text{THz}}$  sampled from 0.7 THz to 58 THz. These are then concatenated and passed through an MLP. Two variants of the model are trained, one outputting the full 300 by 300 pixel LPS image  $\mathbf{y}_{\text{LPS}}$  and the other outputting the 300 ordered samples of bunch's longitudinal current profile  $\mathbf{y}_{\text{I}}$ . The neural network is trained in a supervised setup to minimise the difference between the inferred LPS or current profile, and their respective ground truth values generated from start-to-end simulations.

### 3.1.3 Achieving High Resolution at Any Scale

When predicting either the LPS or the current profile, one has to choose the sample positions in each dimension for which the model is expected to predict the LPS and current magnitudes. However, bunch dimensions can vary greatly, with both bunch length and energy spread potentially varying by roughly an order of magnitude, as can be seen in Fig. 3.3. In previous work on neural network surrogate model-based virtual LPS diagnostics, this produced an inevitable trade-off between the resolution of the model output, the range of permissible bunch dimensions and the computational requirements of the model. In order to cover a bunch of large dimensions, the samples must be spread out over a wide range, while a very large number of samples have to be predicted within that wide range to adequately resolve small bunches. Not only does this needlessly increase the complexity of the neural network model, it also means that a lot of the time, most of the neural network's outputs are not predicting any useful information. This can lead to the model weights getting stuck in a local optimum, where the model always infers 0.0 on these outputs, because most bunches do not cover the entire predicted range and samples at the edge of the range are therefore 0.0 throughout most of the training data.

This work proposes to eliminate the trade-off by adding a second output to the final layer of the neural network that is either the length  $y_s = s$  of the bunch along the longitudinal dimension when only reconstructing the current profile, or the 2-dimensional range  $\mathbf{y}_{\text{range}} = (s, \delta_E)$  of the bunch in both dimensions of the LPS. A fixed number of samples, in this case 300, are then equidistantly arranged over the predicted bunch dimensions. This way, the resolution of the prediction automatically adapts to the dimensions of each bunch, always resolving all relevant features, while ensuring that the entire bunch is captured. The resulting neural network model has the same inputs and outputs as described in Section 3.1.2, but adds the predicted range outputs  $\hat{y}_s$  in the 1-dimensional case and  $\hat{\mathbf{y}}_{\text{range}}$  in the 2-dimensional case. Considering the 2-dimensional LPS prediction, the model  $\mathcal{M}$  is given as

$$(\hat{\mathbf{y}}_{\text{LPS}}, \hat{\mathbf{y}}_{\text{range}}) = \mathcal{M}(\mathbf{x}_{\text{RF}}, \mathbf{x}_{\text{THz}}). \quad (3.2)$$

In between the input and the output layer, there are between 2 and 6 hidden layers, as is illustrated in Fig. 3.2. Each hidden layer is optionally followed by batch normalisation [144] before the activation. The outputs use a different activation from the hidden layers. Namely, *softplus* is chosen as the output activation function. The latter ensures that all outputs

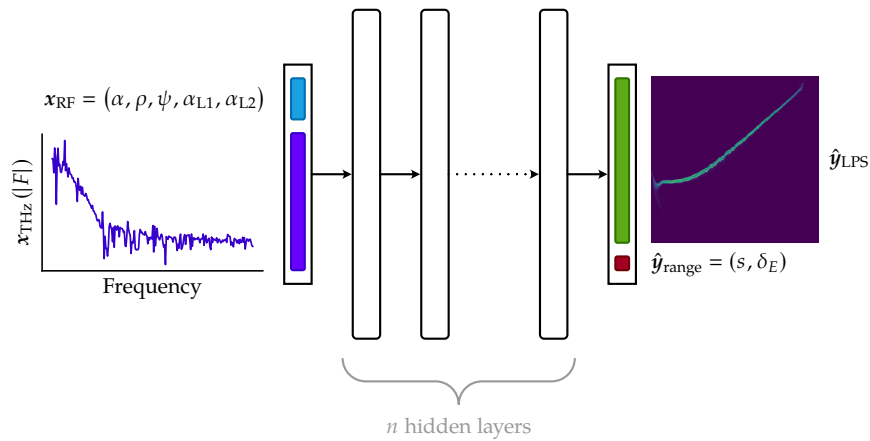


Figure 3.2: Flowchart of the start-to-end neural network architecture at the example of the 2-dimensional LPS prediction [160]. The RF settings and THz spectrum are input to the neural network on the left. The information then flows through multiple hidden layers of the MLP. The bunch dimension and LPS outputs are shown on the right.

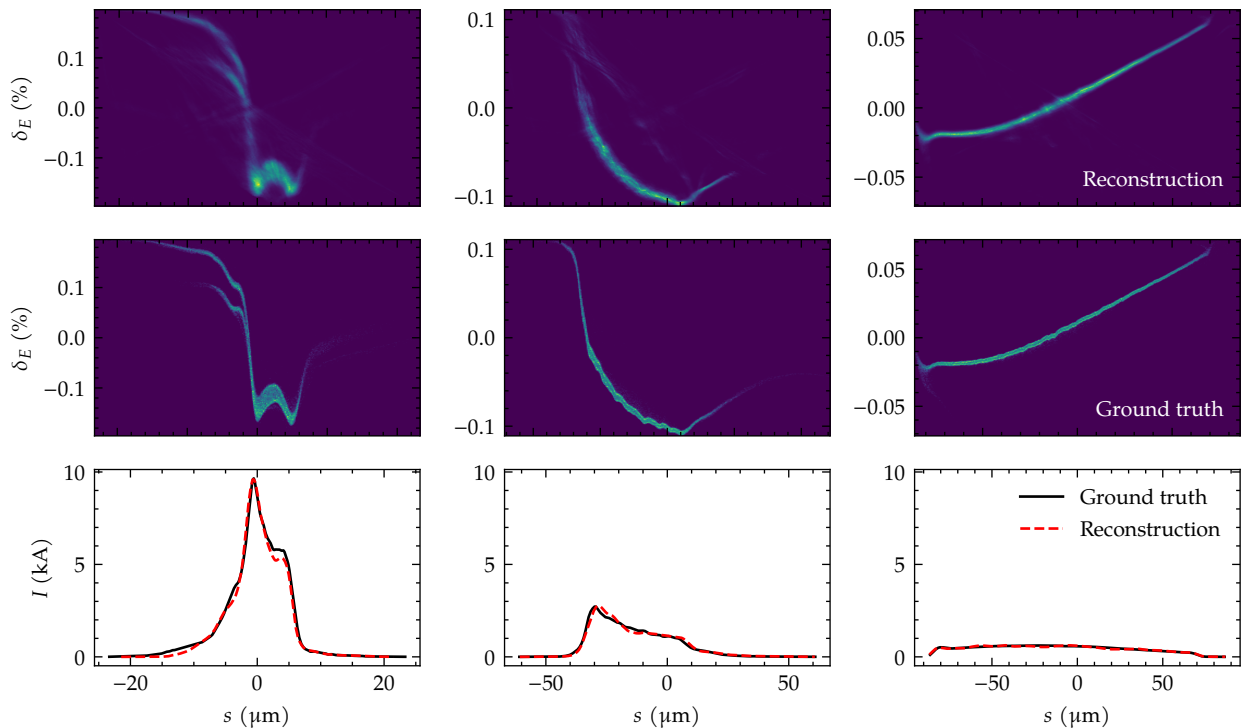


Figure 3.3: Example predictions of the LPS in the first row and the current profile in the second row for three different samples from the test dataset [160]. The ground truth LPSs are shown in the second row. The ground truth current profiles are shown along with the predictions in the third row. Note that the shown current profile predictions are not computed from the LPS predictions, but inferred by a separate neural network trained to output the 1-dimensional current profiles.

Table 3.1: Hyperparameters used during training of the reconstruction models

Hyperparameter	Current profile	LPS
Number of hidden layers	3	4
Hidden layer width	267	681
Hidden activation	ReLU	ReLU
Batch normalisation	Yes	Yes
Number of training epochs	76	95
Learning rate	0.003	0.0008
Batch size	66	70

are strictly positive, as they physically should be, while also ensuring that there remains some gradient when the output should be zero according to the ground truth. If a ReLU activation function were used on the output layer, the gradients would be lost when the model predicts 0.0, resulting in fully trained models with some “dead” output neurons that always predict 0.0. An MSE loss function over all output samples

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \text{MSE}(\mathbf{y}_{\text{range}}, \hat{\mathbf{y}}_{\text{range}}) + \text{MSE}(\mathbf{y}_{\text{LPS}}, \hat{\mathbf{y}}_{\text{LPS}}) \quad (3.3)$$

with

$$\text{MSE}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2, \quad (3.4)$$

here shown for the 2-dimensional LPS prediction, is used with both outputs  $\mathbf{y}_{\text{range}}$  and  $\mathbf{y}_{\text{LPS}}$  weighted equally.

To train the model, the *Adam* optimiser [173] is used with  $\beta_1 = 0.9$  and  $\beta_2 = 0.999$ . Hyperparameters, such as the number and width of hidden layers, or the learning rate, were tuned using BO to minimise the loss over the validation dataset. The final used hyperparameters for both 1-dimensional current profile reconstruction and 2-dimensional LPS reconstruction are listed in Table 3.1. The dataset used for this example contains 32 000 samples and was randomly split into training, validation and test sets according to the ratios (80 %, 10 %, 10 %). The RF settings and THz formfactors are normalised over the training dataset to follow a standard normal distribution. Current profiles, bunch lengths, LPS images and LPS image dimensions are normalised to a range of [0, 1].

### 3.1.4 Results

As can be seen in Fig. 3.3, the outputs of the start-to-end neural network virtual diagnostic models match the ground truth closely on unseen samples for both the 1-dimensional and 2-dimensional cases. Moreover, especially in the 2-dimensional case, no discernible artefacts are present in the reconstructions, setting this method apart from previous neural network-based LPS reconstruction methods. This advantage can be attributed to the adaptive resolution of the model, which ensures that all of the model’s outputs play a meaningful role in the prediction of every training sample, reducing the amount of randomness they encounter during the training and stabilising their outputs in the process.

The results listed in Table 3.2 confirm the high accuracy of both models in terms of four metrics. Here, the *sample error* is the MAE between the predicted and ground truth current profile or LPS samples normalised to the peak value of the respective ground truth

Table 3.2: Results achieved by the reconstruction models

Metric	Current profile		LPS	
	Median	Mean	Median	Mean
Loss	0.000 087	0.000 88 ± 0.0039	0.0012	0.0016 ± 0.0015
Sample error (%)	2.1	4.4 ± 6.9	0.03	0.9 ± 4.3
Length error (%)	2.9	4.0 ± 4.0	3.3	3.9 ± 3.2
Current error (%)	3.3	4.5 ± 4.7	16.5	18.1 ± 11.6

current profile or LPS image. The *length error* is the absolute error between the predicted and ground truth bunch lengths normalised to the ground truth bunch length. Both of these errors are computed on values directly inferred by the models. To test the models' ability to predict physically accurate current profiles and LPS images, the *current error* is computed as the absolute error between the bunch current of the predicted and ground truth current profiles and LPS images, respectively, and normalised to the bunch current of the ground truth current profile or LPS image. All errors are on the order of a few percent and therefore very low. Only the current error of the LPS model is notably higher, but at less than 20 % still reasonably accurate. This higher error is likely caused by the increased difficulty of predicting a 2-dimensional image compared to a 1-dimensional current profile.

Figure 3.4 shows how the performance changes for different bunches with different lengths and currents. The model performance is found to be consistent across the range of bunch lengths and currents in the test dataset. A small increase in the spread of the errors is observed for long bunches and bunches with low currents. This is likely due to the fact that in these cases most samples within the current profile and LPS image have relatively small values, amplifying the relative scale of errors.

Models trained as part of this work have been integrated into a GUI application that has been deployed and tested in the control room at European XFEL. Integrated into the control system in this way, the presented model can inform the tuning process and help human operators set up the accelerator more efficiently and effectively, achieving better beam quality and reducing the time required to get there.

This demonstrates an advanced method of training a start-to-end neural network model as a feed-forward virtual diagnostic. Currently providing diagnostic information to human operators, the output of this model may also be used as input to advanced autonomous tuning methods, potentially allowing them to outperform methods that only have access to less informative diagnostics. Moreover, being a neural network, inference of the model is fast and could be used to build an RL environment to train a policy that tunes the LPS through the RF settings. With neural networks being differentiable, the same tuning task could also be achieved using a gradient-based optimisation method on the surrogate model.

While the presented trained model can only be used for applications involving the LPS at European XFEL, the method is general and can be employed to train surrogates of other diagnostics and facilities through little more than training on the appropriate dataset.

## 3.2 Physics-based High-Speed Differentiable Simulations

Start-to-end neural network surrogate models as presented in Section 3.1 have three main disadvantages: they require retraining and possibly redesign in order to model different parts of an accelerator facility; they require a large amount of data to train; and their

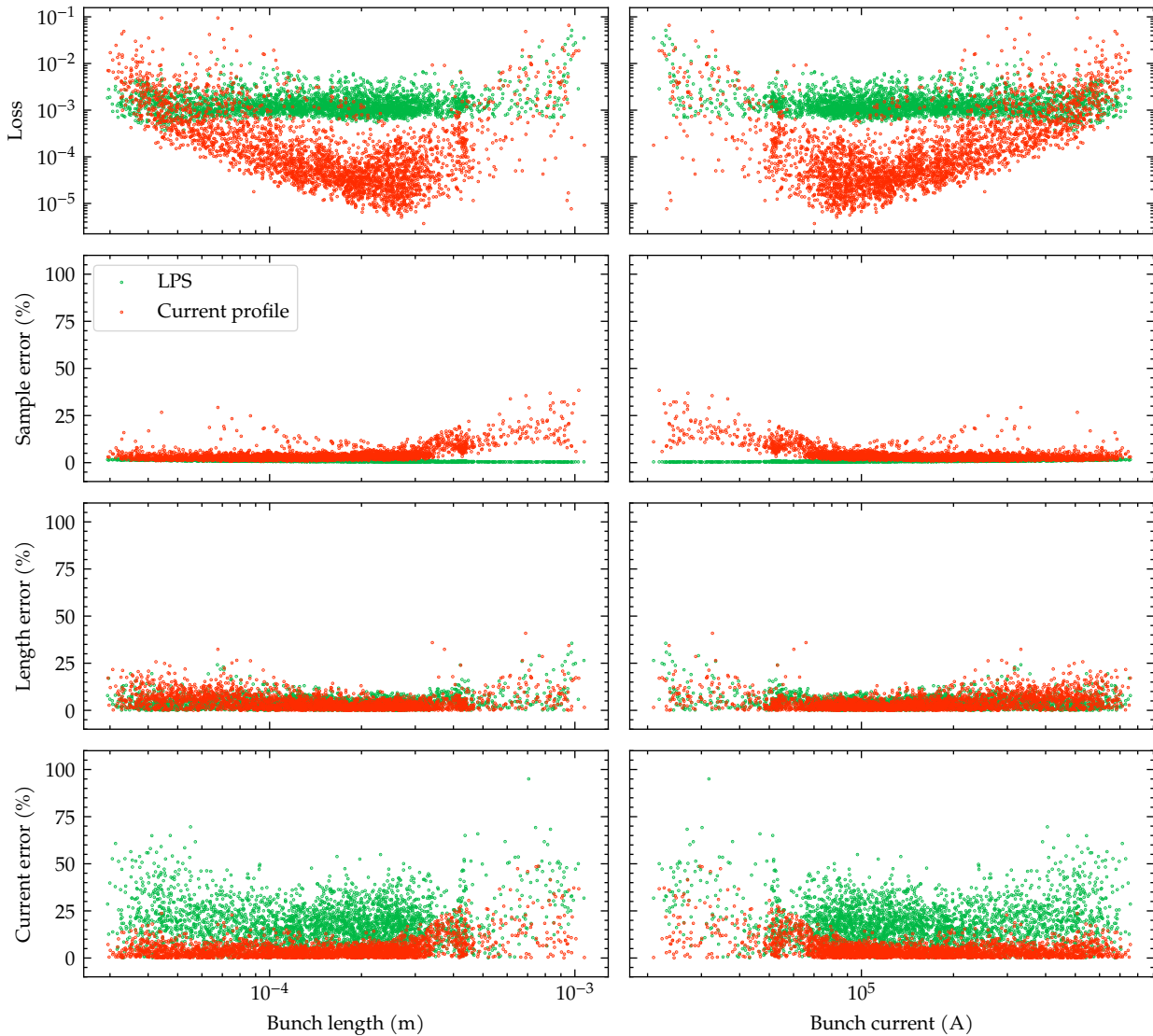


Figure 3.4: Loss, bunch length error and bunch current error between all ground truth samples from the test set and the predictions of the current profile (red) and LPS image (green) reconstruction models plotted over the bunch length and current of the ground truth samples. Bunch length and current errors are normalised to the ground truth values to account for the fact that bunch lengths and currents in the dataset vary by multiple orders of magnitude.

behaviour is not well-defined, with models potentially deviating significantly from the truth, especially in regions of the input space where the model has not seen a lot of data during training. These disadvantages can be addressed by using handcrafted physics-based models. By implementing such models with the help of frameworks like PyTorch [248] or JAX [37] it is possible to take advantage of their automatic differentiation capabilities to calculate the partial derivatives up to arbitrary orders for all the parameters using the chain rule. Moreover, these frameworks are highly optimised for tensorised computation and can be accelerated using dedicated hardware, such as GPUs. Thanks to the very active development efforts by the machine learning community, any model implemented using these frameworks can also easily take advantage of the mature software ecosystem of optimisation algorithm implementations and other utilities surrounding them. This makes PyTorch and JAX ideal candidates for implementing handcrafted high-speed physics-based differentiable models that can be used in a wide range of applications.

In this section, a novel high-speed differentiable simulation code called *Cheetah* is introduced. Cheetah is a Python package implemented in PyTorch that accelerates beam dynamics simulations by multiple orders of magnitude through tensorised computation and several speed optimisation methods, while providing built-in support for automatic differentiation to efficiently compute gradients of beam dynamics. Cheetah was first introduced as a fast beam dynamics simulation code to accelerate data acquisition for machine learning applications in [314], and was later extended with many new features, most notably differentiability, in [162], which is the work that is presented here.

The implementation of Cheetah is introduced in Section 3.2.1, including its overall design, speed optimisations and a benchmark of its speed against other codes. Afterwards, three different example applications of Cheetah are demonstrated in Section 3.2.2, showcasing the use of its automatic differentiation capabilities for gradient-based transverse beam tuning, gradient-based system identification and as a prior for BO. Furthermore, the use of Cheetah for coupling physics-based simulations with neural network surrogates is demonstrated in Section 3.3. In the context of this dissertation, Cheetah serves as an important building block for the training of RL policies for tuning particle accelerators in Chapters 4 and 5. Moreover, Cheetah is used to explore gradient-based RL to train neural network policies for accelerator tuning in Section 5.3.

### 3.2.1 Fast Differentiable Tracking in Python

The overarching goal in implementing Cheetah is to provide a differentiable beam dynamics code with improved speed over existing simulation codes to be used for machine learning applications. Here, the conscious decision is made to trade accuracy to achieve these speed improvements. This means that Cheetah, while faster than existing codes, may be lower fidelity than they are. With machine learning applications this is a worthwhile trade-off. Methods like domain randomisation [323] enable neural networks trained on inaccurate simulated data to effectively generalise to the real-world domain. Moreover, initial training of a machine learning model on cheap low-fidelity data followed by fine-tuning on high-fidelity data is a widely used method to speed up the training of machine learning models. At the same time, Cheetah is designed to integrate seamlessly with popular machine learning tools. Cheetah is intended to be used both as a tool in machine learning applications, e.g. to support the training of neural networks, and as an application of machine learning itself, e.g. through the integration of neural network models in its simulation pipeline. Last but not least, another objective is to make Cheetah easy to use,

easily extensible and to follow best practices in its implementation with high-quality code.

To this end, Cheetah is implemented in the *Python* programming language, which hosts an extensive machine learning ecosystem and is widely used in scientific computing. Cheetah employs the *PyTorch* [248] framework. While the primary purpose of PyTorch is the implementation of machine learning algorithms, its fast tensor compute capabilities, strong GPU support and automatic differentiation features make it an ideal fit for fast parallel scientific computation.

To validate that Cheetah models the physics of beam dynamics accurately and to ensure high code quality, Cheetah makes use of various continuous integration (CI) pipelines. Almost 500 tests are implemented to verify not only that Cheetah runs without errors, but also that its outcomes are physically plausible and match those computed by Ocelot [3, 325], ASTRA [94] and Bmad [275]. Automatic code formatting and linting are also used to enforce a high standard of readability and maintainability, while ensuring that implementations follow the best programming practices of PEP8 [332] and minimising the incidence of elusive future errors. The official GitHub repository [157] is set up with clear contribution guidelines and well maintained in an effort to foster future collaboration. To lower the barrier of entry and ease installation, stable versions of Cheetah are regularly deployed to *PyPI*. Reference documentation and some use case examples for Cheetah are made available via *Read the Docs* [159].

### Beam Tracking in Cheetah

At its core, Cheetah is made up of two main object classes, `Beam` and `Element`, which provide implementations of charged particle beams and accelerator elements, such as magnets and drift sections, respectively. Both of these inherit from PyTorch's `nn.Module`, allowing their parameters to be optimised with the tools provided by PyTorch when set to an `nn.Parameter` instead of a `Tensor`.

Cheetah provides two ways to represent the beam, a `ParticleBeam` with the coordinates of each macroparticle and a `ParameterBeam` with only statistical values representing the particle distribution. Both inherit from a `Beam` base class. In `ParticleBeam`, each particle is represented by a 7-dimensional vector

$$\mathbf{p} = (x, x', y, y', \tau, \delta, 1), \quad (3.5)$$

where  $x$  and  $y$  are the horizontal and vertical positions,  $x'$  and  $y'$  are the slopes in trace space,  $\tau$  is the longitudinal displacement, and  $\delta$  is the momentum offset with respect to the nominal energy. The 6-dimensional vector usually used in beam dynamics is expanded at the end, analogous to an affine space, allowing effects like thin-lens magnets and misalignments to be coherently represented as transfer map multiplications and enabling some of the speed optimisations discussed later in this section. A full `ParticleBeam` is then represented as a tuple

$$\mathcal{B}_{\text{particle}} = (P, \mathbf{q}, E) \quad (3.6)$$

of a matrix of particles  $P \in \mathbb{R}^{n \times 7}$ , where  $n$  is the number of macroparticles, a vector of particle charges  $\mathbf{q} \in \mathbb{R}^n$ , and a reference beam energy  $E$ .

For applications that require faster computations and do not require modelling of the bunch substructures, Cheetah provides a second representation, the `ParameterBeam`. It assumes the particle distribution to be Gaussian and represents it using a 7-dimensional vector  $\mu$  for the position of the reference particle in phase space according to Eq. (3.5) and

a covariance matrix  $\Sigma$ . Consequently, a `ParameterBeam` is represented as a tuple

$$\mathcal{B}_{\text{parameter}} = (\mu, \Sigma, q, E), \quad (3.7)$$

where  $q$  is the total charge of all macroparticles in the beam.

Furthermore, the `Beam` subclasses implemented in Cheetah offer convenient computation of various of their properties. Both beam representations support generating Gaussian beams, and importing beams from files saved by other particle tracking codes, a feature which is further discussed later in this section. In addition, `ParticleBeam` instances can be generated with regularly spaced macroparticles.

The `Element` class represents accelerator beamline elements, such as magnets, drift sections or diagnostic instruments. Each subclass must implement a `track` method that transforms an incoming beam entering the element to an outgoing beam that was affected by the element and is exiting it. To add a new element to Cheetah, one simply subclasses the `Element` base class and implements the `track` method. The `track` method can implement arbitrary computations from simple matrix multiplications for first-order tracking to more complex behaviours like higher-order transfer maps for non-linear elements, beam image computations for diagnostics screens or neural network inference. By default, the `track` method in Cheetah computes linear beam dynamics using a linear transfer map  $R_{\text{Cheetah}} \in \mathbb{R}^{7 \times 7}$  that is already provided as

$$R_{\text{Cheetah}} = \left( \begin{array}{c|c} R_0 & \vdots \\ \hline 0 \cdots 0 & 1 \end{array} \right), \quad (3.8)$$

with  $R_0 \in \mathbb{R}^{6 \times 6}$  being the standard transfer matrix based on [40]. Some elements already present in Cheetah also implement more complex behaviours. For example, the transverse motion in accelerating cavities is modelled according to [264]. In the remainder of this dissertation, all transfer maps  $R$  are assumed to be of the form  $R_{\text{Cheetah}}$ . Transfer maps for some relevant elements used in this dissertation are listed in Appendix A.

To track a beam through an element with a transfer matrix  $R$ , the default implementation either computes the particle matrix  $P_{\text{out}}$  of the outgoing `ParticleBeam` as

$$P_{\text{out}} = P_{\text{in}} R^T \quad (3.9)$$

from the particle matrix  $P_{\text{in}}$  of an incoming `ParticleBeam`, or

$$\begin{aligned} \mu_{\text{out}} &= R \mu_{\text{in}} \\ \Sigma_{\text{out}} &= R \Sigma_{\text{in}} R^T \end{aligned} \quad (3.10)$$

for a `ParameterBeam` with the characteristic parameters  $\mu_{\text{in}}$  and  $\Sigma_{\text{in}}$ .

When adding elements to Cheetah that follow linear beam dynamics, it is therefore sufficient to implement a `transfer_map` method that returns the first order transfer matrix  $R$  of the elements. In this case, the `track` method does not need to be implemented, and Cheetah will automatically compute the outgoing beam using the transfer matrix returned by the `transfer_map` method.

At the time of writing, Cheetah has support for drift sections, dipole magnets with adjustable face angles (e.g. `SBends` and `RBends`), thin-lens corrector magnets, quadrupole magnets, cavities, TDSs, BPMs, markers, diagnostic screen stations, apertures, solenoid magnets, elements with custom transfer maps, and even a first-of-its kind experimental differentiable implementation of a space charge kick. In addition, Cheetah provides a special

---

**Listing 1** Example Python code for tracking an ASTRA beam through a FODO cell using Cheetah.

---

```
# Load initial beam distribution from ASTRA tracking
incoming_beam = ParticleBeam.from_astra("beam_in.ini")

# Create a FODO lattice
fodo_cell = Segment(
    [
        Drift(length=torch.tensor(0.2)),
        Quadrupole(length=torch.tensor(0.2), name="Q1"),
        Drift(length=torch.tensor(0.4)),
        Quadrupole(length=torch.tensor(0.2), name="Q2"),
        Drift(length=torch.tensor(0.2)),
    ]
)

# Change the magnet strengths
fodo_cell.Q1.k1 = torch.tensor(10.0)
fodo_cell.Q2.k1 = torch.tensor(-9.0)

# Tracking through the FODO cell
outgoing_beam = fodo_cell.track(incoming_beam)
```

---

Segment subclass of Element. It represents a sequential lattice of accelerator components and supports nesting of other smaller Segment elements.

An example Python code for tracking a beam distribution loaded from ASTRA results through a FODO cell using Cheetah is shown in Listing 1.

## Speed Optimisations

Cheetah achieves its speed through several automatic and opt-in optimisations. First of these is the use of PyTorch, which is itself implemented in C++ and Compute Unified Device Architecture (CUDA), and is well-optimised thanks to widespread community support. PyTorch holds a key speed advantage over established packages like NumPy in its built-in ability to run on GPUs supporting CUDA or Metal Performance Shaders (MPSs), which can provide significant speed improvements for massively parallel computations such as multi-particle tracking.

Moreover, Cheetah takes advantage of the fact that the transfer matrices of adjacent elements can be combined, if they follow linear beam dynamics and perform no other operations when a beam is tracked through them. Combining the transfer matrices lets Cheetah perform only one computation like those in Eqs. (3.9) and (3.10), rather than having to perform them for each of the combined elements separately. Cheetah automatically identifies sequences of elements that can have their transfer matrices combined. This optimisation is named *dynamic transfer map reduction*. For example, let us consider a Segment is made up of an alternating sequence of dipole magnets and drift sections, which follow linear beam dynamics, followed by an active diagnostic screen station, which records the beam in addition to performing linear beam dynamics, and another sequence of alternating dipole magnets and drift sections. Cheetah will automatically recognise that the transfer matrices ( $R_{M_1}, R_{D_1}, R_{M_2}, R_{D_2} \dots$ ) of the elements upstream of the screen station can be combined into a single transfer matrix  $R_{A1}$ , and that the same can be done for

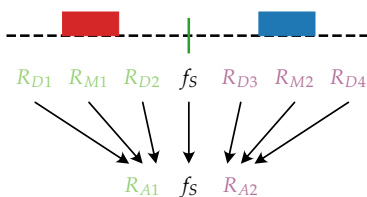


Figure 3.5: Visualisation of a simple example for transfer map reduction. The tracking function of the diagnostic screen is denoted by  $f_s$ . It cannot be reduced along with other transfer maps. The transfer maps of drift sections and magnets upstream of the screen ( $R_{D1}, R_{M1}, R_{D2}$ ) and downstream of the screen ( $R_{D3}, R_{M2}, R_{D4}$ ) can be reduced to two transfer maps  $R_{A1}$  and  $R_{A2}$ , one up- and the other downstream of the screen. A schematic illustration of the example lattice is shown at the top, with the magnets shown in red and blue, the screen station in green, and the drift sections as unoccupied sections on the dashed line.

the sequence of elements downstream of the screen station. A simple example following this description is illustrated in Fig. 3.5. This optimisation can be influenced by the user to some extent. Some elements, such as diagnostic screen stations and BPMs, support being activated or deactivated, based on whether the user intends to use their irreducible functionalities. Cheetah makes transfer map reduction decisions based on the activation status of these elements. Some other elements automatically determine whether they can be optimised through transfer map reduction. Cavities, for example, are effectively drift sections when they are inactive and therefore produce a drift transfer matrix, which is automatically combined with other transfer maps. When they are active, on the other hand, they have more complex effects on the beam and do not take part in transfer map reduction. Because transfer map reduction does not always need to be performed every time a beam is tracked through a Segment, Cheetah provides an opt-in variant of the same optimisation, where the user can tell Cheetah which elements may be changed in the future. All other elements are then frozen, allowing Cheetah to perform *static transfer map reduction*. This optimisation can be very effective when only a few parameters are changed between consecutive simulations on large lattices.

In a similar vein, Cheetah can be commanded to find all inactive elements that are effectively drift sections, and replace them with actual drift sections, which are generally faster to compute. In addition, drift sections in Cheetah are pure linear beam dynamics elements, meaning they can be merged with adjacent linear beam dynamics elements, either dynamically and on-the-fly, or statically before tracking.

Especially when combining these optimisations, they can significantly speed up computations in Cheetah. Critically, all the implemented optimisations maintain the differentiability and correct gradients of the implemented models.

Currently in an experimental stage of development and part of future work is another speed optimisation in Cheetah, where multiple simulations can be vectorised and run in parallel for different parametrisations of the same lattice and beam configuration to take advantage of the parallelism of modern hardware. Early benchmarks indicate that this optimisation delivers at least another order of magnitude speed improvement over the results reported in this dissertation, even on less parallel hardware like CPUs.

## Integration with Other Codes

To facilitate the quick adoption of Cheetah, the ability to load beams and lattices, especially from other particle tracking codes, is crucial. Cheetah's default lattice exchange format is an adapted variant of the interoperable lattice exchange format *LatticeJSON* [6]. Based on the standard JSON format, this makes reading and writing of lattice files which are compatible with Cheetah straightforward in any programming language. Cheetah's modular and simple architecture further simplifies the implementation of converters from other lattice and beam exchange formats. Cheetah currently supports loading beams from ASTRA, lattices from Bmad and Elegant, and both beams and lattices from Ocelot. Support for MAD-X [49] lattices and openPMD [142] beams is planned for future releases.

## Speed Benchmarks

In this section, the execution speed of Cheetah is benchmarked and compared to other simulation codes on the same tracking task. The EA section at ARES, which is introduced in Section 2.2.2, is used as the benchmark lattice. The benchmarks were run on two different computing platforms to account for the potential advantages of different hardware: a laptop with an Apple M1 Pro chip with 10 CPU Cores and 32 GB of random access memory (RAM); and a high-performance computing (HPC) cluster node with two AMD EPYC 7643 having a combined 192 cores, 1 TB of RAM and 4 Nvidia A100 GPUs, each with 80 GB of VRAM. Cheetah was run on both the CPU and the GPU on the cluster node. Note that at the time of writing, Cheetah can only take advantage of a single GPU at any time. Simulation times were averaged over multiple runs using Python's *timeit* package. Cheetah was run in multiple different configurations: tracking a `ParameterBeam`, tracking a `ParticleBeam` on CPU, and tracking a `ParticleBeam` on GPU. For each configuration, the benchmarks were run with and without the opt-in lattice optimisations. Ocelot and ASTRA were each benchmarked with and without space charge effects. Parallel ASTRA was run using 8 performance cores on M1 Pro and 48 cores on EPYC 7643, which were found to be the fastest configurations for this particular benchmark. Bmad-X with a NumPy backend and Xsuite are also included in the benchmarks. The results of the speed benchmark are listed in Table 3.3. For Cheetah with `ParticleBeam` and other simulation codes, a beam with 100 000 macroparticles is used. The benchmarks were run with a pre-release version of Cheetah v0.6.2.

It is found that Cheetah can compute the benchmarked simulation setup up to eight orders of magnitude faster than the other simulation codes. In particular, Cheetah is about 5500 times faster than the fastest ASTRA setup without space charge on the ARM laptop. What is more, Cheetah is over 9000 times faster than Ocelot on the same device. Cheetah also achieves computational speeds around 1900 times faster than the already very fast Bmad-X. Xsuite achieves speeds comparable to Cheetah without Cheetah's opt-in optimisations, but Cheetah is up to two orders of magnitude faster than Xsuite when opt-in optimisations are used. It is important to keep in mind that these speed advantages of Cheetah are by design achieved at the cost of fidelity, as higher-order, collective and similar effects present in other simulation codes are excluded by default to attain the reported speeds. Comparing different configurations of Cheetah, `ParameterBeam` tracking appears to be 2 to 40 times faster than `ParticleBeam` tracking. GPU acceleration is a sensible choice only with `ParticleBeam`, though it is not guaranteed to improve compute times. With lattice-optimisations turned on, Cheetah is found to be 8 times faster on GPU than on CPU. However, without using lattice-optimisations, the simulations running on GPU slowed

Table 3.3: Step computation times of simulation codes in milliseconds [162]

Code	Comment	Laptop	HPC node
ASTRA	space charge	264 000.00	3 605 000.00
	no space charge	109 000.00	183 000.00
Parallel ASTRA	space charge	39 000.00	17 300.00
	no space charge	16 900.00	12 600.00
Ocelot	space charge	22 100.00	21 700.00
	no space charge	182.00	119.00
Bmad-X		40.50	74.30
Xsuite	CPU, no space charge	0.81	2.82
	GPU, no space charge	-	0.57
Cheetah	ParticleBeam	1.60	2.95
	ParticleBeam + optimisation	0.79	0.72
	ParticleBeam + GPU	-	4.63
	ParticleBeam + optimisation + GPU	-	0.09
	ParameterBeam	0.76	1.29
	ParameterBeam + optimisation	0.02	0.04

down due to the overhead of data transfers between CPU and GPU. This is the result of the benchmark beam tracking 100 000 particles. In this case, the overhead induced by sending instructions and data to the GPU outweighs the performance benefits of highly-parallel computation. On the other hand, when the number of tracked particles is increased to 10 million, tracking with optimisations turned on takes 37.5 ms on CPU and 998  $\mu$ s on GPU. With optimisations turned off, Cheetah tracks the same beam in 37.5 ms on CPU and 5.36 ms on GPU. This is a significant improvement, demonstrating the advantages of Cheetah's built-in GPU acceleration. Moreover, opt-in optimisations yielded up to 38 times faster execution on CPU and up to 51 times faster execution on GPU. Note that the opt-in optimisations benchmarked here represent the most extreme case, taking maximum advantage of the optimisations to demonstrate Cheetah at its fastest and at its slowest. In real-world use of opt-in optimisations, the results are expected to be slightly worse than the extreme cases showcased here, as some user-defined exceptions might reduce the effectiveness of Cheetah's optimisations.

For the sub-pC bunch charge at ARES, which is a typical operation mode at this facility and was considered during the real-world studies in Chapter 4, the relative error of the Cheetah simulation result compared to the ASTRA result is below 1%, demonstrating the reliability of the results obtained with Cheetah. An example of how the lack of higher-order effects in Cheetah affects the results for a higher bunch charge or more complex beam shapes is shown in Fig. 3.6. Here, an artificial initial distribution with 5 pC bunch charge is chosen and the transverse phases in the  $(x, x')$  plane are shown using different tracking methods. The Cheetah result agrees with the ASTRA result neglecting the space charge effect. When including the space charge effect, the beam distribution starts to diffuse and the sharp edges are no longer visible, resulting in relative errors of 3% and 7% in  $\sigma_x$  and  $\sigma_{x'}$ , respectively. This error will increase for higher bunch charges or at lower energies. In that case, one could consider training a neural network to compensate for the error as described in Section 3.3.

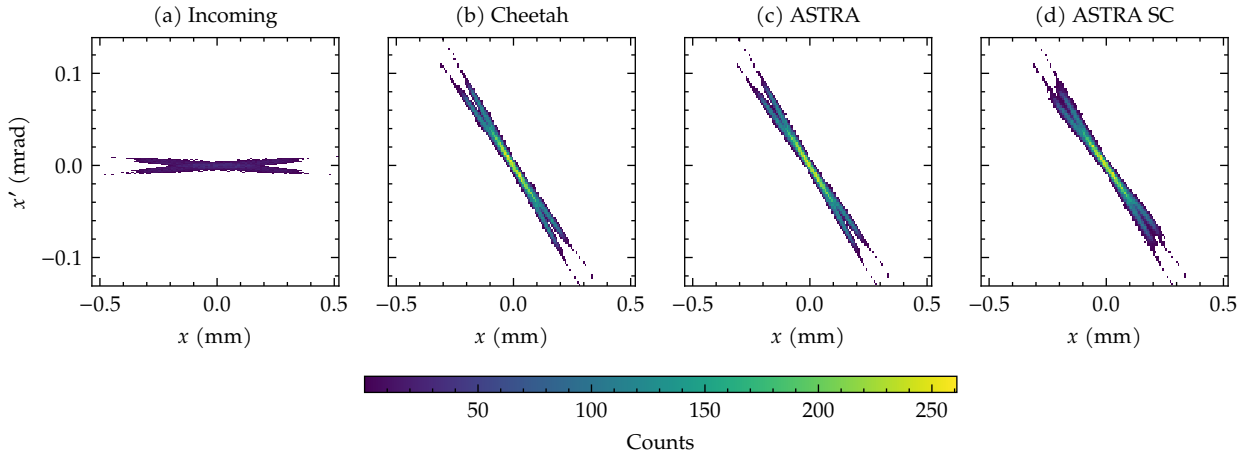


Figure 3.6: Transverse phase spaces of a 5 pC beam in the  $(x, x')$  plane before and after tracking through the EA section [162]. (a) Incoming distribution of the beam at the EA section entrance with an arbitrary non-Gaussian shape. (b) Outgoing distribution using Cheetah tracking. (c) Outgoing distribution using ASTRA tracking without space charge effects. (d) Output distribution using ASTRA tracking with space charge effects.

### 3.2.2 Use Case Examples

In Chapters 4 and 5, Cheetah is used to enable fast RL in simulation to train policies that transfer well to the real world for tuning the transverse beam parameters as well as the FEL intensity in particle accelerators. Furthermore, Cheetah’s automatic differentiation features are used for gradient-based RL on the FEL tuning task such that the true policy gradient can be leveraged to train potentially better policies using fewer samples. Beyond the main topic of this work, Cheetah also enables a variety of other applications, three of which are explored in the following, where Cheetah’s automatic differentiation features are used to perform gradient-based accelerator tuning and system identification, and Cheetah’s utility as a physics-based prior in the context of BO is demonstrated. Finally, the suitability of Cheetah for interleaving physics-based models with modular neural network surrogate models, and how this can be used to combine the advantages of physics-based modelling with those of neural network surrogates similar to those presented in Section 3.1, is shown in Section 3.3. An overview of where Cheetah fits into the proposed applications is illustrated in Fig. 3.7. Note that this is not an exhaustive list of applications for Cheetah, but rather a selection of examples that demonstrate its versatility in different use cases. More applications are expected to be developed in the future as Cheetah is further evolved and adopted by the community.

#### Gradient-Based Beam Tuning and Lattice Optimisation

In the field of particle accelerators, there are various optimisation tasks, ranging from lattice optimisation in the design phase of a facility to tuning actuators at run time. In some cases, these tasks have too many degrees of freedom to be feasibly solvable by black-box optimisation algorithms, such as BO and reinforcement learning-trained optimisation (RLO). However, the model underlying an accelerator optimisation task may be known. In such cases, gradient-based optimisation may be used. The latter has well-understood convergence properties and extensive tooling for it has been developed in the field of machine learning. Using gradient-based tuning on a model of an accelerator can help find

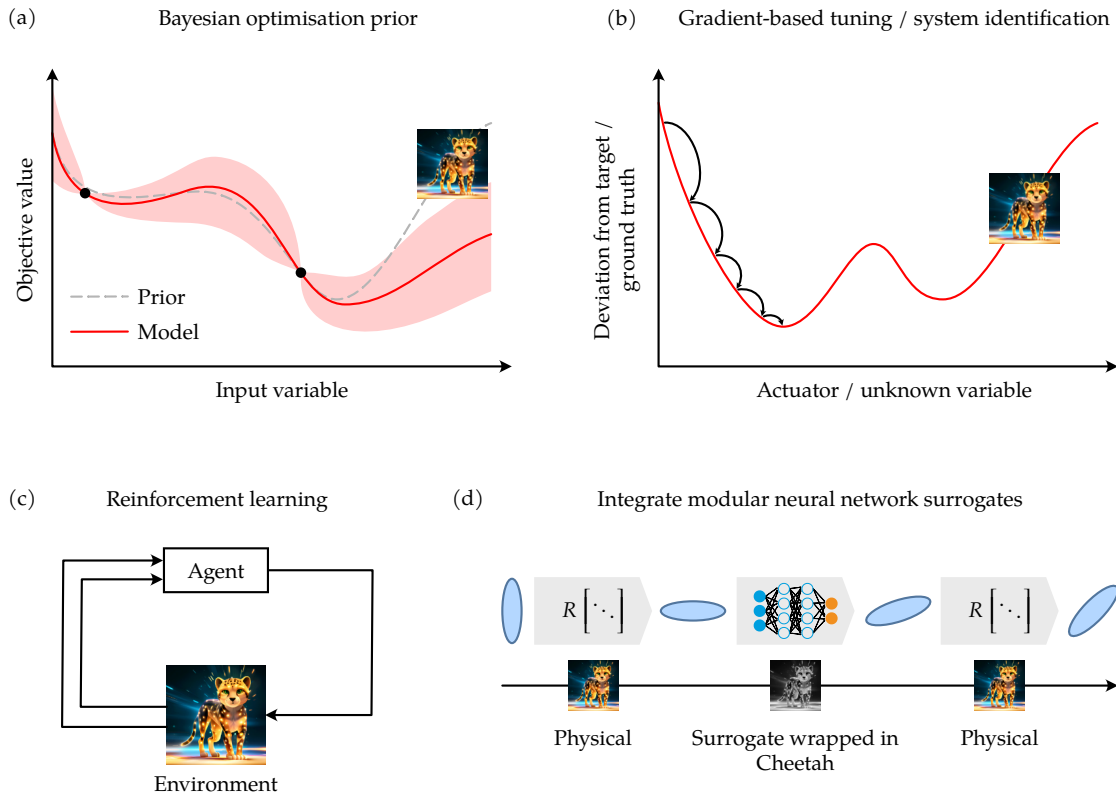


Figure 3.7: Overview of the different roles Cheetah plays in some of the applications considered in this work, with the Cheetah logo marking the use of a model implemented in Cheetah [162]. (a) Cheetah is used as a physical prior for BO as demonstrated in Section 3.2.2. (b) Cheetah provides a differentiable beam dynamics model which can be used for accelerator tuning and system identification as is illustrated in Section 3.2.2. (c) Cheetah enables the implementation of fast beam dynamics environments for training RL agents as is leveraged in Chapters 4 and 5. (d) Cheetah provides the infrastructure to seamlessly integrate modular neural network surrogate models with physical beam dynamics simulations as proposed in Section 3.3.

good setups without the need for beam time. Even in cases where there exists a model mismatch, this offline optimisation approach can provide good starting points close to the optimum, which can then be further optimised online. What is more, gradient-based optimisation methods can also be used to reduce model mismatches through gradient-based system identification as presented in the next section before using the model for gradient-based tuning or lattice optimisation.

In this example, the transverse beam tuning task in the ARES EA section from Section 2.2.2 is considered. It is assumed that unobserved properties, such as the incoming beam and the beamline components' misalignments, are known. This may be the case in simulations during the design stage of an accelerator, if an accelerator is known to deviate very little from its design parameters or if system identification (like the one described in the next section) was performed ahead of time.

The lattice of ARES is loaded as a Cheetah Segment. Because Cheetah defines segments as `PyTorch nn.Module`, all that is needed for PyTorch to automatically compute the gradient of the EA with respect to the five magnet settings, is to define the latter as instances of PyTorch's `nn.Parameter`. A fixed incoming beam is tracked through the EA Cheetah Segment. The resulting beam parameters can then be read from the diagnostic screen

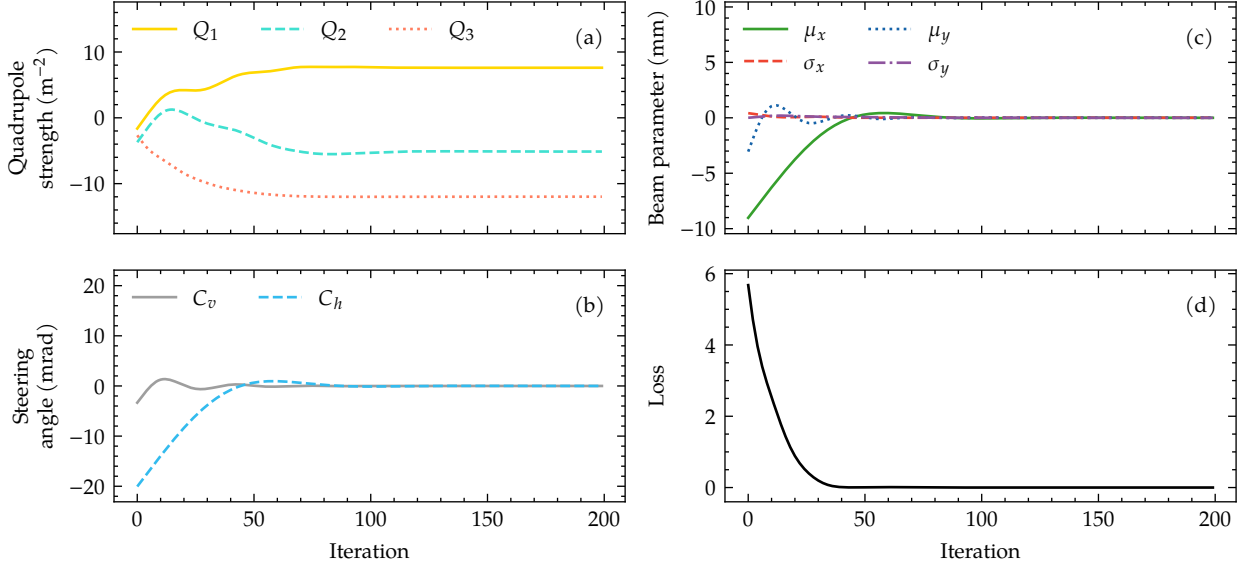


Figure 3.8: Example of gradient-based transverse beam parameter tuning in the EA using Cheetah [162]. (a) and (b) Quadrupole and dipole magnet settings over gradient update iterations. (c) Beam parameters on the diagnostics screen over gradient update iterations. (d) The loss development over gradient update iterations.

station at its end, and an MSE loss, defined as

$$\mathcal{L}_{\text{mse}}(\hat{\mathbf{b}}, \mathbf{b}') = \frac{1}{4} \sum_{i=1}^4 (\hat{b}_i - b'_i)^2, \quad (3.11)$$

with  $\mathbf{b}'$  being the target beam parameters,  $\hat{\mathbf{b}}$  the beam parameters currently estimated by the model at the screen station and  $b_i$  denoting the  $i$ -th element of  $\mathbf{b}$ . Provided the beam tracking function

$$\hat{\mathbf{b}} = f_{\text{EA}}(\mathbf{u} \mid \mathcal{B}_{\text{in}}, \mathbf{m}) \quad (3.12)$$

implemented in Cheetah, the gradient of the loss function with respect to the magnet settings

$$\nabla_{\mathbf{u}} \mathcal{L}_{\text{mse}}(f_{\text{EA}}(\mathbf{u} \mid \mathcal{B}_{\text{in}}, \mathbf{m}), \mathbf{b}') \quad (3.13)$$

can be computed using PyTorch's automatic differentiation features. Adam [173], a variant of SGD, is then used to compute the updates to the magnet settings based on the computed gradient with a learning rate of 0.1.

However, as is, this simple setup results in unstable convergence. This is because the magnet settings are on very different scales, with the maximum quadrupole setting at  $72 \text{ m}^{-2}$  and a maximum dipole magnet setting of  $6.2 \cdot 10^{-3} \text{ rad}$ . To address this, the magnet settings are normalised, i.e. scaled to normally fall into the range  $[-1, 1]$ . With Cheetah, this is easily achieved by wrapping the EA Segment in an outer `nn.Module` with the normalised magnet settings as an `nn.Parameter`. On every call to the module's forward method, the segment's magnet settings  $\mathbf{u}$  are set to

$$\mathbf{u} = \mathbf{u}_{\text{normed}} \odot \boldsymbol{\lambda}, \quad (3.14)$$

i.e. the element-wise product of the normalised actuator parameters  $\mathbf{u}_{\text{normed}}$  and the scaling factors for each respective actuator component  $\boldsymbol{\lambda}$ . For the presented case study,

$\lambda = (5 \text{ m}^{-2}, 5 \text{ m}^{-2}, 6.2 \text{ mrad}, 5 \text{ m}^{-2}, 6.2 \text{ mrad})$  is used. Note that for the quadrupole magnets, the scaling factors are chosen to be smaller than the physical limits of the real magnets, so that they represent the commonly used operational ranges of these magnets in day-to-day operations more accurately.

The resulting convergence of the magnet settings can be seen in Fig. 3.8. In the shown example, the target beam parameters are  $\mathbf{b}' = (0.0 \text{ } \mu\text{m}, 0.0 \text{ } \mu\text{m}, 0.0 \text{ } \mu\text{m}, 0.0 \text{ } \mu\text{m})$ . It is observed that the final magnet settings result in the desired centred and focussed beam. The absolute deviation of the observed transverse beam parameters to the target transverse beam parameters is  $|\Delta\mu_x| = 0.33 \text{ } \mu\text{m}$ ,  $|\Delta\sigma_x| = 6.66 \text{ } \mu\text{m}$ ,  $|\Delta\mu_y| = 0.07 \text{ } \mu\text{m}$ , and  $|\Delta\sigma_y| = 0.85 \text{ } \mu\text{m}$ , resulting in an MAE of  $1.98 \text{ } \mu\text{m}$ . Convergence is generally smooth, with all five magnets converging on their final settings after about 90 gradient steps. Note that because this example is meant as a proof-of-concept, the learning rate and gradient descent algorithm were not tuned for optimal performance. Better results may be possible.

### Gradient-Based System Identification and Virtual Diagnostics

A common challenge with accelerators is that some properties of the beam or the accelerator hardware itself are not observable. Finding these properties usually requires multiple samples at different system states, ideally collected in a structured manner such as a grid scan for best results. Using these samples to reconstruct the hidden properties constitutes an inverse problem. Inverse problems are notoriously difficult to solve. Performing structured measurements to identify hidden properties of an accelerator necessitates an interruption of the beam delivery, making it a costly measurement that is only performed when strictly needed.

Here, a system identification task in the EA is considered. There are two unknowns: the incoming beam and the misalignments of the quadrupoles. The aim is to identify the latter. Knowing the misalignments can be beneficial, either to calibrate the model used in the previous section, or by using the identified values as a guide to better align the physical quadrupole magnets and thereby reduce the dipole effect they have on the beam when used for focussing.

The gradients are computed with respect to the quadrupole misalignments  $\mathbf{m}$ , which are defined as an `nn.Parameter` in the Cheetah model for this task. Similar to the tuning example, the misalignments are normalised by wrapping the EA Segment in a PyTorch Module that holds the normalised quadrupole misalignment  $\mathbf{m}_{\text{normed}}$  as an `nn.Parameter` such that

$$\hat{\mathbf{b}} = f_{\text{EA}}(\mathbf{m}_{\text{normed}} | \mathcal{B}_{\text{in}}, \mathbf{u}). \quad (3.15)$$

A loss function is defined as a measure of the difference between the beam positions predicted by the model  $\hat{\mu}_x = \hat{\mathbf{b}}_1$  and  $\hat{\mu}_y = \hat{\mathbf{b}}_3$  under the estimated misalignments  $\mathbf{m}$ , and the actual observed beam positions  $(\mu_x, \mu_y)$  as

$$\mathcal{L}_{\text{si}}(\mathbf{m}) = (\mu_x - \hat{\mu}_x) + (\mu_y - \hat{\mu}_y). \quad (3.16)$$

Using the gradient  $\nabla_{\mathbf{m}} \mathcal{L}_{\text{si}}(\mathbf{m})$  of the loss function with respect to the misalignments computed through backward mode automatic differentiation, an estimate for the misalignments is found, such that, when this estimate is assumed in the model, the beam positions computed with the latter best match the experimental measurements. In principle, this method can be used online, acquiring new data points and updating the misalignments based on the new measurements, or offline, using a dataset of parasitically acquired measurements

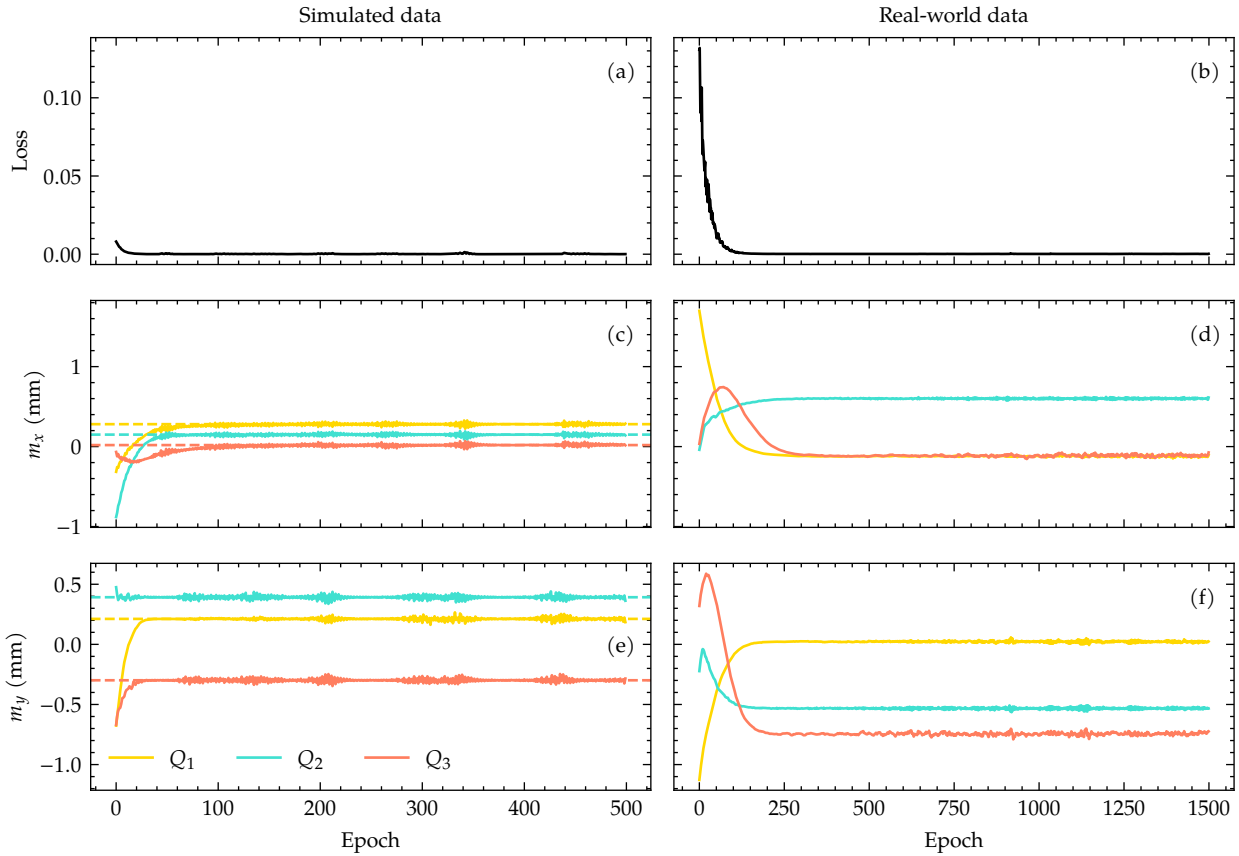


Figure 3.9: Two examples of gradient-based quadrupole misalignment identification [162]. For the first example, using parasitically collected simulated data, it is shown (a) the loss  $\mathcal{L}_{\text{si}}$ , (c) the horizontal quadrupole misalignments  $m_x$  and (e) the vertical quadrupole misalignments  $m_y$  over epochs. Dashed lines signify the ground truth misalignments, which are known in simulation. For the example based on parasitically collected data from the real world, it is shown (b) the loss, (d) the horizontal misalignments and (f) the vertical misalignments.

to identify the misalignments. In this specific demonstration, parasitically acquired measurements collected during evaluations of the RLO policies in Chapter 4 with 151 simulated and 51 real-world samples are used. This means that the presented example is effectively zero-shot.

The method is first validated against simulated data, where the ground truth misalignments are known. This allows for verification that the gradient-based optimisation arrives at correct misalignments. Moreover, when using simulated data, the correct incoming beam is also known and may be assumed in the Cheetah model. As can be seen in Fig. 3.9, the fitted misalignments successfully and quickly converge to the misalignments that were used to generate the simulated data.

Now trusting the misalignment reconstruction, the latter may be tried on data collected from a real-world experiment. With real-world data, however, the incoming beam is often unknown. Therefore, a sensible assumption on the incoming beam must be made. In the EA, the goal is to reduce the misalignments of the quadrupole magnets with respect to the design orbit, which is at the centre of the beam pipe in most cases. It is therefore assumed that the incoming beam position and momentum are at zero, i.e. the observed orbit is assumed to be the design orbit. Doing so effectively sets the origin of the Cheetah model

to the observed orbit in the data, resulting in misalignments being measured as deviations from that orbit. Other properties such as the beam size and energy only marginally affect this particular system identification setup and are therefore not considered. As can be seen in Fig. 3.9, the reconstruction appears to also perform well and smoothly arrive at sensible misalignment estimates under these conditions. However, the results cannot be checked against the ground truth this time, because the ground truth is inherently unknown.

### Physics-Based Prior Mean for Bayesian Optimisation

Thanks to its speed, Cheetah can provide fast predictions of the beam parameters and guide optimisation algorithms during online tuning tasks, ultimately boosting their performance. One particular use case is BO, which utilises a GP model to build a surrogate of the observed data and efficiently optimise the objective function. However, when dealing with high-dimensional tasks, BO tends to over-explore the parameter space to find the global optimum, inevitably increasing the required number of samples [301]. This behaviour is actually observed in the experiments in Section 4.2.3. It limits the tasks which are solvable with classical BO algorithms to those that have less than a few dozen input parameters [271]. Recent studies have shown that the convergence speed of BO can be significantly improved by incorporating prior knowledge of the accelerator system into the GP model, for example by including the correlation of quadrupole magnets into the GP covariance [70] or using a neural network surrogate model as the prior mean function for the GP [356]. In the case of a neural network surrogate, it is important that the model is accurate, because a too large mismatch between the prior mean and the true function will actually worsen the performance of BO. However, training an accurate neural network model requires a large dataset of samples from either simulations or real measurements, which is not always readily available. Cheetah becomes a promising alternative to implement the prior mean function for BO thanks to its fast inference time. In addition, Cheetah's built-in automatic differentiation enables efficient acquisition function optimisation using gradient descent methods in modern BO packages like *BoTorch* [14].

Here the use of Cheetah as a prior mean for BO is demonstrated on a beam-focussing task considering a FODO cell consisting of two quadrupole magnets  $Q_1$  and  $Q_2$  with drift spaces in between and a diagnostic screen at the end. The objective is to minimise the mean beam size measured at the screen

$$O(k_{Q_1}, k_{Q_2}) = \frac{1}{2} (\sigma_x + \sigma_y) \quad (3.17)$$

by changing the quadrupole strengths  $k_{Q_1}$  and  $k_{Q_2}$ , where  $\sigma_x$  and  $\sigma_y$  denote the horizontal and vertical beam sizes as a function of the quadrupole strengths, respectively.

The performance of BO with a Cheetah model as a prior mean function is evaluated with a matched prior, where the distances between the magnets exactly match the prior mean, and a mismatched prior, where the distances are mismatched. The results are further compared to BO with a constant prior mean and Nelder-Mead simplex as baselines. All algorithms are implemented using the *Xopt* package [269]. Both BO variants use a Matérn-5/2 kernel and upper confidence bound (UCB) acquisition function with  $\beta = 2.0$ , which is a standard choice of hyperparameters for BO applications. Each algorithm was run 10 times starting from the same detuned setting. The results are shown in Fig. 3.10. In the case of the matched Cheetah prior mean, BO was able to immediately find the global minimum without exploring much of the parameter space. In the case of the mismatched Cheetah prior mean, BO first sampled around the minimum predicted by the prior mean.

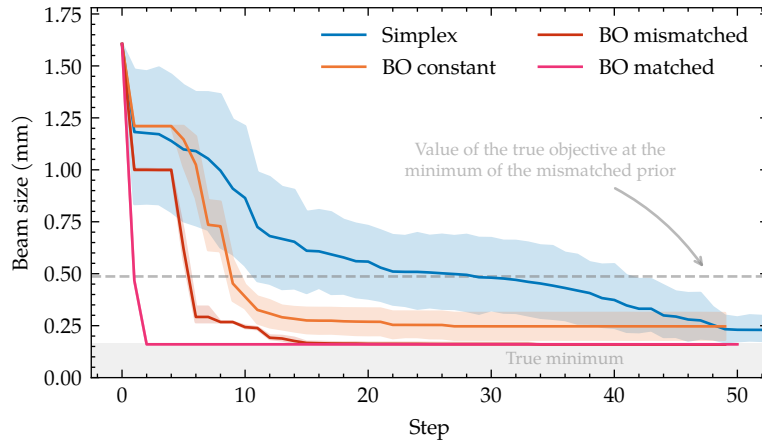


Figure 3.10: Optimisation results from the beam focussing task on a FODO cell [162] using Nelder-Mead simplex (blue), BO with a constant mean function (orange), and BO with a Cheetah model as the prior mean before matching the prior model to the lattice under tuning (dark red) and after matching (magenta). The results are averaged over 10 runs for each algorithm with shaded regions showing one standard deviation around the mean. A dashed line indicates value of the true objective at the magnet settings that minimise the mismatched prior. The true minimum was found through grid scans and is marked by the top of the grey area at the bottom of the plot.

Since there was a difference between the predicted and observed beam sizes, it continued exploring the parameter space and subsequently converged to the minimum.

In both cases, BO with an informed Cheetah prior mean converged to the true minimum within 15 steps and was more sample-efficient than BO with a constant prior mean and the Nelder-Mead simplex algorithm, even on this low-dimensional task. As the dimensionality and complexity of the task increase, the impact of using a physics-based prior mean such as one provided by Cheetah is expected to grow. Furthermore, when BO with a Cheetah prior mean is applied to real-world tasks, one can use system identification to determine the mismatch between the simulated and the real accelerator using the obtained data parasitically, as is shown in the previous section. This further reduces the mismatch between the physics-based prior mean and the real-world system, ultimately improving the performance of BO.

### 3.3 Coupling Physics-based Models and Modular Neural Network Surrogates

While high-speed physics-based models such as those implemented with Cheetah in Section 3.2 are well-behaved, versatile and can easily be used to model novel accelerator setups, some beam dynamics such as collective effects are inherently expensive and slow to compute. Neural networks can be used to create fast-executing surrogate models like those presented in Section 3.1 of otherwise computationally expensive effects, but to date, single neural network surrogates are usually used to build start-to-end models of, for example, specific instruments or lattice setups [73, 160, 255]. As a result, these models have limited versatility and reusability, and novel applications require the design and training of new models, which necessitates further beam time or computation to acquire new training

datasets.

*Modular surrogate models*, which are trained to model single generic and reusable elements of an accelerator on the entire domain of the element's input parameters, can be used to address this issue. Coupled with physics-based models, they allow for high-speed differentiable modelling of accelerators with the versatility of physics-based models and the ability of neural network surrogates to accurately model complex effects at high speed [162, 163, 276]. Because PyTorch is one of the most popular frameworks for implementing neural networks, modular neural network surrogates can seamlessly be wrapped as Cheetah elements and combined with the physics-based elements that Cheetah already provides as well as with other neural network surrogates. Crucially, with both Cheetah models and neural networks being differentiable and implemented in PyTorch, the combined model preserves gradient information and can be used in gradient-based applications such as those presented in Section 3.2.2.

### 3.3.1 Microsecond Space Charge Delta Modelling

Here, an example of modular neural network surrogate coupling with physics-based models is demonstrated, where a Cheetah quadrupole is augmented with space charge effects modelled using a neural network [162, 163]. The straightforward implementation of a neural network-based modular surrogate model would see the track method in Cheetah implemented as a forward pass of a neural network

$$\mathcal{B}_{\text{out}} = f_{\text{SC}}(\mathcal{B}_{\text{in}} | l_Q, k_Q), \quad (3.18)$$

mapping the incoming beam  $\mathcal{B}_{\text{in}}$  and quadrupole parameters  $(l_Q, k_Q)$  to an outgoing beam  $\mathcal{B}_{\text{out}}$ . Fortunately, the effects of space charge in a quadrupole are secondary to the linear beam dynamics, and linear beam dynamics can be modelled easily. To reduce the complexity of the function modelled by the neural network and reduce the time and data required to train it, the tracking function through the quadrupole element is instead reformulated as

$$\mathcal{B}_{\text{out}} = f_{\text{linear}}(\mathcal{B}_{\text{in}} | l_Q, k_Q) + f_{\Delta\text{SC}}(\mathcal{B}_{\text{in}} | l_Q, k_Q), \quad (3.19)$$

where  $f_{\text{linear}}(\mathcal{B}_{\text{in}} | l_Q, k_Q)$  is the handcrafted computation of the linear beam dynamics through a quadrupole magnet already implemented by Cheetah and  $f_{\Delta\text{SC}}(\mathcal{B}_{\text{in}} | l_Q, k_Q)$  is the change induced to the outgoing beam by space charge effects. The neural network model is used to approximate  $f_{\Delta\text{SC}}(\mathcal{B}_{\text{in}} | l_Q, k_Q)$ . An illustration of this process is provided in Fig. 3.11. Note that the presented example considers the tracking of a `ParameterBeam`, but the same principle can be applied to a `ParticleBeam`.

Data for training the modular neural network surrogate model is generated using Ocelot [3] with space charge effects and second-order tracking through a single quadrupole element. Space charge effects are calculated with a mesh size of  $63^3$  and applied at a unit step size of 2 cm. A total of 100 000 samples are collected from uniform distributions over length  $l_Q$  and strength  $k_Q$  of the quadrupole, as well as log-uniform distributions over a subset of the incoming beam parameters

$$\mathbf{x} = (\sigma_x, \sigma_{x'}, \sigma_y, \sigma_{y'}, \sigma_\tau, \sigma_\delta, E, q), \quad (3.20)$$

where  $\sigma_x, \sigma_{x'}, \sigma_y, \sigma_{y'}, \sigma_\tau, \sigma_\delta$  are the standard deviations of the particle distribution in the respective phase space coordinates, i.e. the square root of the diagonal elements of the covariance matrix  $\Sigma$ , and  $E$  and  $q$  are the beam energy and total charge, respectively. The

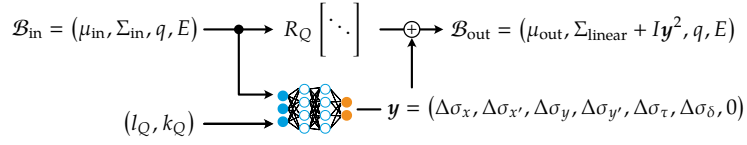


Figure 3.11: Illustration of the operation of the neural network-enhanced space charge quadrupole [162]. The incoming ParameterBeam’s reference particle vector and covariance matrix  $(\mu, \Sigma) \in \mathcal{B}_{\text{in}}$  are multiplied with the magnet’s transfer matrix  $R_Q$  as in classical linear beam dynamics. The neural network model predicts  $f_{\Delta\text{SC}}$  as a vector of changes to some beam parameters describing how the covariance  $\Sigma_{\text{linear}}$  resulting from linear beam dynamics needs to be changed to account for space charge effects, such that  $\Sigma_{\text{out}} = \Sigma_{\text{linear}} + I\mathbf{y}^2$ , where  $I$  is the identity matrix. This change is then added to the linear beam dynamics result, giving the outgoing beam  $\mathcal{B}_{\text{out}}$ .

Table 3.4: Input parameter ranges for dataset generation

Input parameter	Range
$\sigma_x$	[10 $\mu\text{m}$ , 1 mm]
$\sigma_{x'}$	[10 $\mu\text{rad}$ , 1 mrad]
$\sigma_y$	[10 $\mu\text{m}$ , 1 mm]
$\sigma_{y'}$	[10 $\mu\text{rad}$ , 1 mrad]
$\sigma_\tau$	[300 nm, 300 $\mu\text{m}$ ]
$\sigma_\delta$	[ $10^{-5}$ , $10^{-3}$ ]
$E$	[1 MeV, 1 GeV]
$q$	[1 pC, 5 nC]

ranges over which these are sampled for data generation are provided in Table 3.4. A log-uniform distribution was chosen for the beam input parameters because space charge effects depend mainly on their order of magnitude.

The neural network model takes  $\mathbf{x}$  as input, and outputs

$$\mathbf{y} = (\Delta\sigma_x, \Delta\sigma_{x'}, \Delta\sigma_y, \Delta\sigma_{y'}, \Delta\sigma_\tau, \Delta\sigma_\delta, 0), \quad (3.21)$$

the changes to the beam parameters resulting from space charge effects when compared to linear beam dynamics, such that

$$f_{\Delta\text{SC}}(\mathcal{B}_{\text{in}} | l_Q, k_Q) \approx \mathbf{y} = f_{\text{NN}}(\mathbf{x}). \quad (3.22)$$

Note that in Eq. (3.21), the 7-th dimension of the output is not actually inferred by the neural network, but set to a constant 0. An MLP architecture is chosen for the neural network model and the Adam [173] gradient descent algorithm is used for training. Early stopping with a *patience* (number of steps with no improvement before the training is terminated) of 10 was used. The dataset is split 60/20/20 into training, validation and test sets. The logarithm is taken of all beam parameter inputs before they are input into the neural network. All inputs and outputs are scaled to fit a unit-normal distribution with scaling on the beam parameter inputs performed after taking the logarithm. Hyperparameters were tuned over a total of 303 trainings using BO, with *PyTorch Lightning* [87] used to implement the training and *Weights & Biases* [24] for experiment tracking. The best-observed hyperparameters used for the final model are listed in Table 3.5.

Table 3.5: Neural network training hyperparameters

Hyperparameter	Value
Batch size	32
Hidden activation	Sigmoid
Hidden layer width	256
Learning rate	$2 \cdot 10^{-5}$
Number of epochs	959 (max. 10 000)
Number of hidden layers	4
Gradient descent algorithm	Adam

Table 3.6: Mean errors and their standard deviations of the neural network-augmented quadrupole and linear Cheetah model compared to Ocelot’s space charge simulation

Beam parameter	Linear Cheetah	Neural-network augmented
$\sigma_x$ (mm)	$2.4 \pm 3.7$	$0.3 \pm 0.2$
$\sigma_{x'}$ (mrad)	$24 \pm 36$	$2 \pm 2$
$\sigma_y$ (mm)	$1.3 \pm 2.1$	$0.3 \pm 0.3$
$\sigma_{y'}$ (mrad)	$14 \pm 22$	$1 \pm 2$
$\sigma_\tau$ (mm)	$0.8 \pm 2.0$	$0.1 \pm 0.2$
$\sigma_\delta$ (%)	$14 \pm 5$	$1 \pm 1$

### 3.3.2 Results

The results of the neural network-augmented quadrupole implemented in Cheetah are compared to Ocelot’s space charge simulation. To this end, beams with different energies are tracked through both models and the parameters, such as beam sizes, beam divergence, bunch length and energy spread over the bunch, of the outgoing beams are compared, while all other parameters of the incoming beam and the quadrupole parameters are fixed. The results shown in Table 3.6 demonstrate that the neural network-augmented quadrupole model closely matches the results of Ocelot’s space charge simulation, reducing the mean error between the Cheetah model and Ocelot’s space charge computations by 77 % to 93 % when compared to the purely linear Cheetah model.

Figure 3.12 shows the same beam parameters over different beam energies as computed by the neural network-augmented quadrupole implemented in Cheetah, and compares them to Cheetah’s default linear beam dynamics tracking and Ocelot’s space charge simulation. It is observed that the neural network-augmented quadrupole implementation correctly infers larger effects of space charge at low energies when compared to a simple linear beam dynamics model. Congruently, the beam parameters computed using the neural network-augmented quadrupole in Cheetah closely match those computed using Ocelot with space charge tracking, indicating that the neural network-augmented quadrupole model accurately captures space charge effects.

For benchmarking the computational speed gains of the neural network-augmented quadrupole model, the same setup is used with an energy of 6.2 MeV, and compared to Ocelot’s space charge simulation. Ocelot is configured with the same space charge simulation setup as was used to generate the training data. Speed benchmarks are run using Python’s `timeit`, with mean values and standard deviations measured for the linear

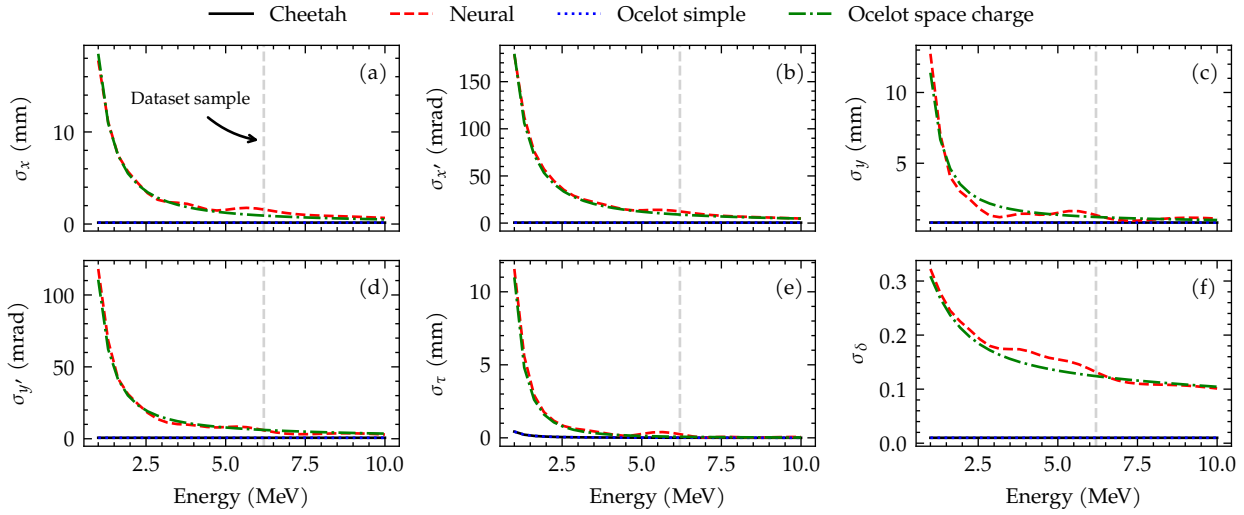


Figure 3.12: Outgoing beam parameters for the same incoming beam over different energies tracked with the default linear quadrupole in Cheetah (black), the neural network-augmented space charge quadrupole (red), Ocelot without space charge calculations (blue), and Ocelot with space charge calculations (green) [162].

Table 3.7: Benchmarked mean compute times required by different quadrupole simulations with and without space charge

Linear Cheetah	Neural-network augmented	Ocelot with space charge
166 $\mu\text{s}$ $\pm$ 383 ns	426 $\mu\text{s}$ $\pm$ 13.3 $\mu\text{s}$	4.12 s $\pm$ 323 ms

Cheetah quadrupole, the neural network-augmented quadrupole and an Ocelot quadrupole with space charge tracking. The results are listed in Table 3.7. It is found that Ocelot takes an average of 4.12 s to perform space charge tracking through a single quadrupole, while modelling the same effect with the neural network-augmented Cheetah element takes an average of 426  $\mu\text{s}$  – a reduction in compute time of almost four orders of magnitude or more than 99.98%. These benchmarks were run on the same CPU of the Apple M1 Pro system on a chip (SoC) considered in the Cheetah speed benchmarks. Unlike Ocelot’s space charge implementation, Cheetah can take advantage of hardware acceleration on GPU, which is expected to result in a further reduction of compute times.

Moreover, because a neural network is used for the modular surrogate modelling, the computation remains fully differentiable, in effect providing differentiable space charge simulations that can seamlessly be integrated with other beam dynamics simulations.

This example serves as a proof of concept for the power of integrating modular neural network surrogates with physics-based models as well as Cheetah’s suitability as a platform for such integrations. Along with parallel work presented in [276], this work is among the first instances of a modular neural network surrogate model being coupled to a physics-based model for particle accelerator simulations, and the first instance where both are fully differentiable. It demonstrates that the integration of modular neural network surrogates in Cheetah enables the building of data-driven, high-speed, high-fidelity models of beam dynamics that would otherwise require computationally expensive models.

### 3.4 Summary

This section introduced three methods for high-speed differentiable beam dynamics modelling, start-to-end neural network surrogates, physics-based high-speed simulations and a coupling of the two. For physics-based modelling in particular, a new Python package called Cheetah was introduced, which provides high-speed differentiable beam dynamics simulations for machine learning applications. These methods have various applications beyond the scope of this dissertation, enabling future research into gradient-based methods for accelerator lattice optimisation, beamline tuning and system identification. In particular, the application of high-speed differentiable modelling to gradient-based tuning, gradient-based system identification and as a physics-based prior for BO was demonstrated, and it was shown how these methods can be used to provide human operators with useful insight into the state of the accelerator. Most importantly, however, the presented methods lay the groundwork for training neural network policies for accelerator tuning in Chapters 4 and 5, which would be infeasible without very fast-executing beam dynamics models, while gradient-based RL leveraging the differentiability of the presented modelling approaches promises to significantly reduce the amount of samples required to train a capable policy, and virtual diagnostics may help serve policies with more informative observations.

# Chapter 4

## Transverse Beam Parameter Tuning

The tuning of transverse beam parameters is a crucial task in the operation of particle accelerators and commonly performed in the vast majority of accelerator facilities.

In this chapter, a specific transverse beam parameter tuning task is considered, that is the tuning of the horizontal and vertical beam positions and sizes on a diagnostic screen using a set of quadrupole and dipole magnets upstream of the screen. This tuning task is regularly performed in the operation of the ARES linear accelerator at DESY in Hamburg, where multiple lattice sections exist that can be tuned like this. These are introduced in Section 2.2.2. Similar sections also exist at many other accelerators. This tuning task is well suited for studying the application of RL to accelerator tuning, as it is complex to solve with no trivial solution in existence and even expert human operators finding it difficult to intuit about, yet still simple enough to analyse and understand the solution strategies found by the RL agents. At the same time, this task takes up a significant amount of accelerator tuning time, and automating it at potentially improved performance over that achieved by expert human operators would have a meaningful impact on the time spent on accelerator tuning as well as on the quality of the resulting beam.

In the following, the development of an RL agent for tuning the transverse beam positions and sizes on a diagnostic screen at the example of the ARES EA section is described in Section 4.1, with a focus on how to overcome the fact that training complex RL agents on real accelerators is often infeasible due to the limited availability of beam time and the potential danger of training agents on real machines. In Section 4.2, the performance of the developed RL agent is evaluated and compared to that of other optimisation algorithms, including BO, which is considered the state of the art in accelerator tuning and very commonly used throughout the accelerator community. The comparison is extended in Section 4.3 to investigate the ability of RLO trained on a static tuning task to run as a feedback system for drift and fault compensation. As part of this comparison, the feedback performance of RLO is compared to that of BO as well. In Section 4.4, three approaches to efficiently transfer the developed solution to novel lattices are explored. The first is akin to [223], demonstrating the generality of the training setup by showing that the RLO agent can be transferred to a novel lattice configuration by retraining while keeping the algorithm, policy model and hyperparameters fixed. A more efficient approach is further explored in the use of domain randomisation as a means of making policies lattice-agnostic to enable fast transfer to novel lattices without the need for retraining from scratch. This section further introduces improvements to the problem formulation and reward function aimed at improving the robustness and speed of the devised training setup. This is followed in Section 4.5 by an exploration of an approach going beyond RL using natural language to

drive particle accelerator tuning at the example of the transverse tuning task in the EA, before this chapter is summarised in Section 4.6.

## 4.1 Zero-Shot Reinforcement Learning-Trained Optimisation

In recent work, it has been shown that RL is capable of solving a variety of problems at sometimes super-human performance levels [12, 305]. But despite continued advances in the field, applying RL to complex real-world control and optimisation problems such as particle accelerator tuning and control has proven difficult [69, 145]. This is due to multiple reasons. The increased complexity of real-world tasks when compared to benchmark problems means that RL agents are generally more difficult to train and require more samples to converge to a well-performing policy. Where the action spaces of benchmark tasks are usually discrete and low-dimensional, the action spaces of real-world problems are typically continuous and high-dimensional. Moreover, the real world is often only partially observable, meaning that the agent cannot directly observe the full state of the environment, which makes solving the task more difficult. Observing as much of the environment as possible commonly involves high-dimensional observations that can be more complex than just a vector of numbers, such as images and other sensor data. Noise is also omnipresent in real-world observations, which means that policies trained on real-world tasks must be robust to it, whereas noise is rarely present in benchmark problems. Finally, not only does training successful policies on complex real-world tasks usually require more samples, the acquisition of these samples is also more difficult, expensive and time-consuming than with benchmark RL tasks, which can rely on simple fast-to-evaluate and parallelisable simulators. Particle accelerators especially are usually one-of-a-kind, meaning that concurrent sample collection is impossible. Moreover, beam time is a scarce resource with some accelerators having waiting times of several months to years for beam time. This problem is amplified by the fact that performing actions on real-world systems like accelerators can at times be slow. For example, changing the quadrupole setting and taking a beam measurement in the EA takes 10 s to 20 s, which would result in years of beam time required to train a successful policy on the task. Safety is also a concern, as RL training inevitably involves the agent trying out suboptimal policies, which can lead to beam losses and potentially damage to the accelerator.

This section introduces a method to train RL agents for particle accelerator tuning that overcomes these challenges by training the agents on a simulation of the accelerator in such a way that the trained policy can be transferred to the real accelerator without requiring any additional training on the latter. The effective result is zero-shot reinforcement learning-trained optimisation (RLO) for particle accelerator tuning, meaning that no beam time is required and the safety of the accelerator is never compromised. This section presents a method and results that were initially introduced in [158].

### 4.1.1 Problem Formulation

In order to develop an RLO solution to tuning the EA, the tuning optimisation problem introduced in Section 2.2.2 is reformulated as an RL task. Four key components are required for the RL formulation: action, observation, reward and environment.

The action defines how the agent interacts with the environment. For the optimisation

task in the EA, the action may intuitively be defined as the magnet settings  $\mathbf{a}_t = \mathbf{u}_t$ . However, experiments have shown that, while this action definition works well in simulation, it stops working when transferring simulation-trained agents to the real world. This is because in the real world there are effects that make the true setting of a magnet deviate from the requested setting, and cause the effect of the magnet on the beam to be different at different times, even when the requested setting is the same. Magnet hysteresis, where the ferromagnetic core of an electromagnet retains some magnetisation when the current in its coils is changed, is one such effect. It causes the magnetic field strength of the magnet to depend not only on the current through the coils at the present time, but also on its history. Accelerator control systems typically have some hysteresis compensation for settings of the magnets' high-level control parameters, such as focussing strength and steering angle, but due to the complex and non-linear nature of hysteresis, this compensation is not very accurate. In addition, high-level control parameters rely on accurate calibration to the magnet hardware and the beam. Consequently, unlike in simulation, both low-level current settings and high-level control parameters cannot be relied upon to map perfectly from the requested settings to the expected effect on the beam in the real world, which would inhibit a successful sim-to-real transfer. Instead, the action is defined as a continuous delta

$$\mathbf{a}_t = \Delta \mathbf{u}_t = (\Delta k_{Q_1}, \Delta k_{Q_2}, \Delta \alpha_{C_v}, \Delta k_{Q_3}, \Delta \alpha_{C_h}), \quad (4.1)$$

i.e. the amounts by which to change the actuator values  $\mathbf{u}_t$  at time step  $t$ . At time step  $t + 1$ , this results in actuator values

$$\mathbf{u}_{t+1} = \mathbf{u}_t + \mathbf{a}_t. \quad (4.2)$$

This action definition allows the agent to dynamically compensate for unexplained deviations from the requested magnet settings and therefore the expected beam. It is also intuitive to how expert human operators change magnet settings during manual tuning. Experiments have shown that the latter definition enables a successful sim-to-real transfer. To ease transferability between the simulation and the real world, these action deltas are defined in terms of the high-level focusing strength and steering angle control parameters.

The observation is based on the internal state  $\mathbf{s}_t$  of the environment. In an ideal case, the state of the transverse tuning task can be fully described by the magnet settings  $\mathbf{u}_t$ , the target beam parameters set by the operator  $\mathbf{b}'$ , the incoming beam  $\mathcal{B}_{\text{in}}$  entering the section from upstream, as well as the misalignments  $\mathbf{m}$  of the quadrupole magnets and the diagnostic screen. Based on these, all other relevant quantities, that is the observed beam parameters on the diagnostic screen  $\mathbf{b}_t$  – and with that their deviation from the target beam parameters  $\mathbf{b}'$  – can, in theory be computed following Eqs. (2.44) and (2.45). While this assumption holds in simulation, the real  $f_b$  is not known exactly and depends on various additional sources of noise and uncertainty. For example, as mentioned above, the effect of the requested magnet settings on the beam is disturbed by calibration errors and hysteresis. As a result, the true state  $\mathbf{s}_t$  of the transverse tuning task is vastly more complex. Most of this state is not observable with the diagnostics available at the real facility. Most significantly, the incoming beam  $\mathcal{B}_{\text{in}}$  and the misalignments  $\mathbf{m}$  are neither measurable nor constant. Hence, the transverse tuning task is partially observable, that is the observation  $\mathbf{o}_t$  is not the same as the state  $\mathbf{s}_t$ . Making use of the diagnostics that are available, the readbacks of the currently applied magnet settings  $\mathbf{u}_t$  can be read from the control system. The transverse beam parameters  $\mathbf{b}_t$  on the diagnostic screen can be inferred by reading the pixel intensities of an image taken by a camera pointed at the screen. An estimate of the beam parameters is derived from the parameters of a 2-dimensional Gaussian fitted to

the pixel intensities. The target beam parameters  $\mathbf{b}'$  are provided directly by the operator. With these, a partial observation of the state is defined as

$$\mathbf{o}_t = (\mathbf{b}_t, \mathbf{u}_t, \mathbf{b}'). \quad (4.3)$$

The most important component of any RL problem definition is the reward. The choice of reward is crucial for an RL algorithm's ability to learn. For the transverse tuning task in the EA, the reward function  $\mathcal{R}$  is defined in terms of the objective function  $O$ . This is convenient as it allows staying close to the original optimisation problem definition. Specifically, it is defined as the difference – intuitively the improvement – of the objective function  $O$  from time step  $t$  to time step  $t + 1$ . While this definition is intuitive and ensures that the RL agent learns to minimise the objective, learning can be slow, and it is possible for agents to traverse potentially dangerous states of high objective in between two states of low objective during the optimisation. Both these issues are caused by the fact that agents can recover most of the reward lost when traversing states of high objective value in between two states of approximately equal value. Rather than defining constraints to solve this issue, which has to be redone for each problem and may be difficult, the reward is slightly redefined such that the agent is discouraged from worsening the objective at any time unless the expected cumulative reward following such an action is very large. Whenever the objective difference is negative, i.e. the new actuators  $\mathbf{u}_{t+1}$  result in a higher objective  $O$  than the previous actuators  $\mathbf{u}_t$ , the reward is multiplied by a factor of 2, making reward recovery more difficult. It was found that this improves the training speed. The final reward function is defined in terms of an objective function  $O(\mathbf{u}_t)$  as

$$r_{t+1} = \mathcal{R}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) = \begin{cases} \Delta O & \text{if } \Delta O > 0 \\ 2 \cdot \Delta O & \text{otherwise,} \end{cases} \quad (4.4)$$

with

$$\Delta O = O(\mathbf{u}_t) - O(\mathbf{u}_{t+1}). \quad (4.5)$$

While the objective function is often already defined for optimisation problems, it was found that issues can arise from using objectives well suited to classical optimisation as a basis for a reward function. This is because typically used objective functions such as MSE and even an MAE result in vanishing rewards near the optimum. As an optimiser improves on these objective functions, the difference between two results in terms of such an objective function becomes orders of magnitude smaller, and hence the improvement in a differential reward, such as the one defined in Eqs. (4.4) and (4.5), as well as the difference between cumulative rewards, when a reward based on the negative objective is used, becomes vanishingly small. As a result, the reward signal in later stages of optimisations is barely strong enough to facilitate learning in an RL context. This issue is illustrated in Fig. 4.1. Resulting trained policies tend to optimise toward the optimum but stop prematurely. At the example of the transverse tuning task this can be observed with policies that perform well, optimising fast and even under difficult conditions, getting close to the target beam when compared to the beam before tuning, but stop short of getting within the measurement accuracy away from the target beam. This problem can be addressed by taking the logarithm of the objective function previously chosen for classical optimisation, causing small improvements near the optimum to result in exponentially improved rewards. In the case of the EA optimisation task, the original objective is a weighted sum over the absolute differences between observed transverse beam parameters

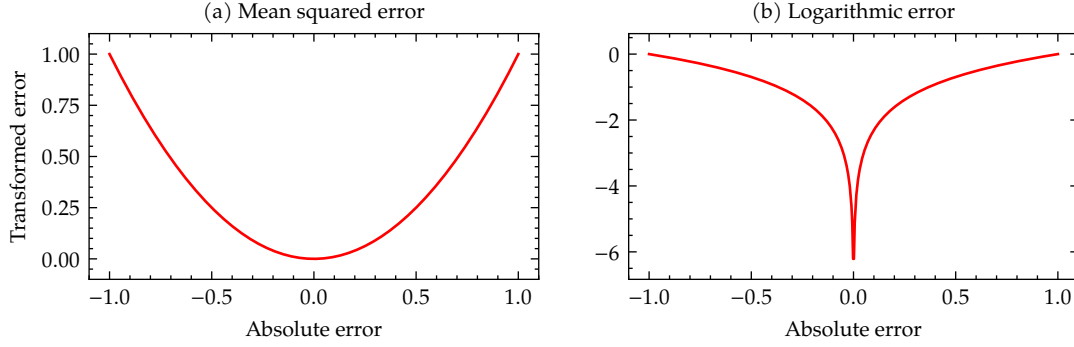


Figure 4.1: 1-dimensional example of a typically used MSE objective compared to the logarithm of an MAE objective [158]. The MSE differential objective shrinks as the input variable converges toward the optimum, i.e. the reward vanishes. This effect is turned around with the logarithmic objective, where the reward magnitude increases as the input variable converges toward the optimum.

and the target beam parameters on the diagnostic screen  $S$ . Taking the logarithm of the objective results in the logarithmic and weighted difference between the observed and target beams

$$O(\mathbf{u}_t) = \ln \frac{1}{4} \sum_{i=1}^4 w_i |\mathbf{b}_i - \mathbf{b}'_i|. \quad (4.6)$$

The environment defines the dynamics of the transverse tuning task as a POMDP, with the next observation computed through an observation function  $\Omega$  in terms of the next state  $\mathbf{s}_{t+1}$ , which is itself a function of the transition probability  $P$ , the current state  $\mathbf{s}_t$  and the action  $\mathbf{a}_t$ , where  $P$  is defined in terms of the EA's beam dynamics  $f_b$ , resulting in

$$\mathbf{o}_{t+1} = \Omega(\mathbf{s}_{t+1}) = (f_b(\mathbf{u}_t | \mathcal{B}_{\text{in}}, \mathbf{m}), \mathbf{u}_{t+1}, \mathbf{b}'). \quad (4.7)$$

An overview of the corresponding RL loop specific to the task in the EA is shown in Fig. 4.2. Note that for a real-world setting,  $f_b$  is not known, but an approximation of its output can be observed through sensor readings. In simulation, an approximation of  $f_b$  similar to that defined in Eq. (3.12) can be used.

### 4.1.2 Training

Training in the real world is difficult and dangerous for a variety of reasons. Firstly, training using the real machine can be expensive as a result of limited availability. Beam time on particle accelerators, for example, is a sought-after resource due to how few of these machines exist. This issue is further amplified by the large number of samples required for training an RL policy and the time it takes to acquire these samples. The policies in this work were trained for 6 million samples, where each sample involves sending new set points to the magnet power supplies, waiting for the power supplies to finish ramping to their set points, and taking multiple images of the diagnostic screen with the beam turned on and with the beam turned off. Altogether this process takes ca. 10 s to 20 s. Consequently, training on the real ARES accelerator would take about 3 years of continuous beam time. Research accelerators are also usually one of a kind, prohibiting the advantages of policy gradient RL algorithms like PPO [292] that can make use of parallelised data collection.

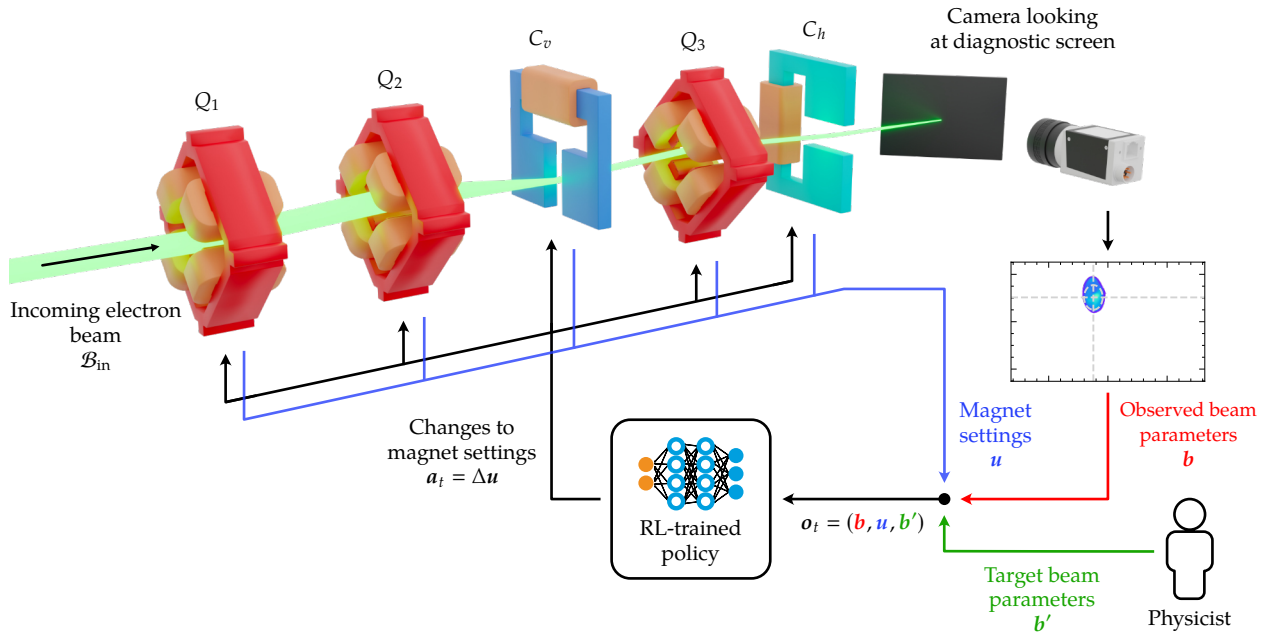


Figure 4.2: Flowchart of the RL loop as used for the transverse beam tuning task in the ARES EA. The policy receives as observation from the environment the current beam parameters  $b$  computed from the diagnostic screen image, the current actuator settings  $u$  and the target beam parameters  $b'$  set by the operator. Based on the observation, the policy outputs an action of the change to the magnet settings  $\Delta u$ .

While the optimisation task on the EA is a particularly strong example for these constraints, they are nonetheless commonplace in the real world. Secondly, it is inherent to RL training that the agent explores unexplored and sometimes extreme regions of the optimisation space. Doing this on a real machine has the potential to result in unsafe conditions, which can cause beam and thereby training interruptions by the machine protection system in the best case, and damage to the machine in the worst.

To address the issues of availability, speed and safety, training is done in simulation, and the trained policies are transferred to the real machine zero-shot, i.e. without any additional training on the real machine. The training setup used here is developed to facilitate this by splitting the logic of the problem environment in two parts: frontend and backend. The frontend provides the general definition of the RL environment, including the definition of the observation, action and reward. The backend is interchangeable and handles the underlying beam dynamics, either by modelling them in simulation or by interfacing with the control system of the real machine. By placing both the simulation and the control system interface inside the same environment, their differences are abstracted away from the RL agent, allowing for seamless switching between simulation and real machine. Figure 4.3 shows an overview of this technical setup. Leveraging the high-speed simulation capabilities of Cheetah, the training of each policy could be sped up to about 14 hours, and with the revised setup in Section 4.4.1 even below 1 hour. This represents a speed-up of about five orders of magnitude by training on a specially developed high-speed simulation instead of the real machine, while training with a conventional simulator like Ocelot would have taken around 14 days, meaning that training with Cheetah is up to two orders of magnitude faster. Most importantly, the use of Cheetah moves the training time from being infeasible to being feasible. An added benefit of training in simulation is that it is impossible for policies to overfit to a specific drift state of the real machine, because

the simulation does not include the effects that cause drift. Though as much as this is an advantage, it may also cause difficulties.

Ideally, the simulation would perfectly represent the real world. Unfortunately, some aspects of the real accelerator, such as the incoming beam  $\mathcal{B}_{\text{in}}$  and the misalignments  $\mathbf{m}$  are not known and can therefore not be modelled accurately in simulation. What is more, the incoming beam differs between working points and must be expected to change from one tuning run to the next. For a successful sim-to-real transfer, it is therefore important to train agents to generalise over these sources of uncertainty. This can be achieved through an approach called *domain randomisation* [239, 323]. To include domain randomisation in the training of the RL agents for transverse beam tuning in the EA, the incoming beam  $\mathcal{B}_{\text{in}}$  and the misalignments  $\mathbf{m}$  are modelled as uniformly distributed random variables over reasonable ranges derived from domain knowledge. At the start of each rollout episode, parameters of the incoming beam are sampled as

$$\begin{aligned}
 E &\sim \mathcal{U}(80 \text{ MeV}, 160 \text{ MeV}) \\
 \mu_x, \mu_y &\sim \mathcal{U}(-1 \text{ mm}, 1 \text{ mm}) \\
 \mu_{x'}, \mu_{y'} &\sim \mathcal{U}(-100 \text{ } \mu\text{rad}, 100 \text{ } \mu\text{rad}) \\
 \sigma_x, \sigma_y &\sim \mathcal{U}(10 \text{ } \mu\text{m}, 500 \text{ } \mu\text{m}) \\
 \sigma_{x'}, \sigma_{y'} &\sim \mathcal{U}(1 \text{ } \mu\text{rad}, 50 \text{ } \mu\text{rad}) \\
 \sigma_\tau &\sim \mathcal{U}(1 \text{ } \mu\text{m}, 50 \text{ } \mu\text{m}) \\
 \sigma_\delta &\sim \mathcal{U}(0.1 \text{ } \%, 1 \text{ } \%)
 \end{aligned} \tag{4.8}$$

and the misalignments are sampled as

$$\mathbf{m}_i \sim \mathcal{U}(-500 \text{ } \mu\text{m}, 500 \text{ } \mu\text{m}), \tag{4.9}$$

such that the agents are trained to generalise to be capable of tuning the beam under a wide range of conditions.

Additionally, the aim is to train an agent that can achieve any reasonable target beam requested by the operator. To this end, the target beam parameters  $\mathbf{b}'$  are also sampled from uniform distributions over reasonable ranges derived from domain knowledge before each training episode. The target beams were randomly sampled from uniform distributions

$$\begin{aligned}
 \mu'_{x'}, \mu'_{y'} &\sim \mathcal{U}(-2 \text{ mm}, 2 \text{ mm}) \\
 \sigma'_{x'}, \sigma'_{y'} &\sim \mathcal{U}(0 \text{ mm}, 2 \text{ mm}).
 \end{aligned} \tag{4.10}$$

These distributions were chosen to cover a wide range of measurable target beam parameters, which are constrained by the dimensions of the diagnostic screen.

In order to accelerate the convergence of the RL training and the optimisation performed by the trained agents, it is important to include domain knowledge where possible. From accelerator theory, it is known that a good beam focus will follow a focus-defocus-focus (FDF) pattern over the quadrupole magnets. This knowledge is leveraged in training and subsequent optimisations by initialising the actuators as

$$\mathbf{u}_0 = \left( 10 \text{ m}^{-2}, -10 \text{ m}^{-2}, 0 \text{ mrad}, 10 \text{ m}^{-2}, 0 \text{ mrad} \right) \tag{4.11}$$

at the start of each episode.

An MLP of two hidden layers is used as a policy model. The layers have widths of 64 and 32 neurons, and use a ReLU activation. The policy takes as input the normalised

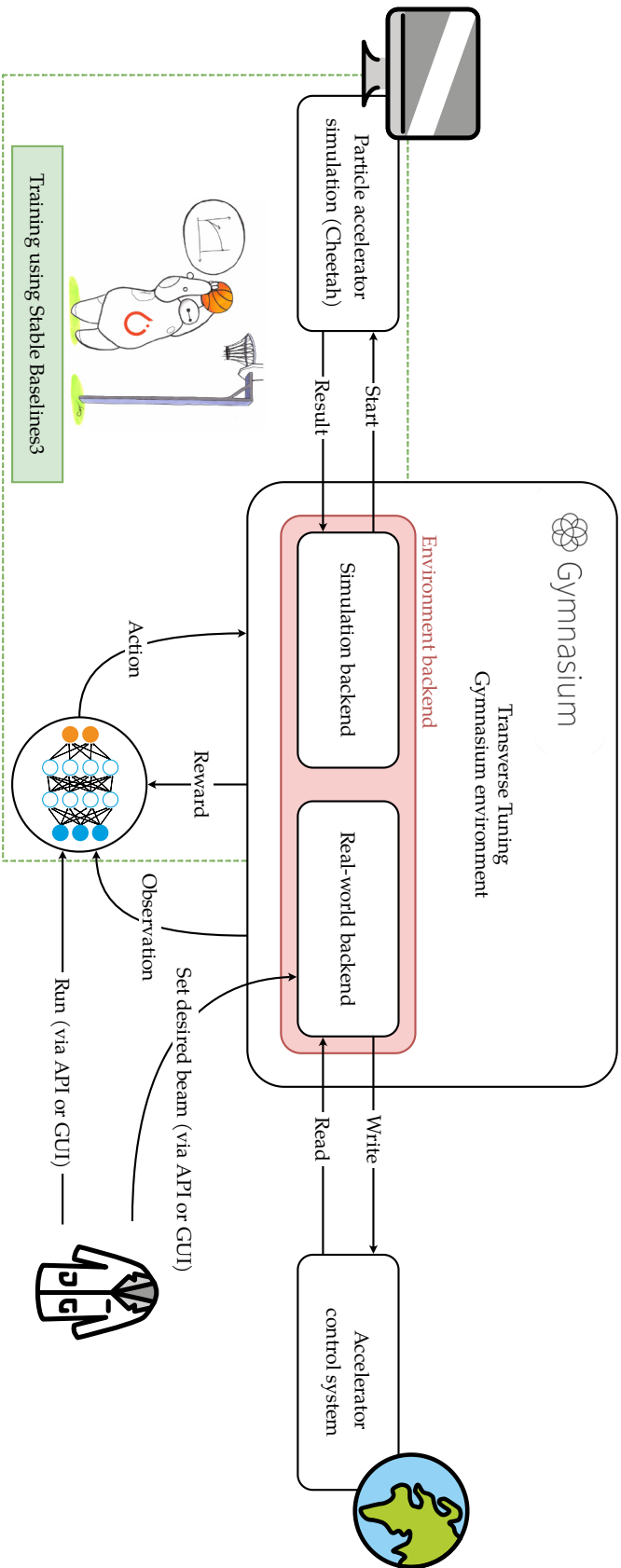


Figure 4.3: *Gymnasium* environment setup used to train policies for transverse beam parameter tuning at ARES [164]. In particular showing how one environment interface facilitates design and training using a simulation of the plant, as well as transferring a developed tuning algorithm to the real-world task without modification.

Table 4.1: Hyperparameters used to train the RLO policies

Parameter	Value
Action noise standard deviation	0.1
Replay buffer length $ \mathcal{D} $	600 000
Discount factor $\gamma$	0.55
Learning rate (actor & critic)	0.001
Polyak update rate $\tau_p$	0.005
Warm-up steps	2000
Layer widths	(64, 32)
Training steps	6 000 000
Maximum episode steps	50
Beam parameter weights $w$	(1, 2, 1, 2)

observed beam parameters  $\mathbf{b}$ , the currently set quadrupole strengths and deflection angles of the magnets  $\mathbf{u}$ , and the target beam parameters  $\mathbf{b}'$ . Its output is defined as normalised changes to the magnet settings  $\mathbf{a}_t = \Delta \mathbf{u}$  and limited to  $\pm 10\%$  of the physical limits of the magnets imposed by their power supplies. The rewards and observations are normalised using a running average during the training. The actions are normalised to the action spaces, which is  $[-3, 3] \text{ m}^{-2}$  for quadrupole strengths,  $[-0.6, 0.6] \text{ mrad}$  for vertical steering magnet, and  $[-0.3, 0.3] \text{ mrad}$  for horizontal steering magnet. The different action ranges of vertical and horizontal magnets are chosen because of their different distances to the diagnostic screen, such that they have approximately the same effect on the beam position.

Three policies were trained with the same hyperparameters and different random seeds on the transverse beam tuning task in the EA. To train policies, the TD3 [100] algorithm is used for its relative training sample efficiency among model-free RL algorithms. Specifically, the implementation provided by the *Stable Baselines3* [254] Python package is employed. Each policy was trained for 6 million steps on the simulated Cheetah model of the EA. The Python package *Gymnasium* [328] (the successor to the previously popular *OpenAI Gym* [38]) is used to define the environments around the simulation and control system backends. Hyperparameters for the TD3 algorithm, the neural networks as well as the environment were selected via a hyperparameter exploration in simulation using *Weights & Biases* [24] for experiment tracking. The best observed hyperparameters were used to train the models for the final benchmarks. The final hyperparameters are listed in Table 4.1. Training was run on a high-performance cluster node with an Intel Xeon Gold 5115 CPU, two Nvidia V100 GPUs and 376 GB of RAM.

## 4.2 Evaluation

In the following, the RLO approach presented in Section 4.1 is evaluated in simulation and in the real world against simple baselines and state-of-the-art methods, most importantly BO. To this end, the considered implementations of other state-of-the-art methods are introduced in Section 4.2.1. In Section 4.2.2 it is then established whether trained RLO policies are capable of transverse beam parameter tuning in the EA by establishing their performance and comparing it to a trivial baseline in simulation, before verifying that these policies as trained in simulation transfer successfully to the real machine. Finally, RLO is compared to the state-of-the-art BO as well as other black-box optimisation methods

commonly used for accelerator tuning in Section 4.2.3. Large parts of this evaluation are based on the work presented in [161, 164].

### 4.2.1 Alternative Algorithms

In the following, four implementations of alternative optimisation algorithms on the transverse beam tuning task in the EA, specifically random search, Nelder-Mead simplex, ES and BO, are introduced. They are used to compare the performance of the RLO approach in Section 4.2.3.

#### Random Search

For the random search baseline, the next magnet settings are sampled uniformly at random from the constrained space of magnet settings. Constraint in this case means that the space is limited to a range of values that are commonly used during operations, instead of the full physical limits of the magnets. At ARES, the latter would be almost an order of magnitude larger than anything ever used in operation and unlikely to result in a beam that is useful for experiments. At the end of the optimisation, after 150 steps, the best magnet settings resulting in the lowest MAE between the observed and the target beam are returned.

#### Nelder-Mead Simplex

The Nelder-Mead simplex optimisation [227] was realised using the implementation provided by the *SciPy* [340] Python package. The initial simplex was tuned in a random search of 405 samples to the one that performed best across the set of 300 trials. Nelder-Mead is allowed to run for a maximum of 150 steps. In the implementation used here, the objective function optimised by Nelder-Mead is the MAE of the measured beam parameters to the target beam parameters

$$O(\mathbf{u}_t) = \frac{1}{4} \sum_{i=1}^4 |\mathbf{b}_i - \mathbf{b}'_i|. \quad (4.12)$$

Note the lack of a logarithm when compared to the objective function considered for RLO in Section 4.1. It was found through experiments that without the logarithm Nelder-Mead performs better on the transverse tuning task.

#### Extremum Seeking

The Xopt [269] implementation of the ES algorithm based on the description from [283] is used for this study. The update rule for each next value of the  $i$ -th actuator component  $\mathbf{u}_{t+1,i}$  is defined as

$$\mathbf{u}_{t+1,i} = \mathbf{u}_{t,i} + \lambda^t \Delta_\tau \sqrt{\alpha_i \omega_i} \cos\left(\omega_i t \Delta_\tau + k O(\mathbf{u}_t) - \frac{\pi}{2} (i \pmod{2})\right), \quad (4.13)$$

where  $\alpha_i$  and  $\omega_i$  are the amplitude and frequency of the oscillation for the  $i$ -th actuator component, respectively,  $k$  is the feedback gain and  $\Delta_\tau$  is the length of the time interval between two actuations.

Following to the original paper, the actuators  $\mathbf{u}$  are normalised to  $[-1, 1]$ , the dithering amplitudes  $\omega$  are linearly spanned in the range  $[1.0, 1.75]$  and

$$\Delta_\tau = \frac{2\pi}{10 \max(\omega_i)}. \quad (4.14)$$

The amplitude of the oscillation  $\alpha_i$  is defined as

$$\alpha_i = \omega_i s^2, \quad (4.15)$$

where  $s$  is the oscillation size.

Extensive hyperparameter tuning was performed across the full set of trials to minimise the MAE at the end of 150 steps. The hyperparameters used for the final evaluation of the ES algorithm are a feedback gain of  $k = 3.7$ , an oscillation size of  $s = 0.11$  and a decay rate of  $\lambda = 0.987$ , where the default settings are  $k = 2.0$ ,  $s = 0.1$  and  $\lambda = 1.0$ . Note that adding a decay rate to the oscillation amplitude is a small deviation from the original implementation, which helps convergence to the target beam parameters in an optimisation setting, where the original implementation from [283] without decay is better suited for stable long-term control. Tuning the oscillation amplitude without the usage of a decay rate  $\lambda$  leads either to local optimisation or large oscillation at the final steps, resulting in a worse final beam error.

### Bayesian Optimisation

The BO variant used for this study was implemented using the *BoTorch* [14] package. A special BO objective

$$O_{\text{BO}}(\mathbf{u}) = -\ln\left(\frac{1}{4} \sum_{i=1}^4 |b_i - b'_i|\right) + w_{\text{visible}}, \quad (4.16)$$

which reflects the fact that BO is used for maximisation rather than minimisation is defined. Here, the logarithm is used to properly weigh the fine improvement when BO approaches the target beam. A constant  $w_{\text{visible}}$  is added to the objective in order to reward when the beam is on the screen with  $w_{\text{visible}} = 10$  and penalise when it is off the screen with  $w_{\text{visible}} = -10$ . Note that this could alternatively be achieved by incorporating a safety constraint. The goal of the BO optimisation is to find the input, i.e. actuators  $\mathbf{u}$ , that maximise the objective function  $O_{\text{BO}}$  as

$$\max_{\mathbf{u}} (O_{\text{BO}}(\mathbf{u})). \quad (4.17)$$

To increase the numerical stability of the GP regression, the previous input settings  $\mathbf{u}$  are min-max normalised to  $[-1, 1]$  and objective values are standardised. The covariance function of the GP models used in this study is the sum of a Matérn-5/2 kernel [210] and a white noise function. The GP hyperparameters, like the length scales and signal noise, are determined dynamically by log-likelihood fits in each step. In each trial, BO is started from the same fixed FDF setting employed by the RLO approach presented in Section 4.1. In the beginning, 5 random samples are taken to initialise the GP model. Based on the posterior prediction of the GP model, an expected improvement (EI) [152] acquisition function automatically balances exploration and exploitation of the objective. The next sample is chosen by maximising the acquisition function. The maximum allowed step sizes

are constrained to 10 % of the magnet range assumed in Section 4.1. In practice, to avoid the time-consuming polarity changes of the quadrupole magnets' power supplies, the quadrupole magnets are clipped to stay in the same polarity, i.e. in the FDF setting. This does not hamper the final performance in simulation studies, as the target beam parameters can still be achieved with this constraint. BO is allowed to run for 150 steps in simulation and 75 steps on the real machine, after which the best magnet settings are returned. Other hyperparameter configurations, such as using UCB as an acquisition function, were tested and found to give comparable performance.

Other capable BO implementations exist, where in the accelerator community, the Xopt package [269] is particularly popular. To ensure that the evaluated implementation matches the state-of-the-art that has evolved for the particle accelerator community, it is compared to two BO variants from Xopt, one using a hard step-size constraint and the other using proximal-biasing [266] as a soft step-size constraint. Both use UCB acquisition with  $\beta = 2$  and perform the default pre-processing steps, i.e. normalising the input to  $[0, 1]$  and standardising the objective values. The hard step-size variant uses the same step-size limit of 0.1 as RLO and the evaluated custom BO implementations. The proximal weight was set to be 0.5, which means the acquisition drops to  $\frac{1}{2}$  over 10 % of the action space. This setting was obtained from a hyperparameter search, optimising for the lowest MAE after 150 steps. It was found that in the transverse beam tuning task, smaller proximal weights would mostly lead to premature convergence, due to many local optima as shown in Fig. 4.11. The results of comparing all three BO implementations are shown in Fig. 4.4. While the custom BO implementation performs worse than both Xopt implementations in terms of the final MAE achieved in simulation, the root mean squared error (RMSE) measured in simulation suggests that it converges notably faster than the proximal-biasing variant and about as fast as the hard step-size constraint variant. More importantly, however, on the real ARES accelerator, the custom BO implementation consistently outperforms both Xopt implementations in terms of both the final beam error and RMSE, meaning better magnet settings are found faster. Hence, it is concluded that the custom BO implementation is representative of the state of the art in BO for particle accelerator tuning.

## 4.2.2 Performance of Reinforcement Learning-trained Optimisation

In this section, the performance of the RLO policies trained according to the setup in Section 4.1 is evaluated. To this end, multiple performance metrics are first introduced, then the results of the evaluation in simulation are presented, followed by the results of the evaluation on the real ARES accelerator. Finally, the performance of the RLO policies is compared to that of expert human operators, before an extensive comparison of RLO to alternative optimisation algorithms is conducted in Section 4.2.3.

### Metrics

The trained policies are evaluated in terms of the quality of the final result they achieve and how quickly they converge to this result. To this end, three metrics are considered.

The final error measures how far the beam parameters observed after tuning are from the target beam parameters in terms of the MAE defined in Eq. (2.45). The final error is computed as

$$\text{Final error} = \mathcal{D}_{\text{mae}}(\mathbf{b}_N, \mathbf{b}'), \quad (4.18)$$

where  $\mathbf{b}_N$  are the observed beam parameters in the  $N$ -th and last step of the optimisation.

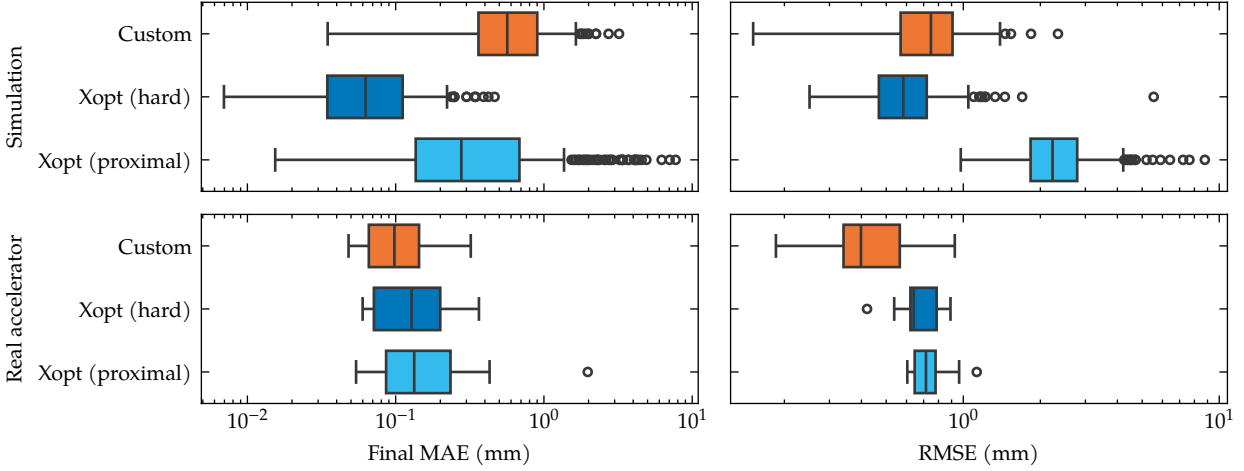


Figure 4.4: Performance of the custom BO implementation used for the comparisons to two other state-of-the-art implementations provided by Xopt [164]. The two plots on the left show the final MAE achieved during the optimisation procedure, while the plots on the right show RMSE over the full optimisation run and serve as an indication of convergence speed. The top row shows results obtained from evaluations in simulation. Results obtained on the real ARES accelerator are shown in the bottom row. The metrics used for comparison are introduced in Section 4.2.2.

In addition, the RMSE over all steps of an optimisation run is considered as a combined measure of how quickly the optimisation converges and how close it gets to the target beam parameters. It is computed as

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{t=1}^N \left( \frac{1}{4} \sum_{i=1}^4 (\mathbf{b}_{t,i} - \mathbf{b}'_i)^2 \right)}, \quad (4.19)$$

where  $\mathbf{b}_{t,i}$  is the  $i$ -th component of the observed beam parameters at step  $t$  of the optimisation.

Another metric is introduced to put the final error in relation to the starting conditions. The normalised beam error improvement is defined as the improvement of the MAE from the initial beam parameters to the final beam parameters, normalised to the initial MAE. It is computed as

$$\text{Improvement} = \frac{\mathcal{D}_{\text{mae}}(\mathbf{b}_0, \mathbf{b}') - \mathcal{D}_{\text{mae}}(\mathbf{b}_N, \mathbf{b}')}{\mathcal{D}_{\text{mae}}(\mathbf{b}_0, \mathbf{b}')}. \quad (4.20)$$

Note that this means that a positive value indicates an improvement in the beam parameters, while a negative value indicates a deterioration.

As an intuitive measure of optimisation speed, the number of steps until the optimisation goal is reached is considered. Based on the measurement accuracy of the diagnostic screen of  $20 \mu\text{m}$ , this metric is computed as the number of steps until the first time that the MAE between the observed beam parameters and the target beam parameters is below  $40 \mu\text{m}$ . In the following this metric is referred to as the *steps to target*. Note that the choice of  $40 \mu\text{m}$  is very strict and presents a high bar for tuning algorithms. Any stricter threshold would be physically unachievable due to hardware constraints. Because experiments have very different demands on the beam parameters, it is likely that a more relaxed threshold would be sufficient for most applications. Thus, this metric measures the speed of the optimisation

algorithm in a worst-case (i.e. most demanding) scenario. None of the algorithms evaluated are always able to meet this threshold. Hence, the proportion of trials where the threshold is met is also reported and the number of steps is only reported for the trials where the threshold was met. Note that while very intuitive, the steps to target on their own do not provide a complete picture of the optimisation speed. Occasionally, the median steps to target can decrease when fewer trials meet the threshold.

To better understand the relationship between the convergence speed and the quality of the result found by the optimisation algorithms, it is helpful to consider the MAE of the final beam in relation to number of steps it took to converge. The latter is expressed as the *steps to convergence*. This metric is defined as the earliest time step  $t'$ , where the difference between the largest error between observed and target beam at a step  $t \geq t'$  and the lowest error at a step  $t \geq t'$  is less than  $40 \mu\text{m}$ .

Note that some optimisation algorithms like RLO, Nelder-Mead simplex and ES keep exploration to a minimum and operate more like a feedback. As a result, one can stop these algorithms at any time and expect to be close to the discovered optimum. Others, such as random search operate almost entirely opposite to this, exploring the parameter space at every point during the optimisation. When stopping random search, one would typically take the best result encountered during the search and return to it. BO is a mix of both, with a balance between exploration and exploitation depending on its configuration. To account for this, in the following, a second variant of the metrics is sometimes considered. Whenever appropriate, and unless otherwise mentioned, the final error, RMSE, improvement, steps to target and steps to convergence are computed with respect to the beam parameters that were actually observed at each step of the optimisation. This is referred to as the *conventional* variant of the metrics. When algorithms like random search and BO are involved, these metrics are in addition computed with respect to the best magnet settings that one would return to if the optimisation was stopped at that point. These would be those settings that produced the beam with the smallest error to the target beam parameters observed at any point before the current time step. This variant of the metrics is henceforth referred to as the *minimum* variant. The difference between the two variants at the example of the MAE over an optimisation with RLO and BO is illustrated in Fig. 4.5. Note that the steps to target are identical in both metric variants because it considers the first encounter of a beam with small-enough error, which happens at the same time in both cases.

When evaluating the performance of a policy or an optimisation algorithm, these metrics are computed for each optimisation run, and aggregated statistics, such as the median, mean and standard deviation, are reported over all runs.

## Simulation

First, the trained policies are evaluated in simulation using the same Cheetah model as used for training. This allows for verification that the trained policies work as expected in a controlled and safe environment, while also making the collection of a large number of benchmark runs feasible. A fixed set of 300 randomly generated *trials* is used for the evaluation. Each trial is a tuple

$$(\mathbf{b}', \mathbf{m}, \mathcal{B}_{\text{in}}, \mathbf{u}_0) \quad (4.21)$$

of the target beam parameters  $\mathbf{b}'$ , the misalignments  $\mathbf{m}$  of the quadrupole magnets and the diagnostic screen, the incoming beam  $\mathcal{B}_{\text{in}}$ , as well as the initial magnet settings  $\mathbf{u}_0$ . The target beams, incoming beam and misalignments were randomly sampled from uniform distributions representing the actual operating ranges of the real-world accelerator as given

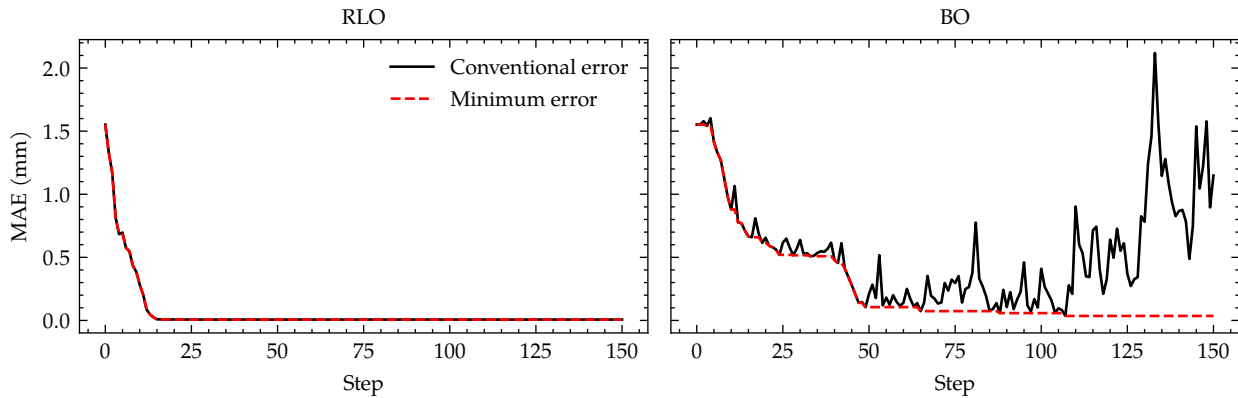


Figure 4.5: Illustration of the difference between the conventional and minimum variants of the metrics used in this work at the example of the error between observed and target beam over time taken from a tuning run performed with RLO and another performed with BO.

in Eqs. (4.8) to (4.10). For reference, the simulation results of the three RLO policies are compared to the trivial baseline of doing nothing, i.e. leaving the magnets at their initial settings  $\mathbf{u} = \mathbf{u}_0$ . A detailed comparison of the performance of RLO to other optimisation algorithms is made in Section 4.2.3. In the evaluation in simulation, the policies are given a maximum of 150 steps to reach the target beam parameters.

The results of the evaluation of the three RLO policies as well as the baseline are given in Table 4.2. They clearly demonstrate that all three RLO policies are able to reduce the MAE by almost three orders of magnitude compared to no tuning. While there are slight differences in performance between the three policies, they all manage to achieve a median final error of less than  $10 \mu\text{m}$ , which is well below the measurement accuracy of the diagnostic screen of  $20 \mu\text{m}$ . In fact, all policies are found to reduce the MAE by 99%. The RMSE of the policies is also found to be low, with  $117 \mu\text{m}$  to  $124 \mu\text{m}$ . This suggests that the policies are able to converge quickly to the target beam parameters. In fact, the policies converge to an MAE lower than  $40 \mu\text{m}$  on over 90% of the trials and within a median of 6 to 7 steps. Figure 4.6 confirms this assessment, showing that the distribution of minimum final beam errors is centred well below the measurement accuracy and 10 steps. The different policies show only slightly different performance, with none of them being outright better than the others in all metrics, and differences generally being within the standard deviation of the results.

## Real World

To verify the approach of training accelerator tuning policies in simulation and transferring them zero-shot without any additional training to the real accelerator, the three trained policies were also evaluated on the real ARES accelerator. The evaluation was done by exchanging the backend of the environment from the simulation model to the interface with the control system of the real accelerator. Because of limited availability of the facility, the evaluation was limited to the first 200 trials. In addition, the misalignments cannot be reasonably changed in the real world, so they were kept constant on the true misalignments of the quadrupoles and the diagnostic screen at ARES at the evaluation beam time. The incoming beam, too, cannot be controlled or measured directly. Instead, the incoming beam depends on the working point and drift of the accelerator present at the time of the

Table 4.2: Performance of RLO with different policies in simulation and on the real machine

Optimiser	Final error ( $\mu\text{m}$ )		RMSE ( $\mu\text{m}$ )		Improvement (%)		Steps to target		
	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Success rate
<i>Simulation</i>									
$\pi_1$	7	$16 \pm 24$	124	$129 \pm 61$	99	$97 \pm 4$	7	$12 \pm 16$	92%
$\pi_2$	10	$21 \pm 61$	118	$125 \pm 57$	99	$97 \pm 8$	6	$10 \pm 12$	91%
$\pi_3$	6	$18 \pm 29$	117	$126 \pm 59$	99	$97 \pm 6$	6	$11 \pm 20$	89%
<i>Real world<sup>1</sup></i>									
$\pi_1$	28	$56 \pm 64$	323	$350 \pm 182$	96	$92 \pm 11$	11	$13 \pm 7$	66%
$\pi_2$	29	$60 \pm 73$	344	$370 \pm 198$	96	$92 \pm 8$	14	$15 \pm 8$	61%
$\pi_3$	51	$70 \pm 60$	357	$373 \pm 184$	93	$92 \pm 6$	9	$11 \pm 6$	45%
<i>Baselines</i>									
Do nothing	1633	$2209 \pm 1823$	2150	$3045 \pm 2688$	0	$0 \pm 0$	-	-	0%

<sup>1</sup>The real-world results were collected from the first 200 trials and limited to 30 steps per trial because of limited beam time availability.

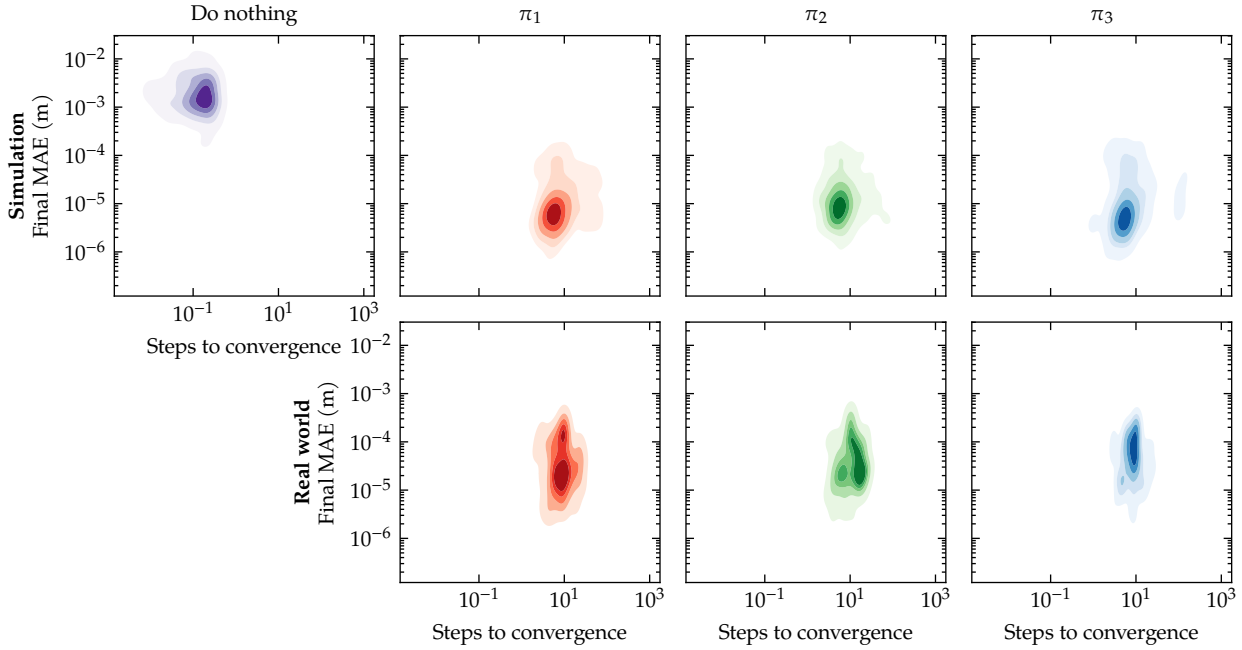


Figure 4.6: Distributions of the final MAEs achieved by the RLO policies during optimisations plotted over the steps to convergence. Kernel density estimations (KDEs) are used to visualise the distributions. Ideally, distributions should be concentrated in the lower left, indicating that policies converge quickly to good solutions. Distributions in the upper left converge quickly but to bad solutions, while distributions in the lower right converge slowly to good solutions. Note the logarithmic scaling of the axes. Because doing nothing always converges in zero steps, a small amount of noise between 0 to 0.3 steps was added to the steps to convergence for doing nothing to enable the visualisation with a KDE. The steps to convergence are measured on the minimum beam to guarantee that all runs converge and are included in the distribution. An equivalent figure with the minimum MAE is available in Appendix B.

evaluation, this was set to an arbitrary normal machine state, usually by leaving it as it was left from previous experiments. This should give a good spread over reasonable working points and drift states.

The results from the real accelerator, listed in Table 4.2, show that the sim-to-real transfer is successful and that the RLO policies perform well in the real world, too, achieving a median final error of 28  $\mu\text{m}$ , 29  $\mu\text{m}$  and 51  $\mu\text{m}$ . This is worse than in simulation, but notably only slightly larger than the measurement accuracy of the diagnostic screen and mostly within the 40  $\mu\text{m}$  threshold. An example of the tuning process of one of the policies on the real accelerator is shown in Fig. 4.7. The RMSE is found to increase less than the final error when transferring to the real accelerator, with values of 323  $\mu\text{m}$  to 357  $\mu\text{m}$ , suggesting that the policies converge onto the target beam about as fast as in simulation. A median of 9 to 14 steps are needed to reach an MAE lower than 40  $\mu\text{m}$ . In accordance with these results, the normalised beam error improvement is found to be 93 % to 96 %, i.e. almost all of the error is removed by the policies. Fewer trials meet the 40  $\mu\text{m}$  threshold in the real world than in simulation, though the reader is reminded of the strictness of the target condition in relationship to the measurement accuracy that likely contributes significantly to this observation. Nevertheless, the slight decrease in performance from simulation to the real accelerator suggests that the policies have slightly overfitted to the simulation, where the

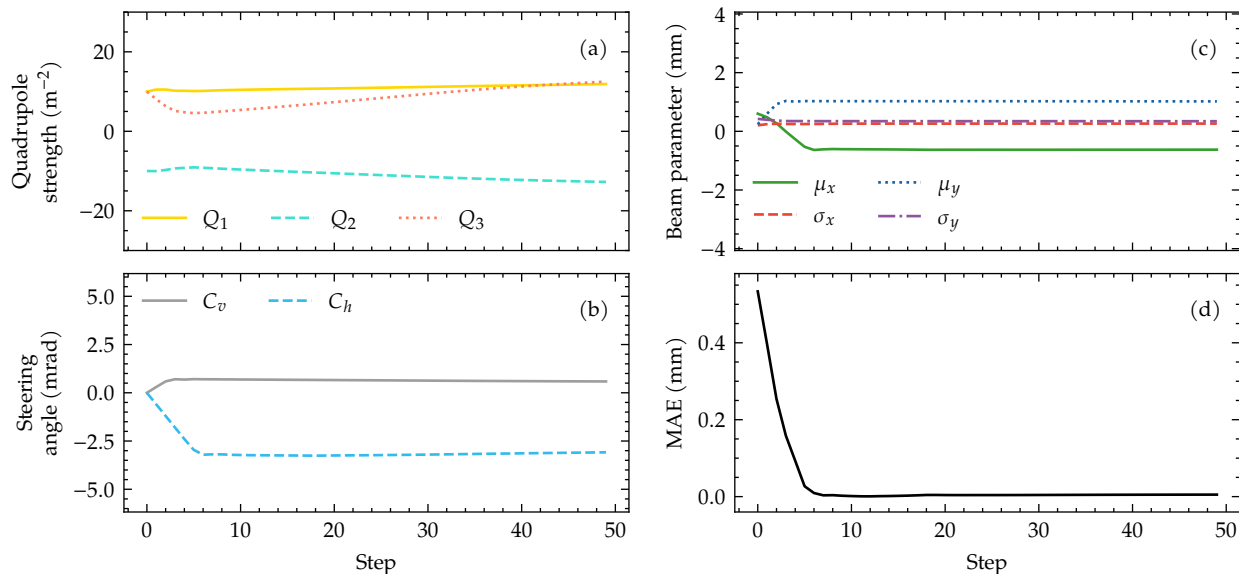


Figure 4.7: A neural network policy trained with RL tuning the transverse beam parameters in the ARES EA, where the target transverse beam parameters of  $\mathbf{b}' = (-0.61 \text{ mm}, 0.26 \text{ mm}, 1.03 \text{ mm}, 0.35 \text{ mm})$  are reached after about 6 steps [162]. (a) and (b) quadrupole and dipole magnet settings over parameter update steps. (c) Beam parameters on the diagnostics screen over parameter update steps. (d) The MAE between observed and target beam over parameter update steps.

real world differs from the assumed Cheetah model. In fact, consulting Fig. 4.6, one can see that the distribution is shifted slightly up and spread out more than in simulation. Though observable, the shift is so small and so narrowly above the measurement accuracy that it is not considered an issue. Note that unlike in simulation, where no policy performed better than the others in terms of all metrics, such a ranking can be made out in the real-world results. This suggests that some policies generalise better while others overfitted more to the simulation. These differences, however, are marginal and within the standard deviation of the results.

Notwithstanding, the results show that RLO training on a high-speed simulation with domain randomisation can be successfully transferred to the real accelerator if the most important sources of uncertainty are included in the domain randomisation.

### Comparison to Expert Human Operators

Furthermore, the three policies trained with different random seeds for the RLO implementation have been compared to two expert human operators on a single trial, with tuning runs being conducted consecutively in order to establish the policies' performance in relation to human-level. Note that RLO can only interact with the accelerator every 10 s to 20 s because it has to wait for magnets to reach their set points and a new beam measurement to be taken before predicting the next action. The human operators are not limited by this and can already take the next action while the magnets are settling and a trend can be identified, thus being able to interact with the accelerator with as much as 1 Hz. Because of this, this particular comparison is made in wall time instead of steps.

The results indicate that, while achieving a similar final error as expert human operators, RLO can do so more quickly and consistently, even though it is constrained by a control

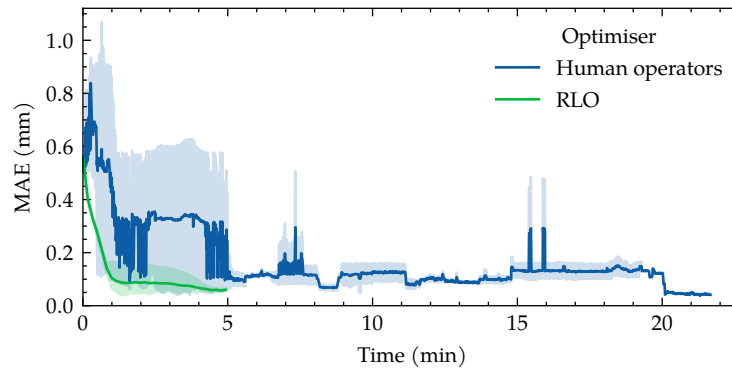


Figure 4.8: Comparison of three RLO policies to two expert human operators, showing distance from the observed beam to the target beam over time [164]. The mean beam distance as the MAE is shown by the solid line. The envelope shows the 95 % confidence interval of the beam distance.

frequency that is 10 to 20 times slower. A plot of the MAE developing over the tuning by RLO compared to the human operators is shown in Fig. 4.8. By the end of their optimisation, the human operators achieved a median MAE of  $39 \mu\text{m}$  compared to  $60 \mu\text{m}$  achieved by the RLO policies. While the final MAE achieved by the operators is slightly lower, the difference is only marginally larger than the measurement accuracy. In addition, it took the human operators on average 18 min to converge on their solution. This is significantly longer than the 1.1 min taken by the RLO policies. Note that the key reason for this is fine adjustments toward the end of the operators' optimisations. Over the first 5 min the operators MAE runs only slightly higher than that of the RLO policies. While the three policies converged consistently, there was a significant variance in MAE over the first few minutes between the human operators. The observed variance can largely be attributed to different tuning strategies used by the operators. Unlike RLO, which adjust all five actuators at the same time in a direct path toward the optimum, the human operators split the task into multiple sub-tasks. The operators, for example, start by fixing either position or focus of the beam first. It was also observed that the human operators adjusted magnets one after the other. In contrast to the RLO policies, the human operators also limited themselves to only using two of the three quadrupoles in the final solution, presumably to simplify the task. Both final solution patterns are considered physics textbook strategies, though in the EA, avoiding the use of the third quadrupole can be advantageous because the vertical steerer is placed upstream of the third quadrupole. The latter almost always introduces a strong dipole moment as a result of the beam trajectory being moved off-centre by the steerer. This dipole moment can be avoided by not using the third quadrupole, while the task remains solvable with the remaining two quadrupoles. Note that on the real accelerator, RLO policies spent most of their time waiting for the function evaluation to finish as the magnets in this accelerator section are controlled by relatively slow power supplies. When faster power supplies are available, the RLO policies can achieve a much more substantial speed advantage over human operators.

Overall, it can be concluded that the RLO policies are able to match expert human operators in the quality of the tuning results and outperform them in terms of the speed of the tuning process, while being more consistent in their performance.

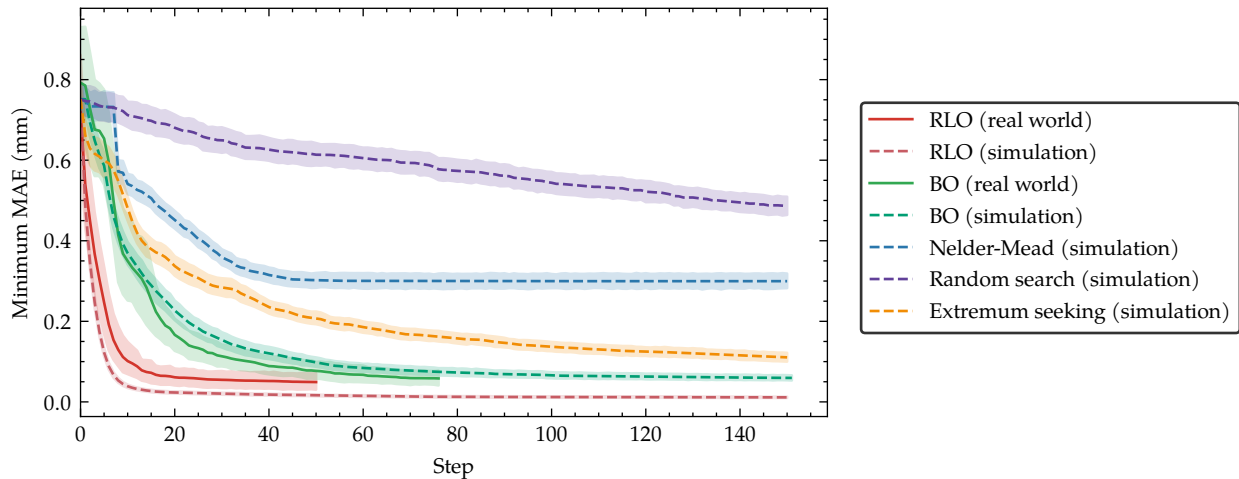


Figure 4.9: Minimum error between observed and target beam over time for different optimisation algorithms [164]. The MAEs of the beam parameters to the target beam are shown for real-world experiments (solid lines) and simulated experiments (dashed lines), averaged over all trials. The envelopes show the 95 % confidence intervals of the beam differences averaged over all trials.

### 4.2.3 Comparing to Alternative Algorithms

In the following, the performance of the RLO policies is compared to the performance of the BO implementation introduced in Section 4.2.1 as well as to the performance of the baseline algorithms, random search, Nelder-Mead simplex and ES. The comparison is done on the same set of 300 trials as used for the evaluation of the RLO policies. The results are listed in Tables 4.3 and 4.4 and illustrated in Fig. 4.9. In addition, aspects like the time it takes each tuning algorithm to infer the next magnet settings, their robustness to real-world disturbances and edge cases are discussed.

#### Simulation Study

Comparing RLO to the other algorithms in simulation, it is observed that RLO clearly outperforms all of them in terms of all conventional metrics listed in Table 4.3. In fact, it presents an almost two order of magnitude improvement in final error over the next-best algorithm, ES. BO is found to achieve a final error that is about 4 times larger than the one achieved by ES and two orders of magnitude larger than the one achieved by RLO, with BO improving the beam error by only 16 % compared to the 99 % improvement achieved by RLO. The final error achieved by random search and Nelder-Mead simplex is found to be worse than the error at the beginning of the optimisation, with random search performing significantly worse. Similarly, RLO reaches the target at about 6 times faster than BO and 7 times faster than ES, while successfully reaching the target more than twice as often as BO and 5 times more often than ES. Random search on the other hand never finds magnet settings that result in a beam error within the target range, while Nelder-Mead simplex only achieves this in 1 % of the trials.

As alluded to earlier, the conventional metrics do not necessarily accurately measure the performance of random search, and to some extent BO, because they do not perfectly reflect the way these algorithms are typically used in tuning setups. Considering instead the minimum metrics in Table 4.4, confirms this. Here, random search achieves an improvement

Table 4.3: Performance of optimisation algorithms in terms of the conventional metrics

Optimiser	Final error ( $\mu\text{m}$ )		RMSE ( $\mu\text{m}$ )		Improvement (%)		Steps to target		
	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Success rate
<b>Simulation (infinite diagnostic screen, no beam alignment)</b>									
Random search	7542	$11\,298 \pm 12\,839$	21 495	$21\,981 \pm 7385$	-940	-	-	-	0%
Nelder-Mead	736	$811 \pm 400$	2964	$3019 \pm 831$	-8	56	55 $\pm$ 7	55 $\pm$ 7	1%
Extremum seeking	145	$166 \pm 102$	482	$536 \pm 214$	79	121	107 $\pm$ 38	107 $\pm$ 38	17%
RLO	7	$16 \pm 24$	124	$129 \pm 61$	99	7	12 $\pm$ 16	12 $\pm$ 16	92%
BO	566	$682 \pm 465$	748	$749 \pm 288$	16	40	48 $\pm$ 26	48 $\pm$ 26	43%
<b>Real world (finite diagnostic screen, beam aligned to quadrupole magnets)</b>									
RLO <sup>12</sup>	34	$63 \pm 61$	178	$240 \pm 147$	93	12	13 $\pm$ 7	13 $\pm$ 7	55%
BO <sup>13</sup>	251	$378 \pm 421$	524	$534 \pm 288$	72	45	48 $\pm$ 12	48 $\pm$ 12	45%
<b>Simulation (finite diagnostic screen, no beam alignment)</b>									
RLO	7	$4768 \pm 82\,170$	123	$17\,021 \pm 128\,429$	99	7	12 $\pm$ 16	12 $\pm$ 16	93%
BO	404	$47\,994 \pm 255\,776$	291 325	$267\,081 \pm 151\,555$	40	44	53 $\pm$ 28	53 $\pm$ 28	53%
<b>Simulation (finite diagnostic screen, beam aligned to quadrupole magnets)</b>									
RLO	6	$14 \pm 22$	114	$119 \pm 58$	99	6	12 $\pm$ 15	12 $\pm$ 15	95%
BO	810	$62\,669 \pm 290\,025$	291 325	$283\,240 \pm 80\,820$	-37	33	37 $\pm$ 19	37 $\pm$ 19	85%

<sup>1</sup>Because of limited beam time availability at the real accelerator, 22 trials were selected for the real-world evaluations.<sup>2</sup>RLO was given 50 steps for the optimisation on the real accelerator in order to perform more evaluation runs with limited beam time.<sup>3</sup>BO was given 75 steps for the optimisation on the real accelerator in order to perform more evaluation runs with limited beam time.

Table 4.4: Performance of optimisation algorithms in terms of the minimum metrics

Optimiser	Final error ( $\mu\text{m}$ )		RMSE ( $\mu\text{m}$ )		Improvement (%)	
	Median	Mean	Median	Mean	Median	Mean
<i>Simulation (infinite diagnostic screen, no beam alignment)</i>						
Random search	464	486 $\pm$ 199	792	800 $\pm$ 310	22	28 $\pm$ 27
Nelder-Mead	266	300 $\pm$ 163	474	510 $\pm$ 213	61	53 $\pm$ 28
Extremum seeking	81	111 $\pm$ 94	310	372 $\pm$ 206	87	84 $\pm$ 13
RLO	4	11 $\pm$ 19	123	127 $\pm$ 60	99	98 $\pm$ 4
BO	45	60 $\pm$ 55	249	290 $\pm$ 168	94	91 $\pm$ 10
<i>Real world (finite diagnostic screen, beam aligned to quadrupole magnets)</i>						
RLO <sup>12</sup>	24	49 $\pm$ 48	178	238 $\pm$ 144	93	93 $\pm$ 5
BO <sup>13</sup>	44	59 $\pm$ 41	306	357 $\pm$ 189	93	91 $\pm$ 6
<i>Simulation (finite diagnostic screen, no beam alignment)</i>						
RLO	4	13 $\pm$ 40	120	10271 $\pm$ 54 830	99	98 $\pm$ 5
BO	38	61 $\pm$ 96	254	14071 $\pm$ 75 157	95	91 $\pm$ 11
<i>Simulation (finite diagnostic screen, beam aligned to quadrupole magnets)</i>						
RLO	4	10 $\pm$ 18	114	118 $\pm$ 58	99	98 $\pm$ 4
BO	23	28 $\pm$ 19	232	237 $\pm$ 109	96	95 $\pm$ 3

<sup>1</sup>Because of limited beam time availability at the real accelerator, 22 trials were selected for the real-world evaluations.

<sup>2</sup>RLO was given 50 steps for the optimisation on the real accelerator in order to perform more evaluation runs with limited beam time.

<sup>3</sup>BO was given 75 steps for the optimisation on the real accelerator in order to perform more evaluation runs with limited beam time.

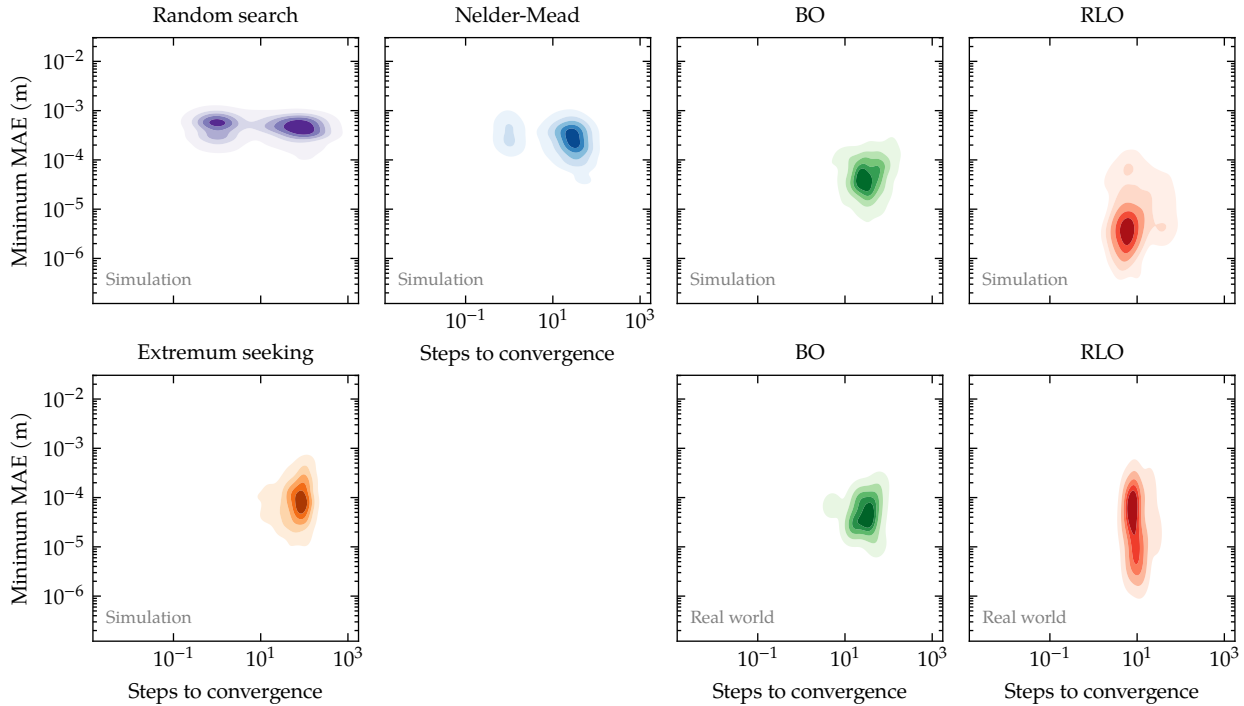


Figure 4.10: Distributions of the minimum MAEs achieved by the different algorithms during optimisations plotted over the steps to convergence. KDEs are used to visualise the distributions. Ideally, distributions should be concentrated in the lower left, indicating that policies converge quickly to good solutions. Distributions in the upper left converge quickly but to bad solutions, while distributions in the lower right converge slowly to good solutions. Note the logarithmic scaling of the axes. The steps to convergence are measured on the minimum beam to guarantee that all runs converge and are included in the distribution. An equivalent figure with the final MAE is available in Appendix B.

of 22 % from the initial beam to the best one encountered during the search. What is more, in terms of the minimum metrics, BO clearly outperforms ES, but is still outperformed itself by RLO. While both BO and RLO achieve over 90 % improvement of the beam error, RLO achieves a final error about an order of magnitude smaller than the one achieved by BO. Nevertheless, the minimum final error achieved by BO is only a little larger than the target threshold of  $40 \mu\text{m}$  and the one achieved by RLO is well below the measurement accuracy, meaning that this difference, while noteworthy, is less relevant in practice.

Figures 4.9 and 4.10 illustrate these results further. Figure 4.9 clearly shows the progressively faster convergence to better beams in terms of the minimum beam error, where random search performs the worst, followed by Nelder-Mead simplex, ES and BO, with RLO performing the best. Similarly, Fig. 4.10 shows that, while algorithms like random search can find their best solution quickly, this solution is reliably worse than those found by RLO, BO and even ES. The demonstrated better performance of RLO compared to BO is clearly visible, though a larger spread in the results of RLO can be observed. It is a clue that the RLO policies might have overfitted to some conditions of the task and perform better on trials that resemble them more closely. BO on the other hand is robust to this kind of overfitting, as it learns directly on the conditions of the present task, which is, however, also why it is slower to converge than RLO.

An important observation from these results is the difference in performance between

the conventional metrics in Table 4.3 and the minimum metrics in Table 4.4. It highlights how, while BO is mostly on par with RLO in a tuning setting, where one can stop at any time and recover the best solution if needed, RLO and even ES are a more suitable choice when this is not possible, such as in a feedback setting, or when strong temporal dependencies like hysteresis inhibit the recovery of a previously found solution.

This is primarily the result of how and when RLO and BO explore. RLO has explored many objective functions of the relevant kind during training, allowing a trained policy to exploit this knowledge to find the optimum quickly and with minimal exploration. From this learned experience, policies appear to even be able to discern that a (local) optimum has been found, and no further exploration is needed. BO on the other hand, is designed to explore the objective space more thoroughly, which it must, given that it has no prior knowledge of the objective function. Even when one local optimum is discovered, depending on the acquisition function used, BO might continue to explore the objective space and find other potentially better optima. This effect can in fact be observed in both Figs. 4.5 and 4.12, where the error to the target beam increases again after a solution with small error was encountered. In fact, the more extensive exploration of the objective space by BO can be seen in Fig. 4.11. Note that the two figures show different slices of the objective function, as they are shown with respect to the final sample found by each algorithm. BO can be seen to explore a larger region of the objective space while finding an optimum near the one found by RLO. The relatively direct line of samples traversed by RLO on the way to the optimum supports the assumption that RLO can take advantage of experience from training encoded in the policy to know in which direction to find the optimum. It is possible to suppress BO's exploration after encountering a local optimum by using an acquisition function that favours exploitation, but experiments have shown that such acquisition functions are more susceptible to objective functions with multiple prominent local optima.

### Real-World Study

In order to evaluate the methods' ability to transfer to the real world and to verify the results obtained in simulation, RLO and BO were benchmarked on the real ARES particle accelerator. This part of the study is crucial, as even with accurate simulations, the gap between simulation and the real world is often wide enough that algorithms performing well in simulation cannot be transferred to the real world [69]. This gap between simulation and experiment was already observed in the early stages of training the RL policy, where trained policies performed well in simulation but failed altogether in the real world. Similarly, when implementing BO for the tuning task, implementations tuned for exploitation showed faster and better optimisation in simulation but failed during the experiments under real-world conditions.

Given the limited availability of the real accelerator, 22 trials of the 300 used for the simulation study were considered for the real-world evaluation. To ensure a fair comparison in benchmarking real-world performances, the algorithms are run back-to-back for each trial. As a result, the conditions experienced by different algorithms are as close as possible. Before every measurement shift, an automated alignment was performed in order to bring the incoming beam close to the centres of the quadrupole magnets and reduce dipole moments induced when the beam passes through a quadrupole magnet off-centre, which can steer the beam too far off the screen in the initial state. This adjustment is needed for BO to find an objective signal in a reasonable time. The effects this alignment (or the

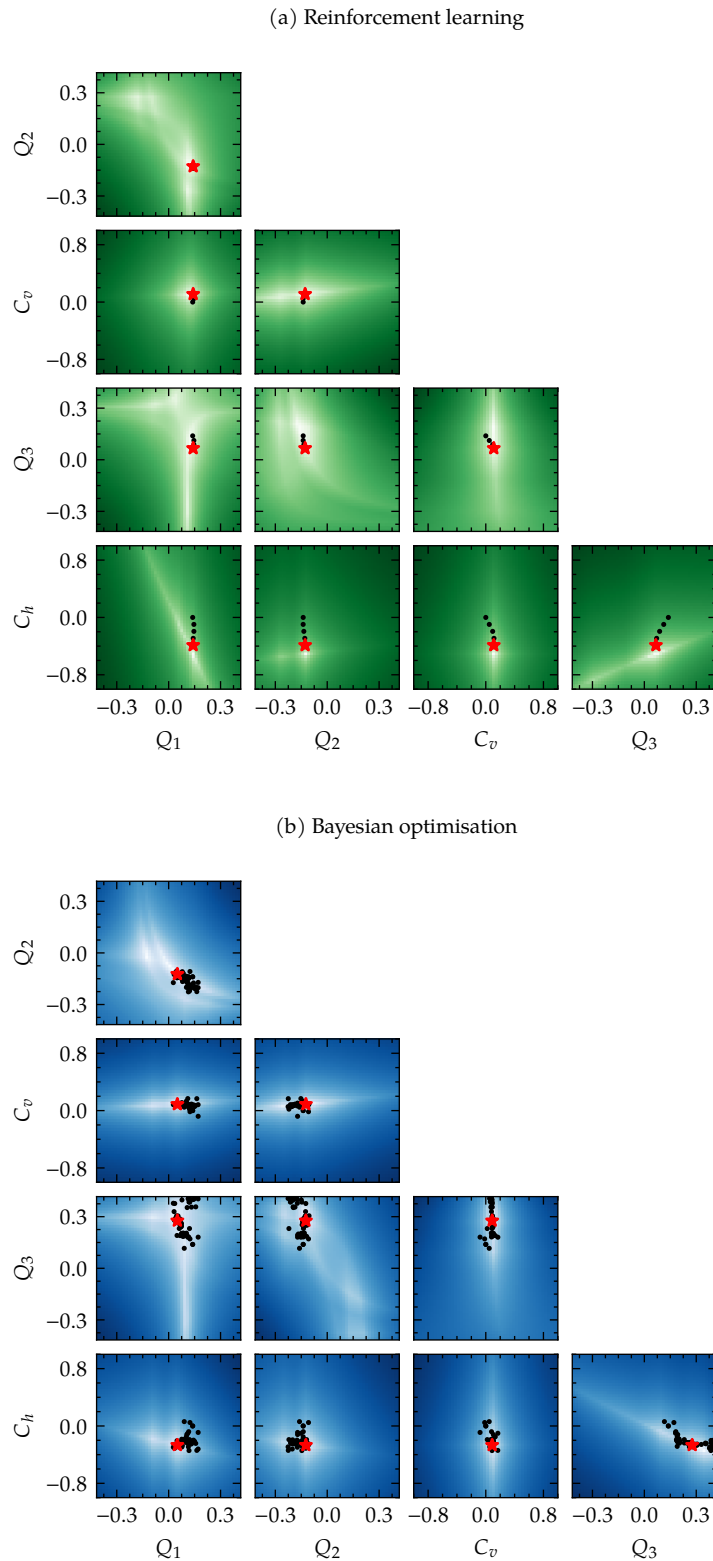


Figure 4.11: Samples taken by RLO and BO on the objective function of a single trial [164]. Black dots mark samples taken before the threshold  $\epsilon$  was reached. Each heat map shows a 2-dimensional slice of the objective space anchored in the final sample, denoted by the red star. Magnet settings are normalised to their permissible ranges, with the final normalised magnet setting for at RLO at  $\mathbf{u} = (0.14, -0.13, 0.11, 0.07, -0.39)$  and for BO at  $\mathbf{u} = (0.05, -0.12, 0.09, 0.28, -0.27)$ . The colour maps are scaled individually for each slice, and shown logarithmically for easier illustration. Lighter shades indicate lower values.

lack thereof) has on the results of this comparison are investigated further below. The experiments comparing the performance of RLO and BO on the real accelerator were conducted on 9 different days over the course of 82 days, running at charges between 2.6 pC and 29.9 pC, and an energy of 154 MeV. Results from the real-world measurements are listed in Tables 4.3 and 4.4. Two example optimisations by RLO and BO on the real accelerator are shown in Fig. 4.12.

On the real particle accelerator, just like in the simulation study, both RLO and BO are found to perform well. Both methods were able to discover minimum beams that were close to the target beam parameters, as can be seen in the beam images in Fig. 4.12, though the final beams that BO stopped at are about 5 times worse in terms of beam parameter error than those of RLO. Just as in simulation, RLO clearly outperforms BO when the final error is considered as in Table 4.3, though the discovered beams are more evenly matched when considering the minimum beam encountered during the optimisation as in Table 4.4. However, even then, RLO is found to find magnet settings that are better than the target threshold almost 4 times faster and 20 % more often. In addition, it can be observed that RLO converges more smoothly than BO. While this has little effect in simulation, in the real world, smooth convergence has various advantages which are discussed further below. Moreover, RLO outperforms BO in 13 out of 22 trials in terms of the best discovered beam, which is a lower proportion than in the simulation study. In fact, RLO performs consistently worse on the real accelerator than in simulation. Some of this can be attributed to limited measurement accuracy, where the minimum beam errors encountered by RLO at a median of 24  $\mu\text{m}$  are only marginally larger than the measurement accuracy of the diagnostic screen. Part of this effect may however also be the result of policies overfitting to the simulation environment. In contrast, BO even performs better on the real accelerator than in simulation by some metrics, though the difference is usually within the standard deviation. This is confirmed by the distributions shown in Fig. 4.10, where the RLO distribution is clearly shifted upward toward a larger minimum error compared to the simulation study, while the BO distribution is almost unchanged.

## Inference Speed

The time it takes to infer the next set of actuator settings depends critically on the chosen algorithm and can have a significant impact on the overall tuning time and whether an algorithm is suitable for a given task. For the transverse tuning task at ARES, the inference time is not a limiting factor, as the magnet power supplies are orders of magnitude slower than inference and there are no real-time requirements brought on by the dynamics of the physical system. With other tuning tasks, where the physical process takes less time, the time taken for tuning may be dominated by the inference time of the tuning algorithm and there might even be real-time requirements [345].

Inference times for RLO and BO can vary greatly depending on the choice of model and other design parameters. Nevertheless, inference time measurements were performed for the specific implementations of RLO and BO presented here. Their average inference times over the 45 000 inferences of the simulation study were measured using a MacBook Pro with an M1 Pro chip running Python 3.9.15. It is observed that BO takes an average of 0.7 s to infer the next actuator settings, while RLO is more than three orders of magnitude faster at 0.0002 s. This reflects the generally expected trend that RLO is capable of faster inference because RLO requires only a forward pass of the neural network policy with a constant  $O(1)$  complexity with respect to the number of steps taken. In contrast, each BO inference

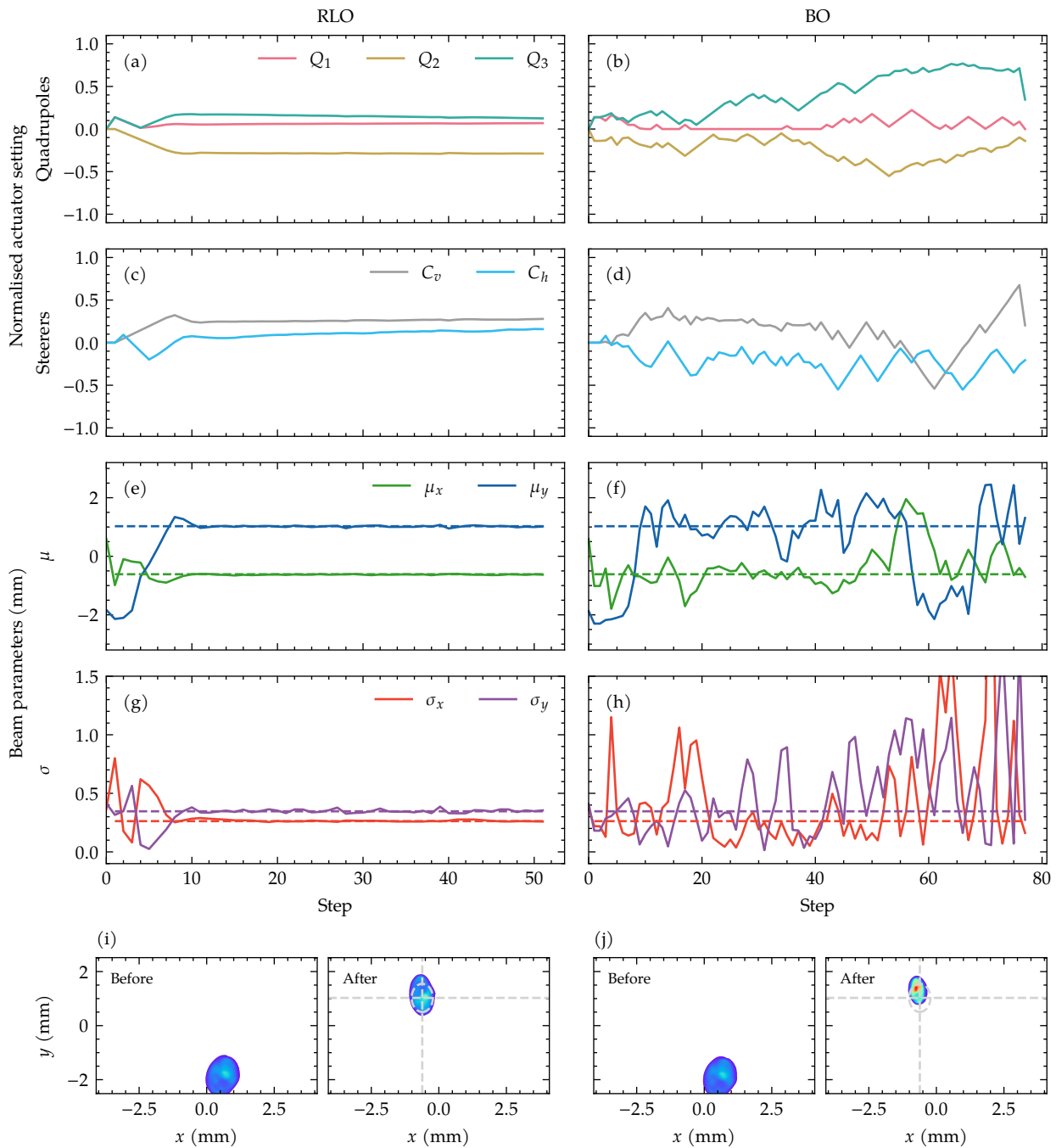


Figure 4.12: Example optimisations on the real particle accelerator [164]. (a), (c), (e) and (g) show the history of an optimisation with RLO. (b), (d), (f) and (h) show the history of an optimisation with BO. (a) and (b) show the steering magnet settings, (c) and (d) show the quadrupole magnet settings, (e) and (f) show the beam positions, and (g) and (h) show the beam sizes. (i) and (j) show the beam images before and after the optimisation respectively. The target beam size and position are indicated with dashed lines. Note that at the end of the shown BO optimisation, the actuators are returned to the best discovered settings. Hence, the "After" screen image shows the beam after attempting to recover the best settings.

step requires a full optimisation of the acquisition function. This involves inferences of the GP model with inference times scaling  $O(n^3)$  with the number of steps taken  $n$ . Even when choosing a different faster model, BO requires an optimisation of the acquisition function in each step, meaning it is generally expected to have slower inference than RLO. Note that the RLO inference can also be sped up significantly by using specialised hardware [295].

### Practical Challenges

Beyond the general performance of the algorithms, there are a number of practical challenges that need to be considered when choosing an algorithm for a given tuning task. These are discussed in the following.

**Sim-to-real transfer** The transfer of a method that works well in a simulation environment to the real world is a large part of developing tuning algorithms for facilities such as particle accelerators. The challenges posed by this so-called sim-to-real transfer impact the choice of tuning algorithm.

Successfully transferring a policy trained for RLO to the real ARES accelerator involved a number of engineering decisions [158] detailed in Section 4.1. While some of the design choices, such as inferring changes to the actuator settings instead of the actuator settings directly, can be applied to other tuning tasks with relative ease, others, such as domain randomisation [239, 323], require specialised engineering for each considered tuning task. Furthermore, all of these require time-consuming fine-tuning to actually achieve a successful zero-shot sim-to-real transfer of a policy trained only in simulation. This is illustrated by the fact that many of the policies trained before the final ones studied here performed well in simulation while sometimes not working at all on the real ARES accelerator.

On the other hand, the evaluated BO implementation transfers to the real world with relatively little effort. Once a method for dealing with faulty measurements was found, the BO implementation performed about as well on the real accelerator as it did in simulation. The method is further discussed in the next section. Only some more specialised design decisions that appear sensible in simulation, such as tuning the acquisition function strongly towards exploitation, do not transfer well, because BO may overfit to falsely positive noisy samples of the objective function. These potential pitfalls need to be considered when first designing a BO solution in simulation. The easier sim-to-real transfer of BO is likely owing to the fact that GP model is learned entirely on the real observations and therefore will not overfit to a different objective function that deviates from the one under optimisation.

One issue that may arise when transferring BO from simulation to the real-world task is over-exploration. RLO naturally converges toward an optimum and then stays there, meaning that, if the optimisation is ended at any time, the environment's state is at or close enough to the best-seen position as predicted by the policy. Depending on the choice of acquisition function, BO is likely to explore further even after finding the global optimum if the parameter space is large enough to encourage exploration. In practice, therefore, it is often necessary to return to the best-seen input when the optimisation is terminated. While this strategy will recover the same objective value in simulation, it is often not ideal for real-world objective functions, which are noisy, not stationary for instance due to slow thermal drifts, or path-dependent. When performing transverse tuning, for example, one encounters magnet hysteresis, where the ferromagnetic core of an electromagnet retains some magnetisation when the current in its coils is turned off.

This causes a path dependence of the beam parameters and consequently the objective function. When returning to a set of electric currents in the coils of the electromagnets that were already seen before, the actual strengths of the magnetic fields may differ, and the objective value may shift.

The transverse tuning task in the EA experiences magnet hysteresis and noisy measurements. It was found that for the studied BO trials, the difference between the beam error when first discovering the best magnets and when recovering them after the tuning deviated by a median of 11  $\mu\text{m}$  and a maximum of 42  $\mu\text{m}$ . This means that the deviation is usually smaller than the screen's estimated accuracy and the noise present in the beam parameter measurements. For the benchmark task, the effects of hysteresis and noise are therefore non-negligible, but also not detrimental to the performance of the tuning algorithms.

Path-dependent effects like magnet hysteresis are amplified by more back-and-forth actions such as those performed by BO in the exploration phase. As already shown in Fig. 4.12, RLO performed much smoother actions in general, which could reduce wear on the actuators and avoid possible additional noise and shifts in the objective function. This further stabilises the tuning process and improves reproducibility. Additionally, it was found that in the case of the hardware installed in the EA, erratic actions sometimes lead to glitches with the electronics of the power supplies driving the magnets. Similar challenges are encountered in most real-world systems, which are always subject to imperfections. Tuning algorithms that produce smoother actions are therefore generally preferred in practice. Both RLO and BO can be designed to produce smoother actions. The smoothness of the actions taken by RLO can be enforced by adding additional terms to the reward function during training. In theory, BO can be made to produce smoother actions through proximal biasing, which has been shown to produce smooth actions respecting the hardware limits of the accelerator [266, 358]. However, as is shown in Fig. 4.4, proximal biasing was found to not improve the performance of BO on the transverse tuning task evaluated here.

**Robustness in the presence of sensor blind spots** In any real system, it is possible to encounter states where the available diagnostics deliver false or inaccurate readings, resulting in erroneous objective values and observations. Transitions to these states can be caused by external factors as well as the tuning algorithm itself. A good tuning algorithm should therefore be robust in the presence of these states and able to recover from them. In the transverse beam tuning task, an erroneous measurement occurs when the electron beam is not visible on the diagnostic screen within the camera's field of view. In this case, the beam parameters computed from the diagnostic screen image are false, also resulting in a faulty objective value.

It was found that, when the beam is not properly observed, RLO can usually recover it in just a few steps. Presumably, the policy can leverage its experience from training to make an educated guess on the beam's position based on the magnet settings even though faulty beam measurements were not part of its training, where RLO always had access to the correct beam parameters.

In contrast, BO struggles to recover the beam when it is off-screen at the beginning of a tuning run, as the GP model is learned completely at application time. The faulty observations lead to wrong predictions of the objective and acquisition functions. When defining the task's objective function as only a difference measure from the current to the target beam, falsely good objective values are predicted in the blind spot region of the actuator space and BO converges towards their locations. The studied BO implementation

alleviates this issue by introducing a constant punishment to the objective function when no beam is detected in the camera's field of view. Nevertheless, the lack of information about the objective function's topology results in BO taking many arbitrary steps before the beam is again detected on the screen and the optimisation starts progressing towards the target beam. While more comprehensive diagnostics can help solve this problem, these are often not available. Although not considered in this study, the sensor blindness issue can be alternatively mitigated by initialising the GP model with a prior mean function learned from simulation or history data, which by design favours settings that result in the beam hitting the diagnostic screen within the camera's field of view. However, this limits the transferability of BO to new problem settings, analogous to an RLO with a pre-trained policy. In addition, this is not feasible for every tuning task. For example, the considered transverse tuning task is only partially observable due to the unknown incoming beam and magnet misalignments. This makes creating an informed prior difficult and in some cases even impossible.

Because of BO's insufficient ability to recover from a system state in which there is no informative objective signal, the presented measurements on the real accelerator were taken with the beam aligned to the quadrupole magnets. As a result, the additional dipole moments induced by the quadrupole magnets when increasing the magnitude of their focussing strength are kept minimal, reducing the chance that the beam leaves the camera's field of view during the initial step of the optimisation. As this alignment would not be performed during nominal operation but may change the observed performance of both algorithms, a study was performed in simulation in order to understand how to interpret the reported results given that the beam was aligned to the centres of the quadrupole magnets before the optimisation. Both algorithms are evaluated on the same 300 trials used in the rest of this comparative study. Unlike in the original simulation study, erroneous beam parameter measurements are modelled when the beam position is detected outside the camera's field of view. RLO and BO are tested once with the original incoming beam and once with an incoming beam that was previously aligned to the quadrupole magnets. The results are reported in Tables 4.3 and 4.4. Considering the minimum metrics, it is found that the reported results on the real particle accelerator would be expected to worsen by about 0 % to 16 % for RLO and by 3 % to 65 % for BO if the electron beam had not been aligned to the centres of the quadrupole magnets at the beginning of measurement shifts. This does not change the relative performance of the two algorithms.

**Edge cases and limitations** Naturally, both tuning algorithms have limitations that could cause them to fail. With the goal of autonomous accelerator operation, it is important to understand these limitations and how they can be mitigated.

In the previously presented experiments, neither RLO nor BO ever produced a final beam that was worse than the beam before the optimisation. Nevertheless, measured metrics vary by more than an order of magnitude between trials for both algorithms. Furthermore, there are a few outlier trials, in which the algorithms perform significantly worse. Note that these are not the same trials for both algorithms and that no clear cause of the worse performance could be identified. These outliers are likely caused by the stochastic components of both algorithms.

In order to better understand under which conditions either algorithm performs worse, grid scans over the target beam space were performed in simulation. Their results are visualised in Figs. 4.13 to 4.15. The target beams were scanned with 20 samples for each of the four beam parameters, resulting in 160 000 different target beams. Misalignments and

incoming beams were kept constant for these scans. The grid scans confirm the presence of stochastic outliers with bad performance for both RLO and BO. They further show that both learning-based algorithms perform worse for large target beams, though this effect is subtle compared to the outliers. Random search does not show the same effect, though any effect may be masked by the inherent randomness of the results. Comparing the grid scans for the three different RLO policies trained with different random seeds in Figs. 4.13 and 4.14 shows that the behaviour is consistent across different policies.

An additional limitation can be the algorithms' reliance on correct accelerator calibration. For example, the investigated RLO policy is trained to set the strengths of the quadrupole magnets  $k$  and the angle of the steering magnets  $\alpha$ . This setup is natural to accelerator physics and straightforward to integrate with beam dynamics simulations. In the real world, the accelerator control system maps quadrupole strengths and steering angles to the correct current set points for the magnets' power supplies. This requires calibration to the beam energy. If that calibration is far off the correct beam energy, RLO may degrade in performance or even fail. For BO, accelerator calibration is mostly a non-issue, because in the case of a zero-mean prior as used in this study, BO makes no prior assumptions on the relationships of actuators to the target function. A false calibration to the beam energy would therefore be transparent to BO. In the case of beam energy calibration on magnets, this calibration is usually known well enough ahead of time for recurring working points, or the beam energy needs to be measured for other reasons anyway. However, in some other cases, where there is a need for regular recalibration, this might influence the selection of the tuning algorithm.

## 4.3 Zero-Shot Feedback for Drift and Fault Compensation

A key advantage of using RLO over other advanced optimisation methods is that RL is also well-suited to training feedback controllers that not only tune the accelerator to a desired state, but can also ensure that this state is maintained over time. This is particularly important for particle accelerators, which are often subject to drifts over time and can sometimes experience component failures. In both of these cases, it is desirable to have a feedback controller that can compensate for these changes and keep the accelerator running optimally.

In the following, it is demonstrated that the RLO policy trained in Section 4.1 is capable of running as a feedback controller under both drifts and fault conditions. Most impressively, the RLO policy is capable of doing this without any additional training or fine-tuning, even though such events were not part of its training procedure. Note that, while interesting as a proof of principle for the capabilities of RL-trained policies, the transverse beam tuning task at ARES is not one where a feedback controller would be deployed in practice, but rather a task performed during the initial setup of the accelerator. The results follow those originally presented in [161].

### 4.3.1 Running as a Feedback

Real-world plants may be subject to drifts over time caused by unmodelled external factors. Moreover, control can be regarded as the continuous optimisation of a dynamic objective function. Consequently, a tuning algorithm that can run as feedback on a dynamic objective function can be used for drift compensation and control in addition to tuning. Thus, a tuning

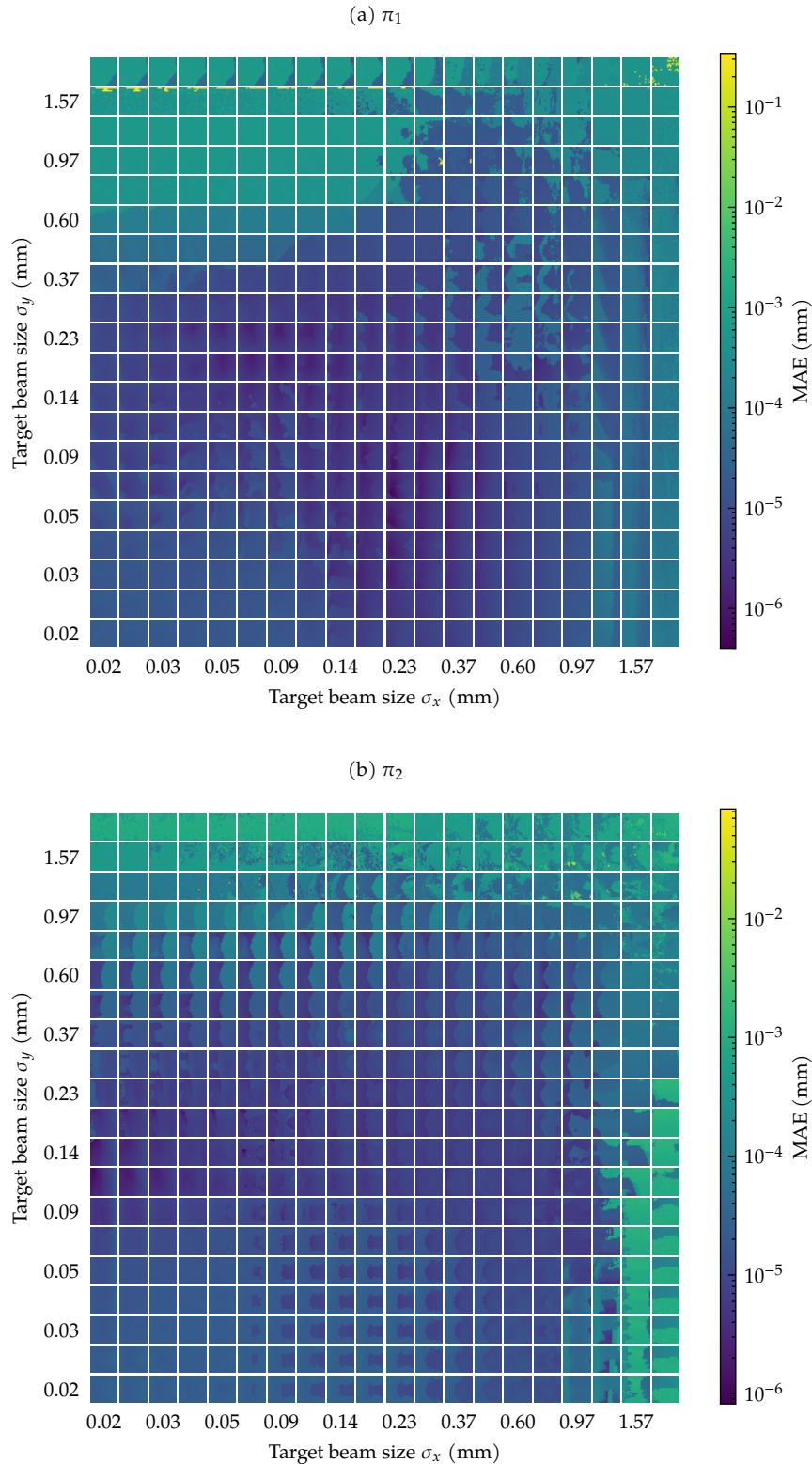


Figure 4.13: Grid scans over target beams using  $\pi_1$  and  $\pi_2$  [164]. The scans were performed over 160 000 different target beams with fixed incoming beam and misalignments. Beam positions in  $x$  and  $y$  are visualised within each box, with the diagnostic screen's centre in their centre and the beam position increasing linearly outwards. Beam sizes are visualised across boxes, starting with beam size  $20 \mu\text{m}$  in the bottom left and increasing logarithmically to the top right. The colour map is scaled logarithmically.

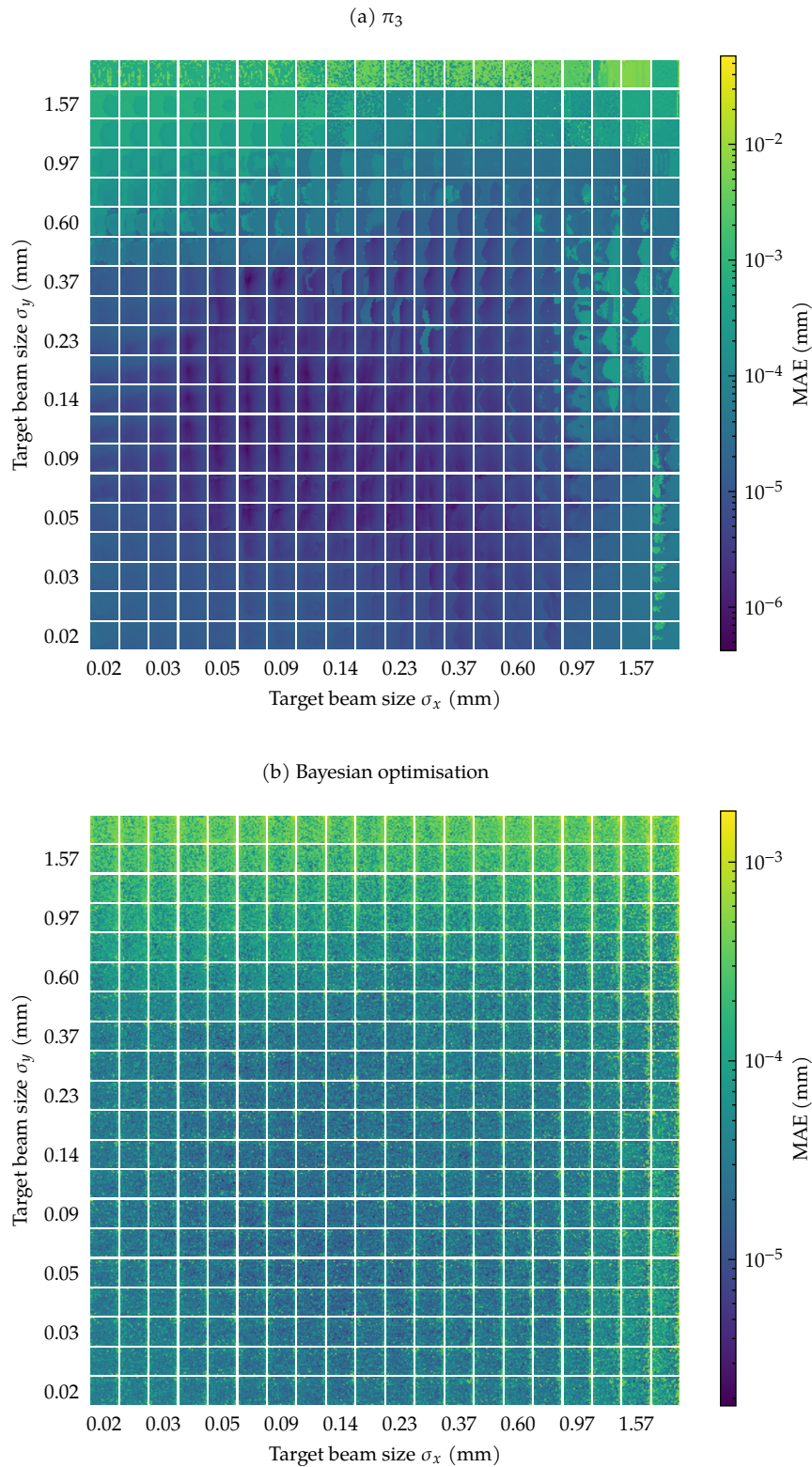


Figure 4.14: Grid scans over target beams using  $\pi_3$  and BO [164]. The scans were performed over 160 000 different target beams with fixed incoming beam and misalignments. Beam positions in  $x$  and  $y$  are visualised within each box, with the diagnostic screen's centre in their centre and the beam position increasing linearly outwards. Beam sizes are visualised across boxes, starting with beam size  $20 \mu\text{m}$  in the bottom left and increasing logarithmically to the top right. The colour map is scaled logarithmically. For BO, the error is taken after recovering the minimum beam, representing a typical tuning scenario.

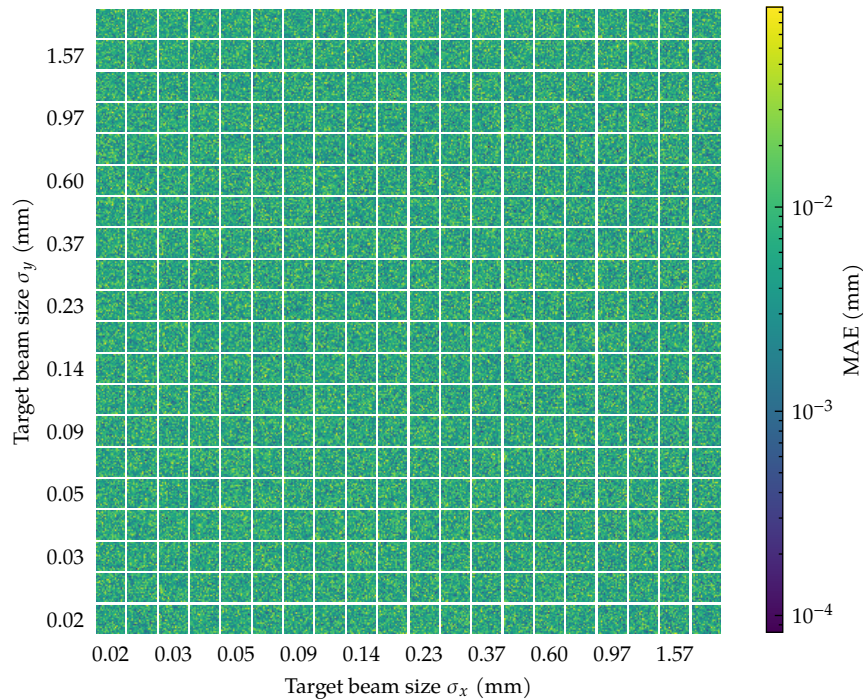


Figure 4.15: Grid scan over target beams using random search [164]. The scan was performed over 160 000 different target beams with fixed incoming beam and misalignments. Beam positions in  $x$  and  $y$  are visualised within each box, with the diagnostic screen’s centre in their centre and the beam position increasing linearly outwards. Beam sizes are visualised across boxes, starting with beam size  $20 \mu\text{m}$  in the bottom left and increasing logarithmically to the top right. The colour map is scaled logarithmically. Note that the error is taken after recovering the minimum beam, representing the typical use of random search in a typical tuning scenario.

method’s ability to operate as a feedback is an interesting subject of further investigation and could impact algorithm selection.

While BO assumes a static objective function, the benchmarked RLO policy does not rely on memorising previously seen objective values or could alternatively learn to adapt to dynamic objective functions during training. It should therefore be possible to use the policy from RLO as an RL-based feedback controller for a dynamic system. To test this, both algorithms were run for 80 steps. After 40 steps, the incoming beam is suddenly changed to a different beam from one step to the next. As a result, a different set of actuator settings is required to achieve the target beam on the diagnostic screen. It is then observed how the RL policy and the BO implementation react to the new incoming beam. If the method manages to recover the machine state, it can be considered capable of running as feedback. As can be seen in Table 4.5 and Fig. 4.16, the RL-based controller can in fact recover the target beam in about the same time it took to perform the original optimisation, with the final beam difference being comparable to the one achieved in optimisation with a static incoming beam. The beam achieved by BO when the beam instantly changes during the optimisation is, as expected, significantly worse than it is with a constant beam. After the incoming beam changed, the GP model based on the previous 40 samples is no longer correct, effectively breaking BO.

However, the system changes that feedbacks need to react to are not always fast. Often, they occur slowly over time, such that the controller must track the change in order to

Table 4.5: Results of feedback and actuator failure studies [161]

Optimiser	Normal	Feedback		Magnet failure	
		Instant	Continuously	Before	During
RLO	16	18	80	40	16
BO	73	350	250	45	50

The values reported are mean final beam differences as MAEs from the observed beam parameters to the target beam parameters in  $\mu\text{m}$ . All optimisations reported in this table were terminated at 80 steps. For comparison results from normal tuning runs (without upstream beam changes or magnet failures) terminated after 80 steps are also reported.

hold the system near the desired state after attaining it. Both algorithms are therefore also evaluated on a setup where the incoming beam changes linearly over the course of 80 steps. The results are listed in Table 4.5. As can be seen in Fig. 4.16, the RL policy is capable of tracking the target beam parameters after attaining them. The reasonably small increase in final MAE is attributed to the fact that the policy requires a few steps to converge on the desired beam parameters, but in the final step only a single step has passed since the last change to the incoming beam, therefore giving the policy only very little time to correct for the change. As with the instant incoming beam change, BO is not capable of tracking the desired beam parameters. Because the incoming beam cannot be included in the GP model, the learned surrogate is ill-defined, tracking a dynamically changing objective function with a static model. Consequently, BO optimises an objective function that diverges from the true objective function of the system.

It needs to be mentioned that the slow drifts of the underlying objective function, like the temperature drift of the magnets, can be tackled by adaptive BO with a spatio-temporal GP model [234] or contextual BO [180]. This would require, however, problem-specific implementation and additional engineering effort.

### 4.3.2 Robustness to Actuator Failure

In real-world plants, one also has to deal with the potential failure of components such as the actuators used for tuning. It would therefore be beneficial if a tuning algorithm could handle such an actuator failure and recover the previous state.

In the following, the RLO policy and the BO implementation are evaluated on their ability to handle two different types of actuator failures. First, the algorithms are tested on their ability to recover from a permanent actuator failure, where the actuator has failed some time before the tuning algorithm was started. Second, the algorithms are tested on their ability to recover from a delayed actuator failure, where the actuator is operational when the tuning starts but fails at a later time during the tuning. Specifically, the third quadrupole magnet is assumed to fail as a result of a defect of its power supply during transverse beam tuning in the EA. As a result of the failure, the quadrupole strength of the third quadrupole magnet is stuck at  $0\text{ m}^{-2}$  after the initial failure. It is assumed that the failed actuator provides a correct readback to the optimisation algorithm because readbacks are commonly provided by separate sensors.

Table 4.5 lists the results of this experiment. It is observed that RLO handles actuator failure well, despite never being trained to do so. With the magnet already failed at the start of the optimisation, RLO finds a set of actuator settings, excluding the failed magnet,

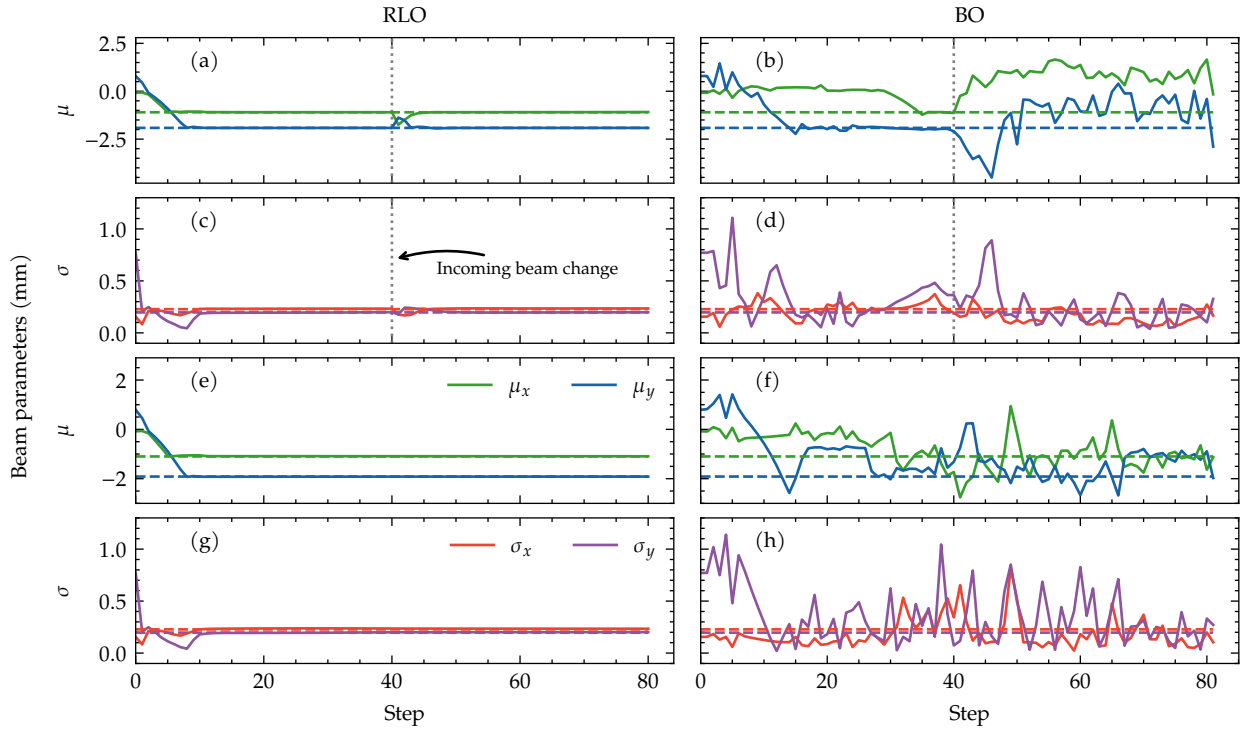


Figure 4.16: Beam parameters over time of RLO and BO running as feedbacks in simulation [161]. (a), (b), (c) and (d) show both algorithms reacting to an instant change of the incoming beam at step 40, denoted by the vertical dotted lines. (e), (f), (g) and (h) show the optimisers tracking the optimum with respect to a continuously changing incoming beam. The horizontal dashed lines denote the target beam parameters.

that is almost as good as the one it would have found without the magnet failure. Note that this shows RLO solving a task that is slightly different from the one it was trained on. With this task being outside the training distribution, it is not surprising that the performance is slightly worse than in the nominal case. However, it is also conceivable that the performance is simply worse because the failed magnet would have been needed to achieve better beam parameters. When the failure occurs during the optimisation, RLO is able to react quickly and recovers the beam parameters in a few steps. An example of RLO reacting to a sudden failure of  $Q_3$  during the optimisation is shown in Fig. 4.17. The performance of BO improves under actuator failure, when compared to the nominal case. This is because the failed magnet reduces the dimensions of the search space. Despite this, BO performs worse than RLO in the presence of actuator failure.

## 4.4 Transfer to Novel Lattices

With transverse tuning tasks being a common task at particle accelerators, it is important to understand how well RLO can be transferred to other lattices, where the basic task is the same, but the underlying lattice is different. Being able to transfer a training setup or trained policy to a new lattice can help the adoption of RLO in practice and enable larger intelligent autonomous accelerator setup schemes. For example, at ARES, there are three more sections similar to the EA, where similar transverse tuning tasks are performed. These sections are the BC, the DL and the SH, as introduced in Section 2.2.2. With trained policies available

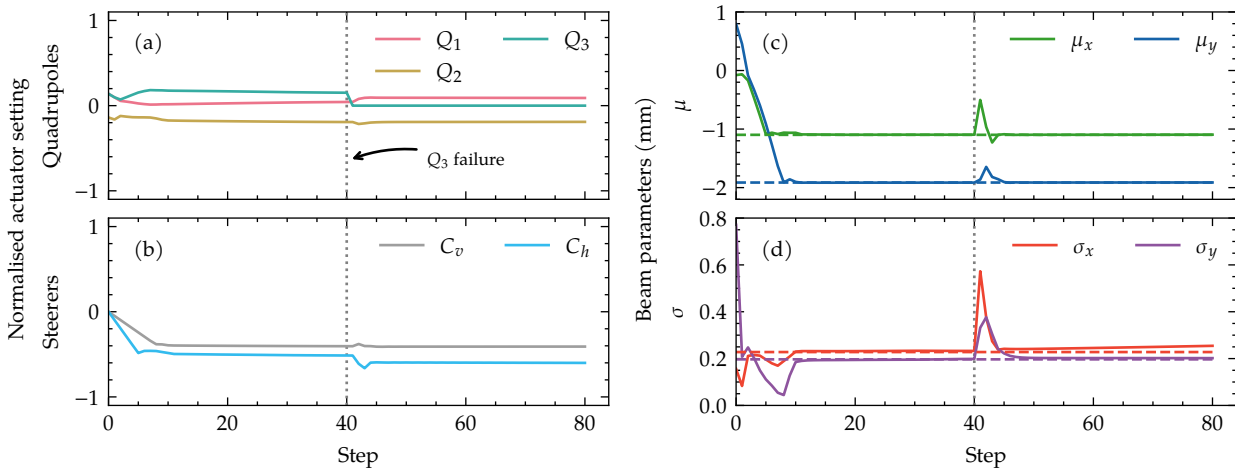


Figure 4.17: RLO reacting to a simulated actuator failure during the optimisation [161]. The third quadrupole magnet fails in step 40, denoted by the vertical dotted lines. In (a), the normalised quadrupole settings are shown, where, when  $Q_3$  fails, the policies increase the strength of  $Q_1$  to compensate. In (b), the normalised steerer settings are shown, where the horizontal steering magnet is used to counter the increased change in beam position as a result of the changing dipole moments induced by the quadrupole magnets. (c) and (d) show the beam positions and sizes, respectively. When  $Q_3$  fails, all beam parameters deviate suddenly from the target values, denoted by the dashed lines, but are quickly recovered by the policy.

for each of these sections, one could imagine implementing a sequential beam threading scheme, where the beam is threaded through the entire beamline by tuning the transverse beam parameters in each section one after the other along the beamline. More advanced tuning schemes are also imaginable, where federated reinforcement learning [121] is employed to train a supervisor policy that coordinates expert policies in each of the sections by deciding when to call them and which target beam parameters to tune towards.

In the following, two different schemes for transferring training setups or trained policies to new lattices are introduced. First, a revised version of the training setup from Section 4.1 developed for the EA is presented. It is shown how a single generalisable training setup can be used to train policies on multiple different tasks without additional engineering at the example of other sections at ARES, akin to the approach introduced in [223]. Second, it is demonstrated how domain randomisation can be used to train a lattice-agnostic policy once which can then be applied zero-shot to a distribution of lattices in Section 4.4.2.

#### 4.4.1 Retraining from Scratch

The most straightforward approach to transferring a training setup to a new lattice is to take the developed training setup, which is known to work for the given task on one lattice, and apply it to the new lattice, where the same or a very similar task is to be solved. This takes advantage of all the development work that has gone into solving the task on the original lattice, defining observations, actions, rewards, and finding well-performing hyperparameters, and only requires that the lattice considered in the environment is changed, and the policy is retrained anew from scratch. The latter, however, is also the most major disadvantage of this approach. While the development effort is mostly eliminated, training RL policies is a notoriously time-consuming process, and retraining a policy from

scratch for each new lattice can be a significant time investment. Nevertheless, this approach demonstrates the generality of a well-designed training setup, and saving the engineering effort is a notable advantage in practice.

### Experience-based Improvements to Algorithm and Reward

Before investigating the transfer to novel lattices through retraining, some improvements to the training setup developed in Section 4.1 are introduced. These are aimed at improving the stability of the training process, and take advantage of the insights gained from the training process of the original policies.

Firstly, the TD3 algorithm is replaced with the PPO algorithm. TD3 was originally chosen for its sample efficiency, to potentially enable training on the real accelerator, and because in initial studies the much slower simulation code Ocelot [3] was used. For the final setup as presented in Section 4.1, however, a training purely in simulation with a zero-shot transfer to the real accelerator was chosen. What is more, training in simulation was done with the specially developed high-speed simulation code Cheetah, introduced in Section 3.2. The significantly reduced cost of collecting samples, opens up the possibility of using the PPO algorithm, which is less sample-efficient, but has been shown to be more stable and robust in practice. In addition, the PPO itself is less computationally expensive than TD3, and allows for more parallelisation of the experience rollouts. Consequently, while PPO might require more samples of the environment to converge to a good policy, the required wall-clock time to train a policy is expected to be lower. In fact, training the PPO policy on the EA lattice took only 58 min on a laptop compared to 14 h on a HPC cluster node for the TD3 policy from Section 4.1.

Secondly, the reward function is modified. The original reward function from Section 4.1 has four potential disadvantages. For one, while the doubling of negative rewards in Eq. (4.4) encourages the policy move as directly as possible towards the target beam, ultimately helping the training process by discouraging the policy to move away from the target beam and recover later, it also makes it more likely that the policy gets stuck in local optima. In addition, while the doubling of negative rewards encourages always moving towards the target beam rather than away from it at any point in time, no component of the reward encourages the policy to move quickly towards the target beam. This is also a potential reason why unusually small discount factors were found to be required in the training process of the TD3 policy, as they are the only way to encourage the policy to move quickly towards the target beam with the differential reward defined in Eq. (4.5). Lastly, the logarithmic scaling of the objective in Eq. (4.6) is a potential source of issues in the training process and another reason why small discount factors were found to be necessary. The logarithmic scaling significantly increases the reward for small improvements of the beam parameters close to the target beam. While this can help ensure that the policy keeps converging further towards the target beam, when the differences to the target beam can span several orders of magnitude during an episode, it can also cause the policy to improve a beam parameter that is already close to the target beam at the expense of another beam parameter that is further away. This effect is further amplified by the fact that rewards are not limited to a certain range. The reward for one beam parameter can therefore be significantly larger than the reward for another beam parameter, making the reward for the beam parameter that is further away from the target nearly irrelevant in comparison. A last issue with the original reward function can be seen in the example run in Fig. 4.7, where the policy continues to change the magnet settings even after the beam parameters

have converged. While it has seemingly learned to do so while still keeping the beam parameters at the requested values, this behaviour is not desired in practice. The original reward does not penalise such behaviour.

To address these issues, a new reward function is developed based on the principle of *transforms* and *combiners* introduced in [59]. The new reward function is based on three objectives: the deviation of the measured beam parameters to the target beam parameters, the change in the magnet settings and whether the beam is visible on the diagnostic screen within the camera's field of view. The beam reward is now defined in terms of a negative objective (difference of the measured beam to the target beam) rather than a differential reward. A clipped linear transform

$$r_{b_i} = \begin{cases} 0 & \text{if } \frac{|b_i - b'_i|}{4 \text{ mm}} \geq 1 \\ 1 - \frac{|b_i - b'_i|}{4 \text{ mm}} & \text{otherwise} \end{cases} \quad (4.22)$$

is applied to each of the beam parameter differences to scale them from a range  $[0, \infty]$  to  $[1, 0]$ , mapping a worst-case error of 4 mm imposed by the field of view dimensions to a reward of 0 and a perfect beam to a reward of 1. This transform is also illustrated in Fig. 4.18. The rewards for each of the beam parameters are then combined into a single beam reward  $r_b$  by taking a uniformly weighted mean

$$r_b = \frac{1}{4} \sum_{i=1}^4 r_{b_i}. \quad (4.23)$$

The magnet change reward  $r_a$  is defined in terms of a sigmoid transform of each action component

$$r_{a_i} = \frac{1}{1 + \exp(-\tilde{a}_{t,i})}, \quad (4.24)$$

where  $\tilde{a}_{t,i}$  is the normalised action component scaled to a range  $[-\ln(19), \ln(19)]$  by

$$\tilde{a}_{t,i} = -\ln(19) - \left(1 - \frac{|a_i|}{\hat{a}_i}\right) (-2 \ln(19)) \quad (4.25)$$

with  $\hat{a}_i$  being the maximum action, which is assumed to be  $30 \text{ m}^{-2}$  for the quadrupole magnets and  $6.1782 \text{ mrad}$  for the steering magnets. An illustration of this transform is also shown in Fig. 4.18. This transform scales the magnet change reward components into a range  $[0, 1]$ . They are then combined through a uniformly weighted mean

$$r_a = \frac{1}{|a_t|} \sum_{i=1}^{|a_t|} r_{a_i}, \quad (4.26)$$

where  $|a_t|$  is the number of elements in the action, i.e. the number of magnets controlled by the policy. The visibility reward  $r_v$  is defined as

$$r_v = \begin{cases} 1 & \text{if the beam is visible on the diagnostic screen} \\ 0 & \text{otherwise.} \end{cases} \quad (4.27)$$

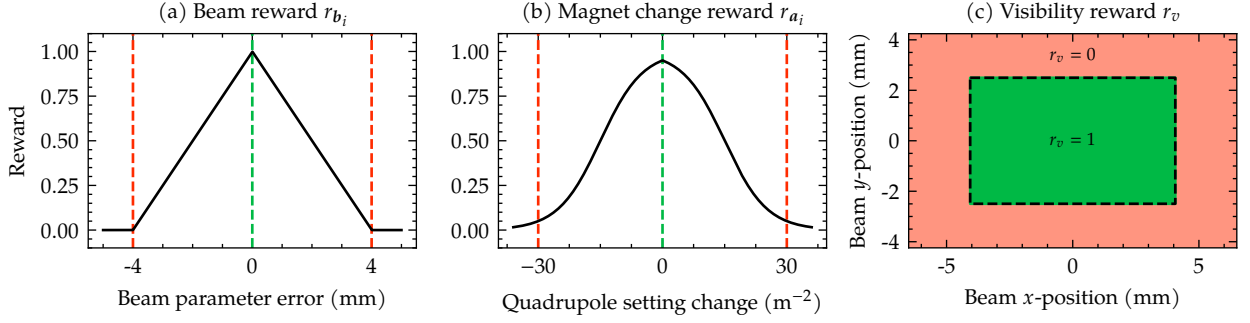


Figure 4.18: Illustration of the transforms applied to the different reward components. (a) shows the clipped linear transform applied to the difference of the measured beam parameters to the target beam parameters in order to map the error to a reward in the range  $[0, 1]$ . (b) shows the sigmoid transform applied to the change in the setting of a quadrupole magnet in order to map the change to a reward in the range  $[0, 1]$ . Note that the settings mapped for a steerer are different. (c) shows the visibility reward over the 2-dimensional beam position. The black dashed line outlines the visible area of the screen. Green areas and green dashed lines in all subplots indicate a good result that should map to a reward near 1, while red areas and red dashed lines indicate a bad result that should map to a reward near 0.

In the Cheetah model, the beam is considered to be visible as long as the beam position  $(\mu_x, \mu_y)$  is within the field of view of the camera. Figure 4.18 shows a visualisation of this reward component. The final reward is then computed as the weighted mean

$$r_t = \frac{w_b r_b + w_a r_a + w_v r_v}{w_b + w_a + w_v}, \quad (4.28)$$

where  $w_b = 3$ ,  $w_a = 0.5$  and  $w_v = 0.5$  are the weights for the different reward components. These reward transforms and combiners were chosen based on experiments from a set of possible transforms and combiners devised in [59] and found to perform well. Similarly, the reward weights were chosen based on experiments.

The FDF magnet initialisation is mapped to novel lattices by applying focus and defocus in turns to the quadrupole magnets. That is, the initial magnet settings for the EA remain

$$\mathbf{u}_0 = \left( 10 \text{ m}^{-2}, -10 \text{ m}^{-2}, 0 \text{ mrad}, 10 \text{ m}^{-2}, 0 \text{ mrad} \right), \quad (4.29)$$

while the initial magnet settings for the SH, for example, are

$$\mathbf{u}_0 = \left( 0 \text{ mrad}, 10 \text{ m}^{-2}, 0 \text{ mrad}, -10 \text{ m}^{-2} \right). \quad (4.30)$$

## Evaluation

The revised training setup is evaluated on the EA lattice using the same setup introduced in Section 4.2.2. In addition, policies trained on the BC, DL and SH sections are evaluated and compared to the EA policy. Their shared hyperparameters are listed in Table 4.6. For comparison, the policy trained on the EA lattice in Section 4.1 is evaluated on all four lattices. The results are listed in Table 4.7. In order to better understand how fast and to how good of a solution the policies converge, the mean beam error over time of the policies

Table 4.6: Hyperparameters used to train the PPO agents

Parameter	Value
Batch size $ \mathcal{B} $	128
Learning rate (actor & critic)	Linearly decaying from 0.0003
Discount factor $\gamma$	0.99
Number of concurrent environments	40
Number of rollout steps	128
Entropy coefficient $c_2$	0.01
Value function coefficient $c_1$	0.5
Number of training epochs	10
Clip range $\epsilon$	0.2
Layer widths	(64, 64)
Training steps	5 000 000
Maximum steps per episode	50
Initial log standard deviation	-2.3

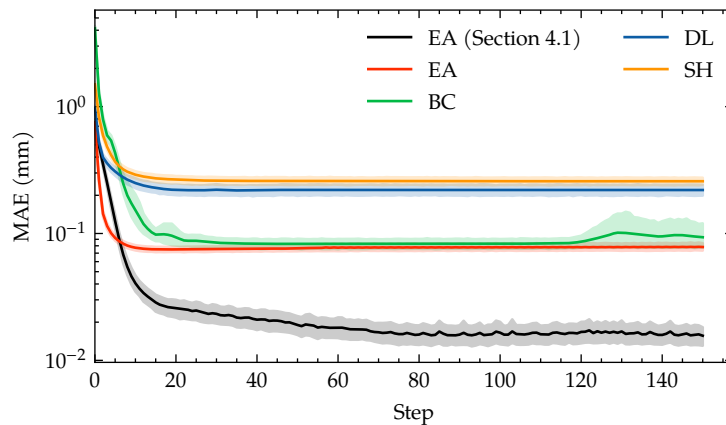


Figure 4.19: MAE between observed and target beam over time for different policies on different lattices. The MAEs of the beam parameters to the target beam are averaged over all trials. The envelopes show the 95 % confidence intervals of the beam differences averaged over all trials. Note the logarithmic scaling of the y-axis.

Table 4.7: Performance of retrained and lattice-agnostic policies on different lattices

Lattice	Final error ( $\mu\text{m}$ )		RMSE ( $\mu\text{m}$ )		Improvement (%)		Steps to target		
	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Success rate
<b>Section 4.1</b>									
<i>Policies trained from scratch</i>									
EA	7	16 $\pm$ 24	124	129 $\pm$ 61	99	97 $\pm$ 4	7	12 $\pm$ 16	92%
BC	6870	8743210 $\pm$ 18999030	10281	5494497 $\pm$ 11148520	-138	-223276 $\pm$ 612547	-	-	0%
DL	225	35490 $\pm$ 302941	317	20037 $\pm$ 151163	61	-4599 $\pm$ 33370	3	3 $\pm$ 1	1%
SH	72835	91616 $\pm$ 104449	83955	103734 $\pm$ 112744	-9926	-15541 $\pm$ 22639	36	37 $\pm$ 10	1%
<i>Geometry-randomised policy</i>									
EA	71	78 $\pm$ 46	127	136 $\pm$ 53	90	87 $\pm$ 11	6	10 $\pm$ 8	17%
BC	63	93 $\pm$ 192	510	523 $\pm$ 225	98	97 $\pm$ 4	11	13 $\pm$ 6	28%
DL	162	220 $\pm$ 189	301	377 $\pm$ 274	83	77 $\pm$ 18	7	9 $\pm$ 7	14%
SH	228	258 $\pm$ 155	396	458 $\pm$ 223	84	80 $\pm$ 16	9	11 $\pm$ 7	1%
<i>Order-randomised policy</i>									
Sample <sup>1</sup>	85	93 $\pm$ 42	148	157 $\pm$ 58	88	86 $\pm$ 9	6	9 $\pm$ 10	16%
EA	85	91 $\pm$ 44	149	157 $\pm$ 56	88	85 $\pm$ 10	8	14 $\pm$ 21	18%
BC	35228	61639 $\pm$ 69674	47385	73082 $\pm$ 78302	-722	-1380 $\pm$ 1609	-	-	0%
DL	1007	1031 $\pm$ 193	1694	1673 $\pm$ 190	-52	-89 $\pm$ 158	-	-	0%
SH	12592	14738 $\pm$ 10315	16453	20285 $\pm$ 14399	-851	-1178 $\pm$ 1128	-	-	0%
Sample <sup>1</sup>	91	99 $\pm$ 45	174	181 $\pm$ 56	88	85 $\pm$ 10	11	14 $\pm$ 7	8%
EA	85	92 $\pm$ 37	164	166 $\pm$ 48	87	84 $\pm$ 12	9	11 $\pm$ 4	7%
BC simplified <sup>2</sup>	1774	49948 $\pm$ 84802	2706	60891 $\pm$ 103232	45	-989 $\pm$ 1794	-	-	0%
BC	2104	56765 $\pm$ 91922	3039	68023 $\pm$ 109672	41	-1046 $\pm$ 1849	6	6 $\pm$ 0	0.3%
DL	98	104 $\pm$ 40	153	156 $\pm$ 48	85	81 $\pm$ 14	4	4 $\pm$ 1	5%
SH simplified <sup>2</sup>	407	494 $\pm$ 380	631	742 $\pm$ 476	67	59 $\pm$ 30	26	32 $\pm$ 21	2%
SH	7402	10596 $\pm$ 6168	10690	14355 $\pm$ 7973	-569	-776 $\pm$ 739	-	-	0%

<sup>1</sup>An arbitrary lattice sample from the geometry-randomised training distribution.<sup>2</sup>Simplified variants of the BC and SH lattices without the apertures and bends present in the full lattices.

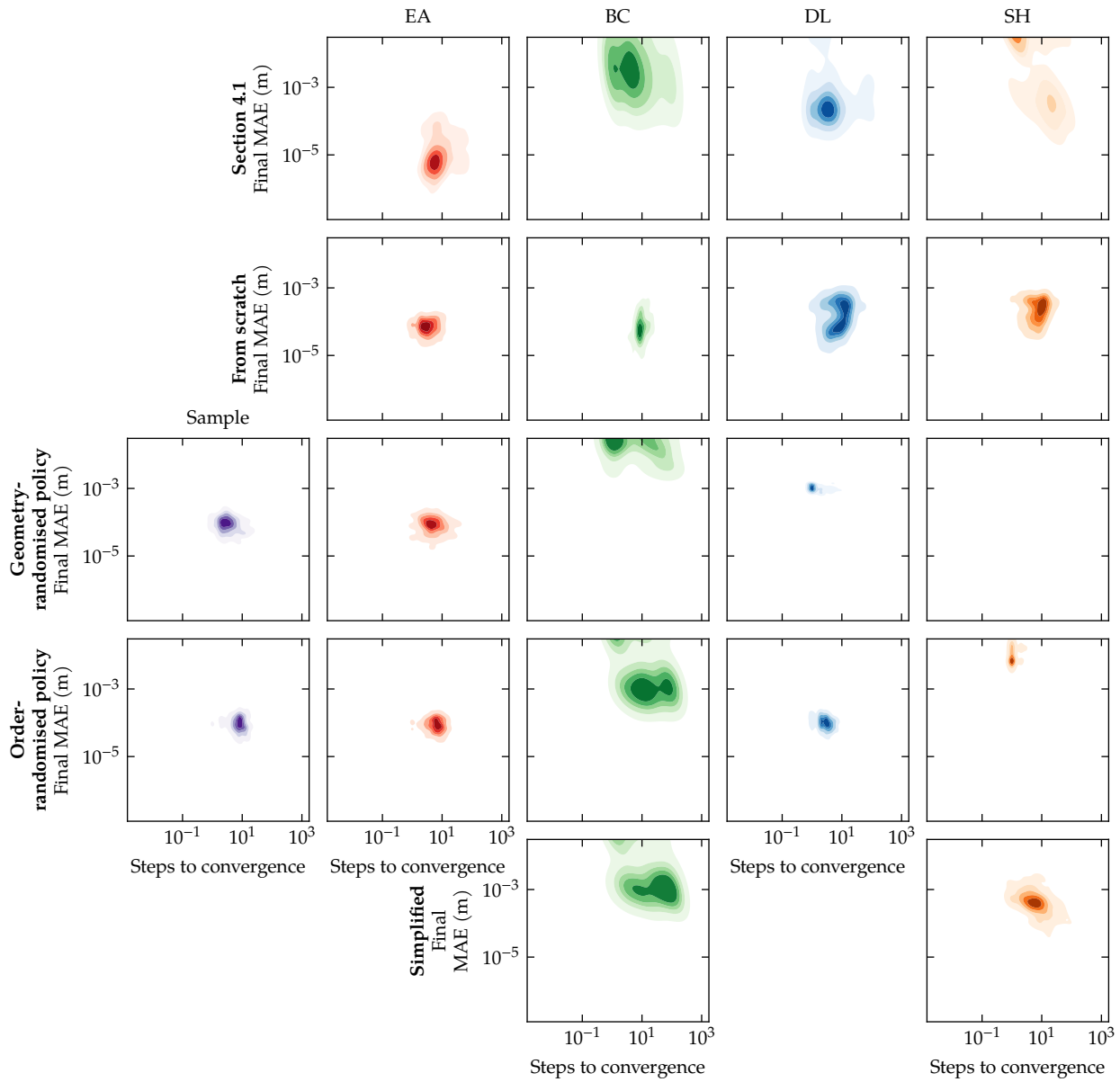


Figure 4.20: Distributions of the final MAEs achieved by specialist and lattice-agnostic policies on different lattices during optimisations plotted over the steps to convergence. KDEs are used to visualise the distributions. Ideally, distributions should be concentrated in the lower left, indicating that policies converge quickly to good solutions. Distributions in the upper left converge quickly but to bad solutions, while distributions in the lower right converge slowly to good solutions. Note the logarithmic scaling of the axes. In the first row,  $\pi_1$  from Section 4.1 trained on the EA is applied to the different lattices. The bottom row pertains to simplified less-out-of-sample variants of the BC and SH lattices, where apertures and the bends are removed. The steps to convergence are measured on the minimum beam to guarantee that all runs converge and are included in the distribution. An equivalent figure with the final MAE is available in Appendix B.

trained on the different lattices is shown in Fig. 4.19. What is more, Fig. 4.20 explores the relation between the speed of the optimisation and the final result.

As expected, the policy from Section 4.1 does not perform well on most other lattices. Interestingly, however, it still performs reasonably well on the DL lattice. The latter differs from the EA lattice in the order of the magnets and in that it has only two quadrupole magnets rather than three. To interface with the policy from Section 4.1, the third quadrupole magnet was always observed to be at  $0 \text{ m}^{-2}$  and the policies outputs for it were ignored. It was already found in Section 4.3.2 that the policy is capable of handling the failure of a quadrupole magnet during the optimisation, and it appears that it can also handle its absence for the entire optimisation. Moreover, the policy seems to be somewhat robust to changes in the magnet positions and order, with only a limited degradation in performance, despite this not being part of its training. The BC and SH lattices differ more significantly from the EA lattice, despite the BC having the same number of quadrupole magnets and the most similar magnet order. In both cases there are dipole magnets and apertures in the beam pipe that affect the beam in ways that the policy from Section 4.1 would have not experienced during training. In the BC the distance between the first two quadrupole magnets is also exceedingly long, which increases the leverage of the first quadrupole and could potentially lead to the policy overusing it.

The policies trained from scratch directly on each of the respective lattices perform, as is to be expected, significantly better. Despite the training setup being originally tailored to the EA lattice, the BC policy trained from scratch even outperforms the policy trained with the same revised setup on the EA lattice. The policies trained from scratch on the DL and SH lattices also perform well, though notably worse than the EA and BC policies. While this could be due to parts of the training setup being less well-suited to these lattices and therefore not generalising well, it is more likely that this indicates that the task is more difficult to solve on these lattices. Note that the component of the training setup most likely to be ill-suited to a novel lattice are the factors used to normalise the observations and actions, specifically those pertaining to the magnet settings. The effect of a magnet depends significantly on the distance of that magnet from the diagnostic screen station as that distance gives it leverage. Hence, the normalisation of the magnet settings should be based on the distance of the magnet from the diagnostic screen station. Despite this, the BC policy trained from scratch with the normalisation tuned to the EA lattice performs well on the BC lattice, which is arguably the most different of the four. This indicates that the training setup generalises well to the other lattices. A key difference between the EA and the BC lattices, where the specialised policies perform their best, and the DL and SH lattices, where the policies perform notably worse, is the placement of at least two quadrupole magnets upstream of any steerers. Having a steerer upstream of a quadrupole magnet means that the dipole moment induced by traversing the quadrupole off-centre must be accounted for every time the quadrupole or steerer are changed. In addition, focussing is theoretically possible with only two quadrupole magnets, such that, if a third one is placed downstream of a steerer, one can simplify the task by not using the third quadrupole magnet. This behaviour can in fact be observed with both RLO policies and real human operators. However, when less than two quadrupole magnets are positioned upstream of all steerers, one cannot take advantage of this simplification, and the task becomes more difficult. This is the case for the DL and SH lattices, where the policies perform worse, demonstrating that the cause of the worse performance is likely the tasks' increased difficulty rather than the training setup not generalising well.

Compared to the training setup from Section 4.1, the revised training setup performs

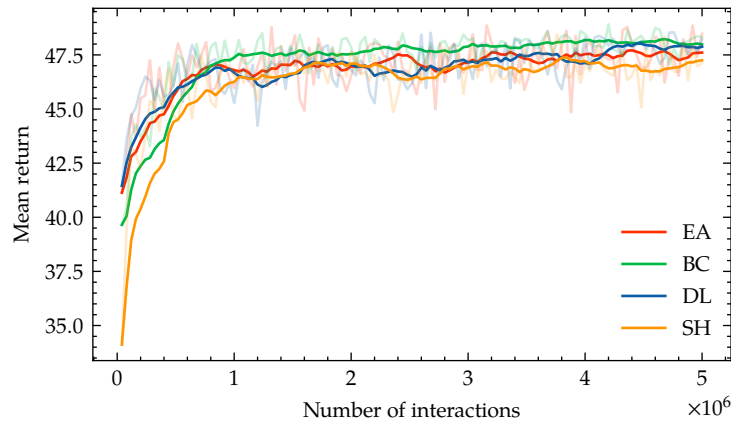


Figure 4.21: Training curves of the return from the training of policies on each ARES section. Solid lines show the mean return over 100 evaluation episodes, and the shaded curves show the actual return of each evaluation episode.

comparably in terms of all but one metric. The final error is found to be about an order of magnitude worse, though still in a range that would be considered good for the task. This is likely the result of the revised reward function being more conservative in its reward assignment. Where the logarithmic definition in Section 4.1 would have encouraged decreasing the error to the target beam ever further, for example by entering a different objective valley, improvements of ever smaller errors eventually become numerically irrelevant in the revised reward function. On the other hand, the revised reward function is found to be profoundly more stable, being much more robust to changes in hyperparameters, and solving among others the issue of forced small discount factors to compensate for exploding rewards and the lack of temporal encouragement. Hence, while the revised reward definition is overall an improvement over the one from Section 4.1, the latter retains the advantage of encouraging better final beams. Outside of policy performance, the revised training setup is found to allow for much faster training in terms of wall-clock time by taking advantage of concurrent rollouts enabled by the PPO algorithm. The training time is reduced from 14 h to under 1 h. Furthermore, thanks to the revised reward function, the meaning of the return is interpretable. For example, because training episodes are 50 steps long, the optimal return is 50, and how close to the optimum the policy is can be assessed by how close to 50 the return gets, with all policies achieving evaluation returns between 46 to 48. The returns averaged over 100 evaluation episodes throughout the training are plotted in Fig. 4.21.

#### 4.4.2 Domain Randomisation for Lattice-Agnostic Policies

Even though the classic retraining approach is straightforward and can be successful, it is also time-consuming and requires a lot of computational resources. It would be much more desirable to train a policy once, which can then be applied to solve the transverse tuning task on a variety of different lattices. In Section 4.1, domain randomisation [239, 323] was used to train a policy that could generalise to different states of the parameters of the real lattice that are not exactly known during training, such as the incoming beam, and the misalignments of magnets and diagnostic screens. In [360], this approach is extended to train policies that generalise to different lattices, as long as the basic task of tuning the transverse beam parameters is the same, and follows the same definitions of observations

and actions, effectively making the policies *lattice-agnostic*. Here, two variants of the lattice-agnostic training setup are introduced and explored for their ability to generalise to the BC, DL, SH and EA sections at ARES.

## Method

The approach is based on the method introduced in Section 4.1, using the exact training setup, including reward definition and hyperparameters from Section 4.4.1. Unlike in the original approach, the policies are trained on a lattice where the magnets and diagnostic screen are arranged as

$$(Q_1, Q_2, Q_3, C_v, C_h, S). \quad (4.31)$$

This is a more common arrangement than the one found in the EA and is a reasonable middle ground from which to generalise to other lattices. Consequently, the actuator and action vectors become

$$\mathbf{u}_t = (k_{Q_1}, k_{Q_2}, k_{Q_3}, \alpha_{C_v}, \alpha_{C_h}) \quad (4.32)$$

and

$$\mathbf{a}_t = (\Delta k_{Q_1}, \Delta k_{Q_2}, \Delta k_{Q_3}, \Delta \alpha_{C_v}, \Delta \alpha_{C_h}), \quad (4.33)$$

respectively.

To make the policies lattice-agnostic, domain randomisation is applied to the lattice in addition to the incoming beam, misalignments and target beam parameters. At the beginning of each training episode, a new lattice configuration is randomly sampled. This forces the algorithm to learn a policy that is robust to variations in the lattice, effectively making it lattice-agnostic. Specifically, two lattice randomisation schemes are explored: *geometry randomisation* and *order randomisation*.

In the geometry randomisation scheme, the lengths of the drift sections in the lattice are randomly sampled from a uniform distribution. That is, the lattice is considered as

$$(D_1, Q_1, D_2, Q_2, D_3, Q_3, D_4, C_v, D_5, C_h, D_6, S), \quad (4.34)$$

with  $D_i$  being a drift section. The length of each drift section  $l_{D_i}$  is randomly sampled from a uniform distribution over the interval [0.1 m, 0.5 m].

In the order randomisation scheme, the geometry randomisation is extended by randomly permuting the order of magnets in the lattice. This captures, for example, the EA section, which does not follow the lattice order of the geometry randomisation scheme. With respect to the latter, the EA is out-of-distribution, as the first steerer is placed between  $Q_2$  and  $Q_3$ . Note that the order of the magnets is only permuted in the lattice, not in the actuator and action vectors. These remain defined according to Eqs. (4.32) and (4.33), regardless of the order of the magnets in the lattice.

## Results

The performance of the lattice-agnostic policies trained following the two domain randomisation schemes is evaluated on the BC, DL, SH and EA sections at ARES, as well as a synthetic lattice sample taken from the geometry-randomised training distribution with

$$(l_{D_1}, l_{D_2}, l_{D_3}, l_{D_4}, l_{D_5}, l_{D_6}) = (0.2 \text{ m}, 0.4 \text{ m}, 0.4 \text{ m}, 0.2 \text{ m}, 0.2 \text{ m}, 0.4 \text{ m}). \quad (4.35)$$

Note that the DL and SH sections only have two rather than three quadrupole magnets. For these sections, the third quadrupole magnet is set to a fixed value of  $0 \text{ m}^{-2}$  in the actuator

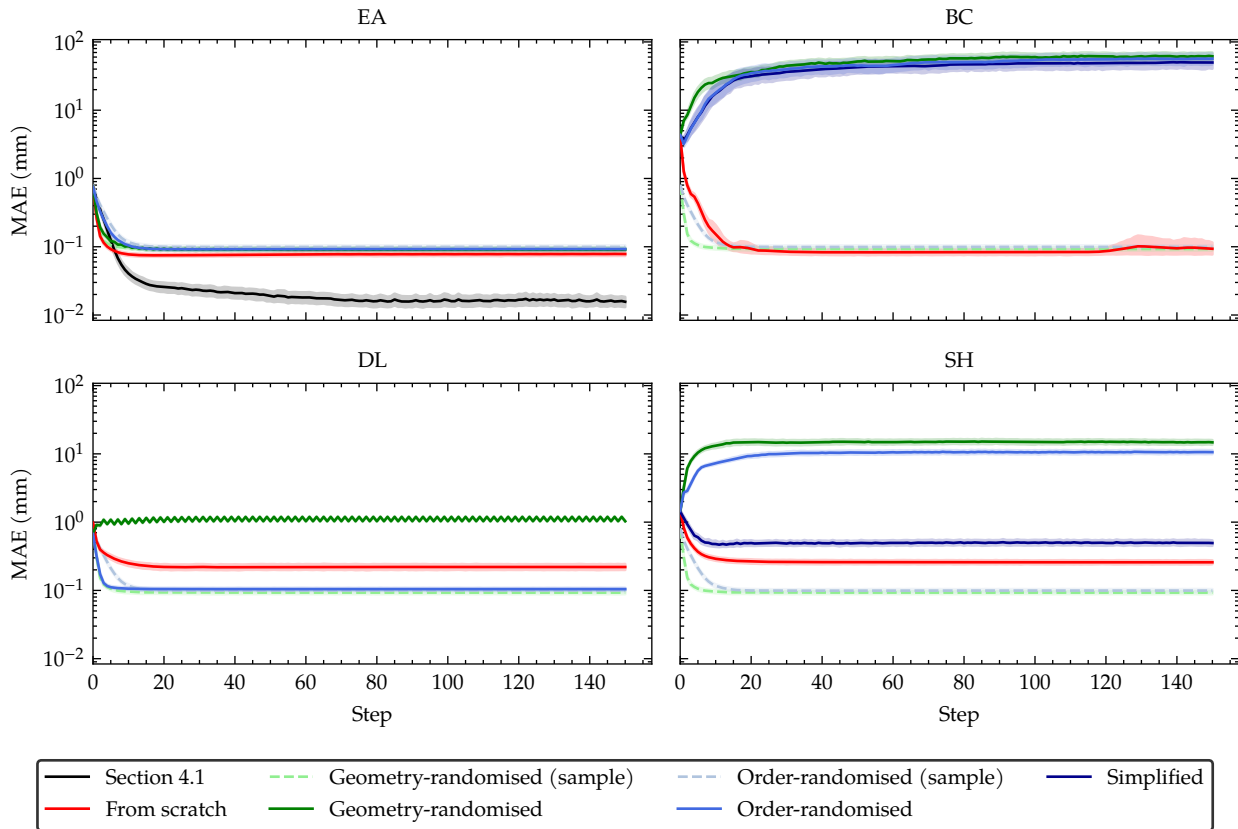


Figure 4.22: Beam errors in MAE over time and averaged over all trials for lattice-agnostic policies on different lattices. The envelopes show the 95 % confidence intervals. Results from the geometry and order-randomised policies are shown in shades of green and blue, respectively. Results from the order-randomised policy are shown in shades of blue. For reference, directly trained expert policies are shown in red, the EA policy from Section 4.1 is shown in black, and results from an in-distribution synthetic lattice sample are shown with dashed lines in each subplot for both the geometry-randomised and order-randomised policies in the lightest shade of the respective colour. Note the logarithmic scaling of the beam error axes.

vector. The results of the evaluation are listed in Table 4.7. MAE convergence over time is shown in Fig. 4.22. The relation between the speed of the optimisation and the final result is explored in Fig. 4.20.

It can be seen that the lattice-agnostic policies perform well on lattices that are in-distribution with respect to the training. Performance on the synthetic lattice sample from the training distribution in the geometry randomisation case, and on both the synthetic lattice samples and the EA in the order-randomised case is comparable to the performance of the policies trained directly on fixed lattices. Crucially, graduating from geometry to order randomisation does not appear to have a negative impact on the performance beyond the standard deviation, indicating that the policy model has the capacity to learn more complex tasks from larger domain randomisation distributions.

More interestingly, the lattice-agnostic policies also perform well on lattice configurations that are out-of-distribution with respect to the training distribution. The EA, for example is out-of-distribution for the geometry-randomised policies, as the first steerer is placed between the second and third quadrupole magnets. Nevertheless, the geometry-

randomised policy achieves similar, even slightly better, performance on the EA than on the in-distribution synthetic lattice sample, suggesting that the policy is to some extent robust to changes in the lattice order despite not having been trained on such changes, aligning with the findings from Section 4.4.1. Similarly, the performance of the order-randomised policy on the DL section is comparable to its performance on the in-distribution synthetic lattice sample and the EA section. The DL is out-of-distribution in two respects: the distance between the steerers  $C_v$  and  $C_h$  is at 0.69 m longer than the maximum distance of 0.5 m in the training distribution, and there are only two quadrupole magnets instead of three. This suggests robustness to the distances between lattice elements and to missing elements in the lattice. In fact, the robustness to a missing quadrupole is in line with the results from Section 4.3.2, where it was found that the policy can still perform well even when an actuator fails during operation, despite not having been trained on such failures. Remarkably, the performance of the order-randomised policy on the DL section is more than a standard deviation better than the performance of the specialised policy trained from scratch. It can be concluded that the domain randomisation over the lattice order has made the policy generally more capable at solving tasks with different lattice arrangements.

However, there are also limitations to the ability of the lattice-agnostic policies to generalise to out-of-distribution lattices. The geometry-randomised policy is found to not perform well on the BC, DL and SH lattices. While the performance on the out-of-distribution EA lattice, which violates the training distribution in terms of lattice order, is good, the other three lattices presumably lie too far outside the training distribution. They additionally violate the latter in terms of maximum element distance and, in the case of the DL and SH sections, the number of quadrupole magnets, as well as the presence of additional elements like bends and apertures in the beam path. Including some of these variations in the domain randomisation during training is found to significantly improve the performance on these sections. This can be seen in the presented experiments at the example of adding order randomisation, which improves the performance on the BC, DL and SH sections. Because only one of these aspects is addressed in the order-randomised policy, it is not surprising that, while much improved, the performance on the BC and SH lattices is still notably worse than on the synthetic sample and the now in-distribution EA section. In particular, it can be seen that the presence of the apertures and bending magnets in the BC and SH sections has a significant impact on the performance of the lattice-agnostic policies. While the performance on the BC section example is generally bad and not much of a difference can be seen between the full lattice, and simplified version without bends and apertures, in the case of the SH section, the performance difference is very pronounced, going from the ability to improve the beam on the simplified lattice to a significant degradation of the beam on the full lattice. This points to the importance of including the relevant lattice elements in the domain randomisation during training, especially when their settings are either unknown or expected to change during operation.

In order to better understand how robust the lattice-agnostic policies are to out-of-distribution lattices, one may look at the correlation between the performance of the policies on different lattices and metrics of how out-of-distribution the lattices are. In Fig. 4.23, three such correlations are shown.

The first metric is defined as the number of factors with respect to which the training distribution is violated. Up to four such factors occur with the sections at ARES: the lengths between elements may be outside the training interval; the order of the magnets may be permuted; the number of quadrupole magnets may be different; and other elements may be present on the drift sections. Note that a permutation of the order is only considered a

violation with the geometry-randomised policy, while it is in-distribution with the order-randomised policy. An overall trend of increasing RMSE with the number of violations can be observed, confirming that the more out-of-distribution a lattice is, the worse the performance of the lattice-agnostic policies.

Similarly, a metric can be defined to measure the out-of-distribution-ness of a lattice in terms of only the distances between the elements in the lattice, giving an indication of how well policies generalise to different lattice geometries. The *geometry distance* is defined as the sum of the absolute differences between the distances in the lattice and the training distribution

$$\mathcal{D}_{\text{geometry}} = \sum_{i=1}^n \min \left( \left| l_{D_i} - l_{D_i}^{\min} \right|, \left| l_{D_i} - l_{D_i}^{\max} \right| \right), \quad (4.36)$$

where  $l_{D_i}^{\min} = 0.1$  m and  $l_{D_i}^{\max} = 0.5$  m are the minimum and maximum drift lengths in the training distribution. Figure 4.23 shows that the median RMSE increases with the geometry distance, meaning that the lattice distances have a significant impact on the out-of-distribution performance of the lattice-agnostic policies. What is more, the gradient of this relationship is larger for the geometry-randomised policy than for the order-randomised policy, suggesting that training on domain-randomised lattice orders has made the latter more robust to changes in the lattice geometry as well.

Finally, the out-of-distribution-ness of a lattice can also be measured as a difference between the element order in the test lattice and the training distribution. For the order-randomised policy this is always zero, as all lattice orders are part of the training distribution. For the geometry-randomised policy, a metric to measure the distance between to lattice orders can be defined in terms of the number of edits required to transform the test lattice sequence into the training sequence as the Levenshtein distance [188] between two sequences  $a$  and  $b$

$$\mathcal{D}_{\text{Levenshtein}}(a, b) = \begin{cases} |a| & \text{if } |b| = 0 \\ |b| & \text{if } |a| = 0 \\ \mathcal{D}_{\text{Levenshtein}}(\text{tail}(a), \text{tail}(b)) & \text{if head}(a) = \text{head}(b) \\ 1 + \min \begin{cases} \mathcal{D}_{\text{Levenshtein}}(\text{tail}(a), b) \\ \mathcal{D}_{\text{Levenshtein}}(a, \text{tail}(b)) \\ \mathcal{D}_{\text{Levenshtein}}(\text{tail}(a), \text{tail}(b)) \end{cases} & \text{otherwise,} \end{cases} \quad (4.37)$$

where  $\text{head}(a)$  is the first element of the sequence  $a$ ,  $\text{tail}(a)$  is the sequence  $a$  without its first element and  $|a|$  is the length of the sequence  $a$ . The correlation of the median RMSE with the Levenshtein distance in Fig. 4.23 is not as strong as with the number of violations or the geometry distance, but can still be observed, especially between the same lattices, which increase in Levenshtein distance when going from an order-randomised policy to a geometry-randomised policy. The outlier of BC is likely the result of the BC section being so far out-of-distribution in terms of lattice geometry.

The above results successfully demonstrate that the domain randomisation approach can be extended to train lattice-agnostic policies that generalise to different lattices while maintaining overall performance, reducing the need for time-consuming and computationally expensive retraining of policies for each new lattice configuration. Most interestingly, these results show that such policies can, to some extent, generalise beyond the training distribution. What is more, the lattice-agnostic policies, can be used as a starting point for further fine-tuning on a specific lattice with very little additional training overhead, as was

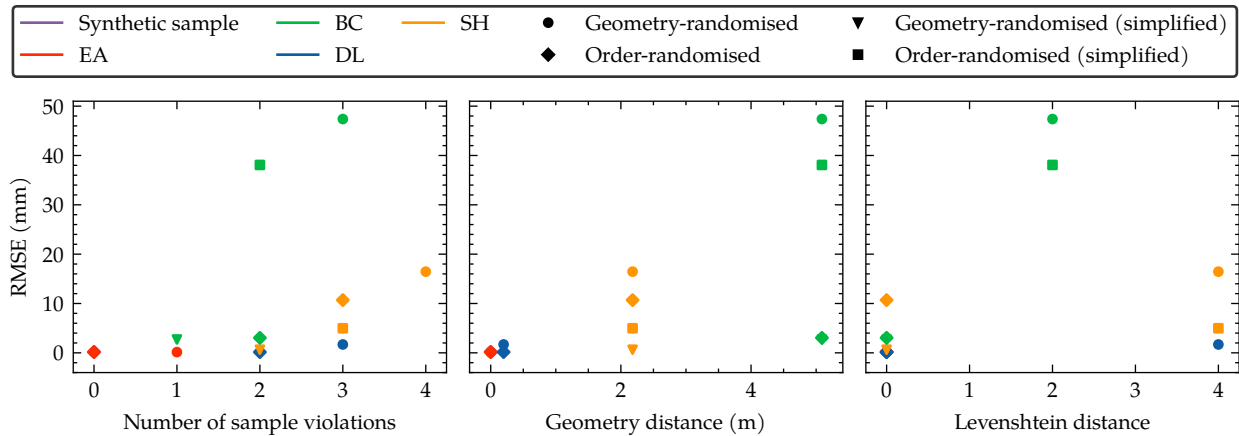


Figure 4.23: Median RMSEs achieved by the two different lattice-agnostic policies on different lattices over different measures of out-of-distribution-ness: the number of distribution violations, the distance of the lattice geometry to the training drift length intervals and the Levenshtein distance as a measure of magnet order difference. Different colours represent different lattices, circles represent the geometry-randomised policy and diamonds represent the order-randomised policy, with triangles and squares representing the same policies applied to simplified lattices without bends and apertures.

demonstrated in [360] using only 2 % of the original training samples. Such lattice-agnostic training followed by lattice-specific fine-tuning closely resembles meta reinforcement learning, which is explored in [133], where the Model-Agnostic Meta-Learning (MAML) algorithm [93, 293] is used to train policies that can quickly adapt to novel working points of the AWAKE facility at CERN.

## 4.5 Natural Language-Driven Particle Accelerator Tuning

While it has been demonstrated in the previous sections that RLO is a very capable solution for particle accelerator tuning, a disadvantage of this method is the effort in engineering and training as well as the RL expertise required to implement it. Numerical optimisation methods like BO are somewhat easier to implement, but this often comes at the cost of performance. What is more, their implementation does not entirely eliminate the need to make informed choices on objective formulation, algorithm selection and hyperparameters. As a result they too require a certain degree of engineering and that practitioners be experts in two separate domains – accelerator physics and optimisation. These challenges have so far slowed the adoption of advanced autonomous tuning algorithms to day-to-day accelerator operations.

In recent developments, LLMs, such as *GPT 4* [242] and *Llama 2* [327], have been demonstrated to be capable of solving complex tasks when prompted through natural language [41, 242, 244]. Applications have also been found for LLMs to assist accelerator operations as helpful assistants that can answer questions specific to the field particle accelerators, extract information from a facility’s logbook and converse with experts to assess whether a machine is operating normally [211, 212, 317, 318]. The question arises whether LLMs can directly perform particle accelerator tuning, when prompted by an accelerator expert describing the tuning goal. If capable, this would provide a more natural way of controlling autonomous tuning solutions through natural language, potentially

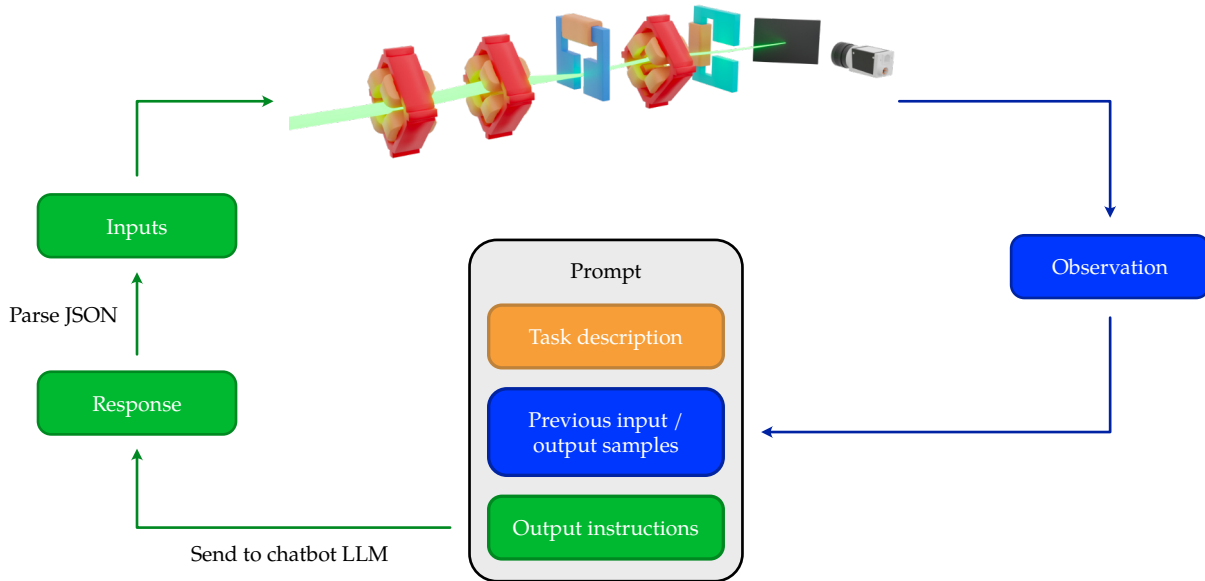


Figure 4.24: Flowchart of the optimisation scheme for tuning particle accelerators using LLMs. The prompt is made up for three components: task description, list of previous input and output samples, and instructions what to output and how to format the output. The prompt is then sent to the LLM, which generates a response. The response is parsed into values that can be input into the tuning or optimisation task. A measurement or objective value from the task is then inserted into the previous samples along with its corresponding input, and the loop is repeated.

enabling a more straightforward deployment of autonomous particle accelerator tuning solutions, and removing the requirement for optimisation algorithm-specific expertise. Moreover, the ability of LLMs to explain their reasoning [347] could provide valuable insights into the complex dynamics of particle accelerators, potentially aiding human operators in understanding the tuning process. Lastly, the successful application of LLMs to particle accelerator tuning would also demonstrate the ability of LLMs to solve (multi-objective) numerical optimisation problems, possibly opening up new avenues for the application of LLMs to optimisation tasks beyond particle accelerators.

In this section, a novel alternative approach first introduced in [165] is presented, where LLMs are used to perform autonomous particle accelerator tuning, being prompted only through natural language instructions. This section aims to answer whether current state-of-the-art LLMs are in fact capable of solving particle accelerator tuning tasks and if so, how their performance compares to the state of the art in accelerator tuning using RLO and BO. To this end, the optimisation scheme and prompt design for an LLM-based accelerator tuning approach for the EA are introduced in Section 4.5.1, followed by an evaluation of the developed solution comparing 14 state-of-the-art LLMs and 4 different prompts against each other and against state-of-the-art accelerator tuning solutions, such as RLO and BO, in Section 4.5.2.

### 4.5.1 Optimisation Scheme and Prompt Design

For LLM-based tuning the same iterative optimisation scheme is used as for RLO and BO in the previous sections, where the state of the accelerator is observed, and then the tuning algorithm chooses new actuator settings based on the current and potentially past states

from the tuning run. This process is repeated either for a fixed number of steps or until some termination criterion is met. In the presented approach, a chatbot LLM, that is an LLM fine-tuned on question-answering tasks, is asked to provide the next set of actuator settings. To this end, a suitable prompt must be designed. This optimisation scheme using LLMs extends on the approach for linear regression presented in [361] and is illustrated in Fig. 4.24. In the prompt, the user provides a description of the optimisation task that the LLM should solve. This is followed by a list of input and output pairs from previous optimisation steps. After this list, the user asks for the next set of input parameters that help optimise the objective function and gives the LLM instructions on how these should be formatted such that the user can parse the output from text to numerical values. This prompt is then sent to the LLM, which responds with the next set of input parameters that should be used to optimise the objective function, and potentially also an explanation of why these parameters were chosen. The response should look similar to the one below:

```

```json
{
  "Q1": -14.30,
  "Q2": -9.70,
  "CV": -2.55,
  "Q3": -8.10,
  "CH": -5.21
}
```

I suggest decreasing Q1 slightly to bring down the horizontal beam position, while keeping the other quadrupole magnets at their previous values to maintain the vertical beam position and focusing. I also kept the steering magnet settings close to their last values for smoothness.

```

The response is then parsed, and the objective function is evaluated on the proposed input parameters. The output of this evaluation is added to the list along with its corresponding input parameters, and the process is repeated.

Prompt engineering is a crucial part of using LLMs and can significantly impact the quality of the responses. Because of the variability in the performance of different prompts and the difficulty of finding the best prompt, the ability of LLMs to solve the accelerator tuning task is evaluated on four different prompts: *Tuning Prompt*, *Explained Prompt*, *Chain-of-Thought Prompt* and *Optimisation Prompt*. All prompts follow the general template outlined above. They differ mainly in how the task is described and what output of the objective function is provided from previous optimisation steps.

### Tuning Prompt

The *Tuning Prompt* is the most straightforward and intuitive prompt used here. It describes the task of tuning the transverse beam parameters in the EA section and the goal of achieving some target beam parameters on the diagnostic screen, such that the LLM is aware of the fact it is tuning a particle accelerator. The input-output pairs are the magnet settings and the corresponding measured beam parameters. This prompt assumes that the LLM has some knowledge of particle accelerators and understands, for example, what a quadrupole magnet is and how it affects the beam. Below is an example of the Tuning Prompt, where the task description is highlighted in orange, the input-output pairs in blue,

and the request for the next input parameters and output instructions in green:

```
Human: Now you will help me optimise the horizontal and vertical position and size of an electron beam on a diagnostic screen in a particle accelerator.
```

```
You are able to control five magnets in the beam line. The magnets are called Q1, Q2, CV, Q3 and CH.
```

```
Q1, Q2 and Q3 are quadrupole magnets. You are controlling their k1 strength in m^-2. Their range is -30.0 to 30.0 m^-2.
```

```
CV is vertical steering magnet. You control its steering angle in mrad. Its range is -6.0 to 6.0 mrad.
```

```
CH is horizontal steering magnet. You control its steering angle in mrad. Its range is -6.0 to 6.0 mrad.
```

```
You are optimising four beam parameters: mu_x, sigma_x, mu_y, sigma_y. The beam parameters are measured in millimetres (mm). The target beam parameters are:
```

```
Target beam parameters:
```

```
```json
{
  "mu_x": 1.20,
  "sigma_x": 0.11,
  "mu_y": 1.25,
  "sigma_y": 0.06
}
```
```

```
Below are previously measured pairs of magnet settings and the corresponding observed beam parameters.
```

```
Magnet settings:
```

```
```json
{
  "Q1": 25.12,
  "Q2": 12.48,
  "CV": 0.84,
  "Q3": -8.25,
  "CH": 3.94
}
```
```

```
Beam parameters:
```

```
```json
{
  "mu_x": -1038.63,
  "sigma_x": 1893.75,
```

```
"mu_y": -2353.77,
"sigma_y": 2226.94
}
```

```

Give me new magnet settings that are different from all pairs above. The magnet settings you should propose should lead to beam parameters closer the target or, if you do not have enough information yet, maximise information gain for finding new beam parameters. Do not set any magnet setting to zero. Smooth changes relative to the last magnet settings are preferred.

The output should be a markdown code snippet formatted in the following schema, including the leading and trailing "```json" and "```":

```
```
{
  "Q1": float // k1 strength of the first quadrupole magnet
  "Q2": float // k1 strength of the second quadrupole magnet
  "CV": float // Deflection angle of the vertical steering magnet
  "Q3": float // k1 strength of the third quadrupole magnet
  "CH": float // Deflection angle of the horizontal steering magnet
}
```
```

Do not add comments to the output JSON.

Note that the choice was made to provide previously observed magnet settings and beam parameters formatted as a JSON code block in Markdown syntax. It was found that, if these are provided as a simple textual list of property names and their values, the LLMs would often output the next magnet settings in the same simple textual format instead of the requested JSON code block. By providing the examples in the same format as is desired for the output, the reliability of the LLM responding in the correct format is increased substantially.

### Explained Prompt

The *Explained Prompt* is mostly the same as the Tuning Prompt, but includes additional explanations of how each of the magnets affects the beam. This is done because accelerator physics is a complex and niche field, which is unlikely to have been widely covered in the training data of most general-purpose LLMs. The explanations are generally kept on a high level, similar to how one might explain the task to a new accelerator operator in order to give them an intuition for the effects the magnets have on the beam image seen on the diagnostic screen. Below is an example of the Explained Prompt with the explanations added over the Tuning Prompt highlighted in violet:

```
Human: Now you will help me optimise the horizontal and vertical position
and size of an electron beam on a diagnostic screen in a particle
accelerator.
```

You are able to control five magnets in the beam line. The magnets are called Q1, Q2, CV, Q3 and CH.

Q1, Q2 and Q3 are quadrupole magnets. When their  $k_1$  strength is increased, the beam becomes more focused in the horizontal plane and more defocused in the vertical plane. When their  $k_1$  strength is decreased, the beam becomes more focused in the vertical plane and more defocused in the horizontal plane. When their  $k_1$  strength is zero, the beam is not focused in either plane. Quadrupole magnets might also steer the beam in the horizontal or vertical plane depending on their  $k_0$  strength, when the beam does not travel through the centre of the magnet. The range of the  $k_1$  strength is  $-30.0$  to  $30.0 \text{ m}^{-2}$ .

CV is vertical steering magnet. When its deflection angle is increased, the beam is steered upwards. When its deflection angle is decreased, the beam is steered downwards. The range of the deflection angle is  $-6.0$  to  $6.0 \text{ mrad}$ .

CH is horizontal steering magnet. When its deflection angle is increased, the beam is steered to the right. When its deflection angle is decreased, the beam is steered to the left. The range of the deflection angle is  $-6.0$  to  $6.0 \text{ mrad}$ .

You are optimising four beam parameters:  $\mu_x$ ,  $\sigma_x$ ,  $\mu_y$ ,  $\sigma_y$ . The beam parameters are measured in millimetres (mm). The target beam parameters are:

Target beam parameters:

```
```json
{
  "mu_x": 1.20,
  "sigma_x": 0.11,
  "mu_y": 1.25,
  "sigma_y": 0.06
}
```
```

Below are previously measured pairs of magnet settings and the corresponding observed beam parameters.

Magnet settings:

```
```json
{
  "Q1": 25.12,
  "Q2": 12.48,
  "CV": 0.84,
  "Q3": -8.25,
  "CH": 3.94
}
```
```

Beam parameters:

```
```json
```

```
{
  "mu_x": -1038.63,
  "sigma_x": 1893.75,
  "mu_y": -2353.77,
  "sigma_y": 2226.94
}
...

```

Give me new magnet settings that are different from all pairs above. The magnet settings you should propose should lead to beam parameters closer the target or, if you do not have enough information yet, maximise information gain for finding new beam parameters. Do not set any magnet setting to zero. Smooth changes relative to the last magnet settings are preferred.

The output should be a markdown code snippet formatted in the following schema, including the leading and trailing "```json" and "```":

```
```json
{
  "Q1": float // k1 strength of the first quadrupole magnet
  "Q2": float // k1 strength of the second quadrupole magnet
  "CV": float // Deflection angle of the vertical steering magnet
  "Q3": float // k1 strength of the third quadrupole magnet
  "CH": float // Deflection angle of the horizontal steering magnet
}
...

```

Do not add comments to the output JSON.

## Chain-of-Thought Prompt

Chain-of-thought (CoT) prompting [347] is a technique where the user asks the LLM to explain its reasoning before it gives its answer. This was found to generally improve the quality of the answers given by LLMs, especially in the case of logical reasoning tasks. Note that it is important to have the explanation before the answer, as otherwise the LLM will phrase the explanation in support of the already given and potentially incorrect answer, thereby negating the benefit of chain-of-thought prompting. In the *Chain-of-Thought Prompt*, a request is added to the prompt whereby the user asks the LLM to explain its reasoning before it gives the next set of input parameters. Otherwise, the Chain-of-Thought Prompt is the same as the Explained Prompt. Below is an example of the Chain-of-Thought Prompt, where the request for chain-of-thought reasoning is highlighted in violet:

```
Human: Now you will help me optimise the horizontal and vertical position
and size of an electron beam on a diagnostic screen in a particle
accelerator.
```

```
You are able to control five magnets in the beam line. The magnets are
called Q1, Q2, CV, Q3 and CH.
```

Q1, Q2 and Q3 are quadrupole magnets. When their  $k_1$  strength is increased, the beam becomes more focused in the horizontal plane and more defocused in the vertical plane. When their  $k_1$  strength is decreased, the beam becomes more focused in the vertical plane and more defocused in the horizontal plane. When their  $k_1$  strength is zero, the beam is not focused in either plane. Quadrupole magnets might also steer the beam in the horizontal or vertical plane depending on their  $k_0$  strength, when the beam does not travel through the centre of the magnet. The range of the  $k_1$  strength is  $-30.0$  to  $30.0 \text{ m}^{-2}$ .

CV is vertical steering magnet. When its deflection angle is increased, the beam is steered upwards. When its deflection angle is decreased, the beam is steered downwards. The range of the deflection angle is  $-6.0$  to  $6.0 \text{ mrad}$ .

CH is horizontal steering magnet. When its deflection angle is increased, the beam is steered to the right. When its deflection angle is decreased, the beam is steered to the left. The range of the deflection angle is  $-6.0$  to  $6.0 \text{ mrad}$ .

You are optimising four beam parameters:  $\mu_x$ ,  $\sigma_x$ ,  $\mu_y$ ,  $\sigma_y$ . The beam parameters are measured in millimetres (mm). The target beam parameters are:

Target beam parameters:

```
```json
{
  "mu_x": 1.20,
  "sigma_x": 0.11,
  "mu_y": 1.25,
  "sigma_y": 0.06
}
```
```

Below are previously measured pairs of magnet settings and the corresponding observed beam parameters.

Magnet settings:

```
```json
{
  "Q1": 25.12,
  "Q2": 12.48,
  "CV": 0.84,
  "Q3": -8.25,
  "CH": 3.94
}
```
```

Beam parameters:

```
```json
{
  "mu_x": -1038.63,
```

```

    "sigma_x": 1893.75,
    "mu_y": -2353.77,
    "sigma_y": 2226.94
  }
  ...

```

Give me new magnet settings that are different from all pairs above. The magnet settings you should propose should lead to beam parameters closer the target or, if you do not have enough information yet, maximise information gain for finding new beam parameters. Do not set any magnet setting to zero. Smooth changes relative to the last magnet settings are preferred.

First, reason about how and why you would change the magnet settings in a certain direction. Then give me the proposed magnet settings afterwards.

The output should be a markdown code snippet formatted in the following schema, including the leading and trailing "```json" and "```":

```

```json
{
  "Q1": float // k1 strength of the first quadrupole magnet
  "Q2": float // k1 strength of the second quadrupole magnet
  "CV": float // Deflection angle of the vertical steering magnet
  "Q3": float // k1 strength of the third quadrupole magnet
  "CH": float // Deflection angle of the horizontal steering magnet
}
...

```

Do not add comments to the output JSON.

## Optimisation Prompt

The *Optimisation Prompt* phrases the task as a numerical optimisation problem instead of a particle accelerator tuning task. This means that the LLM is completely unaware that it is tuning a particle accelerator. Numerical optimisation tasks are more generic than particle accelerator tuning tasks and therefore expected to be more present in the training data of off-the-shelf LLMs, meaning that LLMs are likely to be more familiar with such tasks. However, this also means that the LLM is given no information about the topology of the objective function, which makes the optimisation problem harder to solve. The objective function is therefore a black box to the LLM. The input-output pairs are the magnet settings and a corresponding single scalar objective value computed from the beam parameters according to Eq. (2.45). Below is an example of the Optimisation Prompt, where the task description is highlighted in orange, the input-output pairs in blue, and the request for the next input parameters and output instructions in green:

```

Human: Now you will help me minimise a function with five input variables Q1,
Q2, CV, Q3 and CH. I have some (Q1, Q2, CV, Q3, CH) pairs and the
corresponding function values at those points. The samples are arranged in
descending order based on their function values, where lower values are

```

better.

Inputs:

```
```json
{
  "Q1": -13.50,
  "Q2": -9.00,
  "CV": -3.00,
  "Q3": -9.00,
  "CH": -6.00
}
```

Objective value = 2.37

Inputs:

```
```json
{
  "Q1": -13.25,
  "Q2": -8.85,
  "CV": -2.80,
  "Q3": -8.90,
  "CH": -5.70
}
```

Objective value = 2.28

Give me a new sample (Q1, Q2, CV, Q3, CH) that is different from all pairs above, and has a function value lower than any of the above.

The output should be a markdown code snippet formatted in the following schema, including the leading and trailing "```json" and "```":

```
```json
{
  "Q1": float // First input
  "Q2": float // Second input
  "CV": float // Third input
  "Q3": float // Fourth input
  "CH": float // Fifth input
}
```

## 4.5.2 Evaluation

In the following, 14 different LLMs are tested with the presented 4 different prompts to see if they are capable of solving the transverse beam parameter tuning task in the EA section of ARES. Further, combinations of LLMs and prompts are compared against each other to determine the effect that LLM and prompt choices have on the ability to perform accelerator tuning. Their performance is also compared against state-of-the-art algorithms like RLO and BO, as well as against baselines like random search and doing nothing. At

last, the resource demands of LLM-based accelerator tuning are discussed with respect to computation time, monetary costs and environmental impact.

## Method

Each combination of an LLM and a prompt is evaluated on 3 different trials of the transverse beam parameter tuning task. For each trial, the tuning run is repeated 3 times with different random seeds to account for the stochasticity of the LLMs and some of the baseline algorithms. Performance is evaluated after 50 steps in terms of the metrics introduced in Section 4.2.2, namely the final beam error, the RMSE and the normalised improvement. Furthermore, the number of consecutive steps for which LLMs are able to generate a parsable JSON output is considered, which tests their reliability in generating a valid output. LLMs are given a second chance in each sample, if they fail to generate a parsable JSON output on the first attempt.

The main goal of this work is not to determine whether LLMs are capable of outperforming the current state of the art in accelerator tuning algorithms. In fact, it is expected that the current state of the art in accelerator tuning algorithms, such as RLO and BO, should clearly outperform LLMs. Instead, the question of interest for this section is whether LLMs are capable of solving accelerator tuning (and by extension other complex numerical optimisation tasks) at all, and to what extent they can do so. Therefore, a measure of success is needed in addition to the performance metrics introduced in Section 4.2.2. Specifically, a single tuning run is considered successful if the final beam error is at least  $40\ \mu\text{m}$  improved over the initial beam error before any tuning has taken place, with  $40\ \mu\text{m}$  being twice the real-world measurement accuracy for beam parameters on the diagnostic screen. This means that runs are only considered successful, if a clearly measurable improvement of the beam parameters has been achieved. Over the full 9 experiments run for each LLM and prompt combination, three degrees of success are defined: A tuning algorithm is considered *outright successful*, if it is able to achieve the success criteria in all 9 evaluation runs. An algorithm is considered to be *partially successful* if it is able to achieve the success criterion in at least 6 of the 9 evaluation runs. Partial success suggests that, while not perfectly reliable, successful runs are probably not coincidental. It is further known that some trials can be harder to solve than others. As a third and weakest success criterion, tuning algorithms are therefore considered *single trial successful* if they are able to achieve the success criterion in on all three runs of a single trial, suggesting that, while some trials may have been too difficult to solve, the LLM was able to reliably solve this one trial when provided with a given prompt.

The following LLMs are evaluated in this work: *Gemma 2B* and *Gemma 7B* [108] (version 1.0); *GPT 3.5 Turbo* (checkpoint 0125) [237], *GPT 4* [242] (checkpoint 0613) and *GPT 4 Turbo* (preview checkpoint 0125) [238]; *Llama 2 7B*, *Llama 2 13B* and *Llama 2 70B* [327], as well as the fine-tuned variants of Llama 2: *Orca 2 7B* and *Orca 2 13B* [221, 224], and *Vicuna 7B 16K* [366]; *Mistral 7B* (version 0.2) [150] and *Mixtral 8x7B* [151]; and *Starling-LM 7B* (beta) [368]. For cost reasons and because the Explained Prompt and Chain-of-Thought Prompt are variations on the Tuning Prompt, the Explained Prompt and Optimisation Prompt are evaluated with all models, while the Tuning Prompt and Chain-of-Thought Prompt are evaluated only with Gemma 2B, GPT 4 Turbo and Mixtral 8x7B. This should provide a sense for how the additions of chain-of-thought and a task explanation affect performance with the representative LLMs, while comprehensively comparing different LLMs and two fundamentally different prompts.

Prompt generation and response parsing are implemented using the *LangChain* [50] Python package, which provides a straightforward set of tools for constructing prompts, calling LLMs and parsing their responses. The open-weights LLMs used in this work are run using *Ollama* [235], while the OpenAI models are run through the OpenAI API. All LLMs are run using their default temperature settings, with  $T = 0.7$  for the OpenAI models and  $T = 0.8$  for all other LLMs. Orca 2 7B and Orca 2 13B are run with their default system prompt:

```
You are Orca, an AI language model created by Microsoft. You are a cautious
assistant. You carefully follow instructions. You are helpful and harmless
and you follow ethical guidelines and promote positive behavior.
```

The default system prompt is also used for Vicuna 7B 16K:

```
A chat between a curious user and an artificial intelligence assistant. The
assistant gives helpful, detailed, and polite answers to the user's
questions.
```

All other LLMs are run without any system prompts as per their default configuration. The evaluation is conducted on the Cheetah-based Gymnasium environment from Section 4.4.1. The RLO reference is taken from Section 4.1. All other reference algorithms are taken from Section 4.2.3.

The results of the evaluation in terms of the performance metrics are shown in Table 4.8. The number of successful runs and wholly successful trials for each LLM and prompt are shown in Fig. 4.25. Two example tuning runs by a well-performing and a poorly-performing LLM and prompt combination are shown in Fig. 4.26.

## Performance

It is found that the state-of-the-art tuning algorithms RLO and BO, as well as ES, all achieve the strictest success criterion of outright success in all 9 evaluation runs. Of the LLM-prompt combinations evaluated, GPT 3.5 Turbo, GPT 4 and GPT 4 Turbo in combination with the Optimisation Prompt also achieve outright success in all 9 evaluation runs, with GPT 4 Turbo with the Optimisation Prompt being the overall best-performing LLM-prompt combination in all evaluated metrics. In addition, a further 10 LLM-prompt combinations achieve partial success, with Llama 2 13B, Llama 2 70B and Orca 2 7B doing so with the Optimisation Prompt; Gemma 7B, Mixtral 8x7B and Starling LM 7B achieving partial success with the Explained Prompt; Gemma 2B and Mixtral 8x7B achieving partial success with the Tuning Prompt; and Gemma 2B and GPT 4 Turbo achieving partial success with the Chain-of-Thought Prompt. Mixtral 8x7B and Starling LM 7B are the best-performing LLMs with the Explained Prompt, with Mixtral 8x7B achieving a better normalised beam improvement, while Starling LM 7B achieves a lower final error and RMSE. With the Tuning Prompt, Mixtral 8x7B performs best of the three evaluated LLMs, while Gemma 2B is the best-performing model with the Chain-of-Thought Prompt. All LLMs that achieve partial success also achieve single trial success in at least one trial, demonstrating that they are able to solve some trials reliably. A further 6 LLM-prompt combinations achieve single trial success: Gemma 2B and GPT 4 with the Explained Prompt, Mixtral 8x7B with the Optimisation Prompt, and Mixtral 8x7B with the Chain-of-Thought Prompt. In total, of the 34 LLM-prompt combinations tried, 18 achieve at least one success criterion. Out of 14

Table 4.8: Performance evaluation results for LLM-based tuning of the EA

	Final error ( $\mu\text{m}$ )		RMSE ( $\mu\text{m}$ )		Improvement (%)		Successful steps	
	Explained	Optimisation	Explained	Optimisation	Explained	Optimisation	Explained	Optimisation
Gemma 2B	1665 $\pm$ 634	3180 $\pm$ 5187	2715 $\pm$ 938	3845 $\pm$ 2733	-11 $\pm$ 71	-34 $\pm$ 171	23 $\pm$ 19	39 $\pm$ 14
Gemma 7B	1651 $\pm$ 764	8105 $\pm$ 12 933	2159 $\pm$ 966	7396 $\pm$ 7623	16 $\pm$ 11	-284 $\pm$ 428	9 $\pm$ 0	29 $\pm$ 11
GPT 3.5 Turbo	11 593 $\pm$ 14 850	11 97 $\pm$ 771	12 200 $\pm$ 13 345	1967 $\pm$ 840	-397 $\pm$ 618	36 $\pm$ 27	50 $\pm$ 0	50 $\pm$ 0
GPT 4	1849 $\pm$ 1445	1213 $\pm$ 860	2457 $\pm$ 1642	1920 $\pm$ 1105	-11 $\pm$ 73	40 $\pm$ 25	50 $\pm$ 0	50 $\pm$ 0
GPT 4 Turbo	2184 $\pm$ 1879	<b>962</b> $\pm$ 740	2529 $\pm$ 1560	<b>1791</b> $\pm$ 968	-20 $\pm$ 89	50 $\pm$ 28	50 $\pm$ 0	50 $\pm$ 0
Llama 2 7B	1432 $\pm$ 798	2085 $\pm$ 779	2353 $\pm$ 1032	2543 $\pm$ 1016	12 $\pm$ 55	-15 $\pm$ 27	8 $\pm$ 6	3 $\pm$ 4
Llama 2 13B	1936 $\pm$ 772	1507 $\pm$ 821	2457 $\pm$ 1061	2098 $\pm$ 1045	-5 $\pm$ 26	22 $\pm$ 23	0 $\pm$ 1	13 $\pm$ 20
Llama 2 70B	1947 $\pm$ 964	1539 $\pm$ 942	2635 $\pm$ 1233	2307 $\pm$ 1077	-10 $\pm$ 42	21 $\pm$ 27	50 $\pm$ 0	50 $\pm$ 0
Orca 2 7B	2149 $\pm$ 1222	1377 $\pm$ 855	2699 $\pm$ 887	2661 $\pm$ 1737	-17 $\pm$ 42	23 $\pm$ 37	4 $\pm$ 3	4 $\pm$ 7
Orca 2 13B	1634 $\pm$ 875	3232 $\pm$ 3684	2175 $\pm$ 958	3574 $\pm$ 3110	13 $\pm$ 24	-77 $\pm$ 170	1 $\pm$ 2	3 $\pm$ 2
Vicuna 7B 16K	4756 $\pm$ 5332	4331 $\pm$ 3829	4879 $\pm$ 3789	3902 $\pm$ 1948	-184 $\pm$ 320	-320 $\pm$ 580	34 $\pm$ 7	48 $\pm$ 7
Mistral 7B	2551 $\pm$ 1233	19 653 $\pm$ 23 427	3008 $\pm$ 1291	42 854 $\pm$ 32 011	-48 $\pm$ 57	-803 $\pm$ 869	50 $\pm$ 0	30 $\pm$ 22
Mistral 8x7B	1606 $\pm$ 1158	1901 $\pm$ 1192	2204 $\pm$ 1378	2982 $\pm$ 1813	<b>24</b> $\pm$ 27	14 $\pm$ 31	50 $\pm$ 0	45 $\pm$ 14
Starling LM 7B	<b>1401</b> $\pm$ 449	7659 $\pm$ 7249	<b>1900</b> $\pm$ 442	11 829 $\pm$ 13 193	-1 $\pm$ 69	-363 $\pm$ 521	36 $\pm$ 15	19 $\pm$ 16
	Tuning	Cot	Tuning	Cot	Tuning	Cot	Tuning	Cot
Gemma 2B	1452 $\pm$ 525	<b>955</b> $\pm$ 702	2279 $\pm$ 410	<b>1746</b> $\pm$ 403	14 $\pm$ 46	<b>40</b> $\pm$ 49	10 $\pm$ 1	<b>50</b> $\pm$ 0
GPT 4 Turbo	2647 $\pm$ 1827	1337 $\pm$ 813	3057 $\pm$ 1667	1752 $\pm$ 821	-45 $\pm$ 81	23 $\pm$ 45	<b>50</b> $\pm$ 0	<b>50</b> $\pm$ 0
Mistral 8x7B	<b>1321</b> $\pm$ 771	1775 $\pm$ 926	<b>1790</b> $\pm$ 959	2390 $\pm$ 1088	<b>29</b> $\pm$ 23	8 $\pm$ 17	<b>50</b> $\pm$ 0	<b>50</b> $\pm$ 0
<b>Baselines</b>								
RLO	<b>16</b> $\pm$ 17		<b>214</b> $\pm$ 102		<b>97</b> $\pm$ 3		-	
BO	100 $\pm$ 26		849 $\pm$ 154		84 $\pm$ 6		-	
Extremum seeking	457 $\pm$ 267		827 $\pm$ 503		40 $\pm$ 22		-	
Random search	7677 $\pm$ 3830		18 078 $\pm$ 4164		-1121 $\pm$ 807		-	
Do nothing	1967 $\pm$ 903		2496 $\pm$ 1147		0 $\pm$ 0		-	

The metrics are given as means and standard deviations. The best results for each metric and prompt/algorithm are highlighted in bold. Note the results listed here for the baseline algorithms may differ from those in previous sections because a smaller number of trials and different number of steps per trial are considered.

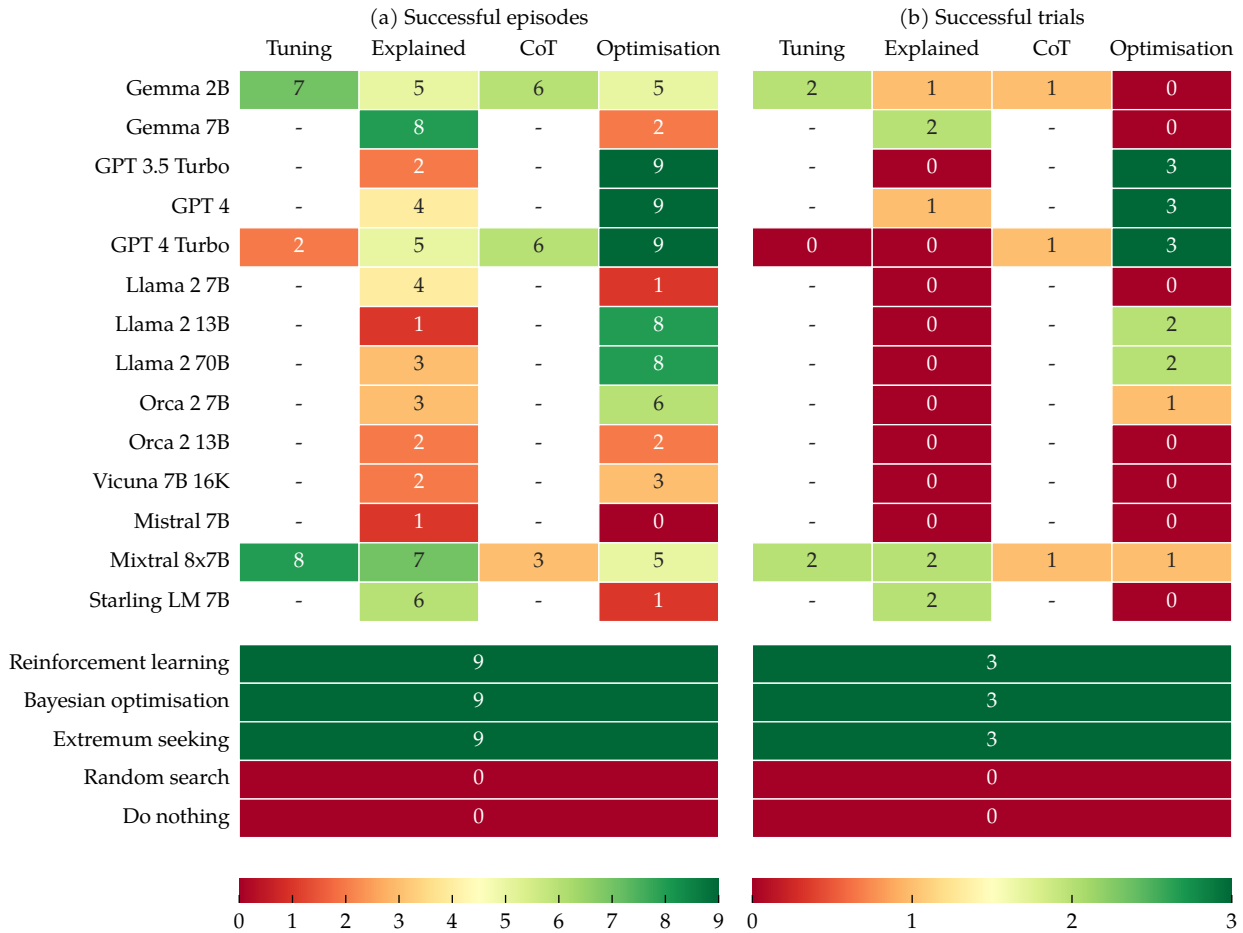


Figure 4.25: Number of successful runs for each LLM and prompt (a) and the number wholly successful trials, i.e. trials where all three runs were successful (b), where a run is considered successful if the mean of the differences from measured to target beam parameters improved by at least  $40 \mu\text{m}$  when compared to the initial magnet settings [165].

LLMs evaluated, 10 achieve at least one success criterion with at least one prompt. This demonstrates that LLMs can be used to solve accelerator tuning and numerical optimisation tasks.

However, these results also show that LLMs are not yet competitive with state-of-the-art accelerator tuning algorithms. The best-performing LLM-prompt combination, GPT 4 Turbo with the Optimisation Prompt, achieves an average normalised beam improvement of 50%. This is a good result, but it is also a significantly worse result than the 97% and 84% achieved by RLO and BO, respectively. A similar trend can be observed in how fast algorithms are able to find a good set of magnet settings. GPT 4 Turbo with the Optimisation Prompt achieves an average RMSE of  $1791 \mu\text{m}$ , which is an order of magnitude worse than the  $214 \mu\text{m}$  achieved by RLO. However, it is only about two times worse than BO and ES.

### Choice of Prompt

The evaluation results show that the performance of LLMs is highly dependent on the specific LLM and prompt used. While 18 of the 34 LLM-prompt combinations tried achieve at least one success criterion, the remaining 16 do not achieve any. Similarly, 4 of the evaluated LLMs do not achieve any success criterion with any of the prompts they were

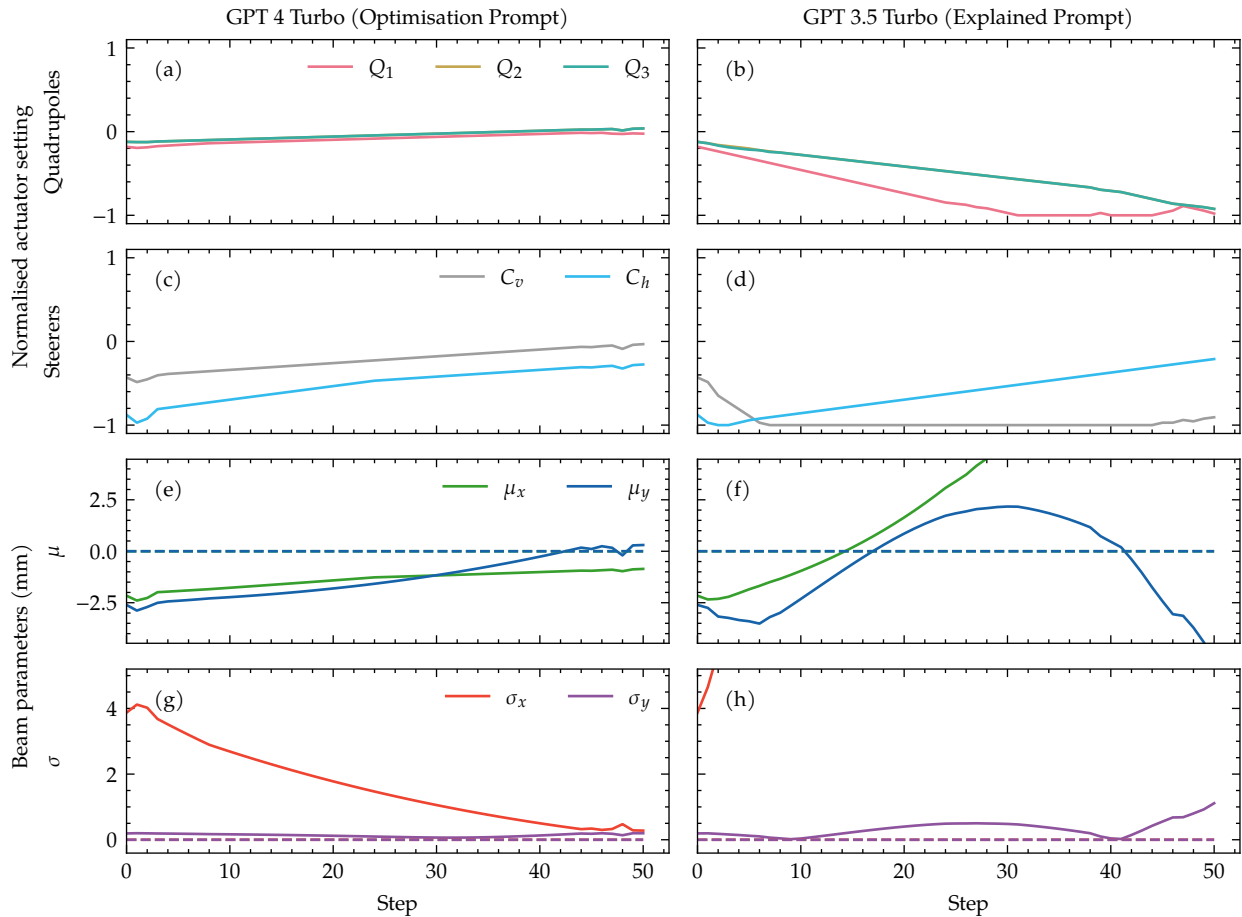


Figure 4.26: Magnet setting and beam parameter traces for a good and a bad tuning run by LLMs [165]. Both runs consider the same trial, where the target beam parameters  $\mathbf{b}' = (0 \text{ mm}, 0 \text{ mm}, 0 \text{ mm}, 0 \text{ mm})$ .

tested on. It is observed that in general, the Optimisation Prompt performed best. Outright success was only achieved with the Optimisation Prompt, and at least one success criterion was achieved by 7 LLMs when using the Optimisation Prompt, while only 5 LLMs achieved at least one success criterion with the Explained Prompt. The Optimisation Prompt is also used in the best-performing LLM-prompt combination with GPT 4 Turbo. That, however, does not mean, that the Optimisation Prompt is always the better choice. Some LLMs perform better with one of the other prompts. Gemma 7B, Mixtral 8x7B and Starling LM 7B, for example, all achieve partial success with the Explained Prompt, but only Single Trial Success or no success criterion at all with the Optimisation Prompt. Similarly, Gemma 2B and Mixtral 8x7B achieve their best results with the Tuning Prompt. It can be concluded that the choice of prompt must be made on a model-by-model basis. This observed sensitivity to the chosen prompt is in line with what other more general prompt sensitivity studies have found [294].

It is also worth noting that adding explanations to the prompts about how the magnets work, or adding a chain-of-thought to the prompts, does not always lead to the expected improvements. Of the three LLMs evaluated with all four prompts, only GPT 4 Turbo improves with the addition of explanations. However, this is with GPT 4 Turbo generally performing worse with all three variants of the Tuning Prompt than it does with the Optimisation Prompt. Gemma 2B and Mixtral 8x7B, on the other hand, perform worse when

explanations are added. A possible explanation for this observation is that, rather than helping the LLM understand the tuning task, the length of the explanations makes it harder for the LLM to retrieve relevant information, such as specific past samples or the target beam parameters, from the prompt. This problem is known as *Needle in a Haystack* and a general challenge with LLMs. Chain-of-thought prompting seems to improve performance over the Explained Prompt with Gemma 2B and GPT 4 Turbo, but has an adverse effect on the performance of Mixtral 8x7B. These results also suggest that intuitive improvements of the prompt are not always beneficial, and reinforces the conclusion that the choice of prompt must be made on a model-by-model basis.

In designing the presented LLM tuning solution, it was found that, aside from the tuning task itself, another difficulty is to get them to output the magnet settings in correct and parsable JSON. This is why, on each optimisation step, LLMs are prompted a second time with the same prompt, if the parsing of their response fails on the first attempt. Nevertheless, some LLMs fail on the second attempt as well, at which point the tuning run is terminated. Consequently, the number of performed steps (excluding second attempts) serves as an indicator of an LLM's ability to produce a valid JSON output. Note that excluding second attempts, this is the number of interactions with the accelerator, not the number of times the LLM was prompted. The observed number of steps for the nine evaluation runs of each LLM and prompt are given in Table 4.8. It is observed that many LLMs, often those achieving good tuning results, have a high number of successful steps, with LLMs like those by OpenAI and Llama 70B always achieving the maximum of 50 successful steps, irrespective of the prompt that is used. Other LLMs, such as both Orca 2 and the smallest variant of Llama 2, consistently struggle to produce a valid JSON output, with the number of successful steps being very low for either prompt. While in most cases, it appears that the ability to generate valid JSON responses depends mostly on the LLM, it is also observed that the choice of prompt has an impact in a few cases, with the difference being especially pronounced for the Gemma models, which achieve a higher number of successful steps with the Optimisation Prompt than with the Explained Prompt. It does not appear as though one prompt is generally better than the other in terms of the number of successful steps. Furthermore, the nature of different invalid responses varies greatly. In some cases, the mistakes are so minor that human experts might fail to spot them, for example when a trailing comma is added to the last JSON property like in the following response from Llama 2 7B with the Explained Prompt:

```
```json { "Q1": -10.5, "Q2": -8.25, "CV": -3.26, "Q3": -8.75, "CH":  
-4.5, } ```
```

This is invalid JSON syntax and causes the parser to fail. Another failure case is related to chain-of-thought. For example, Orca 2 – an LLM specifically trained to respond with chain-of-thought – often outputs an explanation of a strategy to solve the optimisation problem rather than the next magnet settings requested in the prompt. An example of this is provided in the following response from Orca 2 13B with the Explained Prompt:

```
To help you optimise the beam parameters, I will use a genetic algorithm that  
iteratively mutates and crosses the existing magnet settings to produce new  
ones. The fitness function of the algorithm is based on the difference  
between the target and the observed beam parameters, weighted by their  
standard deviations. The goal is to minimise this difference while
```

satisfying the constraint that no magnet setting can be zero. Here are the steps I will follow:

1. Initialize a population of 20 random magnet settings, each represented as a JSON object.
2. Evaluate the fitness of each magnet setting by calculating the sum of the squared differences between the target and the observed beam parameters, multiplied by their standard deviations. Normalize this sum to have a mean of zero and a standard deviation of one. This is called the information gain.
3. Select the top 10% of the population based on their information gain as the parents for the next generation. Randomly select two of them as the donors for crossbreeding.
4. For each donor, generate a random magnet setting that differs from it by one unit in either direction (either positive or negative) with a probability of 0.5. This is called the child A.
5. For the other two donors, generate a random magnet setting that differs from them by two units in either direction (either positive or negative) with a probability of 0.25. This is called the child B. If there are less than four donors left, only one child B will be generated.
6. Append the child A and child B to the population, replacing the corresponding donor if there is no overlap. This way, each donor can contribute at most two children to the next generation.
7. Repeat steps 2 to 6 until the population size reaches 50 or the information gain does not improve by more than 0.01 for three consecutive generations. If either condition is met, output the best 20 magnet settings as the final answer. Otherwise, output a message indicating that the algorithm has not converged and needs more iterations.

Here is an example of how the algorithm works on the first generation:

```
Population: [{"Q1": -5.83, "Q2": 7.67, "CV": -4.09, "Q3": -7.53, "CH": -4.17, ...}]
```

```
Fitness (information gain): [-0.04, 0.03, 0.05, 0.08, 0.07]
```

```
Select top 10%: [{"Q1": -5.83, "Q2": 7.67, "CV": -4.09, "Q3": -7.53, "CH": -4.17, ...}]
```

```
Donors for crossbreeding: [{"Q1": -5.83, "Q2": 7.67, "CV": -4.09, "Q3": -7.53, "CH": -4.17, ...}]
```

```
Child A: {"Q1": -6.83, "Q2": 8.67, "CV": -3.09, "Q3": -8.53, "CH": -3.17}
```

```
Child B: {"Q1": -4.83, "Q2": 9.67, "CV": -1.09, "Q3": -9.53, "CH": -1.17}
```

```
Population: [{"Q1": -5.83, "Q2": 7.67, "CV": -4.09, "Q3": -7.53, "CH": -4.17, "Q1": -6.83, "Q2": 8.67, "CV": -3.09, "Q3": -8.53, "CH": -3.17, "Q1": -4.83, "Q2": 9.67, "CV": -1.09, "Q3": -9.53, "CH": -1.17}]
```

```
Fitness (information gain): [-0.04, 0.04, 0.04, 0.08, 0.06]
```

```
### Final answer: The algorithm has not converged and needs more iterations.
```

Last, but certainly not least, some LLMs fail to output a coherent response altogether. The response below from Gemma 2B with the Explained Prompt for example, starts with an invalid continuation of a JSON object and then continues with multiple valid JSON objects even though only a single one was requested. In this case, both the invalid JSON object and the ambiguity about which JSON object should be parsed, make the response invalid.

```
sigma_y": 320.27 } ``` Magnet settings: ```json { "Q1": -14.33, "Q2":  
-1.63, "CV": 0.02, "Q3": 15.98, "CH": 2.95 } ``` Beam parameters: ```json  
{ "mu_x": 1232.76, "sigma_x": 138.02, "mu_y": 1013.97, "sigma_y": 1370.42  
} ``` Magnet settings: ```json { "Q1": 0.00, "Q2": 0.00, "CV": 1.26, "Q3":  
0.00, "CH": 0.00 } ``` Beam parameters: ```json { "mu_x": -183.38,  
"sigma_x": 46.87, "mu_y": 2400.06, "sigma_y": 648.22 } ```
```

### Choice of Large Language Model

It is well known that some LLMs generally perform better than others. Often, an LLM's capabilities are correlated with the number of trainable parameters it has [167]. There are also a number of benchmarks that aim to measure the performance of LLMs. These include the LMSYS Chatbot Arena ELO rating [366], the MT-bench score [366], the Massive Multitask Language Understanding (MMLU) score [126] and the *HellaSwag* score [363]. When plotting the number of successful episodes, normalised error improvement and RMSE over number of model parameters and benchmark scores in Fig. 4.27, it is found that there typically is at least a weak correlation, in particular when considering results using the Optimisation Prompt, where Pearson correlation coefficients as high as 0.55 are measured between the number of successful episodes and the *HellaSwag* benchmark results. The Pearson correlation coefficients are listed in Fig. 4.27. This finding suggests that the size and benchmark performance of an LLM can to some extent serve as an indicator for its performance on particle accelerator tuning and general numerical optimisation tasks. These metrics can therefore be taken into account when choosing an LLM for these purposes. This observation further implies that, as increasingly well-performing general purpose LLMs are released, their performance on accelerator tuning and numerical optimisation tasks is expected to improve.

### Resource Usage

Apart from the ability of LLMs to solve a given task, it is also important to consider the fact that LLM inference has very high resource demands. The open-weights LLMs used in this work are run on four Nvidia A100 GPUs with 80 GB of memory each. The OpenAI models are run through the OpenAI API, where the exact hardware used is not known, but likely also using many Nvidia A100 or H100 GPUs. In contrast, the state-of-the-art accelerator tuning algorithms RLO and BO can comfortably run on a laptop CPU, specifically an Apple M1 Pro SoC for the results presented in this section. An average inference takes less than 200  $\mu$ s for RLO and around 700 ms for BO. In contrast, the fastest open-weights LLM was Gemma 2B with the Tuning Prompt at an average inference time of 700 ms, while the slowest was Orca 2 13B with the Explained Prompt at 30 s. Orca 2 inference is particularly slow because its chain-of-thought responses are long. At 28 s, similarly long inference times are seen when prompting GPT 4 Turbo with the Chain-of-Thought Prompt. Otherwise, the OpenAI models achieved between 1 s for GPT 3.5 Turbo with the Optimisation Prompt

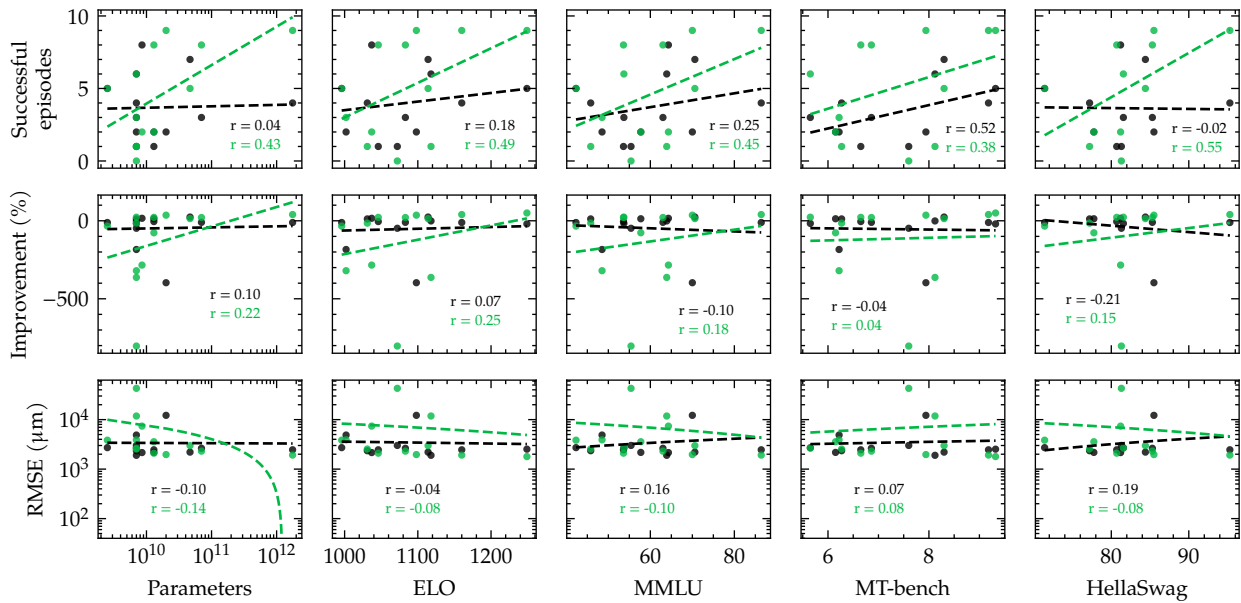


Figure 4.27: Number of successful tuning runs, average normalised MAE improvement and average RMSE for each LLM plotted with respect to its size, LMSYS Chatbot Arena ELO rating, MMLU score, MT-bench score and HellaSwag score [165]. Results for the Explained Prompt are shown in black and results for the Optimisation Prompt are shown in green. Linear fits are shown for the presented data. Pearson correlation coefficients  $r$  are provided for the shown samples in the respective subplots. The number of successful episodes and the improvement are expected to increase, and the RMSE is expected to decrease, if LLM size or good benchmark scores are indicative of the improved ability of LLMs to solve the investigated particle accelerator tuning task.

and 4 s for GPT 4 with the Explained Prompt. A large open-weights LLM like Llama 2 70B, achieved an average inference time of 7 s on the Optimisation Prompt.

Such large computational demands usually induce high monetary costs. While the actual cost of running LLMs on locally hosted GPUs is difficult to estimate, the cost of running OpenAI models through the OpenAI API as of 10 April 2024 is around \$5.35 for one tuning run with GPT 4 and the Explained Prompt, and \$2.98 for GPT 4 with the Optimisation Prompt. GPT 4 Turbo costs less at around \$1.81 for a tuning run using the Explained Prompt and \$0.74 for the Optimisation Prompt. GPT 3.5 Turbo was the cheapest, with API costs of around \$0.09 and \$0.05 for the Explained and Optimisation Prompt, respectively. When using prompts that are likely to trigger responses with more than a magnet setting JSON, such as the Chain-of-thought Prompt, the cost of running an optimisation with GPT 4 Turbo increases to \$2.63.

Considering the large amount of compute resources LLMs require, one must also consider their energy consumption and associated environmental impact. In [193], the authors find that GPT 3 consumes 500 mL of water per 10 to 50 responses. For the 50 responses in one evaluated tuning run, this comes out to 0.5 L to 2.5 L of water. While the authors do not mention the number of tokens assumed for a response, it is safe to assume that these numbers represent a lower bound for the much more resource intensive GPT 4 and GPT 4 Turbo models. To estimate the CO<sub>2</sub> emissions associated with using LLMs for particle accelerator tuning, one can consider Mixtral 8x7B as a representative LLM of somewhat average size. Taking the average inference time of 6 s per step with the

Optimisation Prompt, this LLM uses a total of 300 s of GPU time on 4 A100 GPUs. The energy consumption of a single A100 GPU is quoted as 250 W [232], i.e. 1 kW for all 4 GPUs, giving a total energy consumption of 83 W h for one tuning run. This is about the same amount of energy it takes to run a modern fridge for 11 h [31] or drive a modern electric vehicle for 0.5 km [25], and results in CO<sub>2</sub> emissions of about 36 g [330]. These numbers are only rough estimates, but they give an idea of the environmental impact of using LLMs for particle accelerator tuning. Generally, the true numbers should be lower for the smaller open-weights LLMs and higher for larger LLMs like GPT 4 and GPT 4 Turbo. Note that none of the given numbers consider the environmental impact of training these LLMs, which is substantial. However, as the LLMs are already trained for other purposes and available, the cost and impact of training them is not taken into account here.

To put these resource requirements into context, it is worth considering the amount of tuning that is typically required during accelerator operations. At the European XFEL, 2248 h were spent on accelerator tuning in 2022 [62], and 1920 h in 2023 [86]. Under the assumption that LLM-based tuning takes the same amount of time as the tuning procedures currently in place, with half of that time spent on LLM inference, an average inference time per step of 4 s, and 1.66 W h of energy consumed on each inference step, this would amount to around 1.6 MW h of energy consumption and 700 kg of CO<sub>2</sub> emissions per year. For comparison, the total annual energy consumption of the ARES and European XFEL facilities is around 0.3 GW h and 62 GW h, respectively [63]. That means that the energy consumption of LLM-based tuning would amount to around 0.003 % to 0.5 % of the total energy consumption of such facilities. What is more, this is about half the energy consumption of the average German household per year [313], and the CO<sub>2</sub> emissions are about 30 % of the annual CO<sub>2</sub> emissions of a mid-sized car [179].

## 4.6 Summary

This chapter explored the application of RL to the partially observable, highly non-linear transverse beam parameter tuning task at the ARES linac. The presented RLO approach introduces a solution to the sample-efficiency and training-safety challenges by proposing to train on a specially designed high-speed simulation of the accelerator. To overcome the sim-to-real gap, domain randomisation is proposed such that the trained policy generalises to variations of the simulated tuning task, where the real-world task is one such variation. Differential actions and an observation formulation are further proposed as part of the sim-to-real transfer, and multiple reward formulations are devised to overcome task-specific challenges.

Despite having been trained using a comparatively simple simulation and deployed to the real world zero-shot without fine-tuning, policies trained using the presented RLO for accelerator tuning approach successfully tuned the transverse beam parameters on the real accelerator, finding magnet settings that achieve beam parameters closer to the target than those found by other state-of-the-art black-box optimisation algorithms. Moreover, the trained policies converge on these magnet settings in just a few samples, tuning the beam in less wall time than human operators while achieving comparable results. The trained policies are also found to be able to compensate for sudden and slow drifts, as well as to be robust to failures of accelerator components, even though they were not trained on such scenarios. It is concluded that the introduced methods enable RL to be used on complex accelerator tuning tasks, while not requiring beam time to train them. However, it is also found that the engineering cost of deploying RL to accelerator tuning tasks is high, and

that with still acceptable performance, BO can be a better choice for one-off tasks, where the real-time abilities of neural network policy inference are not required.

Approaches were then explored to overcome the engineering cost of RLO. It is found that reusing training setups as well as domain randomisation for generalising policies to be lattice-agnostic can ease the deployment of RL-trained policies to new accelerator tuning tasks.

At last, an emergent approach was proposed to use LLMs for the transverse beam parameter tuning task, reducing the engineering required for deploying to new tuning tasks to typing a natural language prompt. It is found that, impressively, if the prompt and LLMs are chosen correctly, LLMs can solve the transverse beam parameter tuning task, and in extension accelerator tuning and general optimisation. However, it is also found that the performance of this method lags significantly behind existing methods, especially RLO, and that in light of its resource demands, the results do not justify its use.

# Chapter 5

## FEL Tuning

Taking the lessons learned from developing and evaluating RLO for transverse beam tuning in Chapter 4, the same methodology is applied to the more complex task of maximising the FEL pulse intensity at LCLS in this chapter. The fundamental setup up of this task is introduced in Section 2.2.4. In contrast to the transverse beam tuning task, the FEL tuning task is characterised by a higher-dimensional action space and significantly more complex relationship between the actuators and their effect on the objective, i.e. the pulse intensity. This affects not only the overall difficulty of the task, but also the complexity of modelling its dynamics. Remember that a fast yet accurate-enough model for training in simulation was a crucial prerequisite for the successful results in the transverse beam tuning task. What is more, an ideal solution should be robust enough that at least the training setup, possibly even trained policies, can be transferred to other similar facilities, such as the European XFEL or FLASH in future work, without having to repeat much of the development process. Note also that the results of FEL intensity tuning have a much more immediate impact on the experimental performance and scientific output of the facility than they did with the transverse beam tuning task. Where the latter represents an intermediate step in the process of setting up the machine, the FEL intensity is a direct factor in the quality of the conducted experiments, and one of the main performance indicators of the facility.

In the following, challenges and solutions associated with the more complex modelling of the FEL processes are discussed in Section 5.1, followed by the development and evaluation of an RLO agent in Section 5.2. In Section 5.3, the challenge of high demands for training samples due to the increased complexity is addressed using gradient-based RL on same task, which is made possible by the automatic differentiation capabilities of the Cheetah simulation code introduced in Section 3.2. Finally, the results of both approaches are evaluated and discussed in Section 5.4.

### 5.1 Modelling FEL at LCLS

In order to train a policy in simulation and then transfer it to the real machine, an accurate-enough model of the FEL process is required. For the concrete case of the LCLS FEL, this model is made up of two parts.

The first is the beam dynamics model,

$$\hat{f}_t(\mathbf{u} \mid \mathbf{t}_0) \approx f_t(\mathbf{u} \mid \mathbf{t}_0, \mathbf{v}_{\text{linac}}), \quad (5.1)$$

which approximates  $f_t$  describing the evolution of the electron beam's Twiss parameters just upstream of the first quadrupole magnet to the undulator entrance. This model makes

two simplifying assumptions: Firstly, all actuators other than the 14 quadrupole magnets actuated as part of this tuning task, denoted  $\mathbf{v}_{\text{linac}}$  are assumed to be constant at their design values. Secondly, for computational efficiency, the model assumes linear beam dynamics. The true function  $f_t$  is affected also by collective effects, such as space charge, coherent synchrotron radiation and wakefields, which are not included in the model. At SLAC, a Bmad [275] model is available. However, interfacing between Bmad and a Gymnasium environment in Python is relatively slow when compared to the speed required to make RL feasible. Therefore, a Cheetah model is generated from the Bmad model, which can be used to simulate the beam dynamics at much higher speeds. To this end, Cheetah was extended to support the conversion of a Bmad lattice file to a Cheetah model.

The second part of the model is the FEL process  $f_I(\mathbf{t} | \mathbf{v}_{\text{und}})$  itself. Capable codes, such as *GENESIS* [257], are available to accurately simulate the FEL process. However, these codes take notoriously long to run, sometimes requiring hours for a single simulation. To make RL training on the model feasible, the FEL process is instead modelled through the scaled probability density function of a multivariate Gaussian distribution centred around the design Twiss parameters at the entrance of the undulator, with the undulator settings  $\mathbf{v}_{\text{und}}$  assumed constant at their design values. Considering the probability density function of a multivariate Gaussian distribution

$$f_N(\mathbf{x} | \boldsymbol{\mu}, \Sigma) = \frac{\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)}{\sqrt{(2\pi)^k |\Sigma|}}, \quad (5.2)$$

where  $\boldsymbol{\mu}$  is the mean vector,  $\Sigma$  is the covariance matrix,  $k$  is the dimensionality of the distribution and  $|\Sigma|$  is the determinant of the covariance matrix, an  $f_N$  with appropriate mean and covariance is chosen to match the expected behaviour of the FEL process. That is, the pulse intensity is given by

$$I = f_I(\mathbf{t} | \mathbf{v}_{\text{und}}) \approx \hat{f}_I(\mathbf{t}) = f_N(\mathbf{t} | \hat{\mathbf{t}}, \Sigma) \frac{I^*}{f_N(\hat{\mathbf{t}} | \hat{\mathbf{t}}, \Sigma)}, \quad (5.3)$$

with  $\Sigma$  estimated from experience as

$$\Sigma = \begin{pmatrix} 25 \text{ m}^2 & 0 & 0 & 0 \\ 0 & 0.16 & 0 & 0 \\ 0 & 0 & 625 \text{ m}^2 & 0 \\ 0 & 0 & 0 & 0.16 \end{pmatrix} \quad (5.4)$$

and  $I^* = 5 \text{ mJ}$  the estimated maximum pulse intensity as measured on the GDET. In Eq. (5.3), the factor

$$\frac{I^*}{f_N(\hat{\mathbf{t}} | \hat{\mathbf{t}}, \Sigma)} \quad (5.5)$$

ensures that at the design Twiss parameters  $\hat{\mathbf{t}}$ , the pulse intensity is equal to the optimal pulse intensity  $I^*$ . The combined model mapping actuator settings to the FEL pulse intensity considered in the following sections is consequently given by

$$\hat{f}_{\text{cls}}(\mathbf{u} | \mathbf{t}_0) = \hat{f}_I\left(\hat{f}_{\hat{\mathbf{t}}}(\mathbf{u} | \mathbf{t}_0)\right). \quad (5.6)$$

## 5.2 Gradient-free Reinforcement Learning

As in Section 4.1, solving the FEL tuning task first requires appropriate definitions of the action, observation and reward function that make up the RL problem formulation.

The action definition for FEL tuning follows the principles outlined for the transverse beam tuning task in Section 4.1. The policy is intended to control the 14 quadrupole magnets commonly used for tuning the FEL process at LCLS as introduced in Section 2.2.4. That is, the actuators are

$$\mathbf{u} = (k_{Q_1}, \dots, k_{Q_{14}}), \quad (5.7)$$

with  $k_{Q_i}$  the focusing strength of the  $i$ -th quadrupole magnet in  $\text{m}^{-2}$ . From theory, design values  $\hat{\mathbf{u}}$  for the quadrupole magnet strengths are known. If the real machine were exactly identical to the simulation and in the design state, the optimal actuators would be  $\mathbf{u}^* = \hat{\mathbf{u}}$ , i.e. identical to the design actuators. However, various factors such as manufacturing tolerances, environmental effects and deviations from the design state for specialised working points mean that the optimal actuators  $\mathbf{u}^*$  deviate from the design values. It is reasonable to assume that the optimal actuators are within a range

$$[\hat{\mathbf{u}} - \epsilon_u, \hat{\mathbf{u}} + \epsilon_u] \quad (5.8)$$

around the design values, where for the RL training,  $\epsilon_u$  is chosen as

$$\epsilon_u = (0.13, 0.26, 0.52, 0.26, 0.26, 0.26, 0.26, 0.26, 0.26, 0.26, 0.07, 0.07, 0.07, 0.13) \text{m}^{-2}. \quad (5.9)$$

These ranges are chosen such that changing the design settings by a  $\Delta\mathbf{u}$ , where  $\Delta\mathbf{u}_i = \epsilon_{u,i}$  and  $\Delta\mathbf{u}_j = 0$  for all  $j \neq i$ , for any quadrupole magnet  $Q_i$  results in a pulse intensity  $I \approx 0$  mJ. Following the scheme introduced in Section 4.1, the actions  $\mathbf{a}_t$  are defined to be continuous as the change in strength for each quadrupole magnet settings, i.e.

$$\mathbf{a}_t = \Delta\mathbf{u}, \quad (5.10)$$

such that

$$\mathbf{u}_{t+1} = \mathbf{u}_t + \mathbf{a}_t. \quad (5.11)$$

For the FEL tuning task, the actions are normalised to 10% of the range around the design values, i.e.  $\mathbf{a}_t \in [-0.1\epsilon_u, 0.1\epsilon_u]$ .

The observations  $\mathbf{o}_t$  are defined to include the focussing strengths of the quadrupole magnets  $\mathbf{u}_t$  and the measured pulse intensity  $I_t$  at the end of the undulator line. As the goal is always to maximise the pulse intensity, unlike in the transverse beam tuning task, no target pulse intensity is included in the observation. What is more, while the Twiss parameters at the undulator entrance as well as upstream of the first quadrupole magnet would be useful for tuning, they cannot feasibly be measured on the real machine and are therefore not included in the observation. This gives a final observation definition of

$$\mathbf{o}_t = (I_t, \mathbf{u}_t), \quad (5.12)$$

which is normalised to by rescaling to a range  $[-1, 1]$  assuming the unscaled observation's range is  $[0, I^*]$  for the pulse intensity and  $[\hat{\mathbf{u}} - \epsilon_u, \hat{\mathbf{u}} + \epsilon_u]$  for the quadrupole magnet strengths.

The reward function largely follows the scheme introduced in [59] and Section 4.4.1. The latter allows for the straightforward combination of multiple objectives into a single scalar

reward value between 0 and 1. Measured objective values are transformed into quality measures between 0 and 1 using a non-linear function, such as a softplus or sigmoid. The individual quality measures are then combined into a single scalar reward value using a combiner, such as a weighted smooth max or geometric mean, which allows for weighting the individual components.

Three objectives are considered for the FEL tuning task. The main objective is to maximise the pulse intensity. It is transformed from its range  $[0, I^*]$  through a clipped linear function

$$r_I = \begin{cases} 1 & \text{if } \frac{I}{I^*} \geq 1 \\ \frac{I}{I^*} & \text{otherwise} \end{cases} \quad (5.13)$$

to a quality measure between 0 and 1. This resembles the beam reward in Section 4.4.1, but differs in the input range and the direction of the slope. Note that intensities larger than  $I^* = 5 \text{ mJ}$  should only occur as a result of noise from real sensor readings.

Two secondary objectives are defined to ensure that the magnet settings are only changed if necessary and that they remain reasonably close to the design values. The magnet change reward  $r_a$  is defined the same as it is in Section 4.4.1, but with the action components normalised to a different maximum action of  $0.1\epsilon_u$ . It results

$$\tilde{a}_{t,i} = -\ln(19) - \left(1 - \frac{|a_i|}{0.1\epsilon_{u,i}}\right) (-2\ln(19)) \quad (5.14)$$

such that, when transformed through a sigmoid function

$$r_{a_i} = \frac{1}{1 + \exp(-\tilde{a}_{t,i})} \quad (5.15)$$

a bad action component of  $0.1\epsilon_{u,i}$  is mapped to a partial reward of 0.05 and a good normalised action of 0 is mapped to a partial reward of 0.95. The transformed partial rewards are then combined by taking a uniformly weighted mean

$$r_a = \frac{1}{14} \sum_{i=1}^{14} r_{a_i}. \quad (5.16)$$

Similarly, the absolute value of the normalised magnet settings  $|\mathbf{u}_{\text{normed}}|$ , i.e. the magnet settings shifted onto the design magnet settings  $\mathbf{u} - \hat{\mathbf{u}}$  and scaled from  $[-\epsilon_u, \epsilon_u]$  to  $[-1, 1]$ , are transformed through a softplus function

$$r_{u_i} = 2 \cdot \frac{1}{1 + \exp(\tilde{\mathbf{u}}_i)} \quad (5.17)$$

where

$$\tilde{\mathbf{u}}_i = -\ln(19) \frac{u_i - 0.1\epsilon_{u,i}}{-0.1\epsilon_{u,i}} \ln(19), \quad (5.18)$$

to a quality measure between 0 and 1, such that the design setting results in a reward of 1 and the maximum deviation results in a reward of 0.1. The individual magnet setting reward components are combined by taking a uniformly weighted mean

$$r_u = \frac{1}{14} \sum_{i=1}^{14} r_{u_i}. \quad (5.19)$$

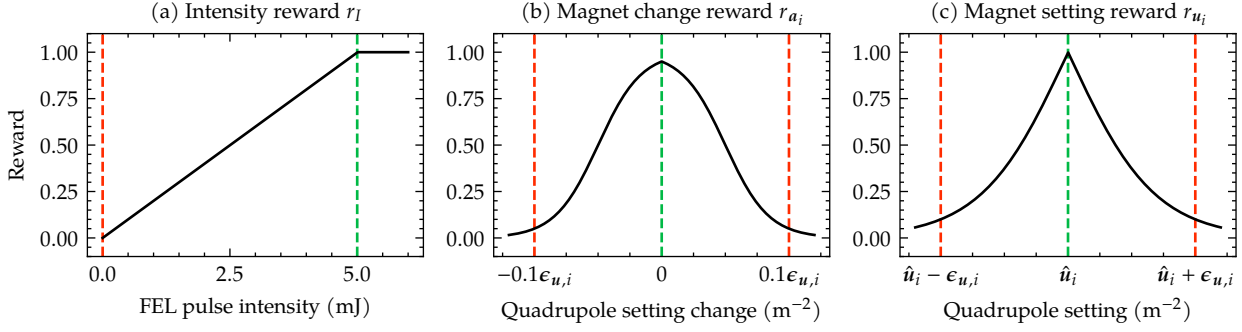


Figure 5.1: Illustration of the transforms applied to the different reward components. (a) shows the clipped linear transform applied to FEL pulse intensity in order to map the error to a reward in the range  $[0, 1]$ . Note that intensities larger than 5 mJ should only occur as a result of noise from real sensor readings. (b) shows the sigmoid transform applied to the change in the setting of a quadrupole magnet in order to map the change to a reward in the range  $[0, 1]$ . (c) shows the softplus transform applied to the quadrupole magnet setting based on the design value in order to map the deviation to a reward in the range  $[0, 1]$ . Green dashed lines in all subplots indicate a good result that should map to a reward near 1, while red areas and red dashed lines indicate a bad result that should map to a reward near 0.

The final reward function is then given by a weighted mean of the three reward components pertaining to the three respective quality measures

$$r_t = \frac{w_I r_I + w_a r_a + w_u r_u}{w_I + w_a + w_u}. \quad (5.20)$$

These reward transforms and combiners were chosen based on experiments from a set of possible transforms and combiners devised in [59] and found to perform well. The significantly lower weights of the secondary objectives compared to the main objective ensure that the policy is primarily optimised to maximise the pulse intensity, and prevents the policy from keeping the quadrupole strengths constant at the design values if this would result in a lower pulse intensity. Visualisations of the transforms applied to the individual reward components  $r_I$ ,  $r_{a_i}$  and  $r_{u_i}$  are shown in Fig. 5.1. The resulting FEL tuning RL loop is illustrated in Fig. 5.2.

As with the transverse beam tuning task in Section 4.1, domain randomisation [239, 323] is used to enable a zero-shot transfer of the policy to the real machine. This becomes especially important for the FEL tuning task, because the strong simplification of the FEL model means that the sim-to-real gap is likely to be larger than for the transverse beam tuning task. The most significant deviation is expected in the incoming beam's Twiss parameters  $t_0$ , which are unlikely to match the design values exactly, cannot be easily observed and may change over time. Based on the design values

$$\hat{t}_0 = (\beta_x, \alpha_x, \beta_y, \alpha_y) = (61.5 \text{ m}, -1.21, 35.4 \text{ m}, 0.67), \quad (5.21)$$

it is found through experiments on the model developed in Section 5.1 that respective changes in either direction of

$$\epsilon_{t_0} = (\Delta\beta_x, \Delta\alpha_x, \Delta\beta_y, \Delta\alpha_y) = (67.1 \text{ m}, 1.05, 67.1 \text{ m}, 2.1) \quad (5.22)$$

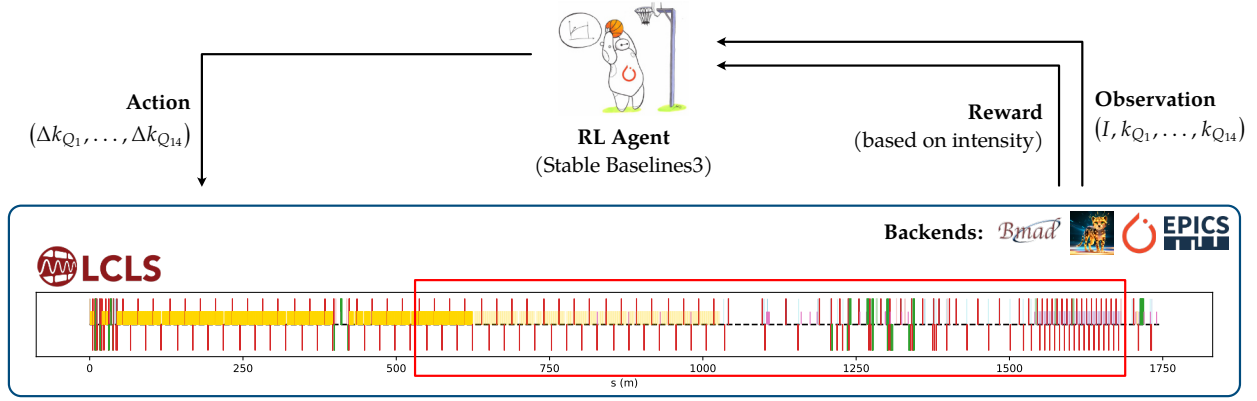


Figure 5.2: RL loop for the FEL intensity tuning task at LCLS. The policy provides an action in the form of quadrupole strength changes to the environment. The environment during training is modelled by a Cheetah simulation of the beam dynamics with a Gaussian approximation of the FEL process, but can also be replaced with a neural network surrogate, a Bmad simulation or connect to the real machine through the EPICS control system. The observation of FEL intensity and quadrupole strengths, as well as the intensity-based reward are returned to the agent.

at the design quadrupole magnet settings  $\hat{u}$  result in a pulse intensity of approximately 0 mJ. Considering that  $\beta_x$  and  $\beta_y$  must physically be positive, the domain randomisation ranges for the incoming Twiss parameters are chosen as

$$[t_{\min}, \hat{t}_0 + \epsilon_{t_0}], \quad (5.23)$$

where

$$t_{\min} = \max\left(\hat{t}_0 - \epsilon_{\hat{t}}, \left(10^{-20} \text{ m}, \hat{t}_{0,2} - \epsilon_{t_{0,2}}, 10^{-20} \text{ m}, \hat{t}_{0,4} - \epsilon_{t_{0,4}}\right)\right). \quad (5.24)$$

The resulting ranges are

$$\begin{aligned} \beta_x &\in [10^{-20} \text{ m}, 128.6 \text{ m}], \\ \alpha_x &\in [-2.26, 0.16], \\ \beta_y &\in [10^{-20} \text{ m}, 102.5 \text{ m}], \\ \alpha_y &\in [-1.43, 2.77]. \end{aligned} \quad (5.25)$$

The relatively high dimensionality of the tuning task makes it exponentially harder to solve than the transverse beam tuning task. This is further amplified by the high non-linearity of the FEL process and its narrow range of Twiss parameters at the undulator entrance that result in lasing. Because of this, the set of actuator settings that lead to meaningful rewards from the pulse intensity objective is very narrow, resulting in sparse rewards. Gathering enough samples to learn a policy that is capable of maximising the pulse intensity consequently requires a very large number of environment interactions. In addition, the domain randomisation range for the incoming Twiss parameters is entirely derived from the simplified model of the FEL process. It is unclear how much the Twiss parameters upstream of the first quadrupole magnet actually deviate in day-to-day operations. A dedicated study to determine the actual range of Twiss parameters would be costly, and the required data cannot be acquired parasitically during regular operation.

To address the issue of uncertain incoming Twiss ranges, and to improve sample efficiency and reduce training times, a simple variant of automatic domain randomisation

(ADR) [239] is employed. With the FEL tuning task, actual lasing in the undulator, and with that a non-zero pulse intensity, is only achieved for a very narrow range around the correct magnet settings. As a result, when the agent starts training with the full range of domain-randomised incoming Twiss parameters, it is very unlikely to encounter meaningful rewards. By starting the training without domain randomisation at the design incoming Twiss parameters and at the design magnet settings, where the FEL is already maximised, and then gradually growing the domain randomisation ranges, the agent is guaranteed to encounter meaningful rewards and can gradually learn to maximise the pulse intensity from increasingly suboptimal starting conditions. This approach resembles *curriculum learning* [20], where the difficulty of the task is incrementally increased on a static schedule to make learning easier. The scheduling of a curriculum, however, is non-trivial and usually requires a lot of tuning to make sure that the curriculum does not increase in difficulty too quickly or too slowly. What is more, how fast agents learn can differ between trainings with different random seeds, making it hard to find one curriculum that generalises well. ADR address these issues by increasing the domain randomisation range adaptively based on the performance of the agent. A difficulty factor  $d$  is introduced that scales the domain randomisation range for the incoming Twiss parameters, such that the actual domain randomisation range adapted from Eqs. (5.23) and (5.24) is given by

$$[\mathbf{t}_{\min}, \hat{\mathbf{t}}_0 + d\boldsymbol{\epsilon}_{t_0}], \quad (5.26)$$

with

$$\mathbf{t}_{\min} = \max\left(\hat{\mathbf{t}}_0 - d\boldsymbol{\epsilon}_{t_0}, \left(10^{-20} \text{ m}, \hat{\mathbf{t}}_{0,2} - d\boldsymbol{\epsilon}_{t_0,2}, 10^{-20} \text{ m}, \hat{\mathbf{t}}_{0,4} - d\boldsymbol{\epsilon}_{t_0,4}\right)\right). \quad (5.27)$$

Starting from an initial difficulty factor  $d_0$ , the difficulty is increased by a step size  $\Delta d$  every time the mean return of the last 100 episodes exceeds a threshold  $R_{\text{threshold}}$ , until a maximum difficulty  $d_{\max} = 1$  is reached.

Note that the policy evaluated in the following was not trained using ADR directly. Instead, it was trained with a fixed difficulty factor of  $d = 0.08$ . The latter was determined through previous trainings with ADR, where  $d = 0.08$  was found to be the largest that policies could reliably solve. For these prior trainings, the curriculum step size was set to  $\Delta d = 0.01$  and the cumulative reward threshold to  $R_{\text{threshold}} = 8$ . Experts at LCLS were consulted and confirmed that the resulting domain randomisation range is reasonable, though this cannot be backed up by data. Most intriguingly, this approach forgoes the need to manually determine domain randomisation ranges that closely match the real conditions. Instead, the ranges are chosen based on the RL algorithm's abilities. Ideally, the ranges found through ADR are wider than the true ranges, and one never has to determine the latter. However, it is possible that the ranges found through ADR are narrower than the true ranges. In this case, the policy will not be able to generalise to the real machine, and one must develop a more capable RL algorithm.

The PPO [292] algorithm is used to train the policy, using the implementation provided by *Stable Baselines3* [254]. The policy is trained for 50 million samples on the simulated model of the FEL at LCLS, which was implemented in a *Gymnasium* [328] environment. Like in Chapter 4, a swappable simulation backend is used to enable the training in simulation and transfer to the real machine. Hyperparameters for the PPO algorithm, the neural networks as well as the environment were selected via a hyperparameter exploration in simulation using *Weights & Biases* [24] for experiment tracking. The final hyperparameters are listed in Table 5.1. Training was run on a high-performance cluster node with a 128-core AMD EPYC 75F3 CPU and 512 GB of RAM, resulting in a total training time of 1 d 16 h.

Table 5.1: Hyperparameters used to train the FEL tuning policies

Parameter	Value
Batch size $ \mathcal{B} $	128
Learning rate (actor & critic)	Linearly decaying from 0.0003
Discount factor $\gamma$	0.99
Number of concurrent environments	40
Number of rollout steps	128
Entropy coefficient $c_2$	0.01
Value function coefficient $c_1$	0.5
Number of training epochs	10
Clip range $\epsilon$	0.2
Layer widths	(64, 64)
Training steps	50 000 000
Maximum steps per episode	10
Initial log standard deviation	-2.3
Reward weights $(w_I, w_a, w_u)$	(3, 0.5, 0.5)

Table 5.2: Performance of FEL tuning policies at LCLS

Metric	Gradient-free	Gradient-based
Final pulse intensity	4.90 mJ	4.85 mJ
RMSE to target intensity	0.77 mJ	0.81 mJ
Normalised pulse intensity recovered	93.7 %	90.0 %
Steps to target	3	7
Proportion of successful trials	98.3 %	65.0 %

## Evaluation

The trained policy was then evaluated on 300 trials defined by randomly sampled incoming Twiss parameters  $t_0$  from the domain randomisation range defined in Eq. (5.26). The results of this evaluation are listed in Table 5.2, reporting the medians over the trials in terms of the metrics from Section 4.2.2 adapted to the FEL tuning task. The relationship between the speed of the policy and the results achieved is visualised in Fig. 5.3. Instead of the final error, the final pulse intensity is reported. The RMSE is computed with respect to the maximum pulse intensity of 5 mJ. The normalised improvement is computed as the proportion of the pulse intensity recovered from the detuned state to the optimal pulse intensity. For the steps to target and the corresponding success rate, a threshold pulse intensity of 4.9 mJ is assumed, with the target considered achieved if a higher pulse intensity is reached. Note that the *minimum* metrics from Chapter 4 may be referred to as *maximum* metrics in this chapter, as the goal of the task considered here is maximisation.

The results show that the trained policy is capable of tuning the FEL to a median pulse intensity of 4.9 mJ, which is only 2 % below the optimal pulse intensity of 5 mJ. Starting from the initial detuned state, the policy is able to improve the pulse intensity by a median of 1.48 mJ, which corresponds recovering 93.7 % of pulse intensity lost by the detuning of the incoming beam. These results are achieved reliably, with the policy achieving a threshold intensity of 4.9 mJ in 98.3 % of the trials, and quickly, with a median of only 3

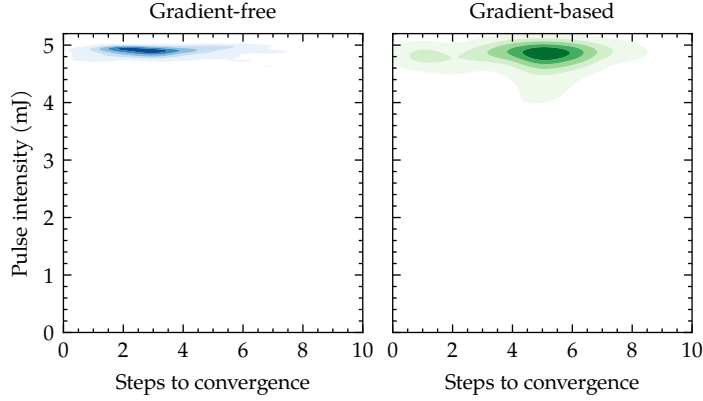


Figure 5.3: Distributions of the final FEL pulse intensity achieved by the gradient-free and gradient-based policies plotted over the steps to convergence. KDEs are used to visualise the distributions. Ideally, distributions should be concentrated in the upper left, indicating that policies converge quickly to good solutions. Distributions in the lower left converge quickly but to bad solutions, while distributions in the upper right converge slowly to good solutions. The steps to convergence are measured on the maximum intensity to guarantee that all runs converge and are included in the distribution. An equivalent figure with the maximum intensity is available in Appendix B.

steps to reach the threshold intensity. An example of a tuning run by a trained policy is shown in Fig. 5.4.

### 5.3 Gradient-based Reinforcement Learning

Most of the commonly used RL algorithms are based on the policy gradient theorem, which states that the policy gradient can be estimated as the empirical average over the gradients of the log-likelihood of the policy multiplied by the cumulative reward or advantage, along the lines of Eq. (2.23). However, this formulation suffers from high variance and relies on large sample sizes to give a stable and accurate estimate of the policy gradient. Nevertheless, such an approach is necessary when the environment is a black box and the exact gradient cannot feasibly be computed.

Fortunately, the model for the FEL process introduced in Section 5.1 is not a black box. What is more, it is constructed using Cheetah, which natively supports automatic differentiation. This means that the exact policy gradient is straightforward to compute as the gradient of the cumulative reward with respect to the policy parameters

$$\nabla_{\theta} R = \nabla_{\theta} \sum_{t=0}^T \mathcal{R}(s_t, \pi_{\theta}(s_t)), \quad (5.28)$$

because the environment and therefore the reward function in addition to the policy are fully differentiable as well as implemented in frameworks that support automatic differentiation. This allows for the use of the true policy gradient, which should make training significantly more sample efficient. The principle of taking the true policy gradient through both the policy and the accelerator model is illustrated in Fig. 5.5.

To train a policy on the FEL tuning task using gradient-based RL, the combined policy-accelerator model and the training are implemented using PyTorch Lightning [87]. In this

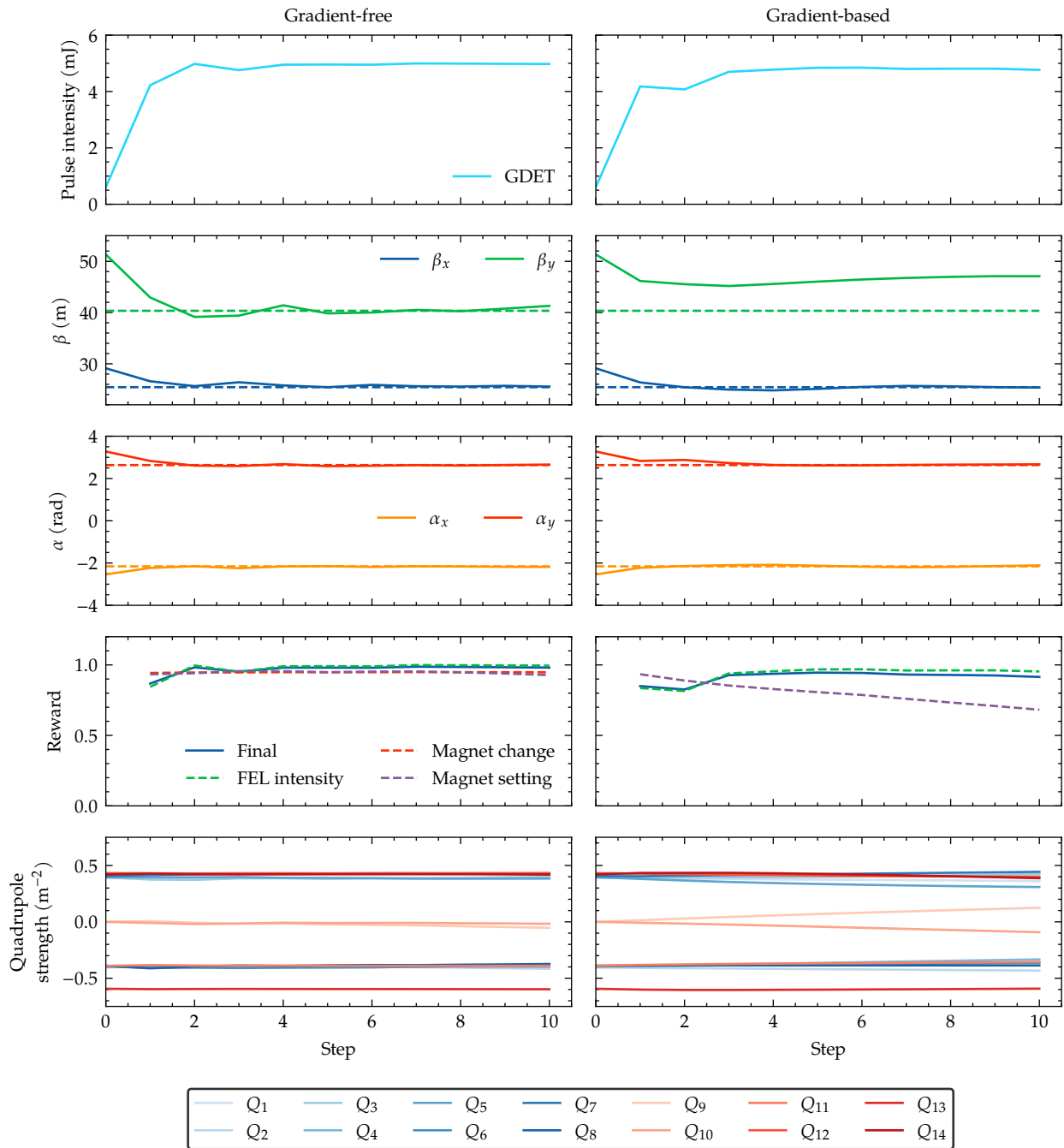


Figure 5.4: Example FEL tuning runs by two trained policies, one trained using a gradient-free approach following Section 5.2 and the other following a gradient-based approach according to Section 5.3. Dashed lines for  $\beta$  and  $\alpha$  indicate their respective design values, which in the model from Section 5.1 result in maximum pulse intensity. Both examples assume the same trial sampled from the evaluated set of 300 trials.

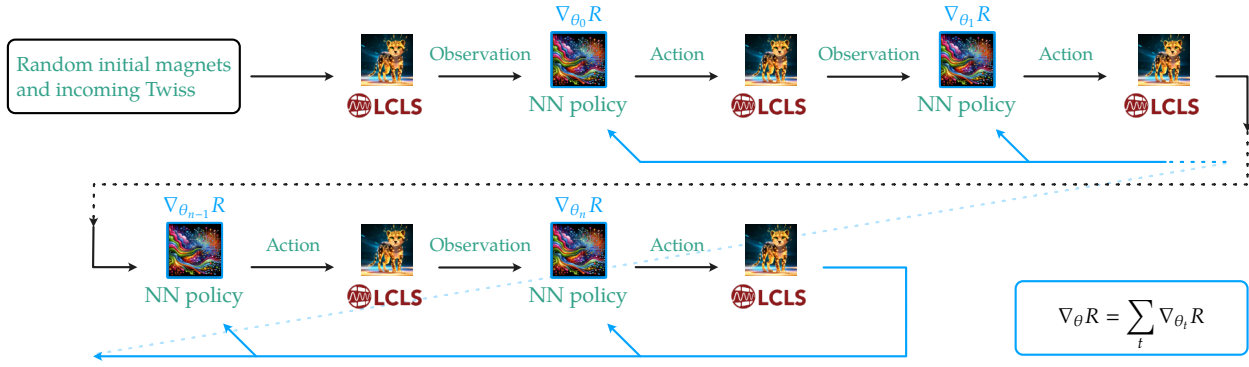


Figure 5.5: Gradient-based RL rollout graph. The computation graph is rolled out over one episode of  $n$  steps, with the policy being used to compute an action from an observation in each step. The gradient of the cumulative reward  $R$  with respect to the policy parameters  $\theta$  is then calculated using backpropagation through the policy and the FEL model, with the final gradient being the sum of all step-wise gradients.

case, the reward function is simplified to

$$r_t = \frac{w_I \frac{1}{I^2} + w_u \sqrt{\sum_{i=1}^{14} \left( \frac{u_i - \hat{u}_i}{\epsilon_{u,i}} \right)^2}}{w_I + w_u}, \quad (5.29)$$

with the reward component weights  $w_I$  and  $w_u$  chosen as 3 and 0.5, respectively, and  $\epsilon_{u,i}$  the half width of the actuator space around the design actuator values  $\hat{u}$ . The loss function for the policy is defined as the negative cumulative reward. The resulting policy gradient algorithm is provided in Algorithm 4.

---

**Algorithm 4** Gradient-based policy optimisation algorithm.

---

**for** iteration = 1, 2, ... **do**

    Compute cumulative reward  $R = \sum_{t=0}^T \mathcal{R}(s_t, \pi_{\theta}(s_t))$

    Compute the policy gradient  $\nabla_{\theta}$  of  $R$  w.r.t.  $\theta$  through automatic differentiation

    Update  $\theta$  following Adam update rule with  $\nabla_{\theta}$  and the learning rate  $\alpha$

**end for**

---

## Evaluation

The policy is trained on 35 510 samples, reaching the same reward threshold in 45 times fewer samples than the gradient-free approach using the PPO algorithm, as can be seen in the return over training curves in Fig. 5.6. The hyperparameters used for the gradient-based RL training are listed in Table 5.3.

The policy trained using the gradient-based approach is evaluated on the same 300 trials as the gradient-free policy from Section 5.2. The results are provided in Table 5.2. While requiring significantly fewer samples and wall-clock time to train, and being trained through a simple policy gradient method, the gradient-based policy is found to achieve comparable performance to the gradient-free policy. The median final pulse intensity is 4.85 mJ, which is very nearly the same as the gradient-free policy. The most significant difference between the two policies is the proportion of trials on which the policies achieve

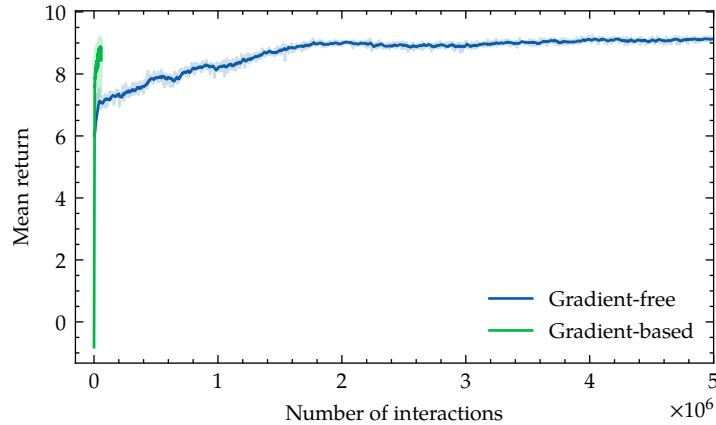


Figure 5.6: Mean return learning curves of gradient-free vs. gradient-based training. Smoothed curves are shown in the respective darker shades of the same colour, while the raw curves are shown in the lighter shades. Note that as a result of the reward formulation and the choice of 10 steps per episode, it is known that the optimal return is 10.

Table 5.3: Hyperparameters used for gradient-based training

Parameter	Value
Batch size $ \mathcal{B} $	16
Hidden activation function	Leaky ReLU
Layer widths	(48, 48, 48)
Learning rate $\alpha$	0.000 15
Maximum episode steps	10
Number of training epochs	10 000

the threshold pulse intensity of 4.9 mJ, which reduces from 98.3% for the gradient-free policy to 65.0% for the gradient-based policy. However, this difference shrinks when considering a threshold pulse intensity of 4.8 mJ, which is achieved in 100% of the trials by the gradient-free policy and in 93.3% of the trials by the gradient-based policy. An example of a tuning run by the gradient-based policy is shown in Fig. 5.4.

## 5.4 Summary

In this chapter, two different approaches to training a policy for tuning the FEL at LCLS were presented, one using a gradient-free RL setup based on the one introduced in Chapter 4 and another using a novel gradient-based approach enabled by a physics-based differentiable model of the FEL tuning RL environment with a Cheetah implementation of the latter that supports automatic differentiation. It is shown that both approaches are capable of training a policy that can tune the FEL to a near-optimal pulse intensity in just a few steps, demonstrating that the RL framework developed in Chapter 4 scales well to significantly more complex tasks, like the highly-relevant FEL tuning task.

What is more, the gradient-based approach is shown to reduce the number of required training samples and consequently the wall-clock time to train the policy by a factor of 45 compared to the gradient-free approach. This is especially relevant in the context of more complex accelerator tuning tasks such as this one, where the time required to train

policies in a gradient-free setting can grow to multiple days and up. More so, the gradient-based approach was able to achieve this improvement using a very simple policy gradient approach over the gradient of the episodic return with respect to the policy parameters, meaning that algorithmic improvements could further reduce the training time required to train a policy for the FEL tuning task.



# Chapter 6

## Conclusion

This chapter concludes the dissertation by summarising the main contributions, results and achievements from the presented work in Section 6.1, discussing the implications of these results in the broader context of accelerator tuning and control in Section 6.2 and providing an outlook on future work and remaining challenges in Section 6.3.

### 6.1 Summary

The goal of the work presented in this dissertation was to overcome the difficulties of applying RL to particle accelerator tuning and control. These difficulties include in particular the challenges of training RL policies on accelerators, both in terms of the cost of beam time and the safety risks associated with training on real accelerators. This work further set out to investigate the suitability of RL in the context of more complex accelerator-related tasks, and compare it to state-of-the-art optimisers like BO.

As such, a simulation-based training setup was developed, which allows training RL policies for accelerator tuning and control on a high-speed simulator, called Cheetah, without the need for real-world training. This setup reduces training times from years of real-world beam time to just hours or minutes of computation time. The setup is augmented with domain randomisation over potentially mismatched parameters of the model, which enables a zero-shot transfer of the trained policies to the real accelerator, completely eliminating the need for beam time to perform RL training.

At the example of a transverse beam tuning task at the ARES accelerator at DESY, it is shown that this setup performs excellently. This validates not only the sim-to-real transfer of the proposed training setup, but also demonstrates that RL is indeed capable of solving more involved accelerator tuning tasks. Most notably, it is demonstrated in studies on the real accelerator that the trained policies find actuator settings that are comparable in quality to those found by expert human operators. Impressively, the RLO policies find these settings about four times faster than the human operators, despite hardware limitations that limit their interaction rate to about an order of magnitude slower than that of the human operators. Moreover, the variance between policies trained on different random seeds is found to be much lower than the variance between different expert human operators, indicating that the use of autonomous tuning methods based on RLO will lead to better reproducibility of experimental results for accelerator users.

An extensive comparative study is conducted, comparing the performance of the RLO setup to state-of-the-art optimisers like BO, as well as more conventional optimisers like ES and Nelder-Mead simplex. The RLO setup is found to outperform all other optimisers

in terms of final tuning results, convergence speed and robustness. Additional tests also showed that, unlike dedicated optimisers, the trained policies are capable of compensating for different timescale drifts and defects of accelerator components, which is not trivially achievable with other optimisers. Most impressively, this is achieved despite the fact that drift compensation and defect correction are not explicitly part of the training and hence out-of-distribution tasks.

Nevertheless, the large requirement for upfront engineering and the limited transferability of the trained policies to different tasks or accelerators are identified as key remaining drawbacks of RLO. To address this issue, multiple avenues of transferability are explored. Akin to work performed on Atari games [223] as well as the games of Go, Chess and Shogi [306], the performance of retraining policies for transverse beam tuning on different lattices is investigated. To this end, policies are trained on three other sections of the ARES accelerator. It is found that the performance of the trained policies remains comparable, all the while keeping the training algorithm, hyperparameters, policy architecture and reward function unchanged. This indicates that, if the target task is somewhat similar in nature, well-designed training setups can generalise, such that one can forgo much of the manual engineering and limit development efforts to only the actual training, which is equated primarily to only computational costs. Moreover, it is shown that the method of domain randomisation can be extended to the lattice geometry. Policies trained in this way are found to be lattice-agnostic, generalising well to different lattices without the need for additional training, even when the lattices are to a limited extent out-of-distribution with respect to the training distribution. However, limitations of this approach are also identified. Most critically, similar tuning tasks on accelerators can differ in many ways, and not all of them can easily be accounted for through domain randomisation. In such cases, the performance of the trained policies may degrade and the training of separate policies may be required.

A further avenue of research is explored, investigating whether LLMs can be used for accelerator tuning. Doing so would potentially reduce the engineering effort required to deploy autonomous accelerator tuning solutions to new tasks to only the formulation of a natural language prompt, i.e. less than is even required for some of the simplest black-box optimisation algorithms, especially in terms of operator expertise about the optimisation algorithm itself. It is shown that, provided the right choice of model and prompt, LLMs can successfully be used for accelerator tuning. This is an intriguing result, as it suggests that LLMs possess the general intelligence required to perform numerical optimisation on multivariate black-box objective functions, only from training for text completion. However, while LLMs are found to be capable of solving the transverse beam tuning task, their performance lags behind that of the best state-of-the-art tuning algorithms like RLO and BO. Moreover, LLMs are found to incur high computational costs, leading to long inference times, high monetary costs and significant environmental impact. Not a viable alternative to state-of-the-art accelerator tuning algorithms yet, it is however found that the accelerator tuning performance of LLMs improves with an LLM's overall performance in terms of LLM benchmarks, suggesting that the fast-paced progress in the field of LLMs will make them a viable alternative in the future.

Based on the findings in the context of the transverse beam tuning task, the RLO setup is transferred to the task of maximising the FEL pulse intensity at LCLS. This is a significantly more complex task and a much more high-impact application of RL to accelerators, as the FEL pulse intensity is a key performance indicator for FELs and directly impacts the scientific output of these facilities. It is found that trained policies are capable of achieving

near-optimal FEL pulse intensity in just a few steps, despite the significantly increased complexity of the task. This demonstrates that the RLO setup is capable of transferring well to more complex tasks, even when the tasks are very different in nature. However, the increased task difficulty is found to increase simulated training times to the order of days. As the complexity of accelerator tuning and control tasks increases, this may become a limiting factor for the applicability of RL. Hence, a gradient-based RL setup is proposed, which uses the differentiable simulator Cheetah to provide the true gradient of the FEL intensity-based cumulative rewards with respect to the policy parameters. This reduces the number of required training samples by almost two orders of magnitude compared to the original gradient-free RLO setup, presumably as a result of the reduced variances and better accuracy of true the policy gradients compared to the policy gradient estimates obtained through the policy gradient theorem.

The RL-based results achieved in this work were only made possible by the development of the high-speed differentiable beam dynamics simulator Cheetah, a side product and integral component of this work. Cheetah is built on top of PyTorch and integrates a number of optimisations as well as inheriting support for GPU acceleration from PyTorch. This makes Cheetah orders of magnitude faster than conventional beam dynamics simulators, a key requirement for collecting the large amounts of data required by RL training. In the case of the transverse beam tuning task, for example, Cheetah enabled training a policy in less than an hour, instead of the multiple weeks it would have taken with the conventional beam dynamics simulator Ocelot and years of real-world beam time.

What is more, Cheetah also inherits PyTorch's automatic differentiation capabilities and native integration with the machine learning ecosystem around PyTorch. As a result, Cheetah is a first-of-its-kind differentiable beam dynamics simulator, making gradients a first-class citizen in accelerator simulations. Access to gradients of beam dynamics models unlocks a wide range of applications and possibilities for accelerator tuning, system identification and virtual diagnostics beyond the RL use case. Some of these are demonstrated in this work, including gradient-based accelerator tuning, gradient-based system identification and physics-based differentiable priors for BO. It is shown, for example, that Cheetah priors in BO can significantly reduce the number of samples required to find the optimal beam focussing through a FODO cell. Moreover, physics-based differentiable models like Cheetah are a critical prerequisite for tractable online reconstruction of the full 6-dimensional phase space of electron beams, a key challenge in accelerator diagnostics [272]. Its open-source availability and ease of use are already attracting interest from the wider accelerator community, and will fast-track the integration of gradient-based optimisation and RL into accelerator operations.

A data-driven path to fast differentiable accelerator modelling is also presented in this dissertation at the example of creating a virtual diagnostic of the LPS and bunch current profile at European XFEL. While an important step to providing the required sensory information for advanced accelerator tuning – both to human operators and as dimensionally reduced observations to RL policies – on their own, this part of the presented work demonstrates also a powerful option for creating fast-executing environments for RL training when otherwise expensive-to-simulate dynamics are involved.

Especially powerful is the combination of physics-based Cheetah models with modular reusable neural network surrogates of computationally expensive dynamics, an integration that is made possible by Cheetah's native interoperability with PyTorch. This work demonstrates an example of this so-called AI/ML coupling, where a modular neural network space charge model is trained to predict how space charge affects the beam when tracked

through a quadrupole magnet. It is shown that the resulting hybrid model accurately estimates space charge effects, while requiring three orders of magnitude less computation time than a conventional space charge simulation and fully preserving differentiability.

## 6.2 Discussion

The results presented in this dissertation mark a significant step forward in the application of RL to accelerator tuning and control, lifting the state of the art from simple tasks in simulation, to more complex high-dimensional tasks in the real world, which are more relevant to day-to-day accelerator operations. Not only do these results demonstrate that RL is capable of solving accelerator tuning tasks as they become more complex, but they also show that RL can outperform state-of-the-art optimisers like BO in terms of final tuning results, convergence speed and robustness. Most importantly, however, the presented approach of training in simulation and using domain randomisation is a key enabler for the application of RL to accelerators as well as other physical facilities.

RL, and RLO specifically for tuning, will play an important role in achieving the full potential of autonomous accelerators. That is not to say that RL will replace human operators, state-of-the-art optimisers like BO, or conventional methods like Nelder-Mead simplex and PID controllers as a one-size-fits-all solution. Even as further achievements are made in reducing the engineering effort required to deploy RL policies, RL will likely continue to require RL experts that develop and maintain the policies, and the cost of this engineering effort is not always justified by the complexity of the task at hand. Instead, RL will always be part of a broader toolkit and multi-tiered solution to accelerator tuning and control, where it will be used for complex and repeated tasks, interacting with conventional controllers on the lower levels of the control hierarchy and human operators on the higher levels, while state-of-the-art turn-key-type optimisers such as BO will be used for one-off tasks on demand. LLM copilots and logbook assistants, as well as federated RL setups might help coordinate these different components and provide a natural language interface for the human operators to control RL policies and other optimisers.

Beyond advancing the state-of-the-art in RL for accelerator tuning and control, the most impactful contribution of this work is the development of the high-speed differentiable beam dynamics simulator Cheetah. Not only is Cheetah a key enabler for RL in the context of accelerators, but it also unlocks a wide range of applications and possibilities for accelerator tuning, system identification and virtual diagnostics by providing straightforward access to gradients of beam dynamics models. The presented examples barely scratch the surface of what is possible with Cheetah, and it is expected that the accelerator community will find many more applications and use cases for it in the future. These will not only be part of building novel autonomous accelerator operation procedures, but also provide key insights to human operators and users about the accelerator's state and performance. Moreover, the speed and differentiability of Cheetah will also help in the development of novel accelerator facilities, where it can markedly reduce the computational cost of exploring the typically immense parameter space of new accelerator designs, while at the same time reducing the number of design iterations required and improving the discovered optimal design through the use of gradient-based optimisation.

Ultimately, the presented advancements in autonomous accelerator tuning and control will help increase the scientific output of accelerator facilities, both enabling so far unattainable operational configurations and reducing the time required for achieving experimental setups. Extrapolating the four-times faster tuning achieved by the presented RLO approach

despite a handicap on the interaction frequency, the deployment of similar RL-based solutions to other tuning tasks has the potential to save over 1500 hours of beam time per year and facility [62], which would then become available for scientific experiments. In addition, new facilities may also be designed faster at reduced cost and to higher performance standards. Beyond the direct impact on the field of accelerators, this has two-fold implications: Firstly, it will help push the boundaries of accelerator-based science, enabling new discoveries and breakthroughs in fields like particle physics, material science, biology and chemistry. Secondly, the presented techniques and tools function as a blueprint for the application of RL and differentiable simulations to other complex physical systems, such as fusion reactors, space propulsion systems and quantum computers, where the same challenges of high-dimensional control, high-speed simulation and safety risks apply.

## 6.3 Outlook

In immediate future work, the FEL intensity maximisation task at LCLS should be further investigated. Particular future goals of this work include the deployment of a trained policy to the real accelerator, as well as a transfer of the training setup to other FEL facilities, such as European XFEL and FLASH. As part of this work, the model mismatch between the simulator and the real accelerator will have to be investigated further, and a more advanced FEL model will have to be integrated into Cheetah, most likely using a modular neural network surrogate model trained on data from GENESIS simulations.

Further plans for future work include the deployment of RL-based solutions to other tasks, such as dump line orbit feedback at European XFEL, in day-to-day accelerator operations. This will be an important step, as RL has yet to gain a foothold in day-to-day use in accelerator control rooms, despite the impressive results achieved in this and other works. An important component of this line of work will be the compilation of the environments developed here and in other works into a benchmark environment package, which will help compare RL solutions and entice RL experts from outside the accelerator community to contribute to the field.

Despite significant progress in solving the challenges of RL training on accelerators, some key challenges remain. The engineering effort required to deploy RL policies remains high, and further study should be devoted to transferable training setups, lattice-agnostic policies and meta RL, where initial studies in this dissertation have shown promising results. Extending the lattice-agnostic approach through the inclusion of lattice information in the observation is a promising direction for future work in this area. The ultimate goal of these studies should be to enable the publication of an RL-based framework, either in the form of training code or even trained policies, that accelerator physicists can use to deploy RL policies to similar tasks at their facilities with minimal effort and RL expertise. An example would be a Python package for training FEL tuning policies, where one needs only to provide the lattice definition and a list of the actuators to tune, in order to train a policy that can be directly deployed as part of the accelerator control system. Scalability is also a key open challenge. This work introduces some ideas on how to scale RL training to more complex tasks, including gradient-based RL, but it is unclear if these methods will be sufficient to scale RL to orders of magnitude larger problems with hundreds of actuators and sensors. In addition, the formulation of a reward for real-world tasks comes with challenges like the order of magnitude reward problem, where the fact that objective values and therefore rewards can differ by orders of magnitude can cause issues in the learning process. Solutions to this problem have been identified in this dissertation,

but they are not without their own disadvantages, and a more wholistic solution would significantly ease the real-world adoption of RL. Lastly, the adoption of RL in accelerator control rooms remains a challenge, not necessarily because of the performance of RL policies and the cost of deploying them, but because accelerator operators need to gain trust and expertise with them as well as a deeper understanding of RL. Some community efforts like the RL4AA collaboration are already underway to address this issue, but ultimately the widespread deployment of RL policies and other autonomous tuning and control solutions in accelerator control rooms will require a cultural shift in the accelerator community. This author expects that the continued success of RL in accelerator tuning and control will help drive this cultural shift, as the benefits of RL become more apparent and the drawbacks are addressed.

Miscellaneous future work on RL for accelerators may include increasing the interaction rate to real-time. With the transverse tuning task at ARES, the RL policies were held back by the limited interaction rate, which was the result of hardware limitations and the fact that comprehensive measurements were taken to provide the observations. In some concurrent work, real-time interaction of RL agents with the accelerator is already being investigated [298], but with problems like the one at ARES, where the observations are constructed from time-consuming measurements, this may remain a challenge. Solving this challenge will require providing higher-dimensional raw observations as input to the policies as well as the modelling of temporal dynamics, such as the control of the magnet power supplies during training. Most likely, this will also require policies with longer time horizons, which may be achieved by using more complex and difficult-to-train RNN or transformer policy architectures.

Avenues for future work pertaining to high-speed differentiable modelling are extensive. Applications for this powerful tool will have to be explored and developed. One such application that is currently under investigation is the online reconstruction of the full 6-dimensional phase space at LCLS based on a method previously implemented with the help of Bmad-X [272]. Efforts are also underway to further improve the capabilities of Cheetah. At the time of writing, a vectorised version of Cheetah is in the final stages of development. Initial results indicate that vectorisation speeds up computation by a factor of about 50 on a CPU, with even larger speed improvements expected on GPUs. The extension of the effects modelled in Cheetah is also underway. Various effects modelled by Bmad-X, such as chromaticity, are currently being integrated into Cheetah. Space charge computations based on Green's functions have also been developed in Cheetah with the goal of investigating the memory requirements of backward mode automatic differentiation over collective effects. Future plans also include the full integration of coupled neural network surrogate models for space charge effects, FEL models and other complex dynamics into Cheetah, providing faster simulations of these and enabling memory-efficient automatic differentiation over them. In particular, the investigation of novel architectures like *PointNet* [252] and graph neural networks [278] for tracking particle clouds through the accelerator as well as more suitable speciality loss functions like structural similarity is a promising direction for future work. A port of Cheetah to JAX [37] has also started development under the name *Lynx* [156], with the goal of further increasing simulation speeds by leveraging features like just-in-time (JIT) compilation. JAX further promises to enable other advanced features like forward-mode automatic differentiation, which are not available in PyTorch.

# Appendix A

## Transfer Maps

The following lists the transfer maps of some of the elements considered in this dissertation, as they are implemented for linear beam dynamics simulations in Cheetah [157, 162].

In the following,  $l$  denotes the length of the respective element,  $\gamma$  is the relativistic Lorentz factor as given by

$$\gamma = \frac{E}{m_e c^2}, \quad (\text{A.1})$$

with

$$m_e c^2 = 0.510\,998\,95 \text{ MeV}, \quad (\text{A.2})$$

the electron mass energy equivalent, where  $m_e$  is the electron mass and  $c$  is the speed of light in a vacuum. In addition,  $\beta$  is the relativistic velocity factor given by

$$\beta = \sqrt{1 - \frac{1}{\gamma^2}}. \quad (\text{A.3})$$

Note further that the listed transfer maps differ slightly from the ones typically used in accelerator physics in that they are 7-dimensional. This is due to the introduction of an additional dimension to the conventional 6-dimensional phase space, analogous to an affine space, allowing Cheetah to coherently express more effects as transfer map multiplications. This is a prerequisite for some of the speed optimisations in Cheetah. Below, the 7-th dimension is indicated by the light grey entries in the transfer maps.

### A.1 Drift Space

A drift space in Cheetah represents a straight section of the beamline, where the particles travel undisturbed by any electromagnetic fields. The transfer map of a drift space is given by

$$R_D = \begin{pmatrix} 1 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & l & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -\frac{l}{\beta^2} \frac{1}{\gamma^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (\text{A.4})$$

## A.2 Steering Dipole Magnet

Steering magnets in Cheetah are modelled as a drift space followed by a thin-lens kick in the respective horizontal or vertical plane that deflects the trajectory of the beam by an angle  $\alpha$ . The transfer map of a horizontal steering dipole magnet is hence given by

$$R_{C_h} = \begin{pmatrix} 1 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & \alpha \\ 0 & 0 & 1 & l & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -\frac{l}{\beta^2} \frac{1}{\gamma^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (\text{A.5})$$

Similarly, the transfer map of a vertical steering dipole magnet is different only in the position of the steering angle  $\alpha$ , resulting in

$$R_{C_v} = \begin{pmatrix} 1 & l & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & l & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & \alpha \\ 0 & 0 & 0 & 0 & 1 & -\frac{l}{\beta^2} \frac{1}{\gamma^2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \quad (\text{A.6})$$

## A.3 Quadrupole Magnet

The transfer map  $R_Q$  of a quadrupole magnet in Cheetah is the product of multiple matrices

$$R_Q = R_{m,\text{out}} R_{t,\text{out}} R_{Q,\text{main}} R_{t,\text{in}} R_{m,\text{in}}, \quad (\text{A.7})$$

a main quadrupole matrix  $R_{Q,\text{main}}$  as well as two sets of matrices ( $R_{m,\text{in}}, R_{m,\text{out}}$ ) and ( $R_{t,\text{in}}, R_{t,\text{out}}$ ) that shift and rotate the beam into a different reference frame at the entrance of the quadrupole and back at the exit of the quadrupole to model the traversal of misaligned and tilted magnets, respectively.

The main quadrupole matrix is given by

$$R_{Q,\text{main}} = \begin{pmatrix} c_x & s_x & 0 & 0 & 0 & \frac{d_x}{\beta} & 0 \\ -ks_x & c_x & 0 & 0 & 0 & \frac{s_x}{\beta} & 0 \\ 0 & 0 & c_y & s_y & 0 & 0 & 0 \\ 0 & 0 & ks_y & c_y & 0 & 0 & 0 \\ \frac{s_x}{\beta} & \frac{d_x}{\beta} & 0 & 0 & 1 & r_{56} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad (\text{A.8})$$

where

$$r_{56} = \frac{l - s_x}{\beta^2 k} - \frac{l}{\beta^2 \gamma^2}, \quad (\text{A.9})$$

$$d_x = (1 - c_x), \quad (\text{A.10})$$



where  $m_x$  and  $m_y$  are the horizontal and vertical misalignments of the quadrupole magnet, respectively.

# Appendix B

## Alternate Tuning Result Over Speed Distributions

Throughout this dissertation, Figs. 4.6, 4.10, 4.20 and 5.3 show the distributions of the MAEs and pulse intensities over the steps to convergence as achieved by different algorithms and policies. Depending on the context, these figures were shown for the conventional or the minimum metrics achieved during the optimisation. For reference, the opposite metric figures are shown in Figs. B.1 to B.4 of this appendix.

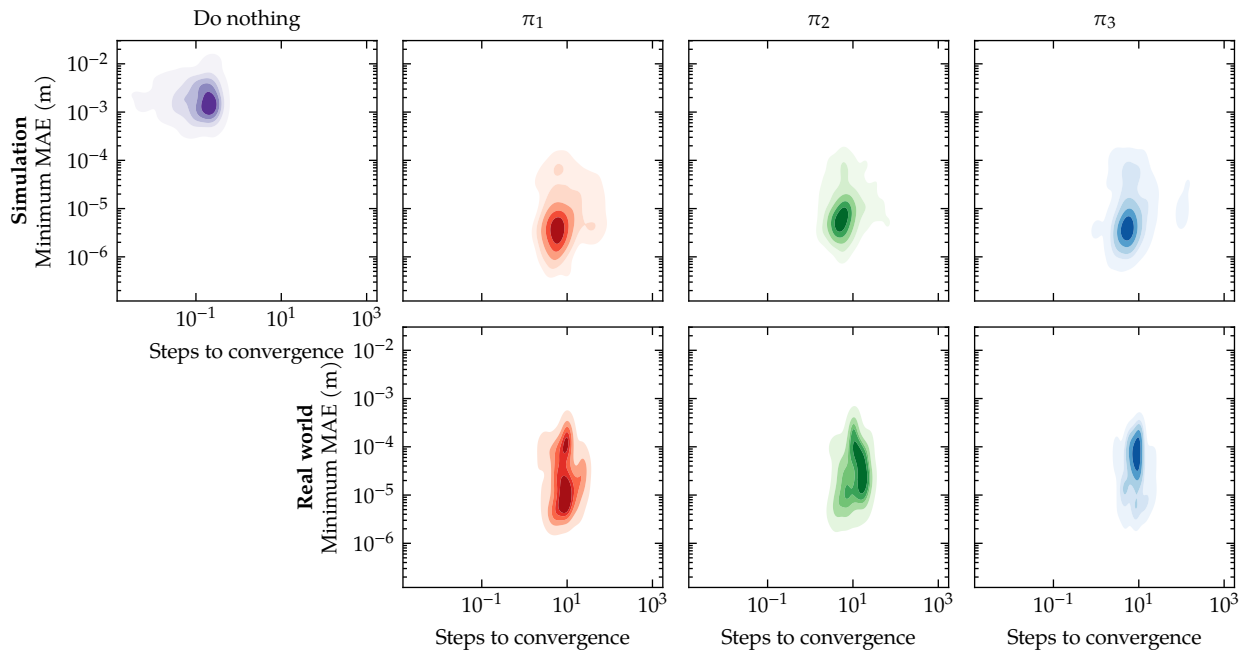


Figure B.1: Distributions of the minimum MAEs achieved by the RLO policies during optimisations plotted over the steps to convergence. KDEs are used to visualise the distributions. Ideally, distributions should be concentrated in the lower left, indicating that policies converge quickly to good solutions. Distributions in the upper left converge quickly but to bad solutions, while distributions in the lower right converge slowly to good solutions. Note the logarithmic scaling of the axes. Because doing nothing always converges in zero steps, a small amount of noise between 0 to 0.3 steps was added to the steps to convergence for doing nothing to enable the visualisation with a KDE. The steps to convergence are measured on the minimum beam to guarantee that all runs converge and are included in the distribution. This figure corresponds to Fig. 4.6.

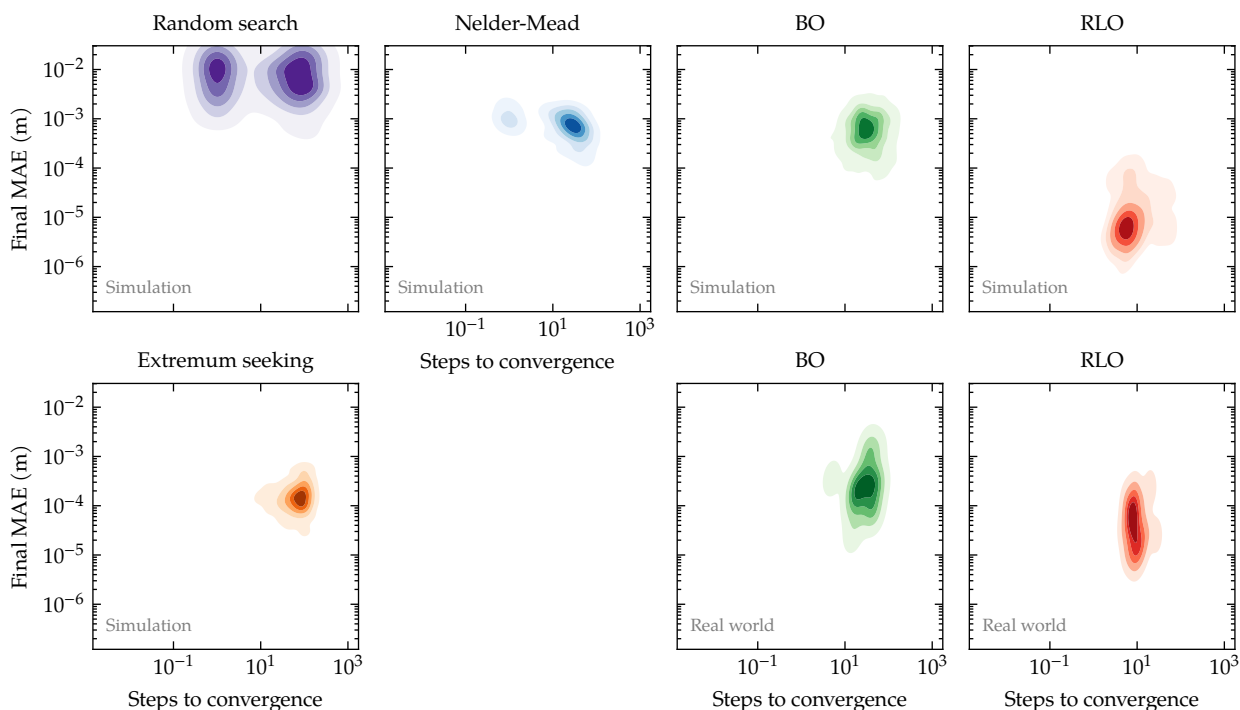


Figure B.2: Distributions of the final MAEs achieved by the different algorithms during optimisations plotted over the steps to convergence. KDEs are used to visualise the distributions. Ideally, distributions should be concentrated in the lower left, indicating that policies converge quickly to good solutions. Distributions in the upper left converge quickly but to bad solutions, while distributions in the lower right converge slowly to good solutions. Note the logarithmic scaling of the axes. The steps to convergence are measured on the minimum beam to guarantee that all runs converge and are included in the distribution. This figure corresponds to Fig. 4.10.

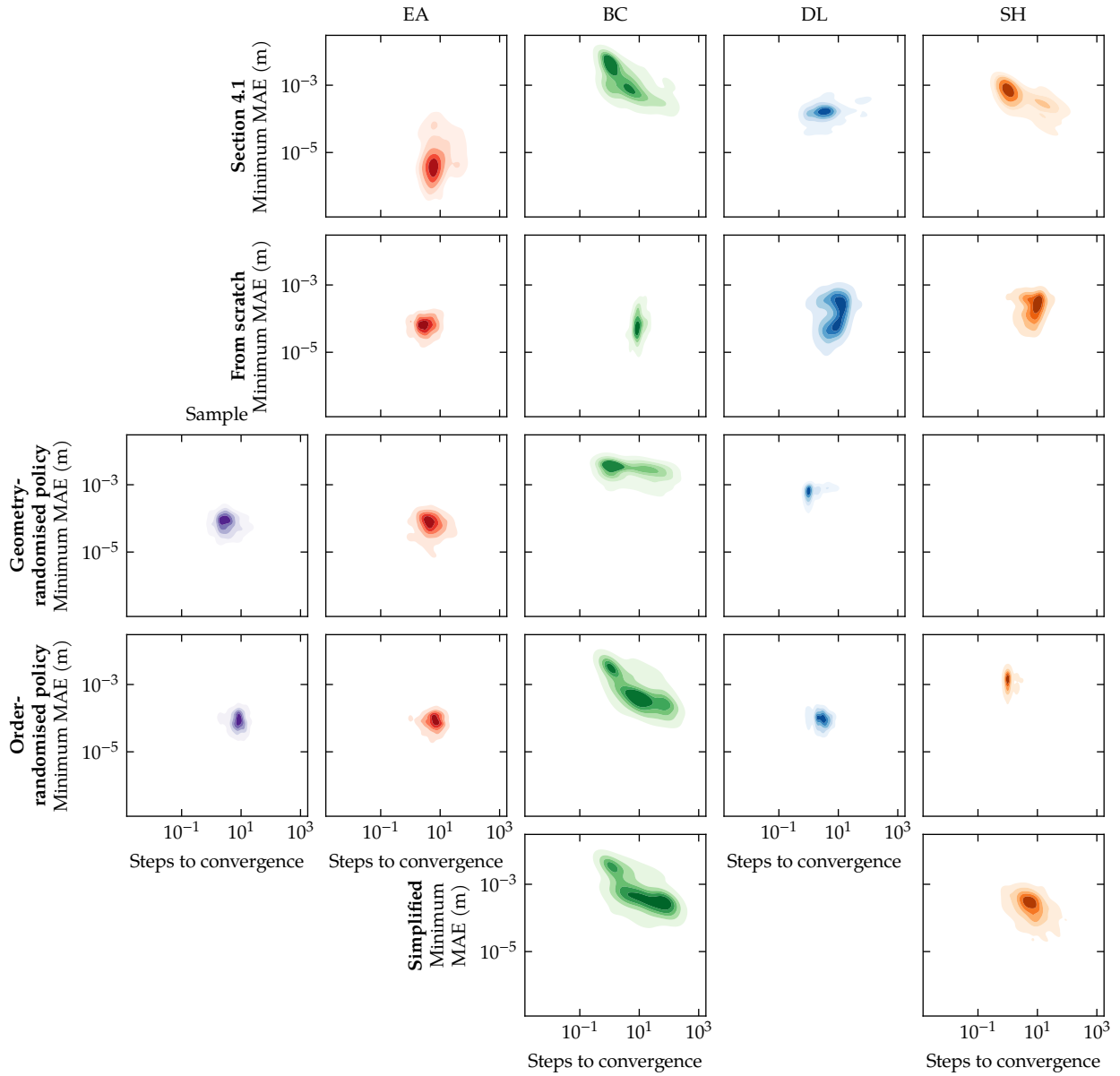


Figure B.3: Distributions of the minimum MAEs achieved by specialist and lattice-agnostic policies on different lattices during optimisations plotted over the steps to convergence. KDEs are used to visualise the distributions. Ideally, distributions should be concentrated in the lower left, indicating that policies converge quickly to good solutions. Distributions in the upper left converge quickly but to bad solutions, while distributions in the lower right converge slowly to good solutions. Note the logarithmic scaling of the axes. In the first row,  $\pi_1$  from Section 4.1 trained on the EA is applied to the different lattices. The bottom row pertains to simplified less-out-of-sample variants of the BC and SH lattices, where apertures and the bends are removed. The steps to convergence are measured on the minimum beam to guarantee that all runs converge and are included in the distribution. This figure corresponds to Fig. 4.20.

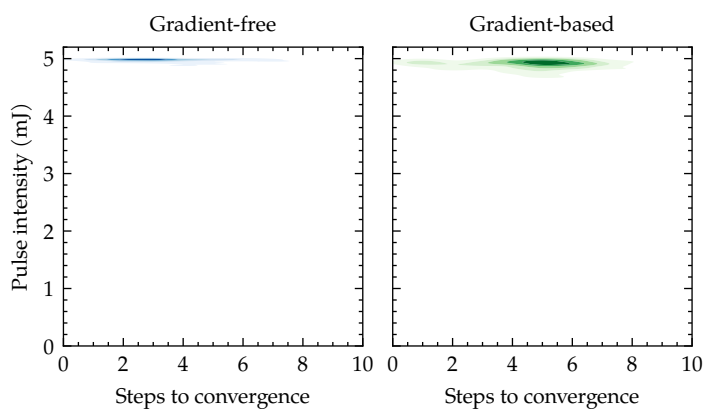


Figure B.4: Distributions of the maximum FEL pulse intensity achieved by the gradient-free and gradient-based policies plotted over the steps to convergence. KDEs are used to visualise the distributions. Ideally, distributions should be concentrated in the upper left, indicating that policies converge quickly to good solutions. Distributions in the lower left converge quickly but to bad solutions, while distributions in the upper right converge slowly to good solutions. The steps to convergence are measured on the maximum intensity to guarantee that all runs converge and are included in the distribution. This figure corresponds to Fig. 5.3.



# Symbols and Abbreviations

## Symbols

$a$	Acquisition function.
$a$	Action vector.
$a_t$	Action at time step $t$ .
$A$	Advantage.
$\mathcal{A}$	Set of possible actions.
$\hat{A}$	Estimate of the advantage.
$\mathbf{b}$	Transverse beam parameter vector $(\mu_x, \sigma_x, \mu_y, \sigma_y)$ . Measured transverse beam parameter vector. Bias vector.
$\hat{\mathbf{b}}$	Estimate of the transverse beam parameter vector $\mathbf{b}$ .
$\mathbf{b}'$	Target transverse beam parameter vector.
$\mathbf{b}_{\text{in}}$	Incoming transverse beam parameter vector.
$\mathbf{b}_{\text{out}}$	Outgoing transverse beam parameter vector.
$\mathcal{B}$	Mini-batch. Abstract notion of a charged particle beam.
$\mathcal{B}_{\text{in}}$	Abstract notion of an incoming charged particle beam.
$\mathcal{B}_{\text{out}}$	Abstract notion of an outgoing charged particle beam.
$C$	Dipole steering magnet.
$C_h$	Horizontal dipole steering magnet.
$C_v$	Vertical dipole steering magnet.
$d$	Difficulty factor. Terminal flag in DDPG.
$D$	Drift space.
$f$	A function with its purpose denoted in the subscript.
$\mathcal{D}$	Dataset. Replay buffer. Distance function.
$E$	Reference beam energy.
$\mathcal{E}$	RL environment.
$\mathbb{E}$	Expectation.
$I$	FEL pulse intensity.
$I^*$	Optimal FEL pulse intensity.
$J$	Expected cumulative reward.
$k$	Quadrupole geometric focusing strength.
$l$	Length of a Cheetah element.
$\mathcal{L}$	Loss function.

$m$	Misalignment vector.
$M$	MDP.
	Neural network for LPS prediction.
$o$	Observation vector.
$o_t$	Observation at time step $t$ .
$O$	Objective function.
$\mathcal{O}$	Set of possible observations.
$p$	7-dimensional Cheetah particle vector $(x, x', y, y', \tau, \delta, 1)$ .
$P$	$N \times 7$ matrix of Cheetah particle vectors.
	State transition probability function.
$q$	Total beam charge.
$\mathbf{q}$	Vector of particle charges.
$Q$	Action-value function.
	Quadrupole magnet.
$\hat{Q}$	Estimate of the action-value function.
$r$	Reward.
$r_t$	Reward at time step $t$ .
$R$	Cumulative reward.
	Linear beam dynamics transfer map.
$\mathcal{R}$	Reward function.
$s$	Environment state.
$s_t$	Environment state at time step $t$ .
$S$	Diagnostic screen station.
$\mathcal{S}$	Set of possible states.
$t$	Twiss parameter vector $(\beta_x, \alpha_x, \beta_y, \alpha_y)$ .
$\hat{t}$	Design Twiss parameter vector.
$t_0$	Twiss parameter vector of the incoming beam.
$T$	LLM temperature parameter.
$u$	Actuator vector.
$\hat{u}$	Design actuator vector.
$u^*$	Optimal actuator vector.
$u_{\text{normed}}$	Normalised actuator vector.
$u_t$	Actuator vector at time step $t$ .
$v_{\text{linac}}$	Settings of the linac that are not controlled.
$v_{\text{und}}$	Settings of the undulator that are not controlled.
$V$	State-value function.
$\hat{V}$	Estimate of the state-value function.
$w$	Weight.
$w_{\text{visible}}$	Constant added to the objective when the beam is visible on the screen.
$\mathbf{w}$	Weight vector.
$W$	Neural network layer weight matrix.
$x$	Horizontal particle position.
$x'$	Horizontal particle momentum.
$x$	Neural network input.
$x_{\text{RF}}$	RF parameter input vector.
$x_{\text{THz}}$	THz spectrum input vector.
$y$	Vertical particle position.
$y'$	Vertical particle momentum.

$y$	Neural network output.
$\hat{y}$	Predicted neural network output.
$y_s$	Bunch length ground truth.
$\hat{y}_s$	Predicted bunch length.
$y_I$	Current profile ground truth.
$\hat{y}_I$	Predicted current profile.
$y_{LPS}$	LPS image ground truth.
$\hat{y}_{LPS}$	Predicted LPS image.
$y_{range}$	Extent of the LPS image ground truth.
$\hat{y}_{range}$	Predicted extent of the LPS image.
$\alpha$	Learning rate.
	Steering angle.
$\delta$	Particle momentum offset.
$\Delta_{SC}$	Abstract notion of change induced by space charge.
$\Delta\Sigma_{SC}$	Covariance matrix change induced by space charge.
$\eta$	TRPO ratio.
$\theta$	Neural network parameters.
	Policy parameters.
$\theta^*$	Optimal policy parameters.
$\theta^-$	Parameters of the target policy.
$\lambda$	Normalisation scale vector.
$\mu_x$	Horizontal beam position.
$\hat{\mu}_x$	Estimate of the horizontal beam position.
$\mu_y$	Vertical beam position.
$\hat{\mu}_y$	Estimate of the vertical beam position.
$\pi$	Policy.
$\pi^*$	Optimal policy.
$\rho_0$	Initial state distribution.
$\sigma_x$	Horizontal beam size.
$\sigma_y$	Vertical beam size.
$\tau$	Longitudinal particle displacement.
$\tau$	Environment rollout trajectory.
$\tau_p$	Polyak update rate.
$\phi$	Parameters of the action-value function.
$\phi^-$	Parameters of the target action-value function.
$\varphi$	Activation function.
$\Omega$	Observation function.

## Acronyms

- A3C** Asynchronous Advantage Actor Critic.  
**ACHIP** Accelerator-on-a-Chip.  
**ADR** automatic domain randomisation.  
**AGS** Brookhaven Alternating Gradient Synchrotron.  
**AI** artificial intelligence.  
**ALE** Arcade Learning Environment.  
**ALPI** Acceleratore Lineare Per Ioni.  
**ARES** Accelerator Research Experiment at SINBAD.

**ATLAS** Argonne Tandem Linear Accelerator System.  
**AWAKE** Advanced Proton Driven Plasma Wakefield Acceleration Experiment.

**BAX** Bayesian Algorithm Execution.  
**BBFS** Beam-based Feedback System.  
**BC** Bunch Compressor.  
**BLM** beam loss monitor.  
**BNL** Brookhaven National Laboratory.  
**BO** Bayesian optimisation.  
**BOBYQA** bound optimisation by quadratic approximation.  
**BPM** beam position monitor.

**CAFe II** China Accelerator Facility for Superheavy Elements.  
**CDR** coherent diffraction radiation.  
**CEBAF** Continuous Electron Beam Accelerator Facility.  
**CERN** Conseil Européen pour la Recherche Nucléaire.  
**CI** continuous integration.  
**CNN** convolutional neural network.  
**COSY** Cooler Synchrotron.  
**CoT** chain-of-thought.  
**CPI** conservative policy iteration.  
**CPU** central processing unit.  
**CSR** coherent synchrotron radiation.  
**CUDA** Compute Unified Device Architecture.

**DAQ** data acquisition.  
**DDPG** deep deterministic policy gradients.  
**DESY** Deutsches Elektronen-Synchrotron.  
**DL** Diagnostic Line.  
**DORIS** Doppel-Ring-Speicher.  
**DQM** data quality monitoring.  
**DQN** deep Q-network.

**EA** Experimental Area.  
**EI** expected improvement.  
**ES** extremum seeking.  
**ESRF** European Synchrotron Radiation Facility.  
**European XFEL** European X-Ray Free-Electron Laser.

**FACET** Facility for Advanced Accelerator Experimental Tests.  
**FAIR** findability, accessibility, interoperability and reusability.  
**FDF** focus-defocus-focus.  
**FEL** Free-Electron Laser.  
**FERL** free energy-based reinforcement learning.  
**FERMI** Free Electron Laser for Multidisciplinary Investigations.  
**FLASH** Free-Electron Laser in Hamburg.  
**FLUTE** Ferninfrarot Linac- und Test-Experiment.  
**FPGA** field-programmable gate array.  
**FPV** first-person view.

**GAIA** general AI assistant for intelligent accelerator operations.

**GAN** generative adversarial network.  
**GELU** Gaussian error linear unit.  
**GMM** Gaussian mixture model.  
**GMPS** gradient magnet power supply.  
**GP** Gaussian process.  
**GPU** graphics processing unit.  
**GS** Gerchberg-Saxton.  
**GUI** graphical user interface.

**HERA** Hadron-Elektron-Ringanlage.  
**HGHG** High-Gain Harmonic Generation.  
**HGVPU** horizontal gap vertical polarisation undulator.  
**HPC** high-performance computing.  
**HVAC** heating, ventilation and air conditioning.

**IMU** inertial measurement unit.  
**IRL** inverse reinforcement learning.

**JIT** just-in-time.

**KARA** Karlsruhe Research Accelerator.  
**KDE** kernel density estimation.  
**KIT** Karlsruhe Institute of Technology.  
**KL** Kullback-Leibler.  
**KNN**  $k$ -nearest neighbours.  
**KSTAR** Korea Superconducting Tokamak Advanced Research.

**LANSCE DTL** Los Alamos Neutron Science Center Drift Tube Linac.  
**LCLS** Linac Coherent Light Source.  
**LHC** Large Hadron Collider.  
**LINAC** linear accelerator.  
**Linac4** Linear accelerator 4.  
**LLM** large language model.  
**LLRF** low-level radio frequency.  
**LPI** light particle injector.  
**LPS** longitudinal phase space.  
**LSTM** long short-term memory.

**MAE** mean absolute error.  
**MAML** Model-Agnostic Meta-Learning.  
**MDP** Markov decision process.  
**MHD** magneto-hydrodynamics.  
**MLP** multilayer perceptron.  
**MMLU** Massive Multitask Language Understanding.  
**MPC** model predictive control.  
**MPO** Maximum a Posteriori Policy Optimisation.  
**MPPI** Model Predictive Path Integral.  
**MPS** Metal Performance Shader.  
**MSBE** mean-squared Bellman error.  
**MSE** mean-squared error.

**Mu2e** Muon-to-Electron Conversion Experiment.

**NLP** natural language processing.

**NSGA-II** nondominated sorting genetic algorithm II.

**OCR** optical character recognition.

**PBT** population-based training.

**PETRA** Positron-Elektron-Tandem-Ringanlage.

**PI** proportional-integral.

**PID** proportional-integral-derivative.

**POMDP** partially observable Markov decision process.

**PPO** Proximal Policy Optimisation.

**PS** Proton Synchrotron.

**PSI** Paul Scherrer Institute.

**QBM** quantum Boltzmann machine.

**RAG** retrieval augmented generation.

**RAM** random access memory.

**RCDS** robust conjugate direction search.

**RCDS-S** safe robust conjugate direction search.

**ReLU** rectified linear unit.

**RF** radio frequency.

**RHIC** Relativistic Heavy Ion Collider.

**RL** reinforcement learning.

**RL4AA** Reinforcement Learning for Autonomous Accelerators.

**RLHF** reinforcement learning from human feedback.

**RLO** reinforcement learning-trained optimisation.

**RMSE** root mean squared error.

**RNN** recurrent neural network.

**SAC** Soft Actor-Critic.

**SASE** self-amplified spontaneous emission.

**SGD** stochastic gradient descent.

**SH** Spectrometer High Energy.

**SLAC** SLAC National Accelerator Laboratory.

**SLM** spatial light modulator.

**SoC** system on a chip.

**SPEA2** strength Pareto evolutionary algorithm 2.

**SPS** Super Proton Synchrotron.

**SVD** singular value decomposition.

**SVM** support vector machine.

**SwissFEL** Swiss Free Electron Laser.

**SXFEL** Shanghai Soft X-Ray Free Electron Laser.

**tanh** hyperbolic tangent.

**TCV** Tokamak à Configuration Variable.

**TD3** twin delayed deep deterministic policy gradients.

**TDS** transverse deflecting structure.

**TESLA** Tera-Electronvolt Energy Superconducting Linear Accelerator.

**TEU-FEL** Twente/Eindhoven University FEL.

**TPSA** truncated power series algebra.

**TRPO** Trust Region Policy Optimisation.

**TurBO** trust region Bayesian optimisation.

**UCB** upper confidence bound.

**VAE** variational autoencoder.

**XFEL** X-ray free-electron laser.



# Bibliography

- [1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow, Large-scale machine learning on heterogeneous systems, 2015.
- [2] J. Achiam. Spinning Up in Deep Reinforcement Learning, 2018.
- [3] I. Agapov, G. Geloni, S. Tomin, and I. Zagorodnov. OCELOT: A software framework for synchrotron light source and FEL studies. *Nuclear Instruments and Methods in Physics Research*, 768, 2014.
- [4] I. Agapov, S. Tomin, W. Decking, M. Scholz, I. Zagorodnov, and G. Geloni. On-line optimization of European XFEL with OCELOT. In *Proceedings of the 15th International Conference on Accelerators and Large Experimental Physics Control Systems (ICALEPCS)*, 2017.
- [5] C. Allaire, R. Ammendola, E. C. Aschenauer, M. Balandat, M. Battaglieri, J. Bernauer, M. Bondi, N. Branson, T. Britton, A. Butter, I. Chahrour, P. Chatagnon, E. Cisbani, E. W. Cline, S. Dash, C. Dean, W. Deconinck, A. Deshpande, M. Diefenthaler, R. Ent, C. Fanelli, M. Finger, M. Finger, E. Fol, S. Furlotov, Y. Gao, J. Giroux, N. C. Gunawardhana Waduge, O. Hassan, P. L. Hegde, R. J. Hernández-Pinto, A. Hiller Blin, T. Horn, J. Huang, A. Jalotra, D. Jayakodige, B. Joo, M. Junaid, N. Kalantarians, P. Karande, B. Kriesten, R. Kunnawalkam Elayavalli, Y. Li, M. Lin, F. Liu, S. Liuti, G. Matousek, M. McEneaney, D. McSpadden, T. Menzo, T. Miceli, V. Mikuni, R. Montgomery, B. Nachman, R. R. Nair, J. Niestroy, S. A. Ochoa Oregon, J. Oleniacz, J. D. Osborn, C. Paudel, C. Pecar, C. Peng, G. N. Perdue, W. Phelps, M. L. Purschke, H. Rajendran, K. Rajput, Y. Ren, D. F. Renteria-Estrada, D. Richford, B. J. Roy, D. Roy, A. Saini, N. Sato, T. Satogata, G. Sborlini, M. Schram, D. Shih, J. Singh, R. Singh, A. Siodmok, J. Stevens, P. Stone, L. Suarez, K. Suresh, A. N. Tawfik, F. Torales Acosta, N. Tran, R. Trotta, F. J. Twagirayezu, R. Tyson, S. Volkova, A. Vossen, E. Walter, D. Whiteson, M. Williams, S. Wu, N. Zachariou, and P. Zurita. Artificial intelligence for the Electron Ion Collider (AI4EIC). *Computing and Software for Big Science*, 8, 2024.
- [6] F. Andreas. LatticeJSON, 2019.
- [7] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas. Learning to learn by gradient descent by gradient descent.

- In *Proceedings of the 30th Conference on Neural Information Processing Systems (NIPS)*, 2016.
- [8] S. Appel and S. Reimann. Beam line optimization using derivative-free algorithms. *Journal of Physics: Conference Series*, 1350, 2019.
- [9] A. Awal, J. Hetzel, and J. Pretz. Injection optimization via reinforcement learning at the cooler synchrotron COSY. In *Proceedings of the 14th International Particle Accelerator Conference (IPAC)*, 2023.
- [10] A. Awal, J. Hetzel, R. Gebel, and J. Pretz. Injection optimization at particle accelerators via reinforcement learning: From simulation to real-world application. *arXiv*, 2406.12735, 2024.
- [11] A. Awal, J. Hetzel, and J. Pretz. Injection optimization via reinforcement learning: From simulation to real-world application. In *Proceedings of the 15th International Particle Accelerator Conference (IPAC)*, 2024.
- [12] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, D. Guo, and C. Blundell. Agent57: Outperforming the atari human benchmark. *arXiv*, 2003.13350, 2020.
- [13] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch. Emergent tool use from multi-agent autotutorials. *arXiv*, 1909.07528, 2019.
- [14] M. Balandat, B. Karrer, D. R. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020.
- [15] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18, 2018.
- [16] I. V. Bazarov and C. K. Sinclair. Multivariate optimization of a high brightness DC gun photoinjector. *Physical Review Special Topics - Accelerators and Beams*, 8, 2005.
- [17] C. Beattie, J. Z. Leibo, D. Teplyaev, T. Ward, M. Wainwright, H. Küttler, A. Lefrancq, S. Green, V. Valdés, A. Sadik, J. Schrittwieser, K. Anderson, S. York, M. Cant, A. Cain, A. Bolton, S. Gaffney, H. King, D. Hassabis, S. Legg, and S. Petersen. DeepMind Lab. *arXiv*, 1612.03801, 2016.
- [18] M. G. Bellemare, J. Veness, and M. Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47, 2013.
- [19] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.
- [20] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.
- [21] W. F. Bergan, I. V. Bazarov, C. J. R. Duncan, D. B. Liarte, D. L. Rubin, and J. P. Sethna. Online storage ring optimization using dimension-reduction and genetic algorithms. *Physical Review Accelerators and Beams*, 22, 2019.

- [22] D. A. Bezgin, A. B. Buhendwa, and N. A. Adams. JAX-Fluids: A fully-differentiable high-order computational fluid dynamics solver for compressible two-phase flows. *Computer Physics Communications*, 2022.
- [23] D. A. Bezgin, A. B. Buhendwa, and N. A. Adams. JAX-Fluids 2.0: Towards HPC for differentiable CFD of compressible two-phase flows. *arXiv*, 2402.05193, 2024.
- [24] L. Biewald. Experiment tracking with Weights and Biases, 2020.
- [25] BMW. BMW 5 Series price list, 2024. URL [https://www.bmw.de/content/dam/bmw/marketDE/bmw\\_de/new-vehicles/pricelists/preisliste-bmw5er-new.pdf.coredownload.inline.pdf](https://www.bmw.de/content/dam/bmw/marketDE/bmw_de/new-vehicles/pricelists/preisliste-bmw5er-new.pdf.coredownload.inline.pdf). Accessed: 2024-04-24.
- [26] T. Boltz. *Micro-Bunching Control at Electron Storage Rings with Reinforcement Learning*. PhD thesis, Karlsruhe Institute of Technology, 2021.
- [27] T. Boltz, M. Brosi, E. Bründermann, B. Haerer, P. Kaiser, C. Pohl, P. Schreiber, M. Yan, T. Asfour, and A.-S. Müller. Feedback design for control of the micro-bunching instability based on reinforcement learning. In *Proceedings of the 10th International Particle Accelerator Conference (IPAC)*, 2019.
- [28] T. Boltz, J. L. Martinez, C. Xu, K. R. L. Baker, R. Roussel, D. Ratner, B. Mustapha, and A. L. Edelen. More sample-efficient tuning of particle accelerators with Bayesian optimization and prior mean models. *arXiv*, 2403.03225, 2024.
- [29] C. Bonnet, D. Luo, D. Byrne, S. Surana, S. Abramowitz, P. Duckworth, V. Coyette, L. I. Midgley, E. Tegegn, T. Kalloniatis, O. Mahjoub, M. Macfarlane, A. P. Smit, N. Grinsztajn, R. Boige, C. N. Waters, M. A. Mimouni, U. A. Mbou Sob, R. de Kock, S. Singh, D. Furelos-Blanco, V. Le, A. Pretorius, and A. Laterre. Jumanji: A diverse suite of scalable reinforcement learning environments in JAX. *arXiv*, 2306.09884, 2024.
- [30] M. Borland. elegant: A flexible SDDS-compliant code for accelerator simulation. In *Proceedings of the 6th International Computational Accelerator Physics Conference*, 2000.
- [31] Bosch. KIR31AD40 EU data sheet, 2021. URL <https://media3.bosch-home.com/Documents/eudatasheet/de-DE/KIR31AD40.pdf>. Accessed: 2024-04-24.
- [32] L. Boukela, A. Eichler, J. Branlard, and N. Z. Jomhari. A two-stage machine learning-aided approach for quench identification at the European XFEL. *arXiv*, 2407.08408, 2024.
- [33] H. Bourslard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59, 1988.
- [34] K. Bourzac. World’s most powerful X-ray laser will ‘film’ chemical reactions in unprecedented detail. *Nature*, 621, 2023.
- [35] E. Bozoki and A. Friedman. Neural networks and orbit control in accelerators. In *Proceedings of the European Particle Accelerator Conference*, 1994.
- [36] E. Bozoki and A. Friedman. Neural network technique for orbit correction in accelerators/storage rings. In *AIP Conference Proceedings*, volume 315, 1994.

- [37] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: Composable transformations of Python+NumPy programs, 2018.
- [38] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym, 2016.
- [39] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv*, 2104.13478, 2021.
- [40] K. L. Brown. A first and second order matrix theory for the design of beam transport systems and charged particle spectrometers. *Advanced Particle Physics*, 1, 1968.
- [41] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, 2020.
- [42] N. Bruchon. *Feasibility Investigation on Several Reinforcement Learning Techniques to Improve the Performance of the FERMI Free-Electron Laser*. PhD thesis, University of Trieste, 2021.
- [43] N. Bruchon, G. Fenu, G. Gaio, M. Lonza, F. A. Pelligrino, and E. Salvato. Toward the application of reinforcement learning to the intensity control of a seeded free-electron laser. In *Proceedings of the 23rd International Conference on Mechatronics Technology*, 2019.
- [44] N. Bruchon, G. Fenu, F. A. Pellegrino, and E. Salvato. Free-electron laser optimization with reinforcement learning. In *Proceedings of the 17th International Conference on Accelerator and Large Experiment Physics Control Systems*, 2019.
- [45] N. Bruchon, G. Fenu, G. Gaio, M. Lonza, F. H. O'Shea, F. A. Pellegrino, and E. Salvato. Basic reinforcement learning techniques to control the intensity of a seeded free-electron laser. *Electronics*, 9, 2020.
- [46] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv*, 2303.12712, 2023.
- [47] F. Burkart, R. Aßmann, H. Dinter, S. Jaster-Merz, W. Kuropka, F. Mayet, and T. Vinatier. The ARES Linac at DESY. In *Proceedings of the 31st International Linear Accelerator Conference (LINAC)*, 2022.
- [48] M. Cai, L. Cao, J. C. Wu, Z. H. Zhu, C. Feng, and Z. T. Zhao. Improving the performance of the SXFEL through proximal policy optimization. In *Proceedings of the 14th International Particle Accelerator Conference (IPAC)*, volume 905, 2023.
- [49] CERN. MAD-X – Methodical accelerator design, 1990.
- [50] H. Chase. LangChain, 2022.

- [51] T. Chen, X. Chen, W. Chen, Z. Wang, H. Heaton, J. Liu, and W. Yin. Learning to optimize: A primer and a benchmark. *Journal of Machine Learning Research*, 23, 2022.
- [52] X. Chen, Y. Jia, X. Qi, Z. Wang, and Y. He. Orbit correction based on improved reinforcement learning algorithm. *Physical Review Accelerators and Beams*, 26, 2023.
- [53] X.-l. Chen, X. Qi, C. Su, Y. He, and Z. Wang. Trend-based SAC beam control method with zero-shot in superconducting linear accelerator. *arXiv*, 2306.13869, 2023.
- [54] M. Chevalier-Boisvert, B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry. Minigrid & Miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *Computing Research Repository*, 2306.13831, 2023.
- [55] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30, 2017.
- [56] J. Citrin, I. Goodfellow, A. Raju, J. Chen, J. Degraeve, C. Donner, F. Felici, P. Hamel, A. Huber, D. Nikulin, D. Pfau, B. Tracey, M. Riedmiller, and P. Kohli. TORAX: A fast and differentiable tokamak transport simulator in JAX. *arXiv*, 2406.06718, 2024.
- [57] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6, 2002.
- [58] W. Decking, S. Abeghyan, P. Abramian, A. Abramsky, A. Aguirre, C. Albrecht, P. Alou, M. Altarelli, P. Altmann, K. Amyan, V. Anashin, E. Apostolov, K. Appel, D. Auguste, V. Ayvazyan, S. Baark, F. Babies, N. Baboi, P. Bak, V. Balandin, R. Baldinger, B. Baranasic, S. Barbanotti, O. Belikov, V. Belokurov, L. Belova, V. Belyakov, S. Berry, M. Bertucci, B. Beutner, A. Block, M. Blöcher, T. Böckmann, C. Bohm, M. Böhner, V. Bondar, E. Bondarchuk, M. Bonezzi, P. Borowiec, C. Bösch, U. Bösenberg, A. Bosotti, R. Böspflug, M. Bousonville, E. Boyd, Y. Bozhko, A. Brand, J. Branlard, S. Briechle, F. Brinker, S. Brinker, R. Brinkmann, S. Brockhauser, O. Brovko, H. Brück, A. Brüdgam, L. Butkowski, T. Büttner, J. Calero, E. Castro-Carballo, G. Cattalanotto, J. Charrier, J. Chen, A. Cherepenko, V. Cheskidov, M. Chiodini, A. Chong, S. Choroba, M. Chorowski, D. Churanov, W. Cichalewski, M. Clausen, W. Clement, C. Cloué, J. A. Cobos, N. Coppola, S. Cunis, K. Czuba, M. Czwalianna, B. D’Almagne, J. Dammann, H. Danared, A. de Zubiaurre Wagner, A. Delfs, T. Delfs, F. Dietrich, T. Dietrich, M. Dohlus, M. Dommach, A. Donat, X. Dong, N. Doynikov, M. Dressel, M. Duda, P. Duda, H. Eckoldt, W. Ehsan, J. Eidam, F. Eints, C. Engling, U. Englisch, A. Ermakov, K. Escherich, J. Eschke, E. Saldin, M. Faesing, A. Fallou, M. Felber, M. Fenner, B. Fernandes, J. M. Fernández, S. Feucker, K. Filippakopoulos, K. Flöttmann, V. Fogel, M. Fontaine, A. Francés, I. F. Martin, W. Freund, T. Freyermuth, M. Friedland, L. Fröhlich, M. Fusetti, J. Fydrych, A. Gallas, O. García, L. Garcia-Tabares, G. Geloni, N. Gerasimova, C. Gerth, P. Geßler, V. Gharibyan, M. Gloor, J. Głowinkowski, A. Goessel, Z. Gołębiewski, N. Golubeva, W. Grabowski, W. Graeff, A. Grebentsov, M. Grecki, T. Grevsmuehl, M. Gross, U. Grosse-Wortmann, J. Grünert, S. Grunewald, P. Grzegory, G. Feng, H. Guler, G. Gusev, J. L. Gutierrez, L. Hagge, M. Hamberg, R. Hanneken, E. Harms, I. Hartl, A. Hauberg, S. Hauf, J. Hauschildt, J. Hauser,

J. Havlicek, A. Hedqvist, N. Heidbrook, F. Hellberg, D. Henning, O. Hensler, T. Hermann, A. Hidvégi, M. Hierholzer, H. Hintz, F. Hoffmann, M. Hoffmann, M. Hoffmann, Y. Holler, M. Hüning, A. Ignatenko, M. Ilchen, A. Iluk, J. Iversen, J. Iversen, M. Izquierdo, L. Jachmann, N. Jardon, U. Jastrow, K. Jensch, J. Jensen, M. Ježabek, M. Jidda, H. Jin, N. Johansson, R. Jonas, W. Kaabi, D. Kaefer, R. Kammering, H. Kapitza, S. Karabekyan, S. Karstensen, K. Kasprzak, V. Katalev, D. Keese, B. Keil, M. Kholopov, M. Killenberger, B. Kitaev, Y. Klimchenko, R. Klos, L. Knebel, A. Koch, M. Koepke, S. Köhler, W. Köhler, N. Kohlstrunk, Z. Konopkova, A. Konstantinov, W. Kook, W. Koprek, M. Körfer, O. Korth, A. Kosarev, K. Kosiński, D. Kostin, Y. Kot, A. Kotarba, T. Kozak, V. Kozak, R. Kramert, M. Krasilnikov, A. Krasnov, B. Krause, L. Kravchuk, O. Krebs, R. Kretschmer, J. Kreutzkamp, O. Kröplin, K. Krzysik, G. Kube, H. Kuehn, N. Kujala, V. Kulikov, V. Kuzminych, D. La Civita, M. Lacroix, T. Lamb, A. Lancetov, M. Larsson, D. Le Pinvidic, S. Lederer, T. Lensch, D. Lenz, A. Leuschner, F. Levenhagen, Y. Li, J. Liebing, L. Lilje, T. Limberg, D. Lipka, B. List, J. Liu, S. Liu, B. Lorbeer, J. Lorkiewicz, H. H. Lu, F. Ludwig, K. Machau, W. Maciocha, C. Madec, C. Magueur, C. Maiano, I. Maksimova, K. Malcher, T. Maltezopoulos, E. Mamoshkina, B. Manschwetus, F. Marcellini, G. Marinkovic, T. Martinez, H. Martirosyan, W. Maschmann, M. Maslov, A. Matheisen, U. Mavric, J. Meißner, K. Meissner, M. Messerschmidt, N. Meyners, G. Michalski, P. Michelato, N. Mildner, M. Moe, F. Moglia, C. Mohr, S. Mohr, W. Möller, M. Mommerz, L. Monaco, C. Montiel, M. Moretti, I. Morozov, P. Morozov, D. Mross, J. Mueller, C. Müller, J. Müller, K. Müller, J. Munilla, A. Münnich, V. Muratov, O. Napoly, B. Näser, N. Nefedov, R. Neumann, R. Neumann, N. Ngada, D. Noelle, F. Obier, I. Okunev, J. A. Oliver, M. Omet, A. Oppelt, A. Ottmar, M. Oublaïd, C. Pagani, R. Paparella, V. Paramonov, C. Peitzmann, J. Penning, A. Perus, F. Peters, B. Petersen, A. Petrov, I. Petrov, S. Pfeiffer, J. Pflüger, S. Philipp, Y. Pienaud, P. Pierini, S. Pivovarov, M. Planas, E. Pławski, M. Pohl, J. Polinski, V. Popov, S. Prat, J. Prenting, G. Priebe, H. Pryschelski, K. Przygoda, E. Pyata, B. Racky, A. Rathjen, W. Ratuschni, S. Regnaud-Campderros, K. Rehlich, D. Reschke, C. Robson, J. Roever, M. Roggli, J. Rothenburg, E. Rusiński, R. Rybaniec, H. Sahling, M. Salmani, L. Samoylova, D. Sanzone, F. Saretzki, O. Sawlanski, J. Schaffran, H. Schlarb, M. Schlösser, V. Schlott, C. Schmidt, F. Schmidt-Foehre, M. Schmitz, M. Schmökel, T. Schnautz, E. Schneidmiller, M. Scholz, B. Schöneburg, J. Schultze, C. Schulz, A. Schwarz, J. Sekutowicz, D. Sellmann, E. Semenov, S. Serkez, D. Sertore, N. Shehzad, P. Shemarykin, L. Shi, M. Sienkiewicz, D. Sikora, M. Sikorski, A. Silenzi, C. Simon, W. Singer, X. Singer, H. Sinn, K. Sinram, N. Skvorodnev, P. Smirnow, T. Sommer, A. Sorokin, M. Stadler, M. Steckel, B. Steffen, N. Steinhau-Kühl, F. Stephan, M. Stodulski, M. Stolper, A. Sulimov, R. Susen, J. Świerblewski, C. Sydlo, E. Syresin, V. Sytchev, J. Szuba, N. Tesch, J. Thie, A. Thiebault, K. Tiedtke, D. Tischhauser, J. Tolkieln, S. Tomin, F. Tonisch, F. Toral, I. Torbin, A. Trapp, D. Treyer, G. Trowitzsch, T. Trublet, T. Tschentscher, F. Ullrich, M. Vannoni, P. Varela, G. Varghese, G. Vashchenko, M. Vasic, C. Vazquez-Velez, A. Verguet, S. Vilcins-Czvitkovits, R. Villanueva, B. Visentin, M. Viti, E. Vogel, E. Volobuev, R. Wagner, N. Walker, T. Wamsat, H. Weddig, G. Weichert, H. Weise, R. Wenddorf, M. Werner, R. Wichmann, C. Wiebers, M. Wiencek, T. Wilksen, I. Will, L. Winkelmann, M. Winkowski, K. Wittenburg, A. Witzig, P. Wlk, T. Wohlenberg, M. Wojciechowski, F. Wolff-Fabris, G. Wrochna, K. Wrona, M. Yakopov, B. Yang, F. Yang, M. Yurkov, I. Zagorodnov, P. Zalden, A. Zavadtsev, D. Zavadtsev, A. Zhirnov, A. Zhukov, V. Ziemann, A. Zolotov, N. Zolotukhina, F. Zummack, and D. Zybin. A MHz-repetition-rate hard X-ray

- free-electron laser driven by a superconducting linear accelerator. *Nature Photonics*, 14, 2020.
- [59] J. Degraeve, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. Casas, C. Donner, L. Fritz, C. Galperti, A. Huber, J. Keeling, M. Tsimpoukelli, J. Kay, A. Merle, J.-M. Moret, and M. Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602, 2022.
- [60] Deutsches Elektronen-Synchrotron DESY. Superconducting resonator. DESY Media Database, 2007. URL <https://media.desy.de/DESYmediabank/catalog/Presse/r/4227/>. Accessed: 2024-08-12, © DESY.
- [61] Deutsches Elektronen-Synchrotron DESY. European X-ray Laser XFEL on 4 January 2017. DESY Media Database, 2017. URL <https://media.desy.de/DESYmediabank/catalog/Presse/r/84568/>. Accessed: 2024-08-12, © DESY 2017.
- [62] Deutsches Elektronen-Synchrotron DESY. Accelerators 2022, 2023.
- [63] Deutsches Elektronen-Synchrotron DESY. Energy consumption at DESY, 2024. URL [https://nachhaltigkeit.desy.de/energy\\_management/energy\\_consumption\\_at\\_desy/index\\_eng.html](https://nachhaltigkeit.desy.de/energy_management/energy_consumption_at_desy/index_eng.html). Accessed: 2024-10-12.
- [64] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. OpenAI Baselines, 2017.
- [65] X. Ding, W. Du, and A. E. Cerpa. MB2C: Model-based deep reinforcement learning for multi-zone building control. In *BuildSys 2020 - Proceedings of the 7th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*, 2020.
- [66] T. Dorigo, A. Giammanco, P. Vischia, M. Aehle, M. Bawaj, A. Boldyrev, P. de Castro Manzano, D. Derkach, J. Donini, A. L. Edelen, F. Fanzago, N. R. Gauger, C. Glaser, A. G. Baydin, L. Heinrich, R. Keidel, J. Kieseler, C. Krause, M. Lagrange, M. Lamparth, L. Layer, G. Maier, F. Nardi, H. E. Pettersen, A. Ramos, F. Ratnikov, D. Röhrich, R. Ruiz de Austri, P. Martínez Ruiz del Árbol, O. Savchenko, N. Simpson, G. C. Strong, A. Taliercio, M. Tosi, A. Ustyuzhanin, and H. Zaraket. Toward the end-to-end optimization of particle physics instruments with differentiable programming. *Reviews in Physics*, 2023.
- [67] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.
- [68] G. Dresdner, D. Kochkov, P. Norgaard, L. Zepeda-Núñez, J. A. Smith, M. P. Brenner, and S. Hoyer. Learning to correct spectral methods for simulating turbulent flows. *arXiv*, 2207.00556, 2022.
- [69] G. Dulac-Arnold, D. Mankowitz, and T. Hester. Challenges of real-world reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.

- [70] J. Duris, D. Kennedy, A. Hanuka, J. Shtalenkova, A. L. Edelen, P. Baxevanis, A. Egger, T. Cope, M. McIntire, S. Ermon, and D. Ratner. Bayesian optimization of a free-electron laser. *Physical Review Letters*, 124, 2020.
- [71] A. L. Edelen. *Neural Networks for Modeling and Control of Particle Accelerators*. PhD thesis, Colorado State University, 2020.
- [72] A. L. Edelen, S. G. Biedron, B. E. Chase, D. Edstrom, S. V. Milton, and P. Stabile. Neural networks for modeling and control of particle accelerators. *IEEE Transactions on Nuclear Science*, 63, 2016.
- [73] A. L. Edelen, S. G. Biedron, J. P. Edelen, S. V. Milton, and P. J. M. Van Der Slot. Using a neural network control policy for rapid switching between beam parameters in an FEL. In *Proceedings of the 38th International Free-Electron Laser Conference (FEL)*, 2017.
- [74] A. L. Edelen, J. P. Edelen, S. G. Biedron, S. V. Milton, and P. J. M. Van Der Slot. Using neural network control policies for rapid switching between beam parameters in a free electron laser. In *Proceedings of the Workshop on Deep Learning for Physical Sciences at the Conference on Neural Information Processing Systems*, 2017.
- [75] A. L. Edelen, C. Mayes, D. Bowring, D. Ratner, A. Adelman, R. Ischebeck, J. Snuv-erink, I. Agapov, R. Kammering, J. Edelen, I. Bazarov, G. Valentino, and J. Wenninger. Opportunities in machine learning for particle accelerators. *arXiv*, 1811.03172, 2018.
- [76] A. L. Edelen, N. Neveu, M. Frey, Y. Huber, C. Mayes, and A. Adelman. Machine learning for orders of magnitude speedup in multiobjective optimization of particle accelerator systems. *Physical Review Accelerators and Beams*, 23, 2020.
- [77] Egm4313.s12 at English Wikipedia. Neuron3.svg, 2019. URL <https://commons.wikimedia.org/wiki/File:Neuron3.svg>. Accessed 2024-08-05. Licensed under CC BY-SA 3.0; modified from original.
- [78] A. Eichler, F. Burkart, J. Kaiser, W. Kuroopka, O. Stein, C. Xu, E. Bründermann, and A. Santamaria Garcia. First steps toward an autonomous accelerator, a common project between DESY and KIT. In *Proceedings of the 12th International Particle Accelerator Conference (IPAC)*, 2021.
- [79] L. Emery, M. Borland, and H. Shang. Use of a general-purpose optimization module in accelerator control. In *Proceedings of the 2003 Particle Accelerator Conference (IPAC)*, volume 4, 2003.
- [80] C. Emma, A. L. Edelen, M. J. Hogan, B. O'Shea, G. White, and V. Yakimenko. Machine learning-based longitudinal phase space prediction of particle accelerators. *Physical Review Accelerators and Beams*, 21, 2018.
- [81] C. Emma, A. L. Edelen, M. Alverson, A. Hanuka, M. J. Hogan, B. O'Shea, D. Storey, G. White, and V. Yakimenko. Machine learning-based longitudinal phase space prediction of two-bunch operation at FACET-II. In *Proceedings of the 8th International Beam Instrumentation Conference*, 2019.
- [82] P. Emma, J. Frisch, and P. Krejcik. A transverse RF deflecting structure for bunch length and phase diagnostic. Technical report, TN-00-12, Stanford Linear Accelerator Center, 2000.

- [83] P. Emma, R. Akre, J. Arthur, R. Bionta, C. Bostedt, J. Bozek, A. Brachmann, P. Bucksbaum, R. Coffee, F. J. Decker, Y. Ding, D. Dowell, S. Edstrom, A. Fisher, J. Frisch, S. Gilevich, J. Hastings, G. Hays, P. Hering, Z. Huang, R. Iverson, H. Loos, M. Messerschmidt, A. Miahnahri, S. Moeller, H. D. Nuhn, G. Pile, D. Ratner, J. Rzepiela, D. Schultz, T. Smith, P. Stefan, H. Tompkins, J. Turner, J. Welch, W. White, J. Wu, G. Yocky, and J. Galayda. First lasing and operation of an ångstrom-wavelength free-electron laser. *Nature Photonics*, 4, 2010.
- [84] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek. Scalable global optimization via local bayesian optimization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.
- [85] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, S. Legg, and K. Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, 2018.
- [86] European XFEL. Annual Report 2023, 2024.
- [87] W. Falcon and The PyTorch Lightning team. PyTorch Lightning, 2019.
- [88] Farama Foundation. Gymnasium, 2022.
- [89] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatin, A. Novikov, F. J. R. Ruiz, J. Schrittwieser, G. Swirszcz, D. Silver, D. Hassabis, and P. Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610, 2022.
- [90] A. Ferran Pousa, S. Jalas, M. Kirchen, A. Martinez de la Ossa, M. Thévenet, S. Hudson, J. Larson, A. Huebl, J.-L. Vay, and R. Lehe. Bayesian optimization of laser-plasma accelerators assisted by reduced physical models. *arXiv*, 2212.12551, 2022.
- [91] A. Ferran Pousa, S. Jalas, M. Kirchen, A. Martinez de la Ossa, M. Thévenet, S. Hudson, J. Larson, A. Huebl, J.-L. Vay, and R. Lehe. Bayesian optimization of laser-plasma accelerators assisted by reduced physical models. *Physical Review Accelerators and Beams*, 26, 2023.
- [92] E. Fiesler and S. R. Campbell. Hybrid neural networks and their application to particle accelerator control. In *Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation II*, volume 3812, 1999.
- [93] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, 2017.
- [94] K. Flöttmann. ASTRA – A space charge tracking algorithm, 1997.
- [95] E. Fol. *Application of Machine Learning in Beam Optics Measurements and Corrections*. PhD thesis, Goethe University Frankfurt, 2021.
- [96] E. Fol, R. Tomás, J. Coello de Portugal, and G. Franchetti. Detection of faulty beam position monitors using unsupervised learning. *Physical Review Accelerators and Beams*, 23, 2020.

- [97] E. Fol, C. Rogers, and D. Schulte. Machine learning-based modeling of muon beam ionization cooling. In *Proceedings of the 13th International Particle Accelerator Conference (IPAC)*, 2022.
- [98] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax - A differentiable physics engine for large scale rigid body simulation, 2021.
- [99] Freie und Hansestadt Hamburg, Landesbetrieb Geoinformation und Vermessung (LGV). Luftbilder Hamburg - DOP Zeitreihe belaubt (DOP 2019, 20 cm, RGB, Hamburg), 2025. URL <https://suche.transparenz.hamburg.de/dataset/luftbilder-hamburg-dop-zeitreihe-belaubt>. Accessed 2025-10-16. Licensed under dl-de/by-2-0.
- [100] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. *arXiv*, 1802.09477v3, 2018.
- [101] K. Fujita. Physics-informed neural network method for space charge effect in particle accelerators. *IEEE Access*, 9, 2021.
- [102] J. N. Galayda. The LCLS-II: A high power upgrade to the LCLS. In *Proceedings of the 9th International Particle Accelerator Conference (IPAC)*, 2018.
- [103] Y. Gao, J. Chen, T. Robertazzi, and K. A. Brown. Reinforcement learning based schemes to manage client activities in large distributed control systems. *Physical Review Accelerators and Beams*, 22, 2019.
- [104] Y. Gao, K. Brown, A. L. Edelen, E. Hamwi, G. H. Hoffstaetter, A. Kasparian, W. Lin, L. Nguyen, D. Sagan, V. Schoefer, M. Schram, J. Unger, Y. Wang, and K. Zeno. Optimization of AGS bunch merging with reinforcement learning. In *Proceedings of the 15th Particle Accelerator Conference (IPAC)*, 2024.
- [105] R. Garnett. *Bayesian Optimization*. Cambridge University Press, 2023.
- [106] S. Gasiorowski, Y. Chen, Y. Nashed, P. Granger, C. Mironov, D. Ratner, K. Terao, and K. V. Tsang. Differentiable simulation of a liquid argon time projection chamber. *arXiv*, 2309.04639, 2023.
- [107] Gemini Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, D. Silver, M. Johnson, I. Antonoglou, J. Schrittwieser, A. Glaese, J. Chen, E. Pitler, T. Lillicrap, A. Lazaridou, O. Firat, J. Molloy, M. Isard, P. R. Barham, T. Hennigan, B. Lee, F. Viola, M. Reynolds, Y. Xu, R. Doherty, E. Collins, C. Meyer, E. Rutherford, E. Moreira, K. Ayoub, M. Goel, J. Krawczyk, C. Du, E. Chi, H.-T. Cheng, E. Ni, P. Shah, P. Kane, B. Chan, M. Faruqui, A. Severyn, H. Lin, Y. Li, Y. Cheng, A. Ittycheriah, M. Mahdieh, M. Chen, P. Sun, D. Tran, S. Bagri, B. Lakshminarayanan, J. Liu, A. Orban, F. Gura, H. Zhou, X. Song, A. Boffy, H. Ganapathy, S. Zheng, H. Choe, A. Weisz, T. Zhu, Y. Lu, S. Gopal, J. Kahn, M. Kula, J. Pitman, R. Shah, E. Taropa, M. Al Meray, M. Baeuml, Z. Chen, L. El Shafey, Y. Zhang, O. Sercinoglu, G. Tucker, E. Piqueras, M. Krikun, I. Barr, N. Savinov, I. Danihelka, B. Roelofs, A. White, A. Andreassen, T. von Glehn, L. Yagati, M. Kazemi, L. Gonzalez, M. Khalman, J. Sygnowski, A. Frechette, C. Smith, L. Culp, L. Proleev, Y. Luan, X. Chen, J. Lottes, N. Schucher, F. Lebron, A. Rrustemi, N. Clay, P. Crone, T. Kocisky, J. Zhao, B. Perz, D. Yu, H. Howard, A. Bloniarz, J. W. Rae, H. Lu, L. Sifre, M. Maggioni,

F. Alcober, D. Garrette, M. Barnes, S. Thakoor, J. Austin, G. Barth-Maron, W. Wong, R. Joshi, R. Chaabouni, D. Fatiha, A. Ahuja, G. S. Tomar, E. Senter, M. Chadwick, I. Kornakov, N. Attaluri, I. n. Iturrate, R. Liu, Y. Li, S. Cogan, J. Chen, C. Jia, C. Gu, Q. Zhang, J. Grimstad, A. J. Hartman, X. Garcia, T. S. Pillai, J. Devlin, M. Laskin, D. de Las Casas, D. Valter, C. Tao, L. Blanco, A. Puigdomènech Badia, D. Reitter, M. Chen, J. Brennan, C. Rivera, S. Brin, S. Iqbal, G. Surita, J. Labanowski, A. Rao, S. Winkler, E. Parisotto, Y. Gu, K. Olszewska, R. Addanki, A. Miech, A. Louis, D. Teplyashin, G. Brown, E. Catt, J. Balaguer, J. Xiang, P. Wang, Z. Ashwood, A. Briukhov, A. Webson, S. Ganapathy, S. Sanghavi, A. Kannan, M.-W. Chang, A. Stjerngren, J. Djolonga, Y. Sun, A. Bapna, M. Aitchison, P. Pejman, H. Michalewski, T. Yu, C. Wang, J. Love, J. Ahn, D. Bloxwich, K. Han, P. Humphreys, T. Sellam, J. Bradbury, V. Godbole, S. Samangoeei, B. Damoc, A. Kaskasoli, S. M. R. Arnold, V. Vasudevan, S. Agrawal, J. Riesa, D. Lepikhin, R. Tanburn, S. Srinivasan, H. Lim, S. Hodgkinson, P. Shyam, J. Ferret, S. Hand, A. Garg, T. Le Paine, J. Li, Y. Li, M. Giang, A. Neitz, Z. Abbas, S. York, M. Reid, E. Cole, A. Chowdhery, D. Das, D. Rogozińska, V. Nikolaev, P. Sprechmann, Z. Nado, L. Zilka, F. Prost, L. He, M. Monteiro, G. Mishra, C. Welty, J. Newlan, D. Jia, M. Allamanis, C. H. Hu, R. de Liedekerke, J. Gilmer, C. Saroufim, S. Rijhwani, S. Hou, D. Shrivastava, A. Baddepudi, A. Goldin, A. Ozturel, A. Casirer, Y. Xu, D. Sohn, D. Sachan, R. K. Amplayo, C. Swanson, D. Petrova, S. Narayan, A. Guez, S. Brahma, J. Landon, M. Patel, R. Zhao, K. Villela, L. Wang, W. Jia, M. Rahtz, M. Giménez, L. Yeung, J. Keeling, P. Georgiev, D. Mincu, B. Wu, S. Haykal, R. Saputro, K. Vodrahalli, J. Qin, Z. Cankara, A. Sharma, N. Fernando, W. Hawkins, B. Neyshabur, S. Kim, A. Hutter, P. Agrawal, A. Castro-Ros, G. van den Driessche, T. Wang, F. Yang, S.-y. Chang, P. Komarek, R. McIlroy, M. Lučić, G. Zhang, W. Farhan, M. Sharman, P. Natsev, P. Michel, Y. Bansal, S. Qiao, K. Cao, S. Shakeri, C. Butterfield, J. Chung, P. K. Rubenstein, S. Agrawal, A. Mensch, K. Soparkar, K. Lenc, T. Chung, A. Pope, L. Maggiore, J. Kay, P. Jhakra, S. Wang, J. Maynez, M. Phuong, T. Tobin, A. Tacchetti, M. Trebacz, K. Robinson, Y. Katariya, S. Riedel, P. Bailey, K. Xiao, N. Ghelani, L. Aroyo, A. Slone, N. Houlsby, X. Xiong, Z. Yang, E. Gribovskaya, J. Adler, M. Wirth, L. Lee, M. Li, T. Kagohara, J. Pavagadhi, S. Bridgers, A. Bortsova, S. Ghemawat, Z. Ahmed, T. Liu, R. Powell, V. Bolina, M. Iinuma, P. Zablotskaia, J. Besley, D.-W. Chung, T. Dozat, R. Comanescu, X. Si, J. Greer, G. Su, M. Polacek, R. Lopez Kaufman, S. Tokumine, H. Hu, E. Buchatskaya, Y. Miao, M. Elhawaty, A. Siddhant, N. Tomasev, J. Xing, C. Greer, H. Miller, S. Ashraf, A. Roy, Z. Zhang, A. Ma, A. Filos, M. Besta, R. Blevins, T. Klimenko, C.-K. Yeh, S. Changpinyo, J. Mu, O. Chang, M. Pajarskas, C. Muir, V. Cohen, C. Le Lan, K. Haridasan, A. Marathe, S. Hansen, S. Douglas, R. Samuel, M. Wang, S. Austin, C. Lan, J. Jiang, J. Chiu, J. Alonso Lorenzo, L. Lowe Sjöstrand, S. Cevey, Z. Gleicher, T. Avrahami, A. Boral, H. Srinivasan, V. Selo, R. May, K. Aisopos, L. Hussenot, L. Baldini Soares, K. Baumli, M. B. Chang, A. Recasens, B. Caine, A. Pritzel, F. Pavetic, F. Pardo, A. Gergely, J. Frye, V. Ramasesh, D. Horgan, K. Badola, N. Kassner, S. Roy, E. Dyer, V. Campos Campos, A. Tomala, Y. Tang, D. El Badawy, E. White, B. Mustafa, O. Lang, A. Jindal, S. Vikram, Z. Gong, S. Caelles, R. Hemsley, G. Thornton, F. Feng, W. Stokowiec, C. Zheng, P. Thacker, c. Ünlü, Z. Zhang, M. Saleh, J. Svensson, M. Bileschi, P. Patil, A. Anand, R. Ring, K. Tsihlias, A. Vezer, M. Selvi, T. Shevlane, M. Rodriguez, T. Kwiatkowski, S. Daruki, K. Rong, A. Dafoe, N. FitzGerald, K. Gu-Lemberg, M. Khan, L. A. Hendricks, M. Pellet, V. Feinberg, J. Cobon-Kerr, T. Sainath, M. Rauh, S. H. Hashemi, R. Ives, Y. Hasson, E. Noland, Y. Cao, N. Byrd, L. Hou, Q. Wang, T. Sottiaux, M. Paganini, J.-B. Lespiau,

A. Moufarek, S. Hassan, K. Shivakumar, J. van Amersfoort, A. Mandhane, P. Joshi, A. Goyal, M. Tung, A. Brock, H. Sheahan, V. Misra, C. Li, N. Rakićević, M. Dehghani, F. Liu, S. Mittal, J. Oh, S. Noury, E. Sezener, F. Huot, M. Lamm, N. De Cao, C. Chen, S. Mudgal, R. Stella, K. Brooks, G. Vasudevan, C. Liu, M. Chain, N. Melinkeri, A. Cohen, V. Wang, K. Seymore, S. Zubkov, R. Goel, S. Yue, S. Krishnakumaran, B. Albert, N. Hurley, M. Sano, A. Mohananey, J. Joughin, E. Filonov, T. Kepa, Y. Eldawy, J. Lim, R. Rishi, S. Badiezadegan, T. Bos, J. Chang, S. Jain, S. G. Sundara Padmanabhan, S. Puttagunta, K. Krishna, L. Baker, N. Kalb, V. Bedapudi, A. Kurzrok, S. Lei, A. Yu, O. Litvin, X. Zhou, Z. Wu, S. Sobell, A. Siciliano, A. Papir, R. Neale, J. Bragagnolo, T. Toor, T. Chen, V. Anklin, F. Wang, R. Feng, M. Gholami, K. Ling, L. Liu, J. Walter, H. Moghaddam, A. Kishore, J. Adamek, T. Mercado, J. Mallinson, S. Wandekar, S. Cagle, E. Ofek, G. Garrido, C. Lombriser, M. Mukha, B. Sun, H. R. Mohammad, J. Matak, Y. Qian, V. Peswani, P. Janus, Q. Yuan, L. Schelin, O. David, A. Garg, Y. He, O. Duzhyi, A. Älgmyr, T. Lottaz, Q. Li, V. Yadav, L. Xu, A. Chinien, R. Shivanna, A. Chuklin, J. Li, C. Spadine, T. Wolfe, K. Mohamed, S. Das, Z. Dai, K. He, D. von Dincklage, S. Upadhyay, A. Maurya, L. Chi, S. Krause, K. Salama, P. G. Rabinovitch, P. Kumar Reddy M, A. Selvan, M. Dektiarev, G. Ghiasi, E. Guven, H. Gupta, B. Liu, D. Sharma, I. Heimlich Shtacher, S. Paul, O. Akerlund, F.-X. Aubet, T. Huang, C. Zhu, E. Zhu, E. Teixeira, M. Fritze, F. Bertolini, L.-E. Marinescu, M. Bölle, D. Paulus, K. Gupta, T. Latkar, M. Chang, J. Sanders, R. Wilson, X. Wu, Y.-X. Tan, L. N. Thiet, T. Doshi, S. Lall, S. Mishra, W. Chen, T. Luong, S. Benjamin, J. Lee, E. Andrejczuk, D. Rabiej, V. Ranjan, K. Styrac, P. Yin, J. Simon, M. R. Harriott, M. Bansal, A. Robsky, G. Bacon, D. Greene, D. Mirylenka, C. Zhou, O. Sarvana, A. Goyal, S. Andermatt, P. Siegler, B. Horn, A. Israel, F. Pongetti, C.-W. L. Chen, M. Selvatici, P. Silva, K. Wang, J. Tolins, K. Guu, R. Yogeve, X. Cai, A. Agostini, M. Shah, H. Nguyen, N. Ó Donnaile, S. Pereira, L. Friso, A. Stambler, A. Kurzrok, C. Kuang, Y. Romanikhin, M. Geller, Z. Yan, K. Jang, C.-C. Lee, W. Fica, E. Malmi, Q. Tan, D. Banica, D. Balle, R. Pham, Y. Huang, D. Avram, H. Shi, J. Singh, C. Hidey, N. Ahuja, P. Saxena, D. Dooley, S. P. Potharaju, E. O'Neill, A. Gokulchandran, R. Foley, K. Zhao, M. Dusenberry, Y. Liu, P. Mehta, R. Kotikalapudi, C. Safranek-Shrader, A. Goodman, J. Kessinger, E. Globen, P. Kolhar, C. Gorgolewski, A. Ibrahim, Y. Song, A. Eichenbaum, T. Brovelli, S. Potluri, P. Lahoti, C. Baetu, A. Ghorbani, C. Chen, A. Crawford, S. Pal, M. Sridhar, P. Gurita, A. Mujika, I. Petrovski, P.-L. Cedoz, C. Li, S. Chen, N. Dal Santo, S. Goyal, J. Punjabi, K. Kappaganthu, C. Kwak, P. LV, S. Velury, H. Choudhury, J. Hall, P. Shah, R. Figueira, M. Thomas, M. Lu, T. Zhou, C. Kumar, T. Jurdi, S. Chikkerur, Y. Ma, A. Yu, S. Kwak, V. Ähdel, S. Rajayogam, T. Choma, F. Liu, A. Barua, C. Ji, J. H. Park, V. Hellendoorn, A. Bailey, T. Bilal, H. Zhou, M. Khatir, C. Sutton, W. Rzadkowski, F. Macintosh, K. Shagin, P. Medina, C. Liang, J. Zhou, P. Shah, Y. Bi, A. Dankovics, S. Banga, S. Lehmann, M. Bredesen, Z. Lin, J. E. Hoffmann, J. Lai, R. Chung, K. Yang, N. Balani, A. Bražinskas, A. Sozanschi, M. Hayes, H. Fernández Alcalde, P. Makarov, W. Chen, A. Stella, L. Snijders, M. Mandl, A. Kärrman, P. Nowak, X. Wu, A. Dyck, K. Vaidyanathan, R. R, J. Mallet, M. Rudominer, E. Johnston, S. Mittal, A. Udathu, J. Christensen, V. Verma, Z. Irving, A. Santucci, G. Elsayed, E. Davoodi, M. Georgiev, I. Tenney, N. Hua, G. Cideron, E. Leurent, M. Alnahlawi, I. Georgescu, N. Wei, I. Zheng, D. Scandinaro, H. Jiang, J. Snoek, M. Sundararajan, X. Wang, Z. Ontiveros, I. Karo, J. Cole, V. Rajashekhar, L. Tumeh, E. Ben-David, R. Jain, J. Uesato, R. Datta, O. Bunyan, S. Wu, J. Zhang, P. Stanczyk, Y. Zhang, D. Steiner, S. Naskar, M. Azzam, M. Johnson, A. Paszke, C.-C. Chiu, J. Sanchez Elias, A. Mohiuddin,

F. Muhammad, J. Miao, A. Lee, N. Vieillard, J. Park, J. Zhang, J. Stanway, D. Garmon, A. Karmarkar, Z. Dong, J. Lee, A. Kumar, L. Zhou, J. Evens, W. Isaac, G. Irving, E. Loper, M. Fink, I. Arkatkar, N. Chen, I. Shafran, I. Petrychenko, Z. Chen, J. Jia, A. Levskaya, Z. Zhu, P. Grabowski, Y. Mao, A. Magni, K. Yao, J. Snaider, N. Casagrande, E. Palmer, P. Suganthan, A. Castaño, I. Giannoumis, W. Kim, M. Rybiński, A. Sreevatsa, J. Prendki, D. Soergel, A. Goedeckemeyer, W. Gierke, M. Jafari, M. Gaba, J. Wiesner, D. Gage Wright, Y. Wei, H. Vashisht, Y. Kulizhskaya, J. Hoover, M. Le, L. Li, C. Iwuanyanwu, L. Liu, K. Ramirez, A. Khorlin, A. Cui, T. LIN, M. Wu, R. Aguilar, K. Pallo, A. Chakladar, G. Perng, E. A. Abellan, M. Zhang, I. Dasgupta, N. Kushman, I. Penchev, A. Repina, X. Wu, T. van der Weide, P. Ponnappalli, C. Kaplan, J. Simsa, S. Li, O. Dousse, F. Yang, J. Piper, N. Ie, R. Pasumarthi, N. Lintz, A. Vijayakumar, D. Andor, P. Valenzuela, M. Lui, C. Paduraru, D. Peng, K. Lee, S. Zhang, S. Greene, D. D. Nguyen, P. Kurylowicz, C. Hardin, L. Dixon, L. Janzer, K. Choo, Z. Feng, B. Zhang, A. Singhal, D. Du, D. McKinnon, N. Antropova, T. Bolukbasi, O. Keller, D. Reid, D. Finchelstein, M. Abi Raad, R. Crocker, P. Hawkins, R. Dadashi, C. Gaffney, K. Franko, A. Bulanova, R. Leblond, S. Chung, H. Askham, L. C. Cobo, K. Xu, F. Fischer, J. Xu, C. Sorokin, C. Alberti, C.-C. Lin, C. Evans, A. Dimitriev, H. Forbes, D. Banarse, Z. Tung, M. Omernick, C. Bishop, R. Sterneck, R. Jain, J. Xia, E. Amid, F. Piccinno, X. Wang, P. Banzal, D. J. Mankowitz, A. Polozov, V. Krakovna, S. Brown, M. Bateni, D. Duan, V. Firoiu, M. Thotakuri, T. Natan, M. Geist, S. t. Girgin, H. Li, J. Ye, O. Roval, R. Tojo, M. Kwong, J. Lee-Thorp, C. Yew, D. Sinopalnikov, S. Ramos, J. Mellor, A. Sharma, K. Wu, D. Miller, N. Sonnerat, D. Vnukov, R. Greig, J. Beattie, E. Caveness, L. Bai, J. Eisenschlos, A. Korchemniy, T. Tsai, M. Jasarevic, W. Kong, P. Dao, Z. Zheng, F. Liu, F. Yang, R. Zhu, T. H. Teh, J. Sanmiya, E. Gladchenko, N. Trdin, D. Toyama, E. Rosen, S. Tavakkol, L. Xue, C. Elkind, O. Woodman, J. Carpenter, G. Papamakarios, R. Kemp, S. Kafle, T. Grunina, R. Sinha, A. Talbert, D. Wu, D. Owusu-Afriyie, C. Du, C. Thornton, J. Pont-Tuset, P. Narayana, J. Li, S. Fatehi, J. Wieting, O. Ajmeri, B. Uria, Y. Ko, L. Knight, A. Héliou, N. Niu, S. Gu, C. Pang, Y. Li, N. Levine, A. Stolovich, R. Santamaria-Fernandez, S. Goenka, W. Yustalim, R. Strudel, A. Elqursh, C. Deck, H. Lee, Z. Li, K. Levin, R. Hoffmann, D. Holtmann-Rice, O. Bachem, S. Arora, C. Koh, S. Hassas Yeganeh, S. Pöder, M. Tariq, Y. Sun, L. Ionita, M. Seyedhosseini, P. Tafti, Z. Liu, A. Gulati, J. Liu, X. Ye, B. Chrzaszcz, L. Wang, N. Sethi, T. Li, B. Brown, S. Singh, W. Fan, A. Parisi, J. Stanton, V. Koverkathu, C. A. Choquette-Choo, Y. Li, T. Lu, A. Ittycheriah, P. Shroff, M. Varadarajan, S. Bahargam, R. Willoughby, D. Gaddy, G. Desjardins, M. Cornero, B. Robenek, B. Mittal, B. Albrecht, A. Shenoy, F. Moiseev, H. Jacobsson, A. Ghaffarkhah, M. Rivière, A. Walton, C. Crepy, A. Parrish, Z. Zhou, C. Farabet, C. Radebaugh, P. Srinivasan, C. van der Salm, A. Fidjeland, S. Scellato, E. Latorre-Chimoto, H. Klimczak-Plucińska, D. Bridson, D. de Cesare, T. Hudson, P. Mendolicchio, L. Walker, A. Morris, M. Mauger, A. Guseynov, A. Reid, S. Odoom, L. Loher, V. Cotruta, M. Yenugula, D. Grewe, A. Petrushkina, T. Duerig, A. Sanchez, S. Yadlowsky, A. Shen, A. Globerson, L. Webb, S. Dua, D. Li, S. Bhupatiraju, D. Hurt, H. Qureshi, A. Agarwal, T. Shani, M. Eyal, A. Khare, S. Rammohan Belle, L. Wang, C. Tekur, M. S. Kale, J. Wei, R. Sang, B. Saeta, T. Liechty, Y. Sun, Y. Zhao, S. Lee, P. Nayak, D. Fritz, M. Reddy Vuyyuru, J. Aslanides, N. Vyas, M. Wicke, X. Ma, E. Eltyshev, N. Martin, H. Cate, J. Manyika, K. Amiri, Y. Kim, X. Xiong, K. Kang, F. Luisier, N. Tripuraneni, D. Madras, M. Guo, A. Waters, O. Wang, J. Ainslie, J. Baldridge, H. Zhang, G. Pruthi, J. Bauer, F. Yang, R. Mansour, J. Gelman, Y. Xu, G. Polovets, J. Liu, H. Cai, W. Chen, X. Sheng, E. Xue, S. Ozair, C. An-

- germueller, X. Li, A. Sinha, W. Wang, J. Wiesinger, E. Koukoumidis, Y. Tian, A. Iyer, M. Gurumurthy, M. Goldenson, P. Shah, M. Blake, H. Yu, A. Urbanowicz, J. Palomaki, C. Fernando, K. Durden, H. Mehta, N. Momchev, E. Rahimtoroghi, M. Georgaki, A. Raul, S. Ruder, M. Redshaw, J. Lee, D. Zhou, K. Jalan, D. Li, B. Hechtman, P. Schuh, M. Nasr, K. Milan, V. Mikulik, J. Franco, T. Green, N. Nguyen, J. Kelley, A. Mahendru, A. Hu, J. Howland, B. Vargas, J. Hui, K. Bansal, V. Rao, R. Ghiya, E. Wang, K. Ye, J. M. Sarr, M. Moranski Preston, M. Elish, S. Li, A. Kaku, J. Gupta, I. Pasupat, D.-C. Juan, M. Someswar, T. M., X. Chen, A. Amini, A. Fabrikant, E. Chu, X. Dong, A. Muthal, S. Buthpitiya, S. Jauhari, N. Hua, U. Khandelwal, A. Hitron, J. Ren, L. Rinaldi, S. Drath, A. Dabush, N.-J. Jiang, H. Godhia, U. Sachs, A. Chen, Y. Fan, H. Taitelbaum, H. Noga, Z. Dai, J. Wang, C. Liang, J. Hamer, C.-S. Ferng, C. Elkind, A. Atias, P. Lee, V. Listík, M. Carlen, J. van de Kerkhof, M. Pikus, K. Zaher, P. Müller, S. Zykova, R. Stefanec, V. Gatsko, C. Hirnschall, A. Sethi, X. F. Xu, C. Ahuja, B. Tsai, A. Stefanoiu, B. Feng, K. Dhandhania, M. Katyál, A. Gupta, A. Parulekar, D. Pitta, J. Zhao, V. Bhatia, Y. Bhavnani, O. Alhadlaq, X. Li, P. Danenberg, D. Tu, A. Pine, V. Filippova, A. Ghosh, B. Limonchik, B. Urala, C. K. Lanka, D. Clive, Y. Sun, E. Li, H. Wu, K. Hongtongsak, I. Li, K. Thakkar, K. Omarov, K. Majmundar, M. Alverson, M. Kucharski, M. Patel, M. Jain, M. Zabelin, P. Pelagatti, R. Kohli, S. Kumar, J. Kim, S. Sankar, V. Shah, L. Ramachandrani, X. Zeng, B. Bariach, L. Weidinger, T. Vu, A. Andreev, A. He, K. Hui, S. Kashem, A. Subramanya, S. Hsiao, D. Hassabis, K. Kavukcuoglu, A. Sadovskiy, Q. Le, T. Strohman, Y. Wu, S. Petrov, J. Dean, and O. Vinyals. Gemini: A family of highly capable multimodal models. *arXiv*, 2312.11805, 2023.
- [108] Gemma Team, T. Mesnard, C. Hardin, R. Dadashi, S. Bhupatiraju, S. Pathak, L. Sifre, M. Rivière, M. S. Kale, J. Love, P. Tafti, L. Hussenot, P. G. Sessa, A. Chowdhery, A. Roberts, A. Barua, A. Botev, A. Castro-Ros, A. Slone, A. Héliou, A. Tacchetti, A. Bulanova, A. Paterson, B. Tsai, B. Shahriari, C. Le Lan, C. A. Choquette-Choo, C. Crepy, D. Cer, D. Ippolito, D. Reid, E. Buchatskaya, E. Ni, E. Noland, G. Yan, G. Tucker, G.-C. Muraru, G. Rozhdestvenskiy, H. Michalewski, I. Tenney, I. Grishchenko, J. Austin, J. Keeling, J. Labanowski, J.-B. Lespiau, J. Stanway, J. Brennan, J. Chen, J. Ferret, J. Chiu, J. Mao-Jones, K. Lee, K. Yu, K. Millican, L. Lowe Sjoesund, L. Lee, L. Dixon, M. Reid, M. Mikuła, M. Wirth, M. Sharman, N. Chinaev, N. Thain, O. Bachem, O. Chang, O. Wahltinez, P. Bailey, P. Michel, P. Yotov, R. Chaabouni, R. Comanescu, R. Jana, R. Anil, R. McIlroy, R. Liu, R. Mullins, S. L. Smith, S. Borgeaud, S. Girgin, S. Douglas, S. Pandya, S. Shakeri, S. De, T. Klimenko, T. Hennigan, V. Feinberg, W. Stokowiec, Y.-h. Chen, Z. Ahmed, Z. Gong, T. Warkentin, L. Peran, M. Giang, C. Farabet, O. Vinyals, J. Dean, K. Kavukcuoglu, D. Hassabis, Z. Ghahramani, D. Eck, J. Barral, F. Pereira, E. Collins, A. Joulin, N. Fiedel, E. Senter, A. Andreev, and K. Kenealy. Gemma: Open models based on Gemini research and technology. *arXiv*, 2403.08295, 2024.
- [109] M. F. Giannelli, G. Kasieczka, C. Krause, B. Nachman, D. Salamani, D. Shih, and A. Zaborowska. Fast calorimeter simulation challenge 2022, 2022.
- [110] J. P. Gonzalez-Aguilera, Y. Kim, R. Roussel, A. L. Edelen, and C. Mayes. Towards fully differentiable accelerator modeling. In *Proceedings of the 13th International Particle Accelerator Conference (IPAC)*, 2023.
- [111] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

- [112] L. Grech. *Renovation of the beam-based feedback systems in the LHC*. PhD thesis, University of Malta, 2021.
- [113] L. Grech, G. Valentino, D. Alves, and S. Hirilaender. Application of reinforcement learning in the LHC tune feedback. *Frontiers in Physics*, 10, 2022.
- [114] A. Grünhagen, J. Branlard, A. Eichler, G. Martino, G. Fey, and M. Tropmann-Frick. Fault analysis of the beam acceleration control system at the European XFEL using data mining. In *2021 IEEE 30th Asian Test Symposium (ATS)*, 2021.
- [115] A. Grünhagen, A. Eichler, M. Tropmann-Frick, and G. Fey. Data-based condition monitoring and disturbance classification in actively controlled laser oscillators. In *Information Modelling and Knowledge Bases XXXV*, 2024.
- [116] D. Guerra-Ramos, J. Trujillo-Sevilla, and J. M. Rodríguez-Ramos. Towards piston fine tuning of segmented mirrors through reinforcement learning. *Applied Sciences (Switzerland)*, 10, 2020.
- [117] P. Guilmin, R. Gautier, A. Bocquet, and E. Genois. dynamiqs: An open-source Python library for GPU-accelerated and differentiable simulation of quantum systems, 2024.
- [118] L. Gupta, A. L. Edelen, N. Neveu, A. Mishra, C. Mayes, and Y.-K. Kim. Improving surrogate model accuracy for the LCLS-II injector frontend using convolutional neural networks and transfer learning. *Machine Learning: Science and Technology*, 2, 2021.
- [119] W. H. Guss, B. Houghton, N. Topin, P. Wang, C. Codel, M. Veloso, and R. Salakhutdinov. MineRL: A large-scale dataset of minecraft demonstrations. *arXiv*, 1907.13440, 2019.
- [120] T. Haarnoja, B. Moran, G. Lever, S. H. Huang, D. Tirumala, J. Humplik, M. Wulfmeier, S. Tunyasuvunakool, N. Y. Siegel, R. Hafner, M. Bloesch, K. Hartikainen, A. Byravan, L. Hasenclever, Y. Tassa, F. Sadeghi, N. Batchelor, F. Casarini, S. Saliceti, C. Game, N. Sreendra, K. Patel, M. Gwira, A. Huber, N. Hurley, F. Nori, R. Hadsell, and N. Heess. Learning agile soccer skills for a bipedal robot with deep reinforcement learning. *Science Robotics*, 9, 2024.
- [121] D. Hafner, K.-H. Lee, I. Fischer, and P. Abbeel. Deep hierarchical planning from pixels. *Advances in Neural Information Processing Systems*, 2022.
- [122] J. Hanten, M. Arnold, J. Birkhan, C. Caliarì, N. Pietralla, and M. Steinhorst. Enhancement of the S-DALINAC control system with machine learning methods. In *Proceedings of the 8th International Beam Instrumentation Conference*, 2019.
- [123] A. Hanuka, J. Duris, J. Shtalenkova, D. Kennedy, A. L. Edelen, D. Ratner, and X. Huang. Online tuning and light source control using a physics-informed Gaussian process. In *Proceedings of the 33rd Conference on Neural Information Processing Systems*, 2019.
- [124] A. Hanuka, C. Emma, T. Maxwell, A. S. Fisher, B. Jacobson, M. J. Hogan, and Z. Huang. Accurate and confident prediction of electron beam longitudinal properties using spectral virtual diagnostics. *Scientific Reports*, 11, 2021.

- [125] D. Hendrycks and K. Gimpel. Gaussian error linear units (GELUs). *arXiv*, 1606.08415, 2023.
- [126] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. *arXiv*, 2009.03300, 2021.
- [127] T. Higo, H. Shoaee, and J. Spencer. Some applications of AI to the problems of accelerator physics. *Proceedings of the 1987 IEEE Particle Accelerator Conference: Accelerator Engineering and Technology*, 1987.
- [128] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable Baselines, 2018.
- [129] G. E. Hinton and R. Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Advances in Neural Information Processing Systems*, volume 6, 1993.
- [130] S. Hirlaender and N. Bruchon. Model-free and Bayesian ensembling model-based deep reinforcement learning for particle accelerator control demonstrated on the FERMI FEL. *arXiv*, 2012.09737, 2020.
- [131] S. Hirlaender, L. Lamminger, Z. Della Porta, and V. Kain. Ultra fast reinforcement learning demonstrated at CERN AWAKE. In *Proceedings of the 14th International Particle Accelerator Conference (IPAC)*, 2023.
- [132] S. Hirlaender, L. Lamminger, S. Pochaba, J. Kaiser, C. Xu, A. Santamaria Garcia, L. Scomparin, and V. Kain. Towards few-shot reinforcement learning in particle accelerator control. In *Proceedings of the 15th International Particle Accelerator Conference (IPAC)*, 2024.
- [133] S. Hirlaender, S. Pochaba, L. Lukas, A. Santamaria Garcia, C. Xu, J. Kaiser, A. Eichler, and V. Kain. Deep meta reinforcement learning for rapid adaptation in linear Markov decision processes: Applications to CERN's AWAKE project. In *Combining, Modelling and Analyzing Imprecision, Randomness and Dependence*, 2024.
- [134] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9, 1997.
- [135] M. W. Hoffman, B. Shahriari, J. Aslanides, G. Barth-Maron, N. Momchev, D. Sino-palnikov, P. Stańczyk, S. Ramos, A. Raichuk, D. Vincent, L. Hussenot, R. Dadashi, G. Dulac-Arnold, M. Orsini, A. Jacq, J. Ferret, N. Vieillard, S. K. S. Ghasemipour, S. Girgin, O. Pietquin, F. Behbahani, T. Norman, A. Abdolmaleki, A. Cassirer, F. Yang, K. Baumli, S. Henderson, A. Friesen, R. Haroun, A. Novikov, S. Gómez Colmenarejo, S. Cabi, C. Gulcehre, T. Le Paine, S. Srinivasan, A. Cowie, Z. Wang, B. Piot, and N. de Freitas. Acme: A research framework for distributed reinforcement learning. *arXiv*, 2006.00979, 2020.
- [136] A. Hofler, B. c. v. Terzić, M. Kramer, A. Zvezdin, V. Morozov, Y. Roblin, F. Lin, and C. Jarvis. Innovative applications of genetic algorithms to problems in accelerator physics. *Physical Review Special Topics - Accelerators and Beams*, 16, 2013.

- [137] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2, 1989.
- [138] J. Howell, C. Barnes, S. Brown, G. Flake, R. Jones, Y. Lee, S. Qian, and R. Wright. Control of a negative-ion accelerator source using neural networks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 293, 1990.
- [139] X. Huang. Robust simplex algorithm for online optimization. *Physical Review Accelerators and Beams*, 21, 2018.
- [140] X. Huang, J. Corbett, J. Safranek, and J. Wu. An algorithm for online optimization of accelerators. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 726, 2013.
- [141] S. Hudson, J. Larson, J.-L. Navarro, and S. M. Wild. libEnsemble: A library to coordinate the concurrent evaluation of dynamic ensembles of calculations. *IEEE Transactions on Parallel and Distributed Systems*, 33, 2022.
- [142] A. Huebl, R. Lehe, J. Vay, D. Grote, I. Sbalzarini, S. Kuschel, and M. Bussmann. openPMD 1.0. 0: A meta data standard for particle and mesh based data, 2017.
- [143] G. Iadarola, R. De Maria, S. Lopaciuk, A. Abramov, X. Buffat, D. Demetriadou, L. Deniau, P. Hermes, P. Kicsiny, P. Kruyt, A. Latina, L. Mether, K. Paraschou, Sterbini, F. Van Der Veken, P. Belanger, P. Niedermayer, D. Di Croce, T. Pieloni, and L. Van Riesen-Haupt. Xsuite: An integrated beam physics simulation framework. *arXiv*, 2310.00317, 2023.
- [144] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv*, 1502.03167, 2015.
- [145] A. Irpan. Deep reinforcement learning doesn't work yet, 2018. URL <https://www.alexirpan.com/2018/02/14/r1-hard.html>. Accessed: 2022-01-26.
- [146] A. Ivanov and I. Agapov. Physics-based deep neural networks for beam dynamics in charged particle accelerators. *Physical Review Accelerators and Beams*, 23, 2020.
- [147] A. Ivanov, I. Agapov, A. Eichler, and S. Tomin. Physics-enhanced reinforcement learning for optimal control. In *Proceedings of the 12th International Particle Accelerator Conference (IPAC)*, 2021.
- [148] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. Garcia Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364, 2019.
- [149] S. Jalas, M. Kirchen, P. Messner, P. Winkler, L. Hübner, J. Dirkwinkel, M. Schnepf, R. Lehe, and A. Maier. Bayesian optimization of a laser-plasma accelerator. *Physical Review Letters*, 126, 2021.

- [150] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de las Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. Renard Lavaud, M.-A. Lachaux, P. Stock, T. Le Scao, T. Lavril, T. Wang, T. Lacroix, and W. El Sayed. Mistral 7B. *arXiv*, 2310.06825, 2023.
- [151] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. Bou Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. Renard Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. Le Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. El Sayed. Mixtral of Experts. *arXiv*, 2401.04088, 2024.
- [152] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13, 1998.
- [153] V. Kain, S. Hirlander, B. Goddard, F. M. Velotti, G. Z. Della Porta, N. Bruchon, and G. Valentino. Sample-efficient reinforcement learning for CERN accelerator control. *Physical Review Accelerators and Beams*, 23, 2020.
- [154] V. Kain, N. Bruchon, S. Hirlander, N. Madysa, I. Vojskovic, P. Skowronski, and G. Valentino. Test of machine learning at the CERN LINAC4, 2021.
- [155] J. Kaiser. A study of generated versus recorded geolocation data, 2019.
- [156] J. Kaiser. Lynx. GitHub, 2024.
- [157] J. Kaiser, C. Xu, C. Hespe, J. P. Gonzalez-Aguilera, R. Lehe, G. Charleux, R. Roussel, F. Theilen, and A. Santamaria Garcia. Cheetah. GitHub, 2021.
- [158] J. Kaiser, O. Stein, and A. Eichler. Learning-based optimisation of particle accelerators under partial observability without real-world training. In *Proceedings of the 39th International Conference on Machine Learning (ICML)*, volume 162, 2022.
- [159] J. Kaiser, C. Xu, A. Eichler, and A. Santamaria Garcia. Cheetah documentation. Read The Docs, 2022.
- [160] J. Kaiser, A. Eichler, S. Tomin, and Z. Zhu. Machine learning for combined scalar and spectral longitudinal phase space reconstruction. In *Proceedings of the 14th International Conference for Particle Accelerators (IPAC)*, 2023.
- [161] J. Kaiser, C. Xu, A. Eichler, A. Santamaria Garcia, O. Stein, E. Bründermann, W. Kur-opka, H. Dinter, F. Mayet, T. Vinatier, F. Burkart, and H. Schlarb. Learning to do or learning while doing: Reinforcement learning and Bayesian optimisation for online continuous tuning. *arXiv*, 2306.03739, 2023.
- [162] J. Kaiser, C. Xu, A. Eichler, and A. Santamaria Garcia. Bridging the gap between machine learning and particle accelerator physics with high-speed, differentiable simulations. *Physical Review Accelerators and Beams*, 27, 2024.
- [163] J. Kaiser, C. Xu, A. Eichler, and A. Santamaria Garcia. Cheetah: Bridging the gap between machine learning and particle accelerator physics with high-speed, differentiable simulations. *arXiv*, 2401.05815, 2024.

- [164] J. Kaiser, C. Xu, A. Eichler, A. Santamaria Garcia, O. Stein, E. Bründermann, W. Kur-opka, H. Dinter, F. Mayet, T. Vinatier, F. Burkart, and H. Schlarb. Reinforcement learning-trained optimisers and Bayesian optimisation for online particle accelerator tuning. *Scientific Reports*, 14, 2024.
- [165] J. Kaiser, , A. Lauscher, and A. Eichler. Large language models for human-machine collaborative particle accelerator tuning through natural language. *Science Advances*, 11, 2025.
- [166] S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning (ICML)*, 2002.
- [167] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv*, 2001.08361, 2020.
- [168] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney. Recurrent experience replay in distributed reinforcement learning. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*, 2019.
- [169] E. Kaufmann, L. Bauersfeld, A. Loquercio, M. Müller, V. Koltun, and D. Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620, 2023.
- [170] H. Kawahara, Y. Kawashima, K. Masuda, I. J. M. Crossfield, E. Pannier, and D. van den Bekerom. Autodifferentiable spectrum model for high-dispersion characterization of exoplanets and brown dwarfs. *The Astrophysical Journal Supplement Series*, 258, 2022.
- [171] Y. Kijima, K. Yoshida, M. Mizota, and K. Suzuki. A beam diagnostic system for accelerator using neural networks. In *Proceedings of the European Particle Accelerator Conference*, 1992.
- [172] K. Kim, K. Shim, J. Choi, M. Cho, W. Namkung, and I. S. Ko. A simulation study of the global orbit feedback system for Pohang light source. *Japanese Journal of Applied Physics*, 39, 2000.
- [173] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.
- [174] J. Kirschner, A. Adelman, N. Hiller, R. Ischebeck, A. Krause, M. Mutný, and M. Nonnenmacher. Bayesian optimisation for fast and safe parameter tuning of SwissFEL. In *Proceedings of 9th International Particle Accelerator Conference (IPAC)*, 2019.
- [175] J. Kirschner, M. Mutný, N. Hiller, R. Ischebeck, and A. Krause. Adaptive and safe Bayesian optimization in high dimensions via one-dimensional subspaces. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, 2019.
- [176] J. Kirschner, M. Mutný, A. Krause, J. C. De Portugal, N. Hiller, and J. Snuverink. Tuning particle accelerators with safety constraints using Bayesian optimization. *Physical Review Accelerators and Beams*, 25, 2022.

- [177] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118, 2021.
- [178] S. Koyamada, S. Okano, S. Nishimori, Y. Murata, K. Habara, H. Kita, and S. Ishii. Pgx: Hardware-accelerated parallel game simulators for reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.
- [179] Kraftfahrt-Bundesamt. CO2 emission and fuel consumption type test values of passenger vehicles with a maximum of nine seats and caravans (SV 2.2.2), 2024. URL [https://www.kba.de/SharedDocs/Downloads/DE/SV/sv222\\_m1\\_kraft\\_pdf.pdf?\\_\\_blob=publicationFile&v=16](https://www.kba.de/SharedDocs/Downloads/DE/SV/sv222_m1_kraft_pdf.pdf?__blob=publicationFile&v=16). Considered vehicle: 2013 Audi A6 Avant 3.0 TDI quattro (HSN: 0588, TSN: ARO000428), Status: 15. September 2024.
- [180] A. Krause and C. Ong. Contextual Gaussian process bandit optimization. In *Advances in Neural Information Processing Systems*, volume 24, 2011.
- [181] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.
- [182] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60, 2017.
- [183] R. Lai, U. Happek, and A. J. Sievers. Measurement of the longitudinal asymmetry of a charged particle bunch from the coherent synchrotron or transition radiation spectrum. *Physical Review E*, 50, 1994.
- [184] R. T. Lange. gymnax: A JAX-based reinforcement learning environment library. GitHub, 2022.
- [185] F. Lanusse, S. Casas, A. Peel, M. Karamanis, D. Kirkby, A. Boucaud, D. Lanzieri, J. Eric, and Y. Li. jax-cosmo. GitHub, 2020.
- [186] Y. LeCun. *Modèles Connexionnistes de l'Apprentissage*. PhD thesis, Université Paris 6, 1987.
- [187] Y. LeCun. Generalization and network design strategies. Technical report, University of Toronto, 1989.
- [188] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10, 1966.
- [189] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17, 2016.
- [190] K. Li and J. Malik. Learning to optimize. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.
- [191] K. Li and J. Malik. Learning to optimize neural nets. *arXiv*, 1703.00441, 2017.
- [192] K. Li, G. Zhou, Y. Liu, J. Wu, M.-f. Lin, X. Cheng, A. A. Lutman, M. Seaberg, H. Smith, P. A. Kakhandiki, and A. Sakdinawat. Prediction on X-ray output of free electron laser based on artificial neural networks. *Nature Communications*, 14, 2023.

- [193] P. Li, J. Yang, M. A. Islam, and S. Ren. Making AI less "thirsty": Uncovering and addressing the secret water footprint of AI models. *arXiv*, 2304.03271, 2023.
- [194] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.
- [195] Y. Li, D. Choi, J. Chung, N. Kushman, J. Schrittwieser, R. Leblond, T. Eccles, J. Keeling, F. Gimeno, A. D. Lago, T. Hubert, P. Choy, C. d. M. d'Autume, I. Babuschkin, X. Chen, P.-S. Huang, J. Welbl, S. Goyal, A. Cherepanov, J. Molloy, D. J. Mankowitz, E. S. Robson, P. Kohli, N. de Freitas, K. Kavukcuoglu, and O. Vinyals. Competition-level code generation with AlphaCode. *Science*, 378, 2022.
- [196] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. RLlib: Abstractions for distributed reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, 2018.
- [197] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2015.
- [198] N. M. Lockmann, C. Gerth, B. Schmidt, and S. Wesch. Noninvasive THz spectroscopy for bunch current profile reconstructions at MHz repetition rates. *Physical Review Accelerators and Beams*, 23, 2020.
- [199] N. M. A. Lockmann. *Noninvasive Measurements of Electron Bunch Current Profiles with Few-Femtosecond Resolution at MHz Repetition Rates*. PhD thesis, Universität Hamburg, 2021.
- [200] J. O. Lübsen, M. Schütte, S. Schulz, and A. Eichler. A safe Bayesian optimization algorithm for tuning the optical synchronization system at European XFEL. In *Proceedings of the 22nd IFAC World Congress*, volume 56, 2023.
- [201] J. O. Lübsen, C. Hespe, and A. Eichler. Towards safe multi-task Bayesian optimization. In *Proceedings of the 6th Annual Learning for Dynamics & Control Conference*, volume 242, 2024.
- [202] N. Madysa, V. Kain, R. A. Fernandez, N. Biancacci, B. Goddard, and F. M. Velotti. Automated intensity optimisation using reinforcement learning at LEIR. In *Proceedings of the 13th International Particle Accelerator Conference (IPAC)*, 2022.
- [203] J. Maldonado, S. Clark, W. Fu, and S. Nemesure. Enhancing electronic logbooks using machine learning. In *Proceedings of the 19th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS)*, 2023.
- [204] A. Mandhane, A. Zhernov, M. Rauh, C. Gu, M. Wang, F. Xue, W. Shang, D. Pang, R. Claus, C.-H. Chiang, C. Chen, J. Han, A. Chen, D. J. Mankowitz, J. Broshear, J. Schrittwieser, T. Hubert, O. Vinyals, and T. Mann. MuZero with self-competition for rate control in VP9 video compression. *arXiv*, 2202.06626, 2022.

- [205] D. Mankowitz, A. Michi, A. Zhernov, M. Gelmi, M. Selvi, C. Paduraru, E. Leurent, S. Iqbal, J.-B. Lespiau, A. Ahern, T. Köppe, K. Millikin, S. Gaffney, S. Elster, J. Broshear, C. Gamble, K. Milan, R. Tung, M. Hwang, and D. Silver. Faster sorting algorithms discovered using deep reinforcement learning. *Nature*, 618, 2023.
- [206] S. Mann, E. Fadel, S. S. Schoenholz, E. D. Cubuk, S. G. Johnson, and G. Romano.  $\partial pv$ : An end-to-end differentiable solar-cell simulator. *Computer Physics Communications*, 272, 2022.
- [207] D. Marcato, L. Bellan, D. Bortolato, M. Comunian, A. Galata, F. Gelain, V. Martinelli, G. Savarese, and G. A. Susto. Demonstration of beam emittance optimization using reinforcement learning. In *Proceedings of the 14th International Particle Accelerator Conference (IPAC)*, 2023.
- [208] J. Martinez-Marin, B. Mustapha, B. Blomberg, E. Letcher, G. Dunn, D. Stanton, and K. Bunnell. Reinforcement learning and bayesian optimization for ion linac operations. In *Proceedings of the 15th International Conference on Heavy Ion Accelerator Technology*, 2022.
- [209] G. Martino, A. Grünhagen, J. Branlard, A. Eichler, G. Fey, and H. Schlarb. Comparative evaluation of semi-supervised anomaly detection algorithms on high-integrity digital systems. In *Proceedings of the 24th Euromicro Conference on Digital System Design (DSD)*, 2021.
- [210] B. Matérn. *Spatial Variation*. Springer New York, 2nd edition, 1986.
- [211] F. Mayet. Building an intelligent accelerator operations assistant. In *1st Large Language Models in Physics Symposium*, 2024.
- [212] F. Mayet. GAIA: A general AI assistant for intelligent accelerator operations. *arXiv*, 2405.01359, 2024.
- [213] F. Mayet, M. Hachmann, K. Flöttmann, F. Burkart, H. Dinter, W. Kuroпка, T. Vinatier, and R. Assmann. Predicting the transverse emittance of space charge dominated beams using the phase advance scan technique and a fully connected neural network. *Physical Review Accelerators and Beams*, 25, 2022.
- [214] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5, 1943.
- [215] M. McIntire, T. Cope, S. Ermon, and D. Ratner. Bayesian optimization of FEL performance at LCLS. In *Proceedings of the 7th International Particle Accelerator Conference (IPAC)*, 2016.
- [216] W. Mead, P. Bowling, S. Brown, R. Jones, C. Barnes, H. Gibson, J. Goulding, and Y. Lee. Optimization and control of a small-angle negative ion source using an on-line adaptive controller based on the connectionist normalized local spline neural network. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, 72, 1992.
- [217] W. Mead, S. Brown, R. Jones, P. Bowling, and C. Barnes. Adaptive optimization and control using neural networks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 352, 1994.

- [218] D. Meier, L. V. Ramirez, J. Völker, J. Viefhaus, B. Sick, and G. Hartmann. Optimizing a superconducting radio-frequency gun using deep reinforcement learning. *Physical Review Accelerators and Beams*, 25, 2022.
- [219] E. Meier, S. Biedron, G. LeBlanc, M. Morgan, and J. Wu. Electron beam energy and bunch length feed forward control studies using an artificial neural network at the Linac Coherent Light Source. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 610, 2009.
- [220] E. Meier, Y.-R. E. Tan, and G. S. LeBlanc. Orbit correction studies using neural networks. In *Proceedings of the 3rd International Particle Accelerator Conference (IPAC)*, 2012.
- [221] A. Mitra, L. Del Corro, S. Mahajan, A. Cudas, C. Simoes, S. Agarwal, X. Chen, A. Razdaibiedina, E. Jones, K. Aggarwal, H. Palangi, G. Zheng, C. Rosset, H. Khanpour, and A. Awadallah. Orca 2: Teaching small language models how to reason. *arXiv*, 2311.11045, 2023.
- [222] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with deep reinforcement learning. *arXiv*, 1312.5602, 2013.
- [223] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518, 2015.
- [224] S. Mukherjee, A. Mitra, G. Jawahar, S. Agarwal, H. Palangi, and A. Awadallah. Orca: Progressive learning from complex explanation traces of GPT-4. *arXiv*, 2306.02707, 2023.
- [225] V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.
- [226] W. Neiswanger, K. A. Wang, and S. Ermon. Bayesian algorithm execution: Estimating computable properties of black-box functions using mutual information. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.
- [227] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7, 1965.
- [228] N. Neveu, J. Larson, J. Power, and L. Spentzouris. Photoinjector optimization using a derivative-free, model-based trust-region algorithm for the Argonne Wakefield Accelerator. In *Journal of Physics: Conference Series*, volume 874, 2017.
- [229] N. Neveu, L. Spentzouris, A. Adelman, Y. Ineichen, A. Kolano, C. Metzger-Kraus, C. Bekas, A. Curioni, and P. Arbenz. Parallel general purpose multiobjective optimization framework with application to electron beam dynamics. *Physical Review Accelerators and Beams*, 22, 2019.

- [230] D. Nguyen, M. Lee, R. Sass, and H. Shoaee. Accelerator and feedback control simulation using neural networks. In *Proceedings of the IEEE Particle Accelerator Conference*, 1991.
- [231] J. Nousiainen, C. Rajani, M. Kasper, T. Helin, S. Y. Haffert, C. Vérinaud, J. R. Males, K. Van Gorkom, L. M. Close, J. D. Long, A. D. Hedglen, O. Guyon, L. Schatz, M. Kautz, J. Lumbres, A. Rodack, J. M. Knight, and K. Miller. Toward on-sky adaptive optics control using reinforcement learning: Model-based policy optimization for adaptive optics. *Astronomy and Astrophysics*, 664, 2022.
- [232] NVIDIA. NVIDIA A100 data sheet, 2020. URL <https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet.pdf>. Accessed: 2024-04-24.
- [233] K. Nweye, B. Liu, P. Stone, and Z. Nagy. Real-world challenges for multi-agent reinforcement learning in grid-interactive buildings. *Energy and AI*, 10, 2022.
- [234] F. M. Nyikosa, M. A. Osborne, and S. J. Roberts. Bayesian optimization for dynamic problems. *arXiv*, 1803.03432, 2018.
- [235] Ollama Team. Ollama, 2023. URL <https://ollama.com>. Accessed: 2024-04-15.
- [236] D. Olsson. Online optimisation of the MAX-IV 3 GeV ring dynamic aperture. *Proceedings of the 9th International Particle Accelerator Conference (IPAC)*, 2281, 2018.
- [237] OpenAI. GPT-3.5 Turbo model documentation. <https://platform.openai.com/docs/models/gpt-3-5-turbo>, 2023. Accessed: 2024-04-15.
- [238] OpenAI. New models and developer products announced at DevDay, 2023. URL <https://openai.com/blog/new-models-and-developer-products-announced-at-devday>. Accessed: 2024-04-15.
- [239] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang. Solving Rubik’s cube with a robot hand. *arXiv*, 1910.07113, 2019.
- [240] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with large scale deep reinforcement learning. *arXiv*, 1912.06680, 2019.
- [241] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39, 2020.
- [242] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, R. Avila, I. Babuschkin, S. Balaji, V. Balcom, P. Baltescu, H. Bao, M. Bavarian, J. Belgum, I. Bello, J. Berdine, G. Bernadett-Shapiro, C. Berner, L. Bogdonoff, O. Boiko, M. Boyd, A.-L. Brakman, G. Brockman,

T. Brooks, M. Brundage, K. Button, T. Cai, R. Campbell, A. Cann, B. Carey, C. Carlson, R. Carmichael, B. Chan, C. Chang, F. Chantzis, D. Chen, S. Chen, R. Chen, J. Chen, M. Chen, B. Chess, C. Cho, C. Chu, H. W. Chung, D. Cummings, J. Currier, Y. Dai, C. Decareaux, T. Degry, N. Deutsch, D. Deville, A. Dhar, D. Dohan, S. Dowling, S. Dunning, A. Ecoffet, A. Eleti, T. Eloundou, D. Farhi, L. Fedus, N. Felix, S. P. Fishman, J. Forte, I. Fulford, L. Gao, E. Georges, C. Gibson, V. Goel, T. Gogineni, G. Goh, R. Gontijo-Lopes, J. Gordon, M. Grafstein, S. Gray, R. Greene, J. Gross, S. S. Gu, Y. Guo, C. Hallacy, J. Han, J. Harris, Y. He, M. Heaton, J. Heidecke, C. Hesse, A. Hickey, W. Hickey, P. Hoeschele, B. Houghton, K. Hsu, S. Hu, X. Hu, J. Huizinga, S. Jain, S. Jain, J. Jang, A. Jiang, R. Jiang, H. Jin, D. Jin, S. Jomoto, B. Jonn, H. Jun, T. Kaftan, L. Kaiser, A. Kamali, I. Kanitscheider, N. S. Keskar, T. Khan, L. Kilpatrick, J. W. Kim, C. Kim, Y. Kim, J. H. Kirchner, J. Kiros, M. Knight, D. Kokotajlo, L. Kondraciuk, A. Kondrich, A. Konstantinidis, K. Kosic, G. Krueger, V. Kuo, M. Lampe, I. Lan, T. Lee, J. Leike, J. Leung, D. Levy, C. M. Li, R. Lim, M. Lin, S. Lin, M. Litwin, T. Lopez, R. Lowe, P. Lue, A. Makanju, K. Malfacini, S. Manning, T. Markov, Y. Markovski, B. Martin, K. Mayer, A. Mayne, B. McGrew, S. M. McKinney, C. McLeavey, P. McMillan, J. McNeil, D. Medina, A. Mehta, J. Menick, L. Metz, A. Mishchenko, P. Mishkin, V. Monaco, E. Morikawa, D. Mossing, T. Mu, M. Murati, O. Murk, D. Mély, A. Nair, R. Nakano, R. Nayak, A. Neelakantan, R. Ngo, H. Noh, L. Ouyang, C. O’Keefe, J. Pachocki, A. Paino, J. Palermo, A. Pantuliano, G. Parascandolo, J. Parish, E. Parparita, A. Passos, M. Pavlov, A. Peng, A. Perelman, F. d. A. B. Peres, M. Petrov, H. P. d. O. Pinto, M. Petrov, M. Pokorny, M. Pokrass, V. H. Pong, T. Powell, A. Power, B. Power, E. Proehl, R. Puri, A. Radford, J. Rae, A. Ramesh, C. Raymond, F. Real, K. Rimbach, C. Ross, B. Rotsted, H. Roussez, N. Ryder, M. Saltarelli, T. Sanders, S. Santurkar, G. Sastry, H. Schmidt, D. Schnurr, J. Schulman, D. Selsam, K. Sheppard, T. Sherbakov, J. Shieh, S. Shoker, P. Shyam, S. Sidor, E. Sigler, M. Simens, J. Sitkin, K. Slama, I. Sohl, B. Sokolowsky, Y. Song, N. Staudacher, F. P. Such, N. Summers, I. Sutskever, J. Tang, N. Tezak, M. B. Thompson, P. Tillet, A. Tootoonchian, E. Tseng, P. Tuggle, N. Turley, J. Tworek, J. F. C. Uribe, A. Vallone, A. Vijayvergiya, C. Voss, C. Wainwright, J. J. Wang, A. Wang, B. Wang, J. Ward, J. Wei, C. Weinmann, A. Welihinda, P. Welinder, J. Weng, L. Weng, M. Wiethoff, D. Willner, C. Winter, S. Wolrich, H. Wong, L. Workman, S. Wu, J. Wu, M. Wu, K. Xiao, T. Xu, S. Yoo, K. Yu, Q. Yuan, W. Zaremba, R. Zellers, C. Zhang, M. Zhang, S. Zhao, T. Zheng, J. Zhuang, W. Zhuk, and B. Zoph. GPT-4 technical report. *arXiv*, 2303.08774, 2023.

- [243] F. H. O’Shea, N. Bruchon, and G. Gaio. Policy gradient methods for free-electron laser and terahertz source optimization and stabilization at the FERMI free-electron laser at Elettra. *Physical Review Accelerators and Beams*, 23, 2020.
- [244] N. Oulianov, P.-L. Biojout, P. L. Venard, and S. Girard. Evaluate LLMs in real time with Street Fighter III. GitHub, 2024.
- [245] X. Pang, S. Thulasidasan, and L. Rybarczyk. Autonomous control of a particle accelerator using deep reinforcement learning. In *Proceedings of the Machine Learning for Engineering Modeling, Simulation, and Design Workshop at Neural Information Processing Systems*, 2020.
- [246] E. Panofski, R. Assmann, F. Burkart, U. Dorda, L. Genovese, F. Jafarinia, S. Jaster-Merz, M. Kellermeier, W. Kuroepka, F. Lemery, B. Marchetti, D. Marx, F. Mayet,

- T. Vinatier, and S. Yamin. Commissioning results and electron beam characterization with the S-band photoinjector at SINBAD-ARES. *Instruments*, 5, 2021.
- [247] O. J. Parra, J. G. Pardiñas, L. D. P. Pérez, M. Janisch, S. Klaver, T. Lehericy, and N. Serra. Human-in-the-loop reinforcement learning for data quality monitoring in particle physics experiments. *arXiv*, 2405.15508, 2024.
- [248] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, 2019.
- [249] E. Pignatelli. Navix: Scaling gridworld navigation with JAX. GitHub, 2023.
- [250] M. Plappert. keras-rl. GitHub, 2016.
- [251] M. J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Technical report, NA2009/06, University of Cambridge, 2009.
- [252] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. *arXiv*, 1612.00593, 2016.
- [253] J. Qiang. Differentiable self-consistent space-charge simulation for accelerator design. *Physical Review Accelerators and Beams*, 26, 2023.
- [254] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22, 2021.
- [255] D. Ratner, F. Christie, J. Cryan, A. L. Edelen, A. Lutman, and X. Zhang. Recovering the phase and amplitude of X-ray FEL pulses using neural networks and differentiable models. *Optics express*, 29, 2021.
- [256] F. Rehm. AccGPT: A vision for AI assistance at CERN's accelerator control and beyond. In *1st Large Language Models in Physics Symposium*, 2024.
- [257] S. Reiche. GENESIS 1.3: A fully 3D time-dependent FEL simulation code. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 429, 1999.
- [258] M. Remta. Batch spacing optimization via reinforcement learning, 2024.
- [259] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22, 1951.
- [260] B. Romera-Paredes, M. Barekatin, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. R. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, P. Kohli, and A. Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625, 2024.
- [261] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Proceedings of the MICCAI 2015: 18th international Conference*. Springer, 2015.

- [262] F. Rosenblatt. The perceptron: A perceiving and recognizing automaton, 1957.
- [263] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, 1958.
- [264] J. Rosenzweig and L. Serafini. Transverse particle motion in radio-frequency linear accelerators. *Physical Review E*, 49, 1994.
- [265] R. Roussel and A. L. Edelen. Applications of differentiable physics simulations in particle accelerator modeling. *arXiv*, 2211.09077, 2022.
- [266] R. Roussel, J. P. Gonzalez-Aguilera, Y.-K. Kim, E. Wisniewski, W. Liu, P. Piot, J. Power, A. Hanuka, and A. L. Edelen. Turn-key constrained parameter space exploration for particle accelerators using Bayesian active learning. *Nature Communications*, 12, 2021.
- [267] R. Roussel, A. Hanuka, and A. L. Edelen. Multiobjective Bayesian optimization for online accelerator tuning. *Physical Review Accelerators and Beams*, 24, 2021.
- [268] R. Roussel, A. L. Edelen, D. Ratner, K. Dubey, J. P. Gonzalez-Aguilera, Y. K. Kim, and N. Kuklev. Differentiable physics modeling for characterization and optimization of particle accelerator systems with hysteresis. *Physical Review Letters*, 128, 2022.
- [269] R. Roussel, A. L. Edelen, A. Bartnik, and C. Mayes. Xopt: A simplified framework for optimization of accelerator problems using advanced algorithms. In *Proceedings of the 14th International Particle Accelerator Conference (IPAC)*, 2023.
- [270] R. Roussel, A. L. Edelen, C. Mayes, D. Ratner, J. P. Gonzalez-Aguilera, S. Kim, E. Wisniewski, and J. Power. Phase space reconstruction from accelerator beam measurements using neural networks and differentiable simulations. *Physical Review Letters*, 130, 2023.
- [271] R. Roussel, A. L. Edelen, T. Boltz, D. Kennedy, Z. Zhang, F. Ji, X. Huang, D. Ratner, A. Santamaria Garcia, C. Xu, J. Kaiser, A. F. Pousa, A. Eichler, J. O. Lübsen, N. M. Isenberg, Y. Gao, N. Kuklev, J. Martinez, B. Mustapha, V. Kain, C. Mayes, W. Lin, S. M. Liuzzo, J. St. John, M. J. V. Streeter, R. Lehe, and W. Neiswanger. Bayesian optimization algorithms for accelerator physics. *Physical Review Accelerators and Beams*, 27, 2024.
- [272] R. Roussel, J. P. Gonzalez-Aguilera, A. L. Edelen, E. Wisniewski, A. Ody, W. Liu, Y.-K. Kim, and J. Power. Efficient 6-dimensional phase space reconstruction from experimental measurements using generative machine learning. *arXiv*, 2404.10853, 2024.
- [273] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323, 1986.
- [274] A. Rutherford, B. Ellis, M. Gallici, J. Cook, A. Lupu, G. Ingvarsson, T. Willi, A. Khan, C. Schroeder de Witt, A. Souly, S. Bandyopadhyay, M. Samvelyan, M. Jiang, R. T. Lange, S. Whiteson, B. Lacerda, N. Hawes, T. Rocktaschel, C. Lu, and J. N. Foerster. JaxMARL: Multi-agent rl environments in JAX. *arXiv*, 2311.10090, 2023.
- [275] D. Sagan. Bmad: A relativistic charged particle simulation library. *Nuclear Instruments and Methods*, 2006.

- [276] R. T. Sandberg, R. Lehe, C. E. Mitchell, M. Garten, A. Myers, J. Qiang, J.-L. Vay, and A. Huebl. Synthesizing particle-in-cell simulations through learning and GPU computing for hybrid particle accelerator beamlines. *arXiv*, 2402.17248, 2024.
- [277] A. Santamaria Garcia, C. Xu, L. Scomparin, A. Eichler, J. Kaiser, M. Schenk, S. Pochaba, and S. Hirllaender. The reinforcement learning for autonomous accelerators collaboration. In *Proceedings of the 15th International Particle Accelerator Conference (IPAC)*, 2024.
- [278] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20, 2009.
- [279] A. Scheinker. Conditional guided generative diffusion for particle accelerator beam diagnostics. *arXiv*, 2407.10693, 2024.
- [280] A. Scheinker, X. Pang, and L. Rybarcyk. Model-independent particle accelerator tuning. *Physical Review Special Topics - Accelerators and Beams*, 16, 2013.
- [281] A. Scheinker, A. L. Edelen, D. Bohler, C. Emma, and A. Lutman. Demonstration of model-independent control of the longitudinal phase space of electron beams in the Linac-Coherent Light Source with femtosecond resolution. *Physical Review Letters*, 121, 2018.
- [282] A. Scheinker, S. Hirllaender, F. M. Velotti, S. Gessner, G. Z. Della Porta, V. Kain, B. Goddard, and R. Ramjiawan. Online multi-objective particle accelerator optimization of the awake electron beam line for simultaneous emittance and orbit control. *AIP Advances*, 10, 2020.
- [283] A. Scheinker, E.-C. Huang, and C. Taylor. Extremum seeking-based control system for particle accelerator beam loss minimization. *IEEE Transactions on Control Systems Technology*, 30, 2022.
- [284] M. Schenk, E. F. Combarro, M. Grossi, V. Kain, K. S. B. Li, M.-M. Popa, and S. Vallecorsa. Hybrid actor-critic algorithm for quantum reinforcement learning at CERN beam lines. *Quantum Science and Technology*, 9, 2024.
- [285] D. Schirmer, P. Hartmann, T. Büning, and D. Müller. Electron transport line optimization using neural networks and genetic algorithms. In *Proceedings of the European Particle Accelerator Conference*, 2006.
- [286] B. Schmidt, S. Wesch, T. Kövener, C. Behrens, E. Hass, S. Casalbuoni, and P. Schmüser. Longitudinal bunch diagnostics using coherent transition radiation spectroscopy. *arXiv*, 1803.00608, 2018.
- [287] B. Schmidt, N. M. Lockmann, P. Schmüser, and S. Wesch. Benchmarking coherent radiation spectroscopy as a tool for high-resolution bunch shape reconstruction at free-electron lasers. *Physical Review Accelerators and Beams*, 23, 2020.
- [288] S. S. Schoenholz and E. D. Cubuk. JAX M.D. A framework for differentiable physics. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

- [289] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, 588, 2020.
- [290] A. Schuett, C. Knochenhauer, M. Mcateer, and P. Schnizer. Optimising non-linear kicker injection parameters using machine learning. In *Proceedings of the 15th Particle Accelerator Conference (IPAC)*, 2024.
- [291] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust Region Policy Optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, volume 37, Lille, France, 2015.
- [292] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv*, 1707.06347, 2017.
- [293] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv*, 1506.02438, 2018.
- [294] M. Sclar, Y. Choi, Y. Tsvetkov, and A. Suhr. Quantifying language models' sensitivity to spurious features in prompt design or: How I learned to start worrying about prompt formatting. *arXiv*, 2310.11324, 2023.
- [295] L. Scomparin, E. Blomley, T. Boltz, E. Bründermann, M. Caselle, T. Dritschler, A. Kopmann, A. Mochihashi, A.-S. Müller, A. Santamaria Garcia, P. Schreiber, J. Steinmann, and M. Weber. KINGFISHER: A framework for fast machine learning inference for autonomous accelerator systems. In *Proceedings of the 11th International Beam Instrumentation Conference*, 2022.
- [296] L. Scomparin, E. Blomley, E. Bründermann, M. Caselle, A. Kopmann, A.-S. Müller, A. Santamaria Garcia, J. L. Steinmann, and M. Weber. A low-latency feedback system for the control of horizontal betatron oscillations. In *Proceedings of the 14th International Particle Accelerator Conference (IPAC)*, 2023.
- [297] L. Scomparin, J. Becker, E. Blomley, E. Bründermann, M. Caselle, T. Dritschler, A. Kopmann, A. Mochihashi, A.-S. Müller, A. Santamaria Garcia, M. Schuh, J. L. Steinmann, M. Weber, and C. Xu. Preliminary results on the reinforcement learning-based control of the microbunching instability. In *Proceedings of the 15th International Particle Accelerator Conference (IPAC)*, 2024.
- [298] L. Scomparin, M. Caselle, A. Santamaria Garcia, C. Xu, E. Blomley, T. Dritschler, A. Mochihashi, M. Schuh, J. L. Steinmann, E. Bründermann, A. Kopmann, J. Becker, A.-S. Müller, and M. Weber. Microsecond-latency feedback at a particle accelerator by online reinforcement learning on hardware. *arXiv*, 2409.16177, 2024.
- [299] J. Seo, Y. S. Na, B. Kim, C. Y. Lee, M. S. Park, S. J. Park, and Y. H. Lee. Feedforward beta control in the KSTAR tokamak by deep reinforcement learning. *Nuclear Fusion*, 61, 2021.
- [300] J. Seo, Y. S. Na, B. Kim, C. Y. Lee, M. S. Park, S. J. Park, and Y. H. Lee. Development of an operation trajectory design algorithm for control of multiple 0D parameters using deep reinforcement learning in KSTAR. *Nuclear Fusion*, 62, 2022.

- [301] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104, 2016.
- [302] R. J. Shalloo, S. J. Dann, J. N. Gruse, C. I. Underwood, A. F. Antoine, C. Arran, M. Backhouse, C. D. Baird, M. D. Balcazar, N. Bourgeois, J. A. Cardarelli, P. Hatfield, J. Kang, K. Krushelnick, S. P. Mangles, C. D. Murphy, N. Lu, J. Osterhoff, K. Pöder, P. P. Rajeev, C. P. Ridgers, S. Rozario, M. P. Selwood, A. J. Shahani, D. R. Symes, A. G. Thomas, C. Thornton, Z. Najmudin, and M. J. Streeter. Automation and control of laser wakefield accelerators using Bayesian optimization. *Nature Communications*, 11, 2020.
- [303] H. Shang and M. Borland. A Parallel Simplex Optimizer and its Application to High-Brightness Storage Ring Design. In *Proceedings of the 2005 Particle Accelerator Conference (IPAC)*, 2005.
- [304] S. W. Shin, M. Ghergherehchi, and J. S. Chai. Using deep reinforcement learning for designing sub-relativistic electron linac. In *Proceedings of the 9th International Particle Accelerator Conference (IPAC)*, 2018.
- [305] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, K. Leach, M. and Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 2016.
- [306] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Van Den Driessche, T. Graepel, and D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550, 2017.
- [307] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362, 2018.
- [308] SLAC National Accelerator Laboratory. Inside the SLAC Linear Accelerator. Flickr, 2012. URL <https://www.flickr.com/photos/slaclab/8282691262/in/album-72157686223232275>. Accessed: 2024-10-31.
- [309] SLAC National Accelerator Laboratory. Undulators. Flickr, 2021. URL <https://www.flickr.com/photos/slaclab/51717533870/in/album-72157720246688390>. Accessed: 2024-10-31.
- [310] S. Sra, S. Nowozin, and S. J. Wright. *Optimization for machine learning*. MIT Press, 2011.
- [311] J. St. John, C. Herwig, D. Kafkes, J. Mitrevski, W. A. Pellico, G. N. Perdue, A. Quintero-Parra, B. A. Schupbach, K. Seiya, N. Tran, M. Schram, J. M. Duarte, Y. Huang, and R. Keller. Real-time artificial intelligence for accelerator control: A study at the Fermilab Booster. *Physical Review Accelerators and Beams*, 24, 2021.

- [312] A. Stanziola, S. R. Arridge, B. T. Cox, and B. E. Treeby. j-Wave: An open-source differentiable wave simulator. *SoftwareX*, 22, 2023.
- [313] Statistisches Bundesamt (Destatis). Stromverbrauch der privaten Haushalte nach Haushaltsgrößenklassen, 2024. URL <https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Umwelt/UGR/private-haushalte/Tabellen/stromverbrauch-haushalte.html>. Accessed: 2024-10-12.
- [314] O. Stein, J. Kaiser, and A. Eichler. Accelerating linear beam dynamics simulations for machine learning applications. In *Proceedings of the 13th International Particle Accelerator Conference (IPAC)*, 2022.
- [315] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13, 1969.
- [316] C. Su, Z. Wang, X. Chen, Y. Jia, X. Qi, and D. Jia. Reinforcement control and matching for LEBT and RFQ of linear accelerators. In *Proceedings of the 14th International Particle Accelerator Conference (IPAC)*, 2023.
- [317] A. Sulc, R. Kammering, A. Eichler, and T. Wilksen. PACuna: Automated fine-tuning of language models for particle accelerators. In *Machine Learning and the Physical Sciences Workshop at NeurIPS*, 2023.
- [318] A. Sulc, G. Hartmann, J. Maldonado, V. Kain, F. Rehm, A. Eichler, J. Kaiser, T. Wilksen, F. Mayet, R. Kammering, H. Tuennermann, S. St. John, H. Hoschouer, K. Hazelwood, T. Hellert, and D. Ratner. Towards unlocking insights from logbooks using AI. In *Proceedings of the 15th International Particle Accelerator Conference (IPAC)*, 2024.
- [319] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.
- [320] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. Lillicrap, and M. Riedmiller. DeepMind Control Suite. *arXiv*, 1801.00690, 2018.
- [321] C. Tennant, A. Carpenter, T. Powers, A. Shabalina Solopova, L. Vidyaratne, and K. Iftekharuddin. Superconducting radio-frequency cavity fault classification using machine learning at Jefferson Laboratory. *Physical Review Accelerators and Beams*, 23, 2020.
- [322] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38, 1995.
- [323] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.
- [324] S. Tomin, G. Geloni, I. Agapov, I. Zagorodnov, Y. Fomin, Y. Krylov, A. Valintinov, W. Colocho, T. M. Cope, A. Egger, and D. Ratner. Progress in automatic software-based optimization of accelerator performance. In *Proceedings of the 7th International Particle Accelerator Conference (IPAC)*, 2016.

- [325] S. Tomin, I. Agapov, M. Dohlus, and I. Zagorodnov. OCELOT as a framework for beam dynamics simulations of X-ray sources. In *Proceedings of the 8th International Particle Accelerator Conference (IPAC)*, 2017.
- [326] S. Tomin, J. Kaiser, N. M. Lockmann, T. Wohlenberg, and I. Zagorodnov. Undulator linear taper control at the European X-Ray Free-Electron Laser facility. *Physical Review Accelerators and Beams*, 27, 2024.
- [327] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. Canton Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. Michael, S. Ranjan, S. Xiaoqing, E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. Llama 2: Open foundation and fine-tuned chat models. *arXiv*, 2307.09288, 2023.
- [328] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis. Gymnasium. *Zenodo*, 2023. URL <https://zenodo.org/record/8127025>.
- [329] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillcrap, N. Heess, and Y. Tassa. dm\_control: Software and tasks for continuous control. *Software Impacts*, 6, 2020.
- [330] Umweltbundesamt. Entwicklung der spezifischen Treibhausgas-Emissionen des deutschen Strommix in den Jahren 1990 - 2022. Technical report, Umweltbundesamt, 2023.
- [331] U.S. Geological Survey, Department of the Interior/USGS. USGS EROS Archive - Aerial Photography - High Resolution Orthoimagery (HRO), 2018. URL <https://doi.org/10.5066/F73X84W6>. Accessed 2025-10-17. Acquisition date 2015-02-20.
- [332] G. van Rossum, B. Warsaw, and N. Coghlan. Style guide for Python code, 2001.
- [333] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [334] B. Veglia and I. Agapov. Neural networks for ID gap orbit distortion compensation in PETRA III. In *Proceedings of the 14th International Particle Accelerator Conference (IPAC)*, 2023.
- [335] B. Veglia, I. Agapov, J. Keil, L. Malina, E. Musa, T. Hellert, N. Carmignani, L. Carver, L. Houmami, S. Liuzzo, and S. White. Extremum seeking for accelerator optimisation. In *Proceedings of the 14th International Particle Accelerator Conference (IPAC)*, 2023.

- [336] B. Veglia, I. Agapov, and J. Keil. Neural networks for ID gap orbit distortion compensation in PETRA III. *arXiv*, 2406.17494, 2024.
- [337] F. M. Velotti, B. Goddard, V. Kain, R. Ramjiawan, G. Z. Della Porta, and S. Hirlander. Towards automatic setup of 18 MeV electron beamline using machine learning. *Machine Learning: Science and Technology*, 4, 2023.
- [338] L. Vidyaratne, A. Carpenter, T. Powers, C. Tennant, K. M. Iftexharuddin, M. M. Rahman, and A. S. Shabalina. Deep learning based superconducting radio-frequency cavity fault classification at Jefferson Laboratory. *Frontiers in Artificial Intelligence*, 4, 2022.
- [339] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vechnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575, 2019.
- [340] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, Í. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17, 2020.
- [341] Waldoalvarez at English Wikipedia. Example of Markov decision process, 2017. URL [https://commons.wikimedia.org/wiki/File:Markov\\_Decision\\_Process.svg](https://commons.wikimedia.org/wiki/File:Markov_Decision_Process.svg). Accessed 2024-09-12. Licensed under CC BY-SA 4.0; adapted recreation of original.
- [342] A. Walls, Y. Park, M. Tortorici, A. Wall, A. McGuire, and D. Veessler. Structure of the SARS-CoV-2 spike glycoprotein (closed state). *Worldwide Protein Data Bank*, 2020. PDB ID: 6VXX.
- [343] A. C. Walls, Y.-J. Park, M. A. Tortorici, A. Wall, A. T. McGuire, and D. Veessler. Structure, function, and antigenicity of the SARS-CoV-2 spike glycoprotein. *Cell*, 181, 2020.
- [344] J. Wan, Y. Hao, C. Ratcliff, J. Qiang, and Y. Kan. Developing nested auto-differentiation tracking code for beam dynamics optimization. In *Proceedings of the 15th International Particle Accelerator Conference (IPAC)*, 2024.
- [345] W. Wang, M. Caselle, T. Boltz, E. Blomley, M. Brosi, T. Dritschler, A. Ebersoldt, A. Kopmann, A. Santamaria Garcia, P. Schreiber, E. Bründermann, M. Weber, A.-S. Müller, and Y. Fang. Accelerated deep reinforcement learning for fast feedback of beam dynamics at KARA. *IEEE Transactions on Nuclear Science*, 68, 2021.

- [346] Z. Wang, P. Veličković, D. Hennes, N. Tomašev, L. Prince, M. Kaisers, Y. Bachrach, R. Elie, L. K. Wenliang, F. Piccinini, W. Spearman, I. Graham, J. Connor, Y. Yang, A. Recasens, M. Khan, N. Beauguerlange, P. Sprechmann, P. Moreno, N. Heess, M. Bowling, D. Hassabis, and K. Tuyls. TacticAI: An AI assistant for football tactics. *Nature Communications*, 2024.
- [347] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. *arXiv*, 2201.11903, 2023.
- [348] S. Wesch, B. Schmidt, C. Behrens, H. Delsim-Hashemi, and P. Schmüser. A multi-channel THz and infrared spectrometer for femtosecond electron bunch diagnostics by single-shot spectroscopy of coherent radiation. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 665, 2011.
- [349] M. Wielgosz, A. Skoczeń, and M. Mertik. Using LSTM recurrent neural networks for monitoring the LHC superconducting magnets. *Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 867, 2017.
- [350] K. Wille. *The Physics of Particle Accelerators: An Introduction*. Clarendon Press, 2000.
- [351] J. A. Wulff. Optimization of RF manipulations in the PS using reinforcement learning, 2021.
- [352] Y. Xiong, L. Guo, Y. Huang, and L. Chen. Intelligent thermal control strategy based on reinforcement learning for space telescope. *Journal of Thermophysics and Heat Transfer*, 34, 2020.
- [353] Y. Xiong, L. Guo, and D. Tian. Application of deep reinforcement learning to thermal control of space telescope. *Journal of Thermal Science and Engineering Applications*, 14, 2022.
- [354] C. Xu, E. Bründermann, A. Eichler, A.-S. Müller, M. Nasse, A. Santamaria Garcia, C. Sax, and C. Widmann. Machine learning based spatial light modulator control for the photoinjector laser at FLUTE. In *Procetings of the 12th International Particle Accelerator Conference (IPAC)*, 2021.
- [355] C. Xu, E. Bründermann, A.-S. Müller, A. Santamaria Garcia, and J. Schäfer. Surrogate modelling of the FLUTE low-energy section. In *Proceedings of the 13th International Particle Accelerator Conference (IPAC)*, 2022.
- [356] C. Xu, R. Roussel, and A. L. Edelen. Neural network prior mean for particle accelerator injector tuning. *arXiv*, 2211.09028, 2022.
- [357] C. Xu, T. Boltz, A. Mochihashi, A. Santamaria Garcia, M. Schuh, and A.-S. Müller. Bayesian optimization of the beam injection process into a storage ring. *Physical Review Accelerators and Beams*, 26, 2023.
- [358] C. Xu, E. Bründermann, A.-S. Müller, A. Santamaria Garcia, and S. Tomin. Bayesian optimization for SASE tuning at the European XFEL. In *Proceedings of the 14th International Particle Accelerator Conference (IPAC)*, 2023.

- [359] C. Xu, J. Y.-C. Hu, A. Narayanan, M. Thieme, V. Nagaslaev, M. Austin, J. Arnold, J. Berlioz, P. Hanlet, A. Ibrahim, D. Nicklaus, J. Mitrevski, J. M. St. John, G. Pradhan, A. Saewert, K. Seiya, B. Schupbach, R. Thurman-Keup, N. Tran, R. Shi, S. Ogresci, A. M.-I. Shuping, K. Hazelwood, and H. Liu. Beyond PID controllers: PPO with neuralized PID policy for proton beam intensity control in Mu2e. *arXiv*, 2312.17372, 2023.
- [360] C. Xu, J. Kaiser, E. Bründermann, A. Eichler, A.-S. Müller, and A. Santamaria Garcia. Beam trajectory control with lattice-agnostic reinforcement learning. In *Proceedings of the 14th International Particle Accelerator Conference (IPAC)*, 2023.
- [361] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen. Large language models as optimizers. *arXiv*, 2309.03409, 2023.
- [362] S. Yatawatta and I. M. Avruch. Deep reinforcement learning for smart calibration of radio telescopes. *Monthly Notices of the Royal Astronomical Society*, 505, 2021.
- [363] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. HellaSwag: Can a machine really finish your sentence? *arXiv*, 1905.07830, 2019.
- [364] Z. Zhang, A. L. Edelen, C. Mayes, J. Garrahan, J. Shtalenkova, R. Roussel, S. Miskovich, D. Ratner, M. Boese, S. Tomin, G. Wang, and Y. Hidaka. Badger: The missing optimizer in ACR. In *Proceedings of the 13th International Particle Accelerator Conference (IPAC)*, 2022.
- [365] Z. Zhang, M. Song, and X. Huang. Optimization method to compensate accelerator performance drifts. *Physical Review Accelerators and Beams*, 25, 2022.
- [366] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. *arXiv*, 2306.05685, 2023.
- [367] Z. Zhou, X. Li, and R. N. Zare. Optimizing chemical reactions with deep reinforcement learning. *ACS Central Science*, 3, 2017.
- [368] B. Zhu, E. Frick, T. Wu, H. Zhu, and J. Jiao. Starling-7B: Improving LLM helpfulness & harmlessness with RLAIIF. In *First Conference on Language Modeling*, 2024.
- [369] J. Zhu, N. M. Lockmann, M. K. Czwalianna, and H. Schlarb. Mixed diagnostics for longitudinal properties of electron bunches in a free-electron laser. *Frontiers in Physics*, 10, 2022.
- [370] Z. H. Zhu, S. Tomin, and J. Kaiser. Application of machine learning in longitudinal phase space prediction at the European XFEL. In *Proceedings of the 40th International Free Electron Laser Conference (FEL)*, 2022.
- [371] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm. Technical report, ETH Zurich, 2001.