INSTITUT FÜR
MATHEMATIK

TUHH
Technische
Universität
Hamburg

Fourier Neural Operators to reconstruct tumor perfusion from 4D ultrasound data

Master's thesis

by
Judith Deimel
Registration number: 619138
Course of Study: Interdisciplinary Mathematics

September 12, 2025

First Examiner: Prof. Dr. Daniel Ruprecht
Second Examiner: Dr. Sebastian Götschel

DOI: https://doi.org/10.15480/882.16021

Declaration of Authorship

I hereby affirm on oath that I have authored the following thesis entitled

> „Fourier Neural Operators to reconstruct tumor perfusion from 4D ultrasound data“

independently and without impermissible assistance from third parties. I have not used any sources or tools other than those stated and have duly cited all quotations whether verbatim or paraphrased. This thesis has not been presented to any examination board in the same or similar form.

Hamburg, Date

Signature

Contents

1 Introduction

Perfusion imaging is becoming increasingly important for monitoring how tumors respond to therapy. It provides information about the tumor's blood flow and vascular state. Small changes in the blood vessels happen before any anatomical changes, e.g. changes in tumor size. Thus, perfusion imaging can help determine early whether a treatment is effective. This is crucial because ineffective treatments can be costly and harmful for the patients [10].

There are different ways to image perfusion. A new approach is three-dimensional dynamic-contrast-enhanced (DCE) ultrasound. Ultrasound is widely available, low-cost and safe to use at the bedside. In 3D DCE ultrasound, a contrast agent is injected and its flow through the tumor is recorded over time. This results in 4D data consisting of the three spatial dimensions and the time evolution [10].

The aim of dynamic imaging is to understand the tumor microcirculations by looking at how the contrast agent concentration changes over time. Tracer-kinetic theory provides a framework to estimate tissue-specific parameters, such as blood flow, from these concentration-time curves. The two-component model, which is used in this thesis, will be described in detail later. From a mathematical point of view, this reconstruction is an inverse problem [25].

An inverse problem refers to the determination of unknown parameters from observed data. For partial differential equations (PDEs), this typically involves estimating quantities, such as velocities or diffusion coefficients, from measured solutions, for example concentration levels. In the context of perfusion imaging, the inverse problem consists of determining the blood flow from the concentration-time curves of the contrast agent. In contrast, the forward problem refers to solving the PDE for prescribed parameters [3].

The thesis begins with an introduction to neural operators, which are a generalization of neural networks designed to learn mappings between function spaces. In particular, neural operators are capable of learning to solve an entire family of PDEs [15]. The focus is on Fourier neural operators (FNOs), whose capabilities and limitations are tested numerically using two PDEs: the advection-diffusion equation and the two-component model. For both cases, the forward problem is addressed first, where the FNO is trained to compute the solution of the PDE for given parameters. Subsequently, the typically more challenging inverse problem is considered, in which the FNO is trained to identify the parameters of the PDE from the solution. This setting directly relates to the application in perfusion imaging described above.

All methods and experiments in this thesis are implemented in Python.

2 Neural Operators

2.1 Neural Operators for PDEs

Partial differential equations arise in many different fields, including medicine, engineering and physics. Conventional numerical solvers, such as finite element methods and finite difference methods, discretize the space to solve the equation. This can be very time-consuming if a fine discretization is required. Machine learning methods can be a lot faster, as they directly learn the trajectory of the family of equations from the data. As classical neural networks map between finite-dimensional spaces, they can only learn solutions tied to a specific discretization. Additionally, they require new training for a new set of parameters of the PDE [15]. In this setting, a new set of parameters refers to a modification of the physical quantities defining the PDE, such as velocities, diffusion coefficients or reaction rates.

Neural operators are able to learn a mapping between two infinite dimensional spaces with a finite number of training samples. Consequently, a single training phase suffices for learning to solve an entire family of PDEs. Moreover, neural operators are able to transfer solutions between different discretizations and they require no knowledge of the underlying PDE [15]. In fact, neural operators are the only known class of models that guarantee universal approximation and discretization-invariance [13]. Universal approximation refers to the ability to approximate arbitrary functions with a desired accuracy [18]. A model with a fixed number of parameters is discretization-invariant if it accepts any discretization of the input function, it can be evaluated at any point of the output domain and it converges to a continuum operator, i.e. an operator that acts on functions defined over a continuous domain, as the discretization is refined [13]. However, in practice the implementation of neural operators remains discrete, as the inputs, outputs and intermediate representations are all discrete. Thus, discretization mismatch errors may occur, which refer to discrepancies between two discrete approximations of the same continuous function [7].

2.2 Mathematical Description

The content of this section and the subsequent section 2.3 is based on [13] and [15].

2.2.1 Problem Setting

Let $D \subset \mathbb{R}^d$ be an open, bounded set. Let $\mathcal{A} = \mathcal{A}(D; \mathbb{R}^{d_a})$ and $\mathcal{U} = \mathcal{U}(D; \mathbb{R}^{d_u})$ be separable Banach spaces, i.e. complete normed spaces that contain a countable dense subset. We consider the map $\mathcal{G}^{\text{true}} : \mathcal{A} \to \mathcal{U}$, which is typically non-linear and arises as the solution operator of parametric PDEs. For example, consider the following generic family of PDEs:

$$(L_a u)(x) = f(x), \qquad x \in D, \tag{2.1}$$

$$u(x) = 0, \qquad x \in \partial D, \tag{2.2}$$

for some $a \in \mathcal{A}$ and $f \in \mathcal{U}^*$, where $\mathcal{U}^*$ denotes the topological dual space of $\mathcal{U}$. The solution $u : D \to \mathbb{R}$ lives in the space $\mathcal{U}$ and the differential operator $L_a : \mathcal{U} \to \mathcal{U}^*$ depends on the parameter $a \in \mathcal{A}$. From this family of PDEs the operator $\mathcal{G}^{\text{true}} : \mathcal{A} \to \mathcal{U}$, $\mathcal{G}^{\text{true}}(a) := L_a^{-1} f$ arises. This solution operator maps a parameter $a \in \mathcal{A}$ to the corresponding solution $u \in \mathcal{U}$.

The goal is to get an approximation of $\mathcal{G}^{\text{true}}$ by defining a parametric map $\mathcal{G}_\theta : \mathcal{A} \to \mathcal{U}, \theta \in \Theta$ for some finite-dimensional parameter space Θ by finding $\theta_{\text{best}} \in \Theta$ so that $\mathcal{G}_{\theta_{\text{best}}} \approx \mathcal{G}^{\text{true}}$. The parameter space Θ corresponds to the set of trainable parameters of the machine learning model.

Let $\{a_j, u_j\}_{j=1}^N$ denote the set of observations, where $a_j \in \mathcal{A}$, $u_j = \mathcal{G}^{\text{true}}(a_j)$ for $j \in \{1, \ldots, N\}$. In general, a_j and u_j are functions. In order to work with them numerically it is assumed that $a_j|_{D_j} \in \mathbb{R}^{n \times d_a}$ and $u_j|_{D_j} \in \mathbb{R}^{n \times d_u}$ for some n-point discretization $D_j = \{x_1, \ldots, x_n\} \subset D$. Despite this, the neural operator is discretization-invariant, meaning it can predict the solution $u(x)$ for any $x \in D$. These observations are used to determine θ_{best} and to test the accuracy of the approximation $\mathcal{G}_{\theta_{\text{best}}}$. This setup is known as supervised learning.

2.2.2 Definition of a Neural Operator

The full architecture of a neural operator is shown in figure 1.

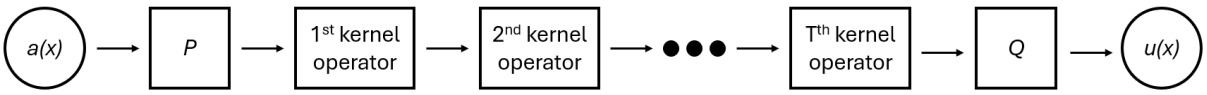


Figure 1: architecture of a neural operator

First, the input $a \in \mathcal{A}$ is mapped to a higher dimensional representation $v_0(x) = P(a(x))$ by a local transformation P, which is implemented as a fully-connected linear layer. A transformation is said to be local if, for each point $x \in D$, the output value $v_0(x)$ depends solely on the input value $a(x)$ at that point, and not on the values of a at any other point. Second, multiple iterations of updates $v_t \to v_{t+1}$ are applied.

Definition 2.1. The **iterative update** $v_t \to v_{t+1}$ is defined by

$$v_{t+1}(x) := \sigma(W v_t(x) + (\mathcal{K}(a;\phi)v_t)(x)),\ \forall x \in D, \tag{2.3}$$

where $\mathcal{K} : \mathcal{A} \times \Theta_{\mathcal{K}} \to \mathcal{L}(\mathcal{V}(D;\mathbb{R}^{d_v}), \mathcal{V}(D;\mathbb{R}^{d_v}))$ is a kernel integral operator, see definition 2.2, $\mathcal{V}(D;\mathbb{R}^{d_v})$ is a separable Banach space, $W : \mathbb{R}^{d_v} \to \mathbb{R}^{d_v}$ is a linear transformation and $\sigma : \mathbb{R} \to \mathbb{R}$ is a component-wise nonlinear activation function [15].

Definition 2.2. The **kernel integral operator** $\mathcal{K}$ is defined by

$$(\mathcal{K}(a;\phi)v_t)(x) := \int_D \kappa(x, y, a(x), a(y); \phi) v_t(y)\, \mathrm{d}y,\ \forall x \in D, \tag{2.4}$$

where $\kappa_\phi : \mathbb{R}^{2(d+d_a)} \to \mathbb{R}^{d_v \times d_v}$ is a neural network parameterized by $\phi \in \Theta_{\mathcal{K}}$ [15].

Each update is the composition of the sum of a local linear operator W and a non-local integral operator $\mathcal{K}$ with a nonlinear activation function σ, see definition 2.1. Last, v_T is projected by the local transformation Q to obtain the output $u(x) = Q(v_T(x))$. In summary, a neural operator is an iterative architecture $v_0 \mapsto v_1 \mapsto \cdots \mapsto v_T$, where v_t for $t \in \{0, \ldots, T\}$ is a sequence of functions mapping into $\mathbb{R}^{d_v}$.

The important difference between the neural operator architecture and a classical feed-forward neural network is that in case of the neural operator all operations are directly defined in function space. Therefore, there is no dependence on a specific discretization of the data.

2.3 Fourier Neural Operators

Although different neural operator layers may be used, this thesis focuses specifically on Fourier neural operators. A Fourier neural operator is obtained by substituting the kernel integral operator $\mathcal{K}$ with a convolutional operator defined in Fourier space. In fact, by removing the dependence on the function a in the definition of the kernel integral operator $\mathcal{K}$ and by imposing $\kappa(x, y; \phi) = \kappa(x - y; \phi)$ we obtain a convolution operator in equation (2.4). Under these assumptions, definition 2.2 can be rewritten as

$$(\mathcal{K}_\phi v_t)(x) = \int_D \kappa_\phi(x - y) v_t(y)\, \mathrm{d}y = (\kappa_\phi * v_t)(x),\ \forall x \in D. \tag{2.5}$$

Thus, it is feasible to parameterize κ_ϕ directly in Fourier space and use the Fast Fourier Transform (FFT) for efficient computation of the convolution.

Definition 2.3. The **Fourier Transform** $\mathcal{F}$ of a function $f \in L^1(\mathbb{R}^n, \mathbb{R}^m)$ with $n, m \in \mathbb{N}$ is defined by

$$(\mathcal{F}(f))_j(k) = \int_{\mathbb{R}^n} f_j(x) e^{-2i\pi\langle x,k\rangle} \, \mathrm{d}x, \ \forall j \in \{1, \dots, m\}, \tag{2.6}$$

where i denotes the imaginary unit, and the **Inverse Fourier Transform** $\mathcal{F}^{-1}$ of a function $f \in L^1(\mathbb{R}^n, \mathbb{R}^m) \cap L^2(\mathbb{R}^n, \mathbb{R}^m)$ is defined by

$$(\mathcal{F}^{-1}(f))_j(k) = \int_{\mathbb{R}^n} f_j(x) e^{2i\pi\langle x,k\rangle} \, \mathrm{d}x, \ \forall j \in \{1, \dots, m\} \quad [26]. \tag{2.7}$$

The convolution theorem states that for two integrable functions f and g, there holds

$$\mathcal{F}(f * g) = \mathcal{F}(f) \cdot \mathcal{F}(g) \quad [26]. \tag{2.8}$$

Applying the inverse Fourier transform to equation (2.8), one obtains

$$f * g = \mathcal{F}^{-1}(\mathcal{F}(f) \cdot \mathcal{F}(g)). \tag{2.9}$$

This can now be applied to equation (2.5) to observe

$$(\mathcal{K}_\phi v_t)(x) = \mathcal{F}^{-1}(\mathcal{F}(\kappa_\phi) \cdot \mathcal{F}(v_t))(x), \ \forall x \in D, \tag{2.10}$$

which leads to the following definition of a Fourier integral operator.

Definition 2.4. The **Fourier integral operator** $\mathcal{K}$ is defined by

$$(\mathcal{K}_\phi v_t)(x) := \mathcal{F}^{-1}(R_\phi \cdot \mathcal{F}(v_t))(x), \ \forall x \in D, \tag{2.11}$$

where R_ϕ is the Fourier transform of a periodic function $\kappa : D \to \mathbb{R}^{d_v \times d_v}$ parameterized by $\phi \in \Theta_{\mathcal{K}}$ [15].

Note that, for a frequency mode $k \in \mathbb{Z}^d$, there holds $(\mathcal{F}v_t)(k) \in \mathbb{C}^{d_v}$ and $R_\phi(k) \in \mathbb{C}^{d_v \times d_v}$. Since κ has a Fourier series expansion, as it is assumed to be periodic, the discrete frequency modes $k \in \mathbb{Z}^d$ can be used. The Fourier series expansion is truncated at a maximal number of modes $k_{\max} = |Z_{k_{\max}}| = |\{k \in \mathbb{Z}^d : |k_j| \leq k_{\max,j}, j \in \{1, \dots, d\}\}|$ to get a finite-dimensional parametrization. This choice of $Z_{k_{\max}}$ allows for easy, efficient implementation. Therefore, R_ϕ can be directly parameterized as a $(k_{\max} \times d_v \times d_v)$-tensor.

The structure of a Fourier layer is illustrated in figure 2. In the top path, the input $v_t(x)$ is first transformed by the Fourier transform $\mathcal{F}$. Next, a linear transform R filters out the high-frequency Fourier modes, before applying the inverse Fourier transform $\mathcal{F}^{-1}$. In the bottom path, a local linear transformation W processes the input directly in the spatial domain. This transformation is implemented as a convolutional layer. The outputs of both paths are then combined and a nonlinear activation function σ is applied to the result [16].

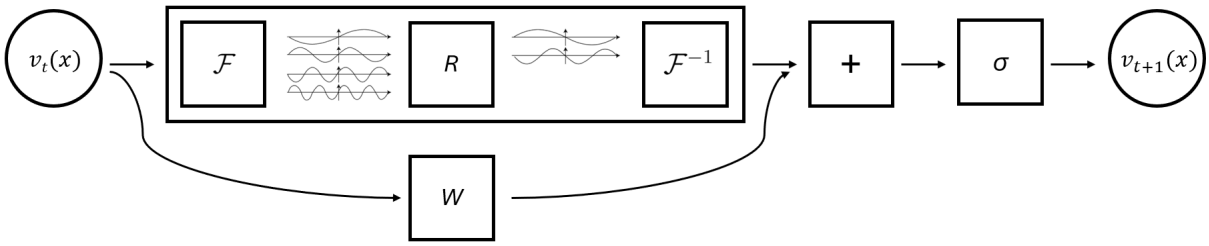


Figure 2: structure of a Fourier layer

2.3.1 Physics-informed Fourier Neural Operators

Physics-informed Fourier neural operators add physics information to the FNO architecture. Physics information is understood to be the PDE, the initial condition, the boundary conditions or other conservation laws. The network learns these physical laws by including the violations into the loss function. Thus, the loss function is a weighted sum of the mean squared error from the data, the residual of the PDE and possibly the deviation from other constraints, such as the initial condition or boundary conditions. This additional physics information has the potential to help the network to learn the solution operator faster and with less training data [23].

3 Forward Problem

This chapter describes several numerical experiments to evaluate the performance of the Fourier neural operator when solving PDEs. In the forward problem, the parameters of the PDE, the initial condition and possibly boundary conditions are given. The goal is to compute the corresponding solution of the PDE. Like most neural networks, the Fourier neural operator needs training data to learn its model parameters. In order to generate ground truth solutions, the PDEs where solved numerically.

3.1 Advection-Diffusion Equation

The advection-diffusion equation is a partial differential equation and it describes the process of a substance being transported by a flow and distributed due to concentration differences.

For $u : \mathbb{R}^3 \to \mathbb{R}, (x, y, t) \mapsto u(x, y, t)$ the advection equation, also called transport equation, is given by

$$u_t + V_1 u_x + V_2 u_y = 0, \tag{3.1}$$

where $u_t = \frac{\partial u(x,y,t)}{\partial t}$, $u_x = \frac{\partial u(x,y,t)}{\partial x}$, $u_y = \frac{\partial u(x,y,t)}{\partial y}$ are the partial derivatives and $V_1, V_2 \in \mathbb{R}$ are constants. Physically, V_1 and V_2 describe the velocity in x- resp. y-direction and could be space-dependent functions, but in this chapter they are assumed to be constant for simplicity [17].

The diffusion equation in two spatial dimensions with constant diffusion coefficients $D_1, D_2 \in \mathbb{R}$ is given by

$$u_t - (D_1 u_{xx} + D_2 u_{yy}) = 0, \tag{3.2}$$

where $u_{xx} = \frac{\partial^2 u(x,y,t)}{\partial x^2}$, $u_{yy} = \frac{\partial^2 u(x,y,t)}{\partial y^2}$ denote the second-order partial derivatives. Physically, the diffusion equation describes processes like heat conduction or the spread of a dye in a stationary fluid [17].

Combining equation (3.1) and equation (3.2) we get the two-dimensional advection-diffusion equation

$$u_t = D_1 u_{xx} + D_2 u_{yy} - V_1 u_x - V_2 u_y. \tag{3.3}$$

Considering two different constants, D_1 and D_2, and respectively V_1 and V_2, allows taking into account diffusion and velocity that differ in the two spatial directions.

3.1.1 Data Generation

In this numerical experiment the two-dimensional advection-diffusion equation (3.3) is studied over the spatial domain $(x, y) \in [0, 5] \times [0, 4]$ and the time interval $t \in [0, 1]$. The spatial grid is uniformly discretized with a resolution of 30×25 points, while the time domain is divided into 16 equally spaced time steps.

For each training sample the parameters V_1, V_2, D_1 and D_2 are randomly chosen. The velocities V_1 and V_2 are sampled uniformly from the interval $[0.25, 1.25)$, the diffusion coefficients D_1 and D_2 from the interval $[0.01, 0.15)$.

The initial condition is given by

$$u(x, y, 0) = h\, e^{-\frac{(x-x_i)^2+(y-y_i)^2}{w}}, \tag{3.4}$$

where the coordinates (x_i, y_i) are uniformly sampled from $[0.5, 1.5) \times [0.5, 0.75)$, the parameter h from the interval $[0.5, 5)$ and the parameter w from $[0.05, 0.2)$.

The spatial derivatives are computed using the pseudospectral method [24]. This method relies on the following theorem:

Theorem 3.1. Let f be infinitely often continuously differentiable and periodic. Then,

$$\mathcal{F}(\frac{d^n}{dx^n} f(x))(k) = (2i\pi k)^n \mathcal{F}(f)(k) \quad \text{for } n \in \mathbb{N} \text{ [26].}$$

The data is first transformed to the frequency domain via the Fast Fourier Transform. In this domain, differentiation is carried out efficiently by exploiting theorem 3.1, which states that differentiation in the spatial domain corresponds to multiplication by the appropriate frequency variable in the Fourier domain. Finally, the result is transformed back to the spatial domain using the inverse FFT. In the data generation process, this is implemented using the function *scipy.fftpack.diff* from the *SciPy*-package [27]. Note that, the FFT is a method to efficiently compute the Discrete Fourier Transform (DFT), which inherently assumes that the input data corresponds to one period of a periodic function [2]. Thus, applying FFT-based pseudospectral differentiation introduces periodic boundary conditions.

This spatial discretization reduces the PDE to a system of ordinary differential equations (ODEs), which can be numerically solved using the function *scipy.integrate.solve_ivp* from the *SciPy*-package. For time integration, the 'Radau' method is used. This implicit Runge-Kutta method of order 5 provides high accuracy and stability, particularly for stiff problems [8]. The solution is stored only at the previously mentioned 16 time steps, even though the solver may work with intermediate time steps internally.

3.1.2 Implementation Details of the FNO

The main setup of the FNO has already been described in section 2.3. The network consists of a linear input layer, four stacked Fourier layers with GELU activation function and a linear output layer. The Gaussian error linear unit (GELU) activation function is a nonlinear transformation, suitable for a variety of machine learning applications. The function is given by

$$\mathrm{GELU}(x) = x \cdot \Phi(x), \quad \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} \, \mathrm{d}t, \tag{3.5}$$

where $\Phi(x)$ is the Gaussian cumulative distribution function [14]. The maximum number of frequency modes $k_{\max,j}$ is set to 12 in both spatial dimensions, and the layer width d_v is fixed at 32. Each input and output channel is discretized on the spatial grid with $n = 30 \cdot 25 = 750$ points. Thus, following the notation introduced in subsection 2.2.1, the network input can be represented as $a_j|_{D_j} \in \mathbb{R}^{750\times 7}$ and the output as $u_j|_{D_j} \in \mathbb{R}^{750\times 16}$, i.e. $d_a = 7$ and $d_u = 16$. The 7 input channels correspond to the coordinates x, y, the initial condition and the parameters V_1, V_2, D_1, D_2, which are broadcast to the full spatial grid to match with the size of the other channels. The output consists of 16 channels, each representing the predicted solution at one time step. This results in a model with a total of 1.18 million trainable parameters. Note that, the total number of trainable parameters depends on the number of frequency modes, the width of the layer, and the number of input and output channels, but it is independent of the number of spatial grid points used in the discretization.

The FNO is trained with 5000 training samples, which are generated as described in the previous subsection. The training samples are split into a training set (≈80%, 4040 samples), a validation set (≈10%, 480 samples) and a test set (≈10%, 480 samples). To improve training efficiency, the data is processed in batches, using a fixed batch size of 32.

The loss function is composed of three terms:

$$\mathrm{Loss} = \mathrm{Loss}_{\mathrm{MSE}} + 0.01 \cdot \mathrm{Loss}_{\mathrm{PDE}} + \mathrm{Loss}_{\mathrm{IC}}. \tag{3.6}$$

The scaling factor 0.01 is chosen to ensure that all three components have approximately the same magnitude. The first term, $\mathrm{Loss}_{\mathrm{MSE}}$, represents the mean squared error between the model's prediction and the training data. It is implemented using the *PyTorch*-function *torch.nn.MSELoss* [19]. The second term, $\mathrm{Loss}_{\mathrm{PDE}}$, is the mean squared residual of the advection-diffusion equation, i.e.

$$\frac{1}{16 \cdot 30 \cdot 25} \sum_{i=1}^{16} \sum_{j=1}^{30} \sum_{k=1}^{25} (u_t + V_1 u_x + V_2 u_y - D_1 u_{xx} - D_2 u_{yy})^2 \Big|_{t_i, x_j, y_k}. \tag{3.7}$$

The spatial derivatives are computed using the pseudospectral method to maintain consistency with the data generation process. However, the pseudospectral differentiation

is here implemented manually using *PyTorch*-operations, because the previously used *scipy.fftpack.diff* function is not compatible with the *PyTorch*-framework. The time derivative is approximated using central differences with fourth-order accuracy. Lastly, the initial condition loss, Loss_{IC}, is computed as the mean squared error between the predicted solution at time $t = 0$ and the true initial condition.

The training process is initialized with a learning rate of 0.001, which is halved every 100 epochs. Training is carried out for a total of 500 epochs. After each epoch, the loss on the validation set is computed. If it is lower than the previously obtained minimum, the current model parameters are saved and the loss on the test set is computed to evaluate the model. Optimization is performed using the Adam optimizer, which is an accelerated and adaptive variant of stochastic gradient descent (SGD) [12].

The training and validation losses are examined to assess the learning behavior. Figure 3 illustrates the evolution of the scaled individual loss components during training, verifying that the scaling factors in equation (3.6) are chosen appropriately. As shown in figure 4a, the combined training loss and the validation loss consistently decrease over the epochs, which indicates that the model successfully learns from the data and is also able to generalize. Figure 4b provides a more detailed view of the second half of the training process and it confirms that both losses continue to steadily decrease. In all plots the y-axis is logarithmically scaled.

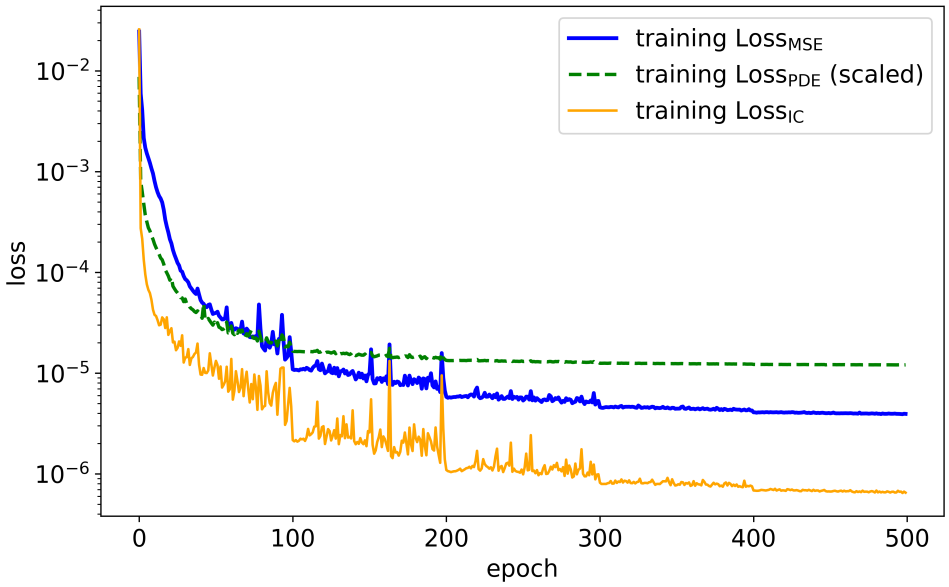
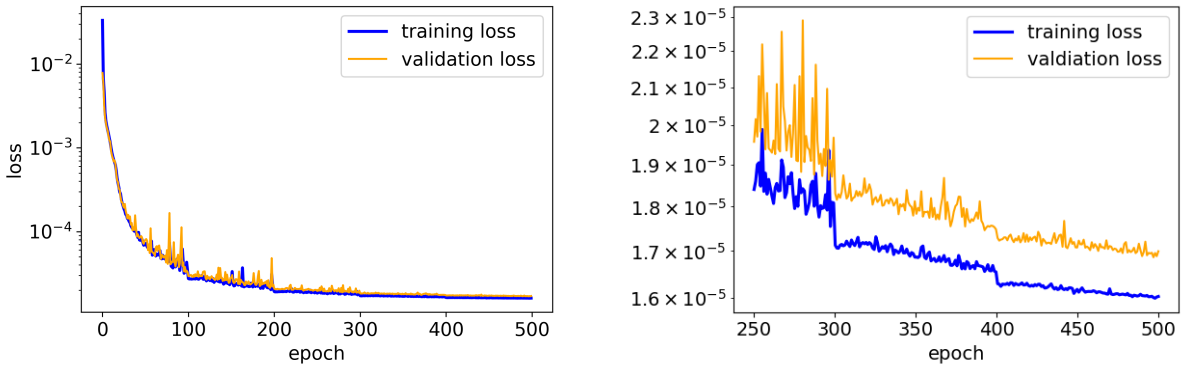


Figure 3: scaled training loss components over all 500 epochs



(a) all 500 epochs (b) last 250 epochs

Figure 4: training and validation loss over the epochs

3.1.3 Results

After the training process is completed, the model's performance is evaluated. First, to quantify the model's accuracy the relative ℓ^2 error between the predicted and the true solution is computed. The relative ℓ^2 error is defined as

$$\frac{\|u_{\text{true}} - u_{\text{pred}}\|_2}{\|u_{\text{true}}\|_2}, \tag{3.8}$$

where $\|.\|_2$ is the Euclidean norm over all grid points and time steps [21].

Table 1 demonstrates that the relative error remains nearly constant, if the spatial resolution in the test set is changed. This confirms that the FNO is able to predict solutions on a finer grid, although it was only trained on the coarse grid data. However, increasing the resolution does not result in a lower relative error, in contrast to what is typically observed with standard numerical solvers.

resolution	**relative error**
30×25	0.00951
60×50	0.01137
120×100	0.01335

Table 1: relative ℓ^2 error for different resolutions

Second, to get a visual impression, figure 5 and figure 6 present the predicted solution along with the corresponding absolute error compared to the true solution at the first, last, and one intermediate time step for two samples from the test set. Figure 5 shows the sample with the lowest relative ℓ^2 error, and its parameters are summarized in table 2. Analogously, figure 6 visualizes the sample with the highest relative ℓ^2 error, and its

parameters are listed in table 3. The color bars are adjusted individually for each time step.

V_1	V_2	D_1	D_2
0.4474	0.7371	0.0515	0.0579

Table 2: parameters for the test sample with the lowest error

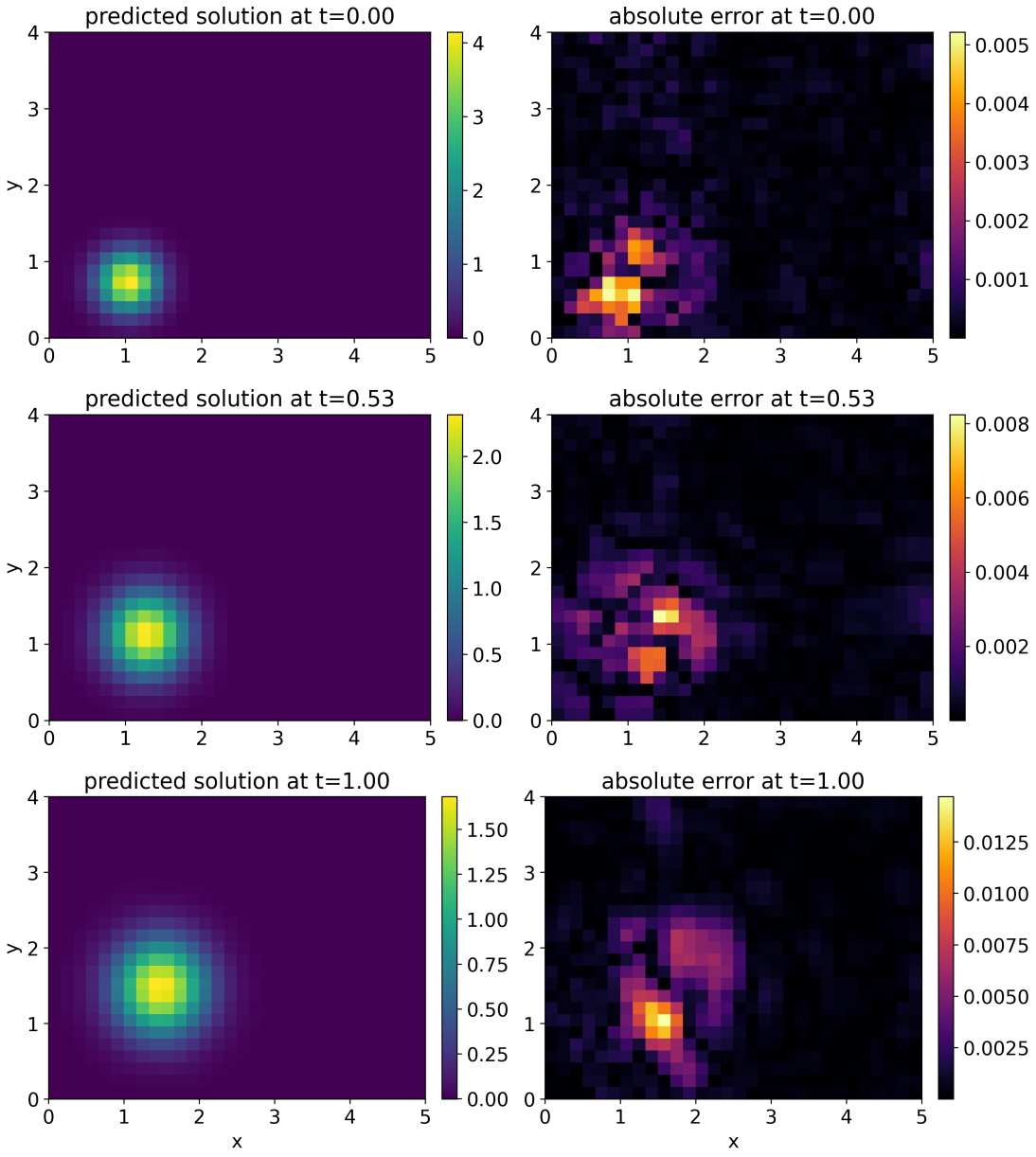


Figure 5: predicted solution and absolute error for test sample with lowest error

V_1	V_2	D_1	D_2
0.2952	0.3058	0.0534	0.0106

Table 3: parameters for the test sample with the highest error

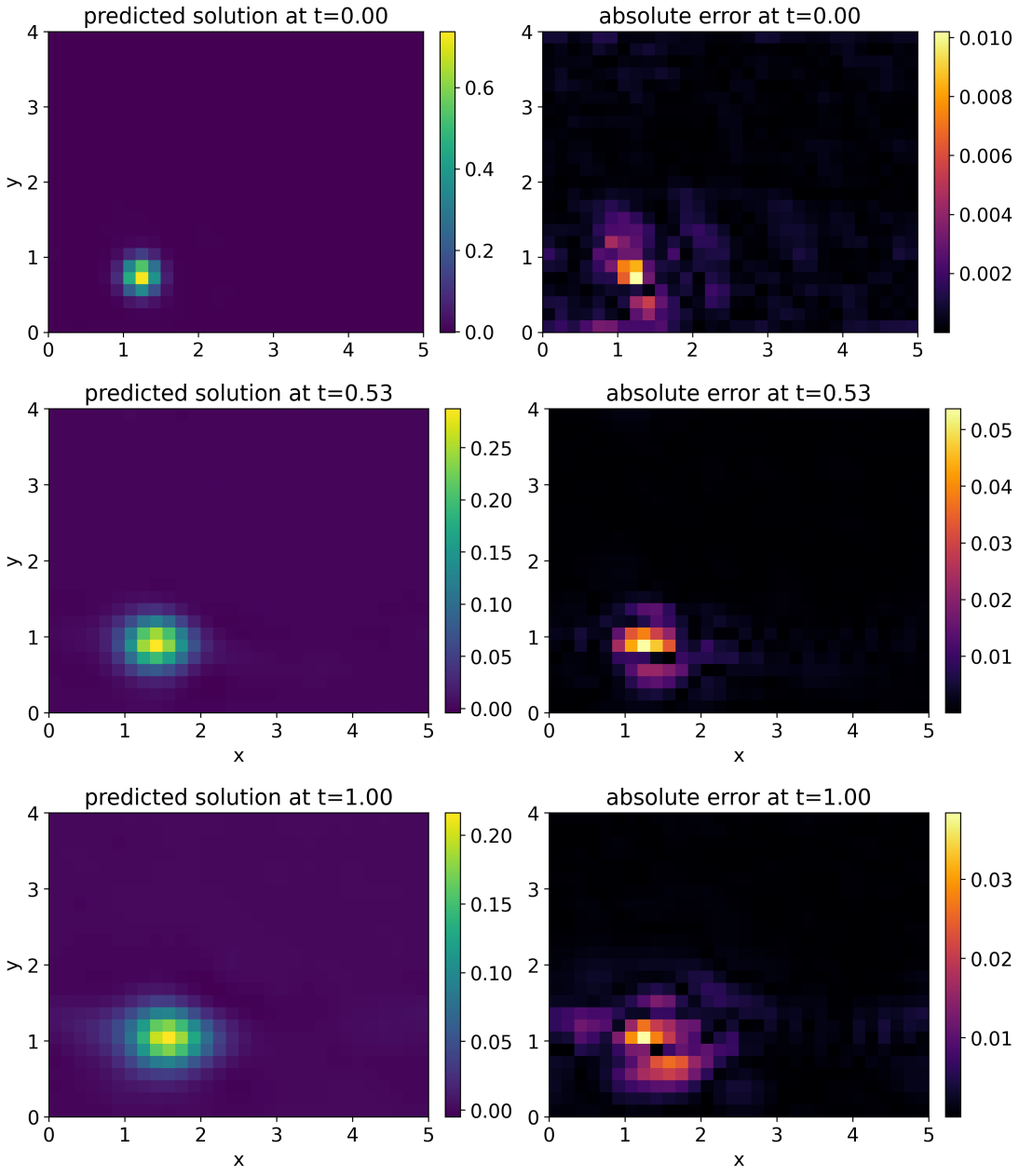


Figure 6: predicted solution and absolute error for test sample with highest error

As illustrated in figure 5 and figure 6, the errors are mainly concentrated in the regions where the solutions are nonzero. For both test samples, the errors increase over time.

3.2 Two-Component Model in 2d

The two-component model was developed to simulate the transportation of a tracer through an organ with the blood flow [25]. It is based on an advection equation without diffusion in two spatial dimensions. The first component u^1 represents the tracer in the oxygenated blood, while the second component u^2 represents the tracer in the deoxygenated blood. To obtain a realistic model at the inflow boundary there is only oxygenated blood and at the outflow boundary there is only deoxygenated blood. Within the domain oxygenated blood is converted into deoxygenated blood at a rate $c(x, y)$, which may vary spatially. Additionally, the two components u^1 and u^2 move at spatially varying velocities $v^1(x, y) = \begin{pmatrix} v^{1,x}(x, y) \\ v^{1,y}(x, y) \end{pmatrix}$ and $v^2(x, y) = \begin{pmatrix} v^{2,x}(x, y) \\ v^{2,y}(x, y) \end{pmatrix}$. The velocity component $v^{1,x}$ is high at the inflow boundary and decreases to zero at the outflow boundary. On the other hand, $v^{2,x}$ is low at the inflow, but increases towards a fast outflow value. This leads to the following system of equations:

$$
\begin{aligned}
u_t^1(x, y, t) + \nabla \cdot f(u^1) &= -c(x, y)\, u^1(x, y, t) && (3.9)\\
u_t^2(x, y, t) + \nabla \cdot g(u^2) &= c(x, y)\, u^1(x, y, t) && (3.10)\\
f(u^1) &= v^1(x, y)\, u^1(x, y, t) \\
g(u^2) &= v^2(x, y)\, u^2(x, y, t),
\end{aligned}
$$

with $(x, y) \in D$ and $t \geq 0$, where the symbol "$\cdot$" denotes the dot product [4].

3.2.1 Choice of Parameters

In this numerical experiment, the two-component model is studied over the spatial domain $(x, y) \in [0, 5] \times [0, 4]$ and the time interval $t \in [0, 1]$. The spatial grid is uniformly discretized with a resolution of 60×50 points, while the time domain is divided into 16 equally spaced time steps.

For each training sample the parameters are randomly chosen. The region where the conversion from oxygenated to deoxygenated blood occurs is defined as a vertical stripe. Specifically, this stripe is given by $(x, y) \in [x_c - l, x_c + l] \times [0, 4]$, where the center position x_c is uniformly sampled from the interval $[1.75, 2)$ and the half-width l is drawn from $[0.5, 1)$. Within this region, the conversion rate is set to a constant value κ, which is uniformly sampled from the interval $[1.5, 2.75)$. To ensure numerical stability, the edges of the stripe are smoothened using logistic functions:

$$
c(x, y) = \kappa \left(\frac{1}{1 + e^{-10(x-(x_c-l))}} - \frac{1}{1 + e^{-10(x-(x_c+l))}} \right). \tag{3.11}
$$

A visualization of the resulting conversion rate for one example is shown in figure 7.

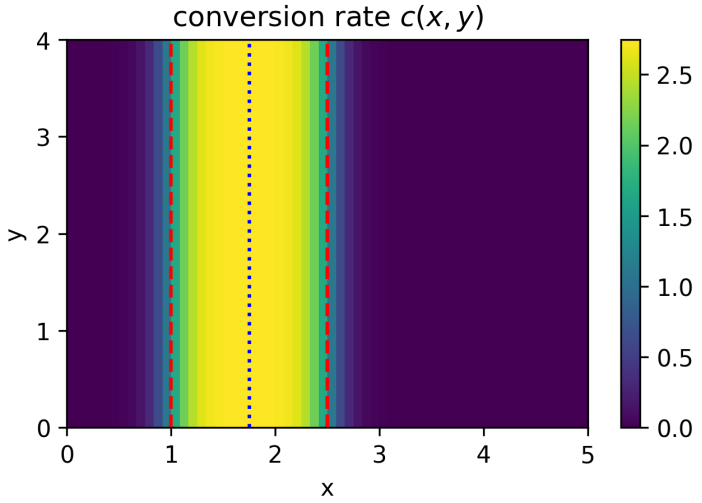


Figure 7: conversion rate with $x_c = 1.75$, $l = 0.75$, $\kappa = 2.75$

The velocity component $v^{1,x}$ is defined as a continuous, piecewise linear function:

$$v^{1,x}(x, y) := \begin{cases} a(x - (x_c + l)) & \text{if } x \leq x_c + l \\ 0 & \text{if } x > x_c + l, \end{cases} \tag{3.12}$$

with slope a sampled uniformly from the interval $[-1.75, -1.25)$.

In contrast, the velocity component $v^{2,x}$ is an increasing linear function, also depending only on the x-coordinate. Its slope is uniformly sampled from the interval $[0.01, 0.5)$ and the intercept is drawn from $[1, 1.25)$. Similarly, the velocity components $v^{1,y}$ and $v^{2,y}$ are also linear functions, they vary only in the y-direction. Their slopes are sampled uniformly from the interval $[-0.25, 0.25)$ and their intercepts from $[-0.5, 0.5)$. Thus, these velocities may have positive or negative values and can be either decreasing or increasing.

The initial condition for the first component u^1 is given by

$$u^1(x, y, 0) = h\, e^{-\frac{(x-x_i)^2+(y-y_i)^2}{w}}, \tag{3.13}$$

where the coordinates (x_i, y_i) are uniformly sampled from $[0.75, 1) \times [1.85, 2.15)$, the parameter h from the interval $[0.5, 5)$ and the parameter w from $[0.05, 0.2)$. The second component u^2 is initially set to zero.

3.2.2 Finite Volume Method

The spatial derivatives are now computed using the finite volume method, as described in [5]. This numerical technique divides the space into small cells and ensures that the inflow into each cell equals the outflow from that cell. The finite volume method is well-suited for the numerical simulation of various types of conservation laws.

First, the domain $D \subset \mathbb{R}^2$ is partitioned into rectangles $V_{i,j} = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}] \times [y_{j-\frac{1}{2}}, y_{j+\frac{1}{2}}]$ where $x_{i\pm\frac{1}{2}} = x_i \pm \frac{\Delta x}{2}$ and analogously $y_{j\pm\frac{1}{2}} = y_j \pm \frac{\Delta y}{2}$. To enforce boundary conditions, an additional layer of ghost cells, i.e. one additional row or column, is added to the physical domain along each edge. The solution at the cell centers (x_i, y_j) is computed as the cell average of the unknown u^1 in cell $V_{i,j}$, i.e.

$$U^1_{i,j}(t) = \frac{1}{|V_{i,j}|} \int_{V_{i,j}} u^1(x, y, t) \, \mathrm{d}x \, \mathrm{d}y, \tag{3.14}$$

where $|V_{i,j}|$ denotes the area of the rectangle $V_{i,j}$. At the center of the ghost cells the solution is set to zero, which corresponds to homogeneous Dirichlet boundary conditions.

The first step in deriving the finite volume method is to integrate the PDE (3.9) for u^1 over a cell $V_{i,j}$:

$$\begin{aligned} \int_{V_{i,j}} u^1_t(x, y, t) \, \mathrm{d}x \, \mathrm{d}y + \int_{V_{i,j}} \nabla \cdot f(u^1) \, \mathrm{d}x \, \mathrm{d}y &= -\int_{V_{i,j}} c(x, y) \, u^1(x, y, t) \, \mathrm{d}x \, \mathrm{d}y \quad \Leftrightarrow \\ \int_{V_{i,j}} \frac{\partial u^1(x, y, t)}{\partial t} \, \mathrm{d}x \, \mathrm{d}y + \int_{V_{i,j}} \operatorname{div}(f(u^1)) \, \mathrm{d}x \, \mathrm{d}y &= -\int_{V_{i,j}} c(x, y) \, u^1(x, y, t) \, \mathrm{d}x \, \mathrm{d}y. \end{aligned}$$

Provided that the integrands and the domain are sufficiently smooth, the time derivative may be interchanged with the integral and the divergence theorem can be applied to convert the volume integral into a surface integral [6] [17]:

$$\begin{aligned} \frac{d}{dt} \int_{V_{i,j}} u^1(x, y, t) \, \mathrm{d}x \, \mathrm{d}y + \int_{\partial V_{i,j}} f(u^1) \cdot \mathbf{n} \, \mathrm{d}s &= -\int_{V_{i,j}} c(x, y) \, u^1(x, y, t) \, \mathrm{d}x \, \mathrm{d}y \quad \Leftrightarrow \\ \frac{d}{dt} U^1_{i,j}(t) + \frac{1}{|V_{i,j}|} \int_{\partial V_{i,j}} f(u^1) \cdot \mathbf{n} \, \mathrm{d}s &= -\frac{1}{|V_{i,j}|} \int_{V_{i,j}} c(x, y) \, u^1(x, y, t) \, \mathrm{d}x \, \mathrm{d}y, \end{aligned}$$

where $\mathbf{n}$ is the outward pointing unit normal vector.

The source term on the right-hand side is approximated by

$$-\frac{1}{|V_{i,j}|} \int_{V_{i,j}} c(x, y) \, u^1(x, y, t) \, \mathrm{d}x \, \mathrm{d}y \approx C_{i,j} \, U^1_{i,j}(t),$$

where $C_{i,j} = \frac{1}{|V_{i,j}|} \int_{V_{i,j}} c(x, y) \, \mathrm{d}x \, \mathrm{d}y$ is the cell-averaged conversion rate.

The surface integral over the cell boundaries is the sum of the integrals over the four cell edges:

$$\int_{\partial V_{i,j}} f(u^1)\cdot\mathbf{n}\,\mathrm{d}s = \underbrace{\int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} f^x(u^1(x_{i+\frac{1}{2}},y,t))\,\mathrm{d}y}_{\approx F^1_{i+\frac{1}{2},j}(t)} - \underbrace{\int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} f^x(u^1(x_{i-\frac{1}{2}},y,t))\,\mathrm{d}y}_{\approx F^1_{i-\frac{1}{2},j}(t)} + \underbrace{\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} f^y(u^1(x,y_{j+\frac{1}{2}},t))\,\mathrm{d}x}_{\approx G^1_{i,j+\frac{1}{2}}(t)} - \underbrace{\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} f^y(u^1(x,y_{j-\frac{1}{2}},t))\,\mathrm{d}x}_{\approx G^1_{i,j-\frac{1}{2}}(t)}, \tag{3.15}$$

where $f^x(u^1) = v^{1,x}u^1$ and $f^y(u^1) = v^{1,y}u^1$. The signs in front of the integrals are determined by the direction of outward pointing unit normal vector $\mathbf{n}$.

For the computation, the fluxes across the edges in equation (3.15) are approximated by numerical fluxes according to the upwind scheme, which is a common choice for advection equations [5]. The upwind scheme uses the value of the solution u^1 from the upstream direction, i.e. the direction from which the information is coming. This respects the physical propagation of information and avoids instabilities but it typically introduces numerical diffusion. The numerical approximations for the fluxes in the horizontal direction are given by the following definitions:

$$F^1_{i+\frac{1}{2},j}(t) := \begin{cases} v^{1,x}(x_{i+\frac{1}{2}},y_j)\,U^1_{i,j}(t) & \text{if } v^{1,x}(x_{i+\frac{1}{2}},y_j) > 0 \\ v^{1,x}(x_{i+\frac{1}{2}},y_j)\,U^1_{i+1,j}(t) & \text{if } v^{1,x}(x_{i+\frac{1}{2}},y_j) < 0 \\ 0 & \text{if } v^{1,x}(x_{i+\frac{1}{2}},y_j) = 0 \end{cases} \tag{3.16}$$

$$F^1_{i-\frac{1}{2},j}(t) := \begin{cases} v^{1,x}(x_{i-\frac{1}{2}},y_j)\,U^1_{i-1,j}(t) & \text{if } v^{1,x}(x_{i-\frac{1}{2}},y_j) > 0 \\ v^{1,x}(x_{i-\frac{1}{2}},y_j)\,U^1_{i,j}(t) & \text{if } v^{1,x}(x_{i-\frac{1}{2}},y_j) < 0 \\ 0 & \text{if } v^{1,x}(x_{i-\frac{1}{2}},y_j) = 0. \end{cases} \tag{3.17}$$

Analogously, the numerical approximations for the fluxes in the vertical direction are given by the following definitions:

$$G^1_{i,j+\frac{1}{2}}(t) := \begin{cases} v^{1,y}(x_i,y_{j+\frac{1}{2}})\,U^1_{i,j}(t) & \text{if } v^{1,y}(x_i,y_{j+\frac{1}{2}}) > 0 \\ v^{1,y}(x_i,y_{j+\frac{1}{2}})\,U^1_{i,j+1}(t) & \text{if } v^{1,y}(x_i,y_{j+\frac{1}{2}}) < 0 \\ 0 & \text{if } v^{1,y}(x_i,y_{j+\frac{1}{2}}) = 0 \end{cases} \tag{3.18}$$

$$G^1_{i,j-\frac{1}{2}}(t) := \begin{cases} v^{1,y}(x_i,y_{j-\frac{1}{2}})\,U^1_{i,j-1}(t) & \text{if } v^{1,y}(x_i,y_{j-\frac{1}{2}}) > 0 \\ v^{1,y}(x_i,y_{j-\frac{1}{2}})\,U^1_{i,j}(t) & \text{if } v^{1,y}(x_i,y_{j-\frac{1}{2}}) < 0 \\ 0 & \text{if } v^{1,y}(x_i,y_{j-\frac{1}{2}}) = 0. \end{cases} \tag{3.19}$$

In conclusion, the discretization of PDE (3.9) for u^1 is given by the following expression:

$$\frac{d}{dt}U^1_{i,j}(t) = \frac{F^1_{i-\frac{1}{2},j}(t) - F^1_{i+\frac{1}{2},j}(t)}{\Delta x} + \frac{G^1_{i,j-\frac{1}{2}}(t) - G^1_{i,j+\frac{1}{2}}(t)}{\Delta y} - C_{i,j}\, U^1_{i,j}(t), \tag{3.20}$$

where Δx and Δy denote the stepsize in horizontal resp. vertical direction.

Applying the finite volume method and using analogous definitions for the numerical approximations of the fluxes yields the following discretization of PDE (3.10) for u^2:

$$\frac{d}{dt}U^2_{i,j}(t) = \frac{F^2_{i-\frac{1}{2},j}(t) - F^2_{i+\frac{1}{2},j}(t)}{\Delta x} + \frac{G^2_{i,j-\frac{1}{2}}(t) - G^2_{i,j+\frac{1}{2}}(t)}{\Delta y} + C_{i,j}\, U^1_{i,j}(t). \tag{3.21}$$

Lastly, the ODE (3.20) for the first component u^1 with respect to time is solved using the function *scipy.integrate.solve_ivp*. Here, the method 'LSODA' is employed, which automatically detects stiffness and switches between an explicit and an implicit solver accordingly [9].

To solve the ODE (3.21) for the second component u^2, the previously computed solution u^1 is interpolated using *scipy.interpolate.interp1d*, which is also part of the *SciPy*-library. Then, *scipy.integrate.solve_ivp* can be used to compute the solution u^2.

3.2.3 Comparison with Analytical Solution

In this subsection, the parameters are chosen such that an analytical solution is available. This makes it possible to validate the numerical solver based on the previously described finite volume method. The PDE (3.9) for the first component u^1 in two spatial dimensions can be rewritten as follows:

$$u_t = -(c + v^{1,x}_x + v^{1,y}_y)\, u - v^{1,x} u_x - v^{1,y} u_y, \tag{3.22}$$

where u denotes u^1 for brevity and $v^{1,x}_x = \frac{\partial v^{1,x}(x,y,t)}{\partial x}$, $v^{1,y}_y = \frac{\partial v^{1,y}(x,y,t)}{\partial y}$ are the partial derivatives.

The velocities and the conversion rate are set to be $v^{1,x}(x,y) = x$, $v^{1,y}(x,y) = 0$ and $c(x,y) = \kappa$ with $\kappa \in \mathbb{R}$ for all $(x,y) \in D$. The initial condition at time $t = 0$ is set to

$$u(x,y,0) = e^{-((x-0.5)^2+(y-2)^2)}, \quad \forall (x,y) \in D. \tag{3.23}$$

Under these assumptions, the equation (3.22) simplifies to

$$u_t = -(\kappa + 1)\, u - x u_x \tag{3.24}$$

and the analytical solution is given by

$$u(x, y, t) = e^{-((xe^{-t}-0.5)^2+(y-2)^2)}\, e^{-(\kappa+1)t}, \tag{3.25}$$

for all $(x, y) \in D$ and time $t \geq 0$ [4].

For the following numerical experiment, $\kappa = 1.5$ is chosen as the constant conversion rate. Here, the PDE is studied on the spatial domain $[1, 3] \times [1, 3]$ over the time interval $[0, 1.3]$. A spatial resolution of 60×50 is used to ensure consistency with the resolution of the FNO training data. Figure 8 presents a visual comparison of the analytical and the numerical solution, the absolute error between the two solutions is illustrated in figure 9. Finally, figure 10 confirms that the finite volume method achieves second-order convergence with respect to spatial refinement [22].

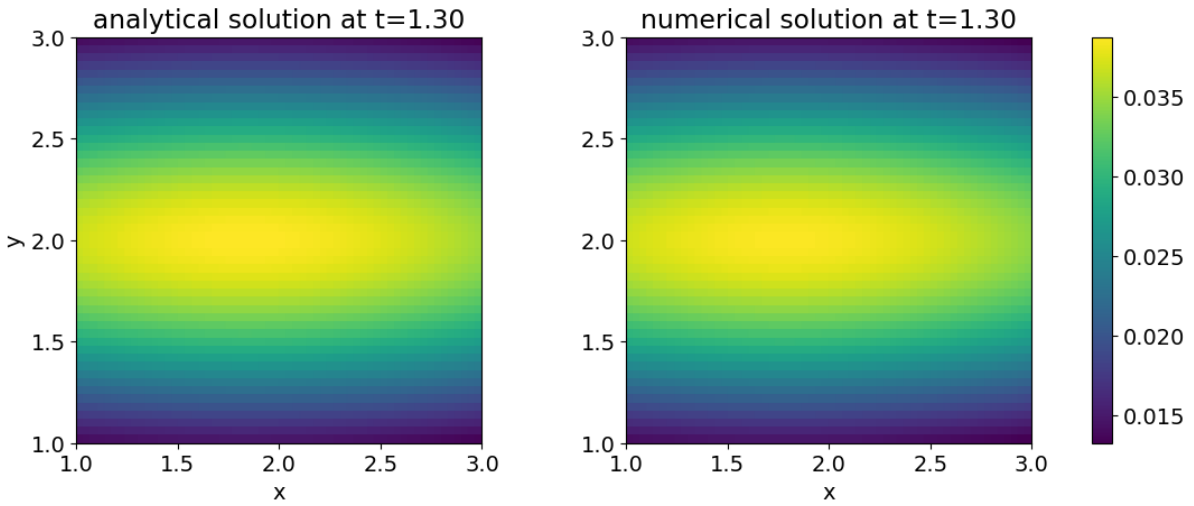


Figure 8: comparison between analytical and numerical solution

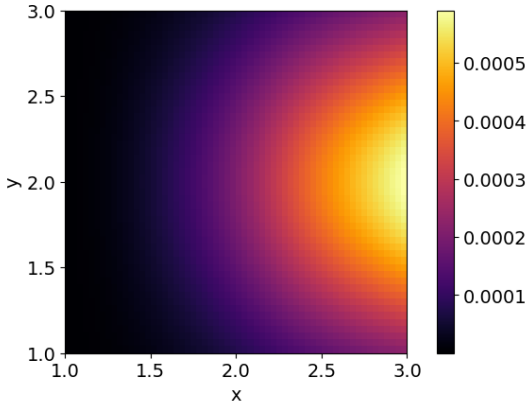


Figure 9: absolute error between analytical and numerical solution

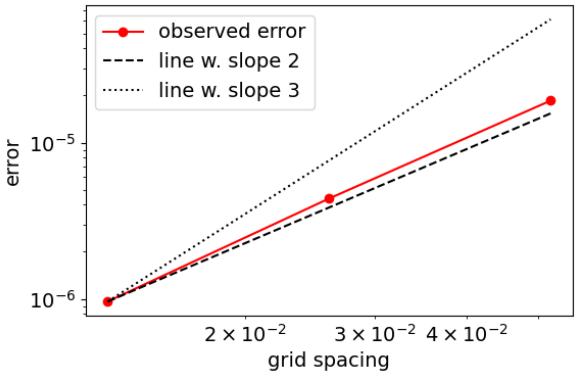


Figure 10: convergence plot using ℓ^2 error

3.2.4 Implementation Details of the FNO

The overall architecture and training process of the FNO are analogous to those used for solving the advection-diffusion equation. The main differences lie in the input and output channels, as well as in the computation of the loss function. The network for the two-component model has 8 input channels, which correspond to the velocities $v^{1,x}(x,y)$, $v^{1,y}(x,y)$, $v^{2,x}(x,y)$, $v^{2,y}(x,y)$, the conversion rate $c(x,y)$, the coordinates x, y and the initial condition $u^1(x,y,0)$. Since u^2 is initially set to zero, the network does not require $u^2(x,y,0)$ as an input. The output consists of 32 channels, with the first 16 representing the predicted solution u^1 and the last 16 representing the predicted u^2, each at one time step.

The FNO is again trained with 5000 training samples, which are split into a training set, a validation set and a test set. The loss function is defined as follows:

$$\text{Loss} = \text{Loss}_{\text{MSE}} + 0.001 \cdot \text{Loss}_{\text{PDE}} + \text{Loss}_{\text{IC}}. \tag{3.26}$$

The scaling factor is again chosen to ensure that all three components have approximately the same magnitude. The term Loss_{PDE} is the average of the mean squared PDE residuals for u^1 and u^2. Similarly, the initial condition loss is defined as the average of the initial condition losses for u^1 and u^2.

Figure 11 illustrates the evolution of the training and validation losses over the epochs, showing that both losses steadily decrease. The y-axis is logarithmically scaled.

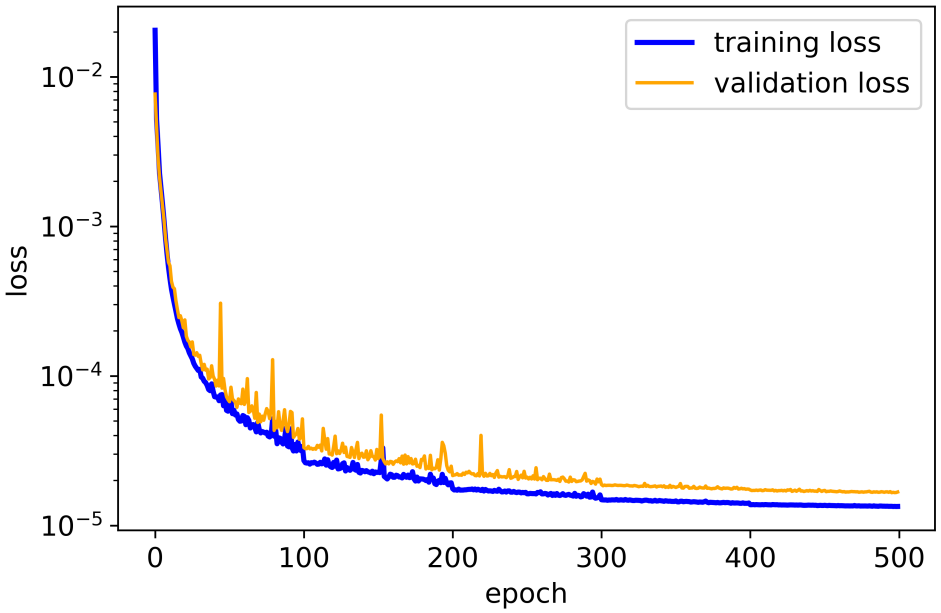


Figure 11: training and validation loss over all 500 epochs

3.2.5 Results

For a visual comparison, figure 13 visualizes the predicted solutions for u^1 and u^2 along with the corresponding absolute errors compared to the true solutions at the initial and the final time step for the test sample with the lowest relative ℓ^2 error. The corresponding velocity fields and the conversion rate are shown in figure 12. The velocity components $v^{1,x}$ and $v^{2,x}$ depend solely on the x-coordinate, which results in a constant profile along the vertical direction. In contrast, $v^{1,y}$ and $v^{2,y}$ vary only in the y-coordinate and are therefore constant along the horizontal direction. Figure 14 illustrates the parameters, and figure 15 the predicted solutions and absolute errors, for the test sample with the highest relative ℓ^2 error. The color bars in the following figures are adjusted individually for each plot.

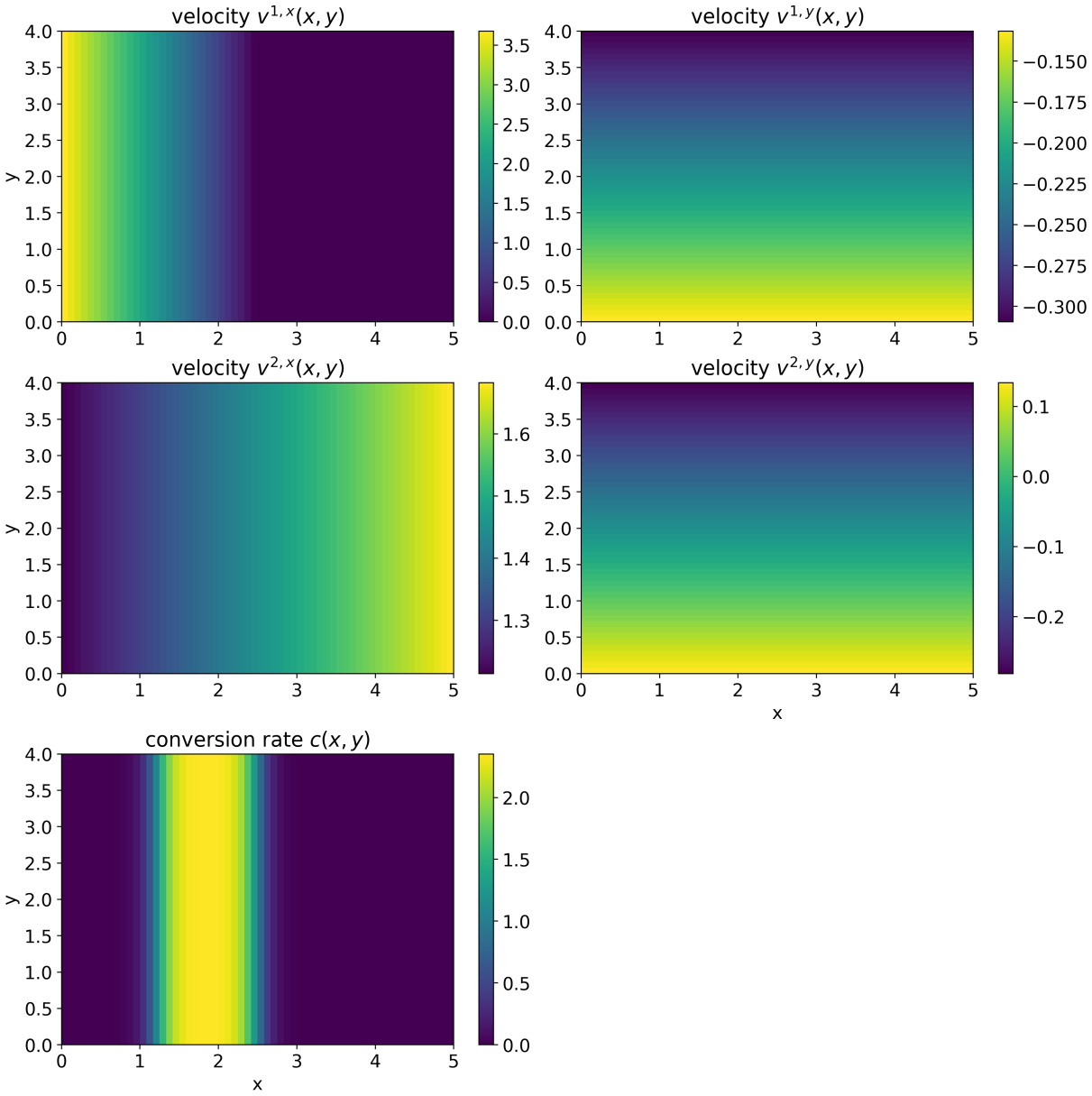


Figure 12: parameters for the test sample with the lowest error

Figure 12 shows that the velocity component $v^{1,x}$ attains high values for small x-coordinates and decreases toward zero as x approaches the right boundary of the conversion region. In contrast, $v^{2,x}$ increases with increasing x-coordinate. The velocity component $v^{1,y}$ is negative and the downward flow becomes slower as y decreases. At small y-coordinates, $v^{2,y}$ is positive, while for larger y it takes on negative values.

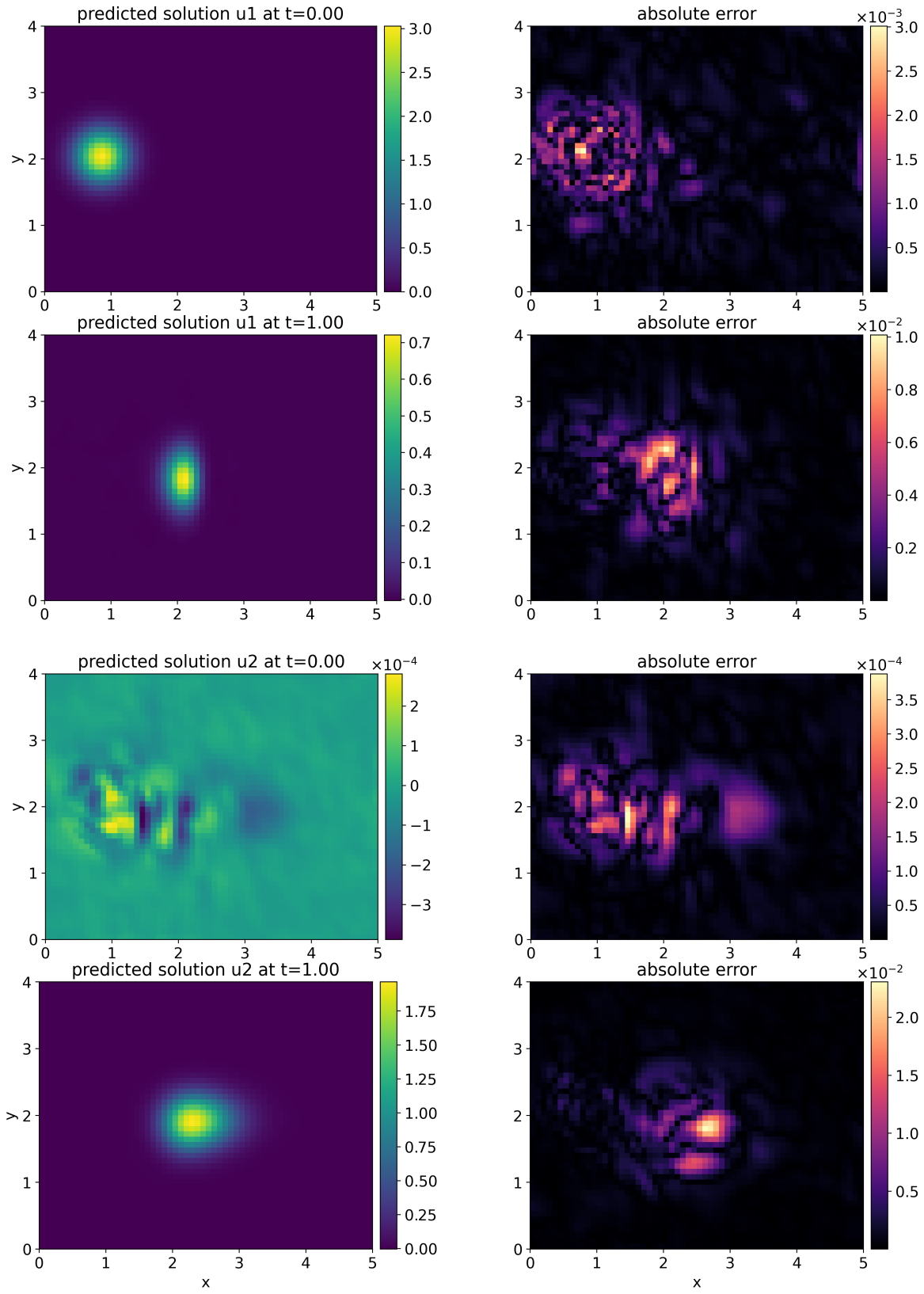


Figure 13: predicted solutions and absolute error for test sample with lowest error

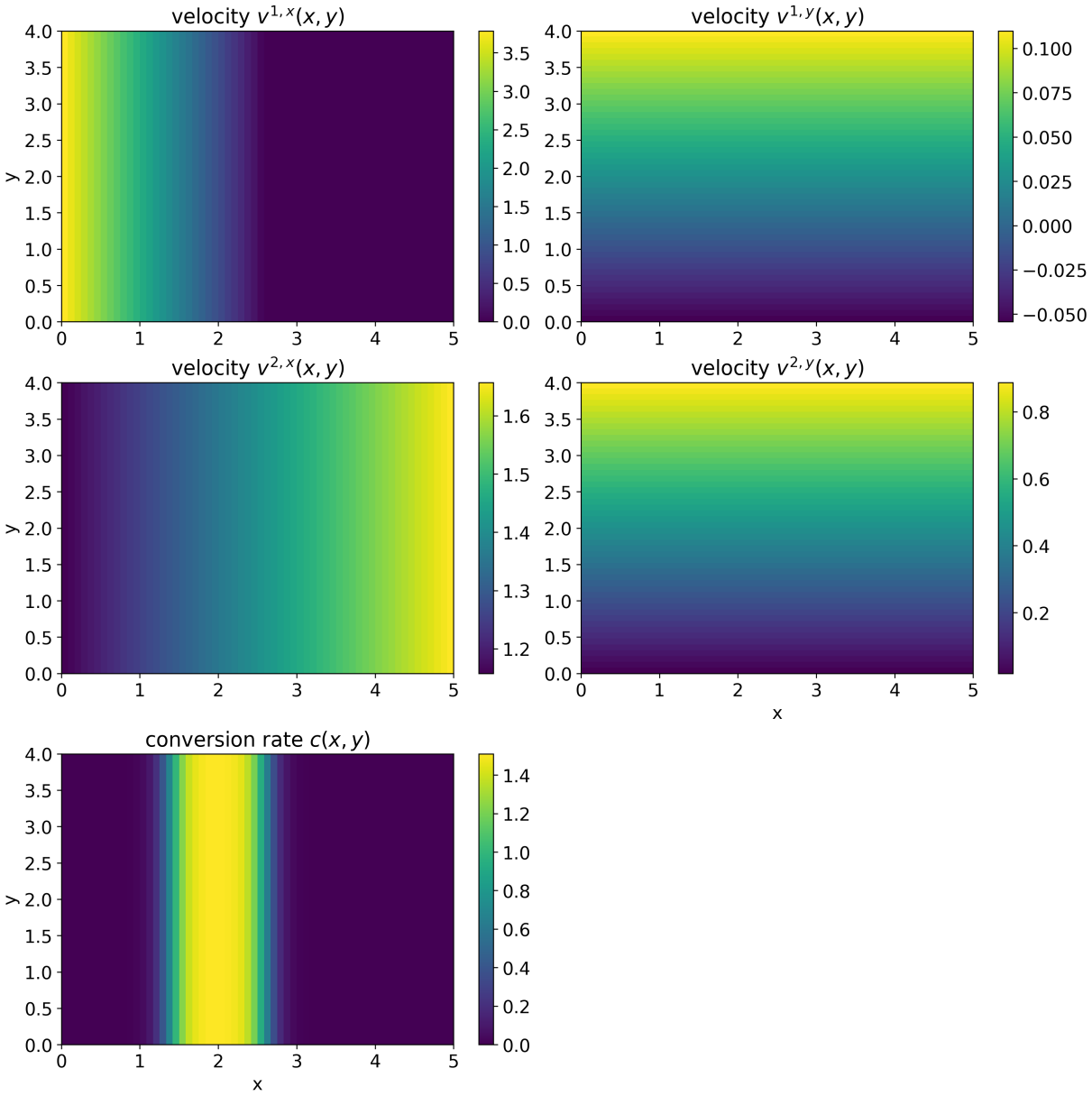


Figure 14: parameters for the test sample with the highest error

Compared to the previous test sample, figure 14 illustrates that the velocity components $v^{1,x}$ and $v^{2,x}$ behave similarly in the worst-case sample. Here, $v^{1,y}$ attains both positive and negative values, whereas $v^{2,y}$ is consistently positive and grows with increasing y-coordinate.

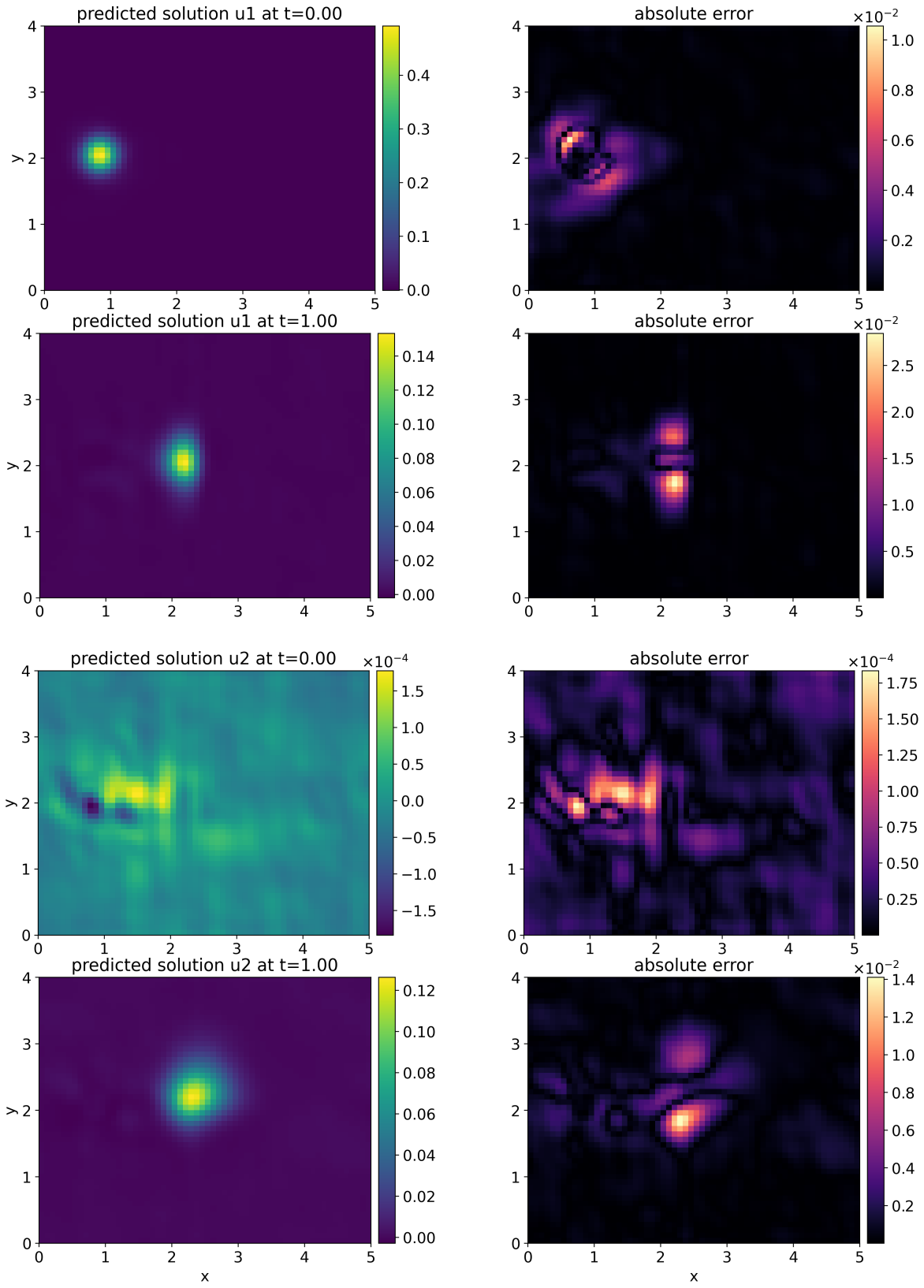


Figure 15: predicted solutions and absolute error for test sample with highest error

Figure 13 and figure 15 both show that, despite including the initial condition loss in the loss function, the model does not fully accurately predict the solution u^2 at the initial time $t = 0$. Nonetheless, with values below 0.001 the absolute error is sufficiently small.

Additionally, table 4 presents the impact of changing the spatial resolution in the test set on the relative error. The training set resolution is fixed at 60×50. The results clearly show that the model is significantly less accurate when predicting the solution on coarser or finer grids. In contrast, for the advection-diffusion equation the error remains nearly constant across different resolutions.

resolution	**relative error**
30×25	0.24296
60×50	0.03983
120×100	0.15811

Table 4: relative ℓ^2 error for different resolutions

To investigate the observed decrease in accuracy, figure 16 und figure 17 show the true and predicted solutions u^1 and u^2 for test samples with resolutions 30×25 and 120×100, respectively. For the sample with lower resolution, the FNO overestimates the solution concentrations. This may be due to the fact that the finite volume method used to generate the ground truth solutions, as described in subsection 3.2.2, introduces numerical diffusion. Increasing the spatial resolution reduces this numerical diffusion [20]. Thus, the FNO trained on the 60×50 grid may have learned to reproduce a certain amount of numerical diffusion present in its training data. When applied to a higher-resolution test sample with less inherent numerical diffusion, the FNO underestimates the solution concentrations. This is another indication that the FNO reflects the numerical diffusion present in its training data.

Lastly, to confirm that the discrepancies observed in table 4 are caused by differences in the numerical diffusion between resolutions, an additional test dataset was generated. The test samples were first computed with the finite volume method on a 59×49 grid and subsequently downsampled by selecting every second grid point in both spatial directions, resulting in a resolution of 30×25. Starting with a 59×49 grid ensures that the boundary points are preserved after downsampling. This procedure keeps the numerical diffusion in the lower-resolution sample comparable to that of the training data. In this setting, the FNO achieves a relative ℓ^2 error of 0.03115 on the coarser grid, which is nearly identical to the error obtained on the test set with the same resolution as the training data. In conclusion, the FNO is able generalize across varying resolutions, provided that the numerical errors in the training and test samples are comparable.

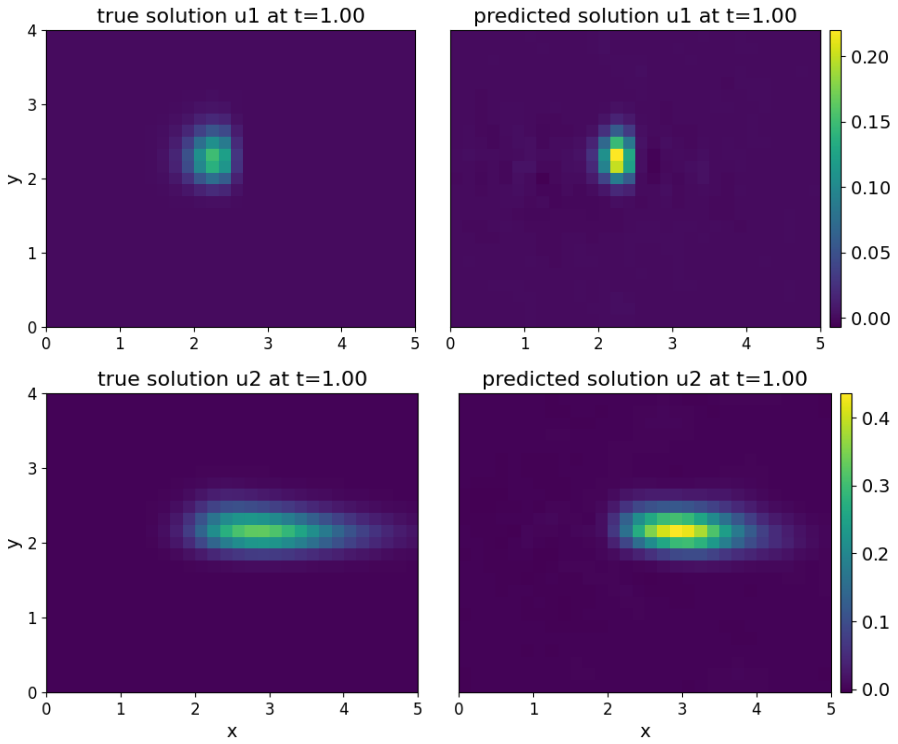


Figure 16: solution comparison for test sample with lower resolution at final time $t = 1$

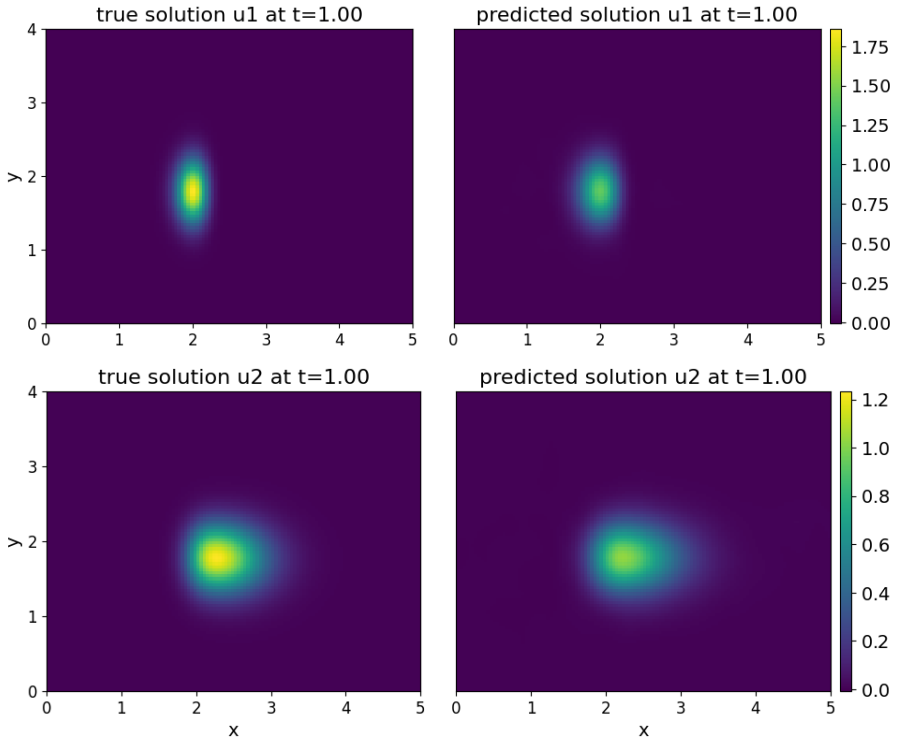


Figure 17: solution comparison for test sample with higher resolution at final time $t = 1$

3.3 Two-Component Model in 3d

The two-component model introduced in section 3.2 can be extended to three spatial dimensions. The resulting three-dimensional model is governed by the following set of equations:

$$u_t^1(x,y,z,t) + \nabla \cdot f(u^1) = -c(x,y,z)\, u^1(x,y,z,t) \tag{3.27}$$
$$u_t^2(x,y,z,t) + \nabla \cdot g(u^2) = c(x,y,z)\, u^1(x,y,z,t) \tag{3.28}$$
$$f(u^1) = v^1(x,y,z)\, u^1(x,y,z,t)$$
$$g(u^2) = v^2(x,y,z)\, u^2(x,y,z,t),$$

with $(x,y,z) \in D$ and $t \geq 0$. The velocity fields v^1 and v^2 are defined by

$$v^1(x,y,z) = \begin{pmatrix} v^{1,x}(x,y,z) \\ v^{1,y}(x,y,z) \\ v^{1,z}(x,y,z) \end{pmatrix} \text{ resp. } v^2(x,y,z) = \begin{pmatrix} v^{2,x}(x,y,z) \\ v^{2,y}(x,y,z) \\ v^{2,z}(x,y,z) \end{pmatrix}.$$

Here, the two-component model is studied over the spatial domain $(x,y,z) \in [0,5] \times [0,4] \times [0,3]$ and the time interval $t \in [0,1]$. Since the solution varies with three spatial coordinates and time, the resulting data is in fact four-dimensional. The spatial grid is uniformly discretized with a resolution of $30 \times 25 \times 20$ points, while the time domain is again divided into 16 equally spaced time steps.

The parameters for each training sample are again selected randomly. Unless specified otherwise, they are chosen as described in subsection 3.2.1 for the two-dimensional case. The region where the conversion from oxygenated to deoxygenated blood occurs is defined as a band restricted only in the x-direction. The additional velocity components $v^{1,z}$, $v^{2,z}$ are linear functions depending solely on the z-coordinate. Their slopes and intercepts lie within the same region as those of $v^{1,y}$ and $v^{2,y}$. The initial condition for the first component u^1 is given by

$$u^1(x,y,z,0) = h\, e^{-\frac{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}{w}}, \tag{3.29}$$

where the coordinates (x_i, y_i, z_i) are uniformly sampled from $[0.75, 1) \times [1.85, 2.15) \times [1.35, 1.65)$, the parameter h from the interval $[0.5, 5)$ and the parameter w from $[0.05, 0.2)$.

The training samples are computed using the finite volume method, described in subsection 3.2.2, which is implemented for the three-dimensional case following the same methodology. Due to computational constraints, the total number of training samples is reduced to 500. These are split into a training set (≈80%, 404 samples), a validation set (≈10%, 48 samples) and a test set (≈10%, 48 samples). A fixed batch size of 16 is used.

The overall architecture and training process of the FNO remain unchanged. The Fourier layers are adapted to enable processing three-dimensional input. The network now has

11 input channels, corresponding to the velocities $v^{1,x}$, $v^{1,y}$, $v^{1,z}$, $v^{2,x}$, $v^{2,y}$, $v^{2,z}$, the conversion rate $c(x, y, z)$, the coordinates x, y, z and the initial condition $u^1(x, y, z, 0)$. The output still consists of 32 channels, which represent the predicted solutions u^1 and u^2 at each time step. The maximum number of frequency modes $k_{\max,j}$ is kept at 12 in all three spatial dimensions, but the layer width d_v is reduced to 16. This results in a model with a total of 3.54 million trainable parameters.

The initial learning rate is set to 0.01 for this numerical experiment. After the training process is completed, the model's performance is evaluated. The FNO achieves a relative ℓ^2 error of 0.09536 on the test dataset. A visualization of the parameters as well as a visual comparison of the true and predicted solution for one sample from the test set can be found in appendix A.

Example 3.1. In this example, the conversion region is defined as a rectangular cuboid, i.e. it is restricted in all three spatial dimensions. As previously, the edges are smoothened, and the corresponding function is given by

$$\begin{aligned} c(x, y, z) = \kappa \big(\sigma(x-(x_c-l^x)) - \sigma(x-(x_c+l^x))\big) \big(\sigma(y-(y_c-l^y)) - \sigma(y-(y_c+l^y))\big) \\ \big(\sigma(z-(z_c-l^z)) - \sigma(z-(z_c+l^z))\big), \end{aligned} \tag{3.30}$$

where $\sigma(s) = (1 + e^{-10s})^{-1}$ and $\kappa \in [1.5, 2.75)$ is a constant value. Depending on the physical situation being modeled, this choice for the conversion region may be more appropriate. The cuboid center coordinates are uniformly sampled from the intervals $x_c \in [1.75, 2)$, $y_c \in [1.5, 1.75)$ and $z_c \in [1.25, 1.5)$. The half-widths in each direction l^x, l^y and l^z are independently drawn from $[0.5, 1)$. The three velocity components $v^{1,x}$, $v^{1,y}$ and $v^{1,z}$ are defined such that they decrease to zero at the boundaries of the conversion region in the corresponding direction, analogous to equation (3.12). Using these parameter settings, 500 training samples are generated. The FNO architecture and the training procedure remain unchanged. For this example, the FNO achieves a relative ℓ^2 error of 0.09421 on the test dataset.

4 Inverse Problem

As described in the previous chapter, the forward problem is: Given the PDE, the parameters of the PDE, the initial condition and possibly boundary conditions, find the solution of the PDE. In contrast, the inverse problem refers to determining the parameters of the PDE, given the equations and its solution. In the context of the FNO, formulating the inverse problem instead of the forward problem can essentially be interpreted as an interchange of the input and output channels.

4.1 Advection-Diffusion Equation

4.1.1 Implementation Details and Experimental Setup

In the context of the two-dimensional advection-diffusion equation (3.3), the inverse problem is to identify the parameters V_1, V_2, D_1 and D_2, given the PDE and the solution $u(x, y, t)$.

The overall architecture and training process of the FNO remain unchanged compared to the forward problem, except that the network now includes two linear output layers. For the inverse problem, the input consists of 18 channels, representing the coordinates x, y and the solution u at the 16 time steps. The output consists of 4 channels corresponding to the constants V_1, V_2, D_1 and D_2. As these parameters are spatially invariant, a global average pooling layer is added prior to the fully connected output layers to reduce the dimensionality. This results in a model with a total of 1.18 million trainable parameters, similar to the model used for the forward problem.

The training data generated for the forward problem, as described in subsection 3.1.1, can also be used to train the model for the inverse problem. The loss function is composed of two terms:

$$\text{Loss} = \text{Loss}_{\text{MSE}} + 0.001 \cdot \text{Loss}_{\text{PDE}}. \tag{4.1}$$

The initial condition loss is not applicable in the inverse setting, as the full solution, including the initial state $t = 0$ is provided as input to the model. In contrast, Loss_{PDE} remains meaningful, as it measures how well the predicted parameters reproduce the PDE satisfied by the given ground truth solution. The training process is initialized with a learning rate of 0.01, which is halved every 100 epochs.

4.1.2 Results

The model is trained on the complete dataset containing 5000 samples, which is split into training, validation and test sets. Training was performed over the course of 500 epochs using the parameters specified in the previous subsection. To evaluate the learning behavior, the training and validation losses are illustrated in figure 18. The y-axis is logarithmically scaled.

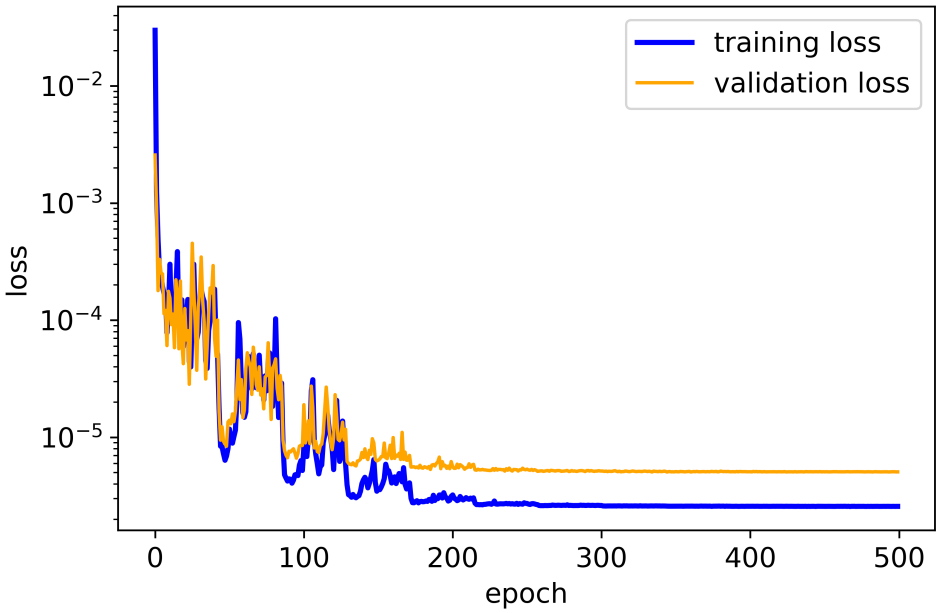


Figure 18: training and validation loss over all 500 epochs

To illustrate the model's performance, the true and predicted parameters are presented in table 5 for two samples from the test dataset. The first two rows present the true and predicted parameters for the test sample achieving the lowest relative ℓ^2 error, while the last two rows show the parameters of the sample with the highest error.

	V_1	V_2	D_1	D_2
true parameters	1.0403	0.8528	0.0662	0.0446
prediction	1.0404	0.8531	0.0663	0.0444
true parameters	0.2952	0.2615	0.0469	0.1401
prediction	0.2971	0.2703	0.0511	0.1372

Table 5: true and predicted parameters for two test samples

Additionally, figure 19 and figure 20 provide a visual impression of the solution obtained by solving the advection-diffusion equation, where the coefficients in the PDE are given by the parameters predicted by the FNO, and the absolute error compared to the true solution. The PDE is solved using the same procedure described for the data generation process in subsection 3.1.1.

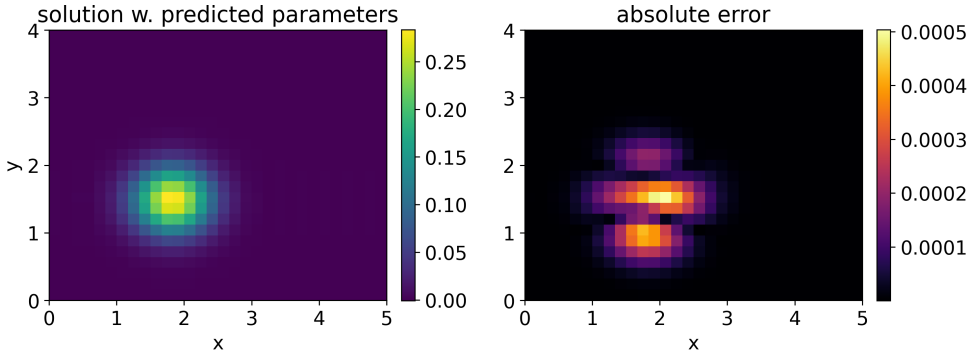


Figure 19: comparison of solutions for test sample with lowest error at final time t=1

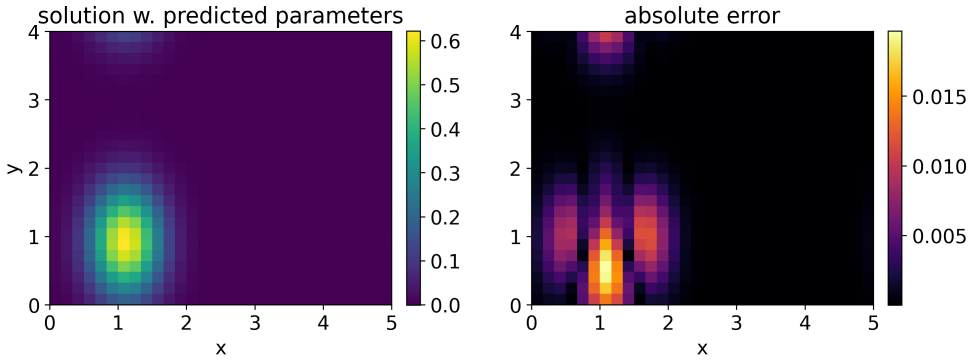


Figure 20: comparison of solutions for test sample with highest error at final time t=1

4.1.3 Impact of Training Set Size and Physics Loss

In practice, particularly in the medical field, obtaining large labeled training datasets is often challenging [1]. To investigate how a reduced training set size affects the model's predictive performance, training is repeated multiple times using randomly selected subsets of the original training dataset, each with a different size. For the smallest training dataset the batch size is reduced from 32 to 16. To ensure a consistent evaluation, the validation and test dataset remain unchanged and are not subsampled. To examine the influence of incorporating the physics loss into the loss function, the model is trained using both the combined term $\text{Loss} = \text{Loss}_{\text{MSE}} + 0.001 \cdot \text{Loss}_{\text{PDE}}$ and the mean squared error loss alone. Furthermore, training was also performed using only the physics loss term as the loss function for the various training subsets. The relative ℓ^2 errors between the true and predicted parameters for the different training setups are shown in table 6.

As presented in table 6, a sufficient training set size is very important. Regardless of the chosen loss function, the relative ℓ^2 error on the test set increases when the size of the training set is reduced. For all training set sizes, except for the smallest, the mean squared error loss alone yields a slightly lower relative error compared to the combined loss function. Figure 21 illustrates the evolution of the individual loss components during

training for the complete dataset. While the mean squared error loss, Loss_{MSE}, decreases steadily over the epochs, the physics loss, Loss_{PDE}, decreases sharply at the beginning, but then quickly stabilizes. This behavior is not unexpected, as Loss_{PDE} is not zero even for the ground truth or analytical solution. The limited decrease in Loss_{PDE} is likely due to numerical errors, particularly in approximating the time derivative based on a small number of timesteps.

training set size	**combined Loss**	**only Loss_{MSE}**	**only Loss_{PDE}**
4040	0.002493	0.002391	0.064844
1000	0.006070	0.006029	0.066261
500	0.011091	0.010763	0.067597
250	0.023281	0.022743	0.070886
100	0.064146	0.064749	0.085617

Table 6: relative ℓ^2 error for different training set sizes and different loss functions

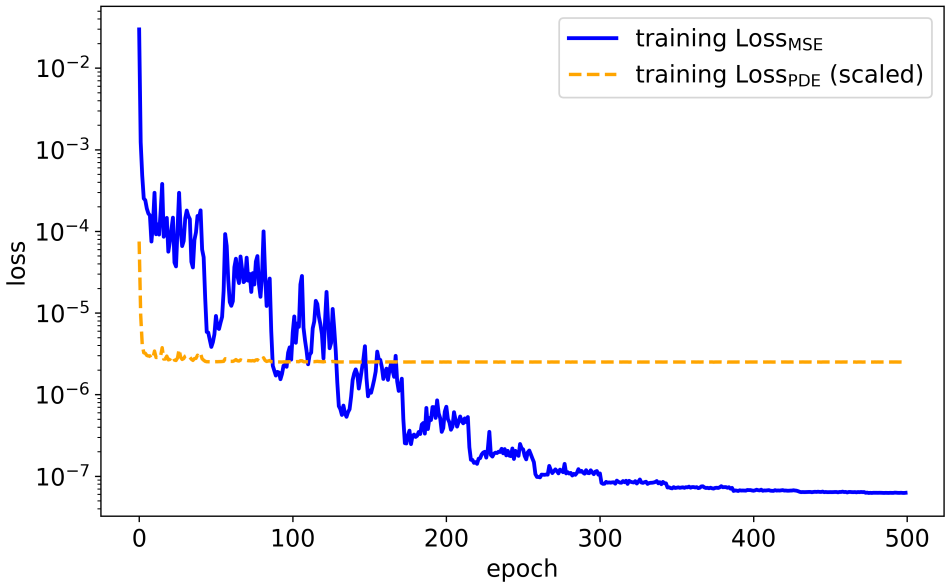


Figure 21: scaled training loss components over all 500 epochs

Table 6 further indicates that when the FNO is trained using only the physics loss term, the size of the training dataset does not have a major impact on the model's performance. This training setup is called unsupervised learning, as the model is trained solely on unlabelled data [11]. To illustrate the results of this approach, table 7 presents the true and predicted parameters for two samples from the test dataset, using the model trained with only 100 unlabelled training samples. The first two rows present the true and predicted parameters for the test sample achieving the lowest relative ℓ^2 error, while the last two rows show the parameters of the sample with the highest error. Moreover, figure 22 and figure 23 provide a visual comparison between the true solution and the solution

obtained by solving the advection-diffusion equation, where the coefficients in the PDE are given by the parameters predicted by the FNO. A significant discrepancy is visible for the worst-case sample, but the model performs well for the best-case sample.

	V_1	V_2	D_1	D_2
true parameters	0.9507	0.3897	0.1376	0.0690
prediction	0.9469	0.3852	0.1509	0.0764
true parameters	0.2952	0.3058	0.0534	0.0106
prediction	0.4903	0.6246	0.0987	0.0513

Table 7: true and predicted parameters for two test samples

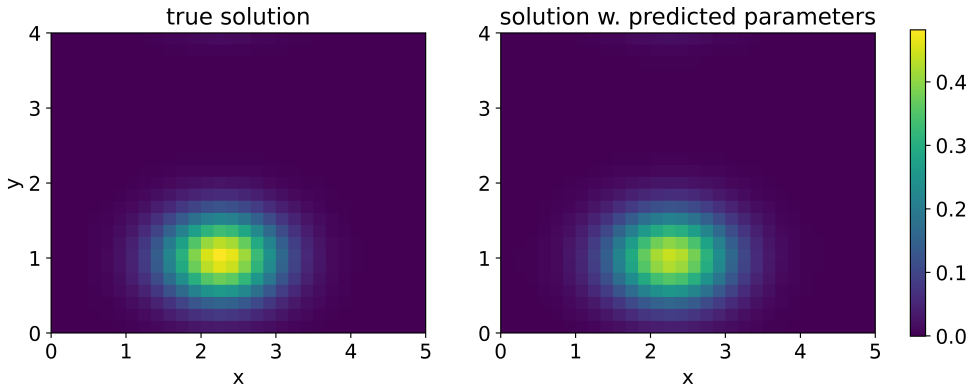


Figure 22: comparison of solutions for test sample with lowest error at final time t=1

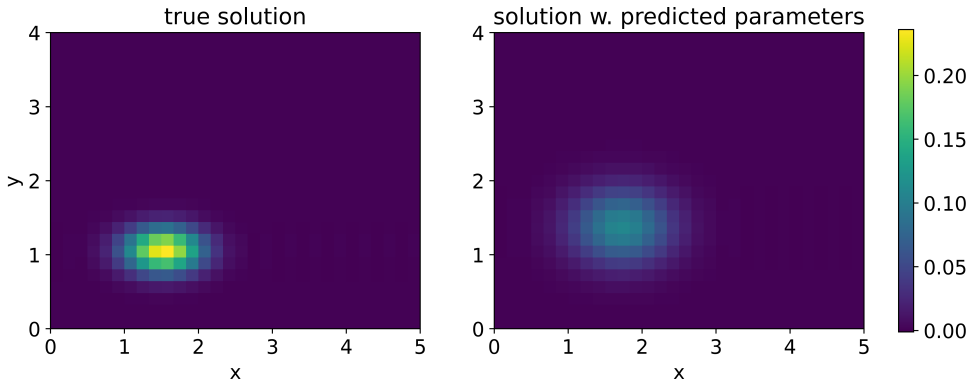


Figure 23: comparison of solutions for test sample with highest error at final time t=1

4.1.4 Experimental Evaluation of Model Limitations

In this subsection, we revisit the model trained on the complete dataset using the combined loss function. Previously, the FNO was evaluated on test samples whose parameters are within the same ranges as those used during training. In the following example, the model is tested on test samples with parameters that lie outside the training range.

Example 4.1. In this experiment, the model is evaluated with velocities V_1, V_2 that lie outside the range of the training dataset. Specifically, the model was trained on velocities within the interval $[0.25, 1.25)$ and is here tested on velocities in the interval $[1.25, 1.5)$ as well as $[0.1, 0.25)$. The results for two test samples with higher velocities are presented in table 8, and those for two samples with lower velocities are shown in table 9. In both tables, the first two rows present the true and predicted parameters for the test sample achieving the lowest relative ℓ^2 error, while the last two rows show the parameters of the sample with the highest error. These results demonstrate that the model has limited extrapolation capabilities, as it is not able to correctly predict values outside the training range.

	V_1	V_2	D_1	D_2
true parameters	1.4774	1.2594	0.1285	0.0713
prediction	1.2526	1.2319	0.1160	0.0823
true parameters	1.2629	1.4989	0.0865	0.0584
prediction	1.0584	1.0950	0.1029	0.0595

Table 8: parameters for two test samples with velocities higher than the training velocities

	V_1	V_2	D_1	D_2
true parameters	0.2199	0.1659	0.0982	0.1041
prediction	0.2633	0.2160	0.1328	0.1140
true parameters	0.1118	0.1176	0.0837	0.0483
prediction	0.2302	0.2202	0.1473	0.0454

Table 9: parameters for two test samples with velocities lower than the training velocities

4.2 Two-Component Model in 2d

4.2.1 Implementation Details and Experimental Setup

In the context of the two-component model, the inverse problem is to identify the velocities $v^{1,x}(x,y)$, $v^{1,y}(x,y)$, $v^{2,x}(x,y)$, $v^{2,y}(x,y)$ and the conversion rate $c(x,y)$, given the PDEs as well as the solutions $u^1(x,y,t)$ and $u^2(x,y,t)$. Since the velocities and the conversion rate are spatially dependent, the FNO is required to predict their values at each grid point. In contrast, for the advection-diffusion equation, the model only needs to predict four constant values. For the two-component model, the FNO has 34 input channels. The first 16 represent the solution u^1, the next 16 represent the solution u^2, each at one time step, and the last 2 channels represent the coordinates x, y. The 5 output channels correspond to the predicted velocity fields and the prediction of the conversion rate.

The model is trained for 750 epochs with an initial learning rate of 0.01, which is halved every 100 epochs. The loss function is given by

$$\text{Loss} = \text{Loss}_{\text{MSE}} + 0.001 \cdot \text{Loss}_{\text{PDE}}. \tag{4.2}$$

The training and validation losses are depicted in figure 24 with an logarithmically scaled y-axis.

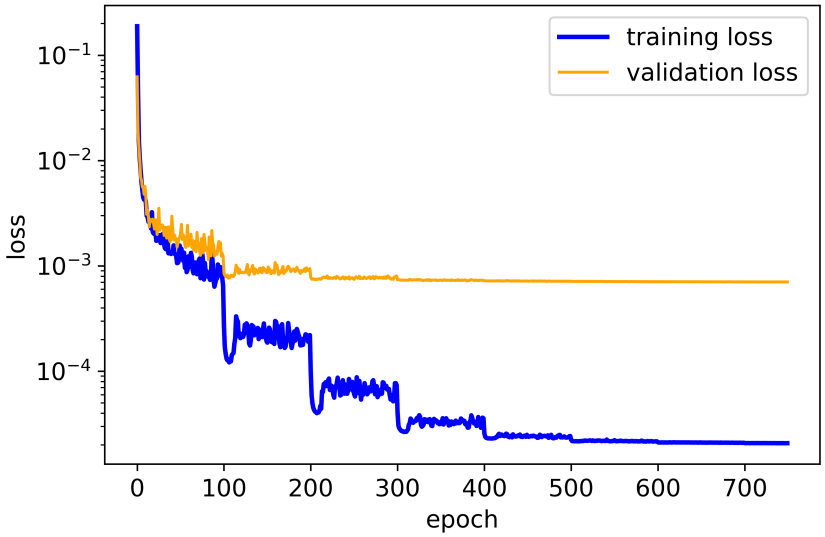


Figure 24: training and validation loss over all 750 epochs

4.2.2 Results

After the training process is completed, the model achieves an average relative ℓ^2 error of 0.01867 on the test set. Figures 25, 26 and 28, 29 show the predicted parameter fields, along with the corresponding absolute error, for the test samples with the lowest and highest relative ℓ^2 errors, respectively. To illustrate the impact of the errors in the parameters on the PDE solutions, figure 27 and figure 30 present the solutions computed using the predicted parameters and the absolute error compared to the true solution for the two test samples.

The FNO provides accurate predictions of the velocity fields, it effectively captures the underlying dynamics and recognizes that the velocities remain constant along one spatial dimension. Furthermore, it reliably identifies the conversion region, although in the worst-case example, the predicted maximum conversion rate κ is not entirely accurate.

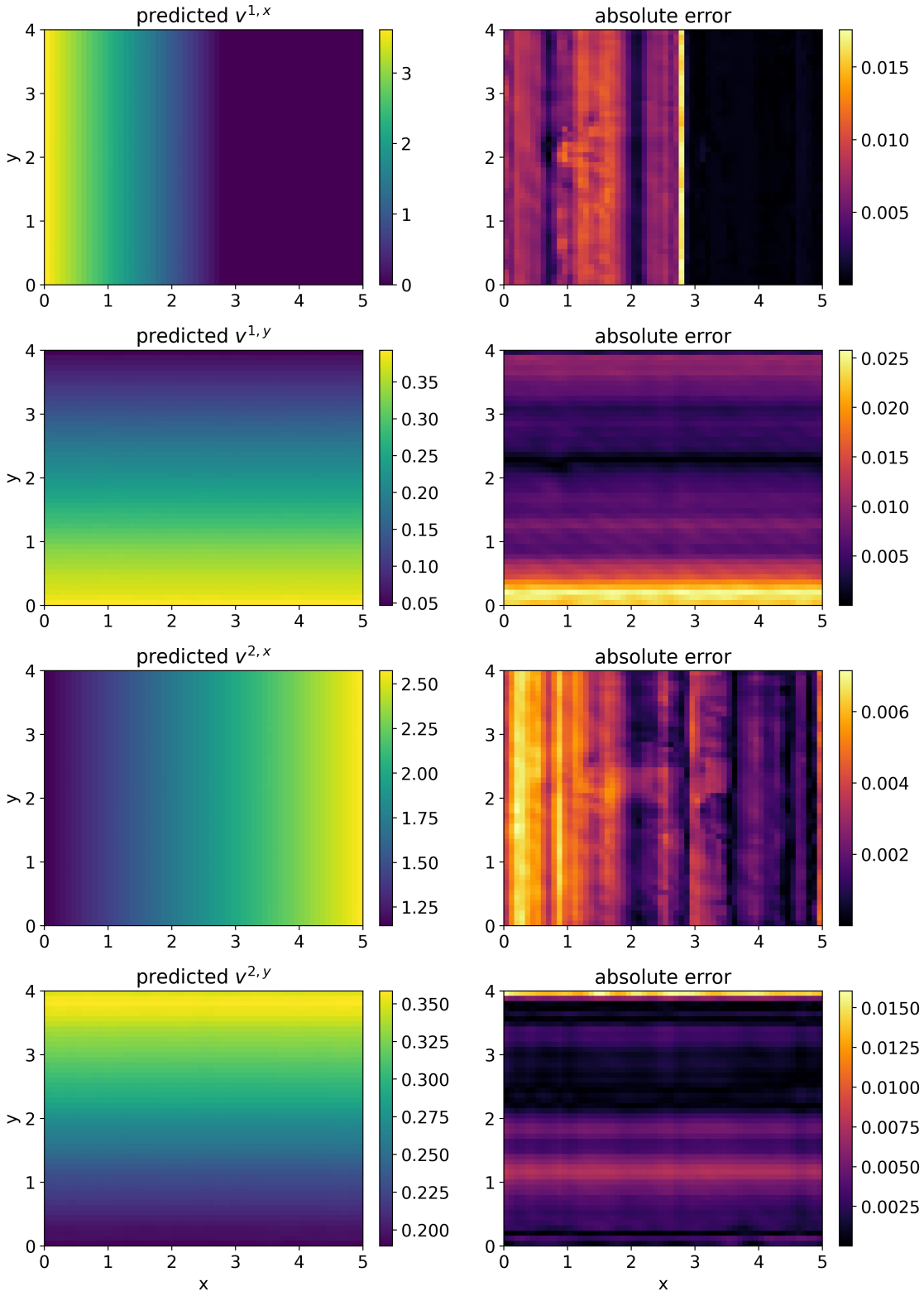


Figure 25: predicted velocities and absolute error for test sample with lowest error

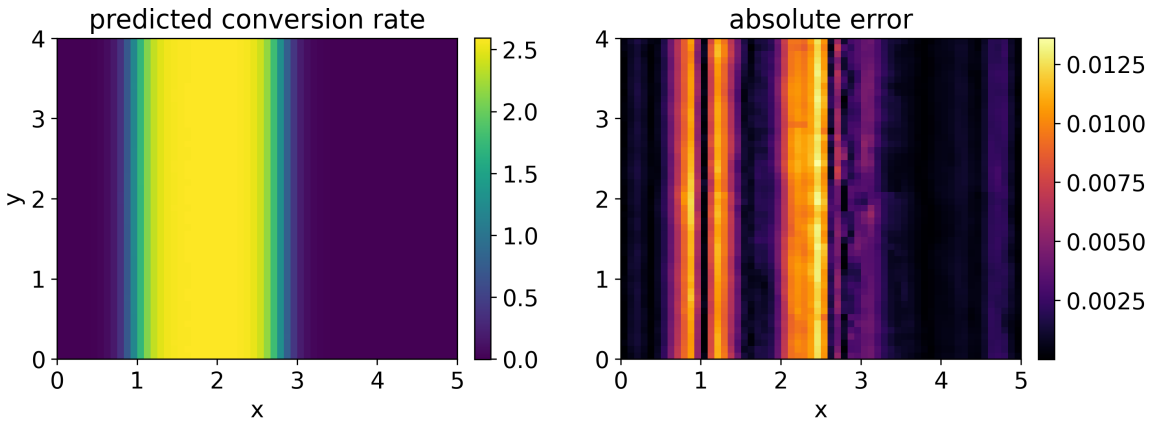


Figure 26: predicted conversion rate and absolute error for test sample with lowest error

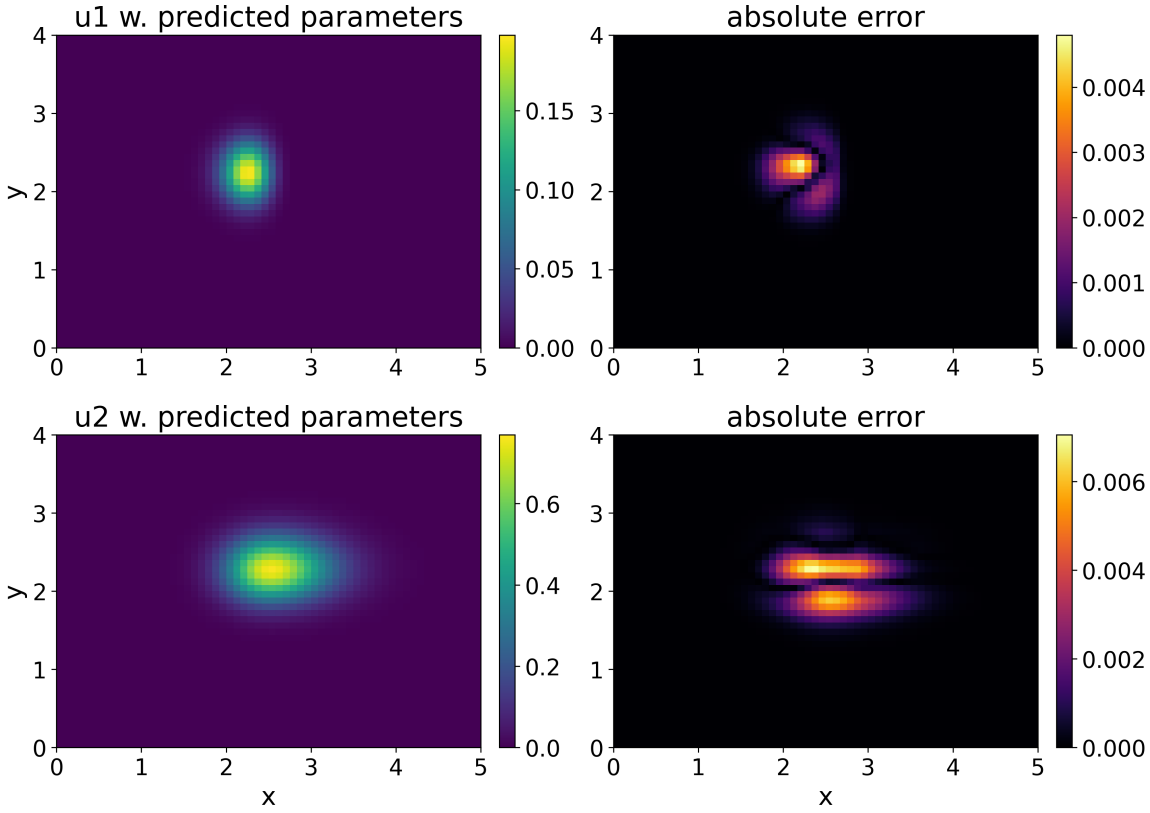


Figure 27: comparison of solutions for test sample with lowest error at final time $t = 1$

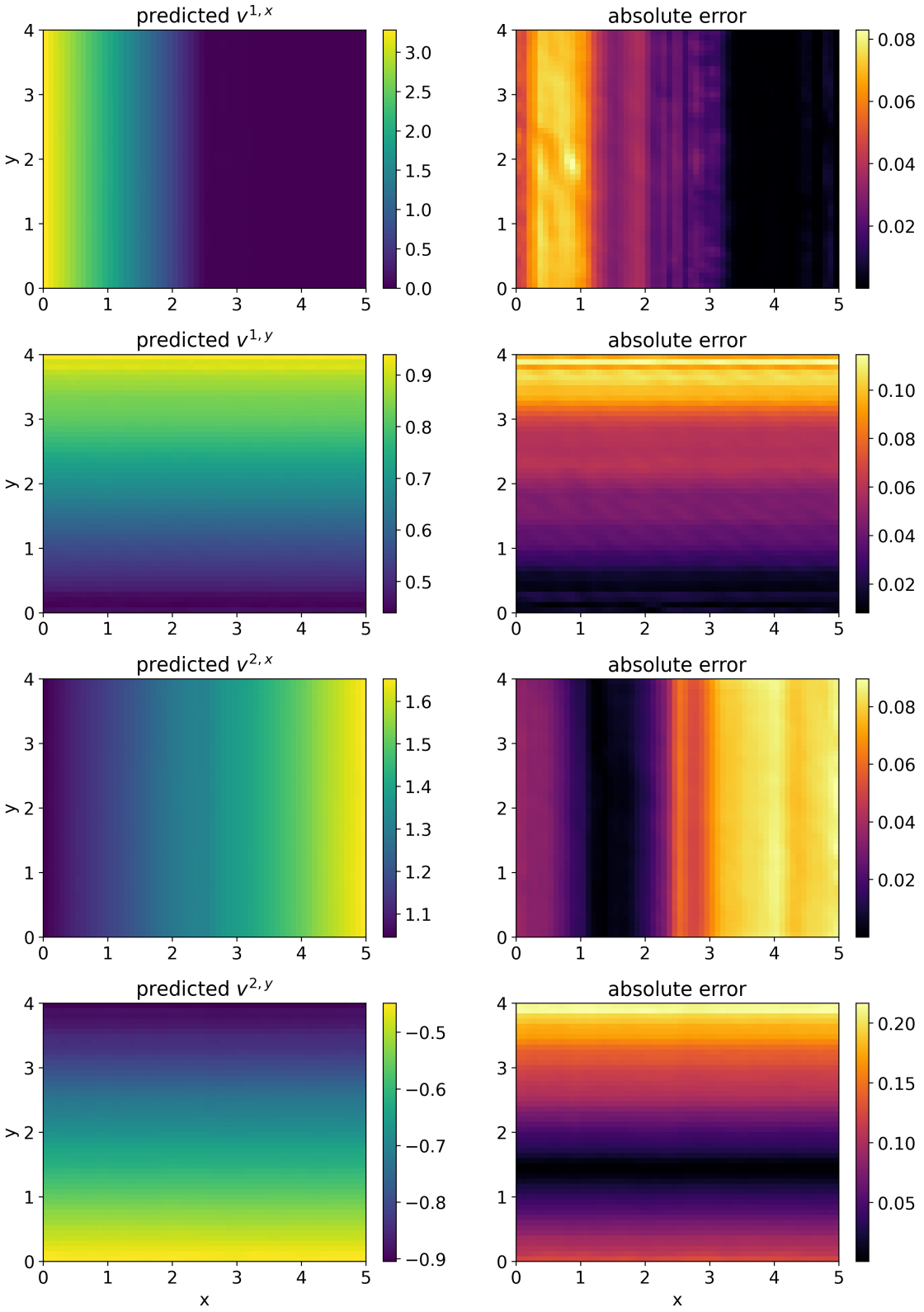


Figure 28: predicted velocities and absolute error for test sample with highest error

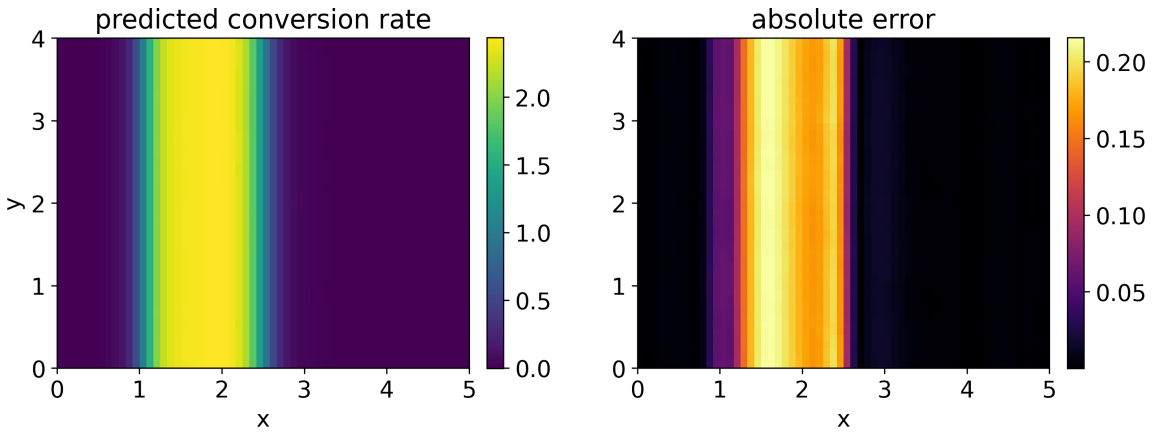


Figure 29: predicted conversion rate and absolute error for test sample with highest error

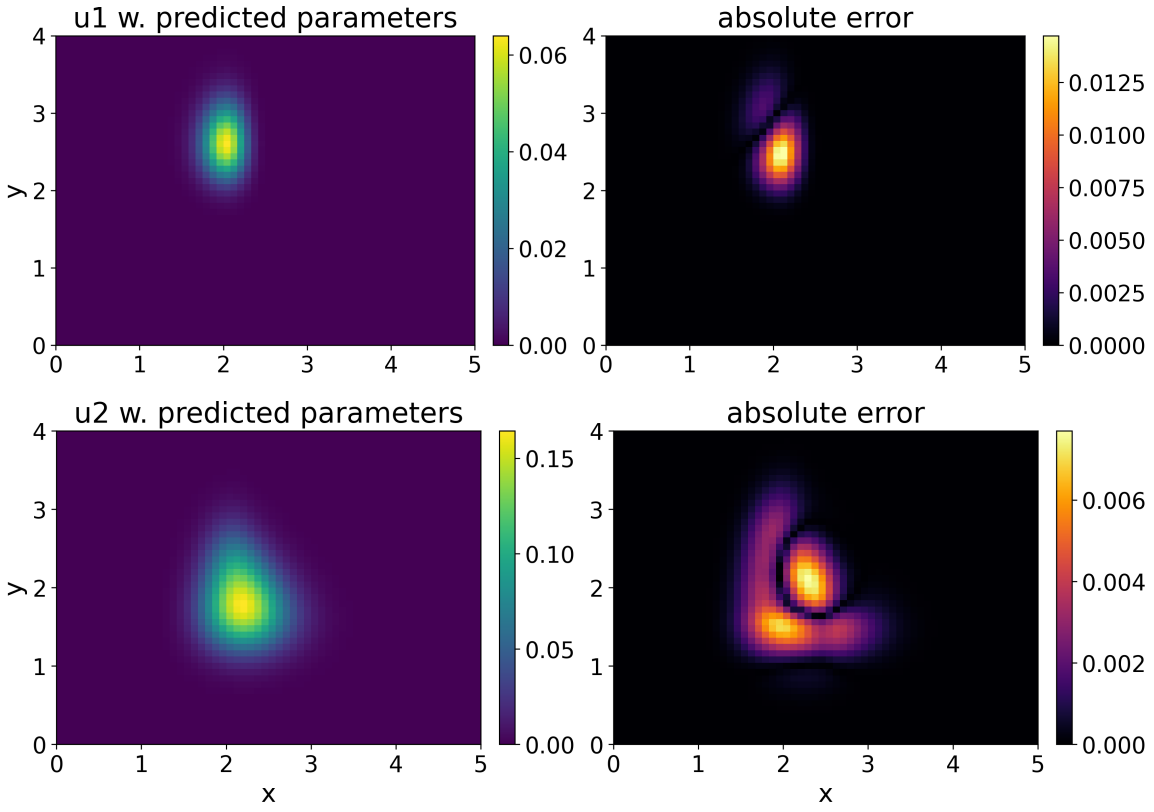


Figure 30: comparison of solutions for test sample with highest error at final time $t = 1$

4.2.3 Experimental Evaluation of Model Limitations

In this chapter, numerical experiments are performed to investigate potential limitations of the FNO trained to solve the inverse problem for the two-component model.

Example 4.2. In this example, the velocity $v^{1,x}$ is set to zero for small y-values. This modification is restricted to a region where the concentrations u^1 and u^2 are consistently zero, and therefore it does not affect the PDE solutions. Figure 31 presents the true solutions, as well as the solutions computed with the predicted parameters for this sample at the final time step. The resulting solutions are consistent with those observed in previous examples, confirming that the local modification of the velocity field does not cause changes in the PDE solutions.

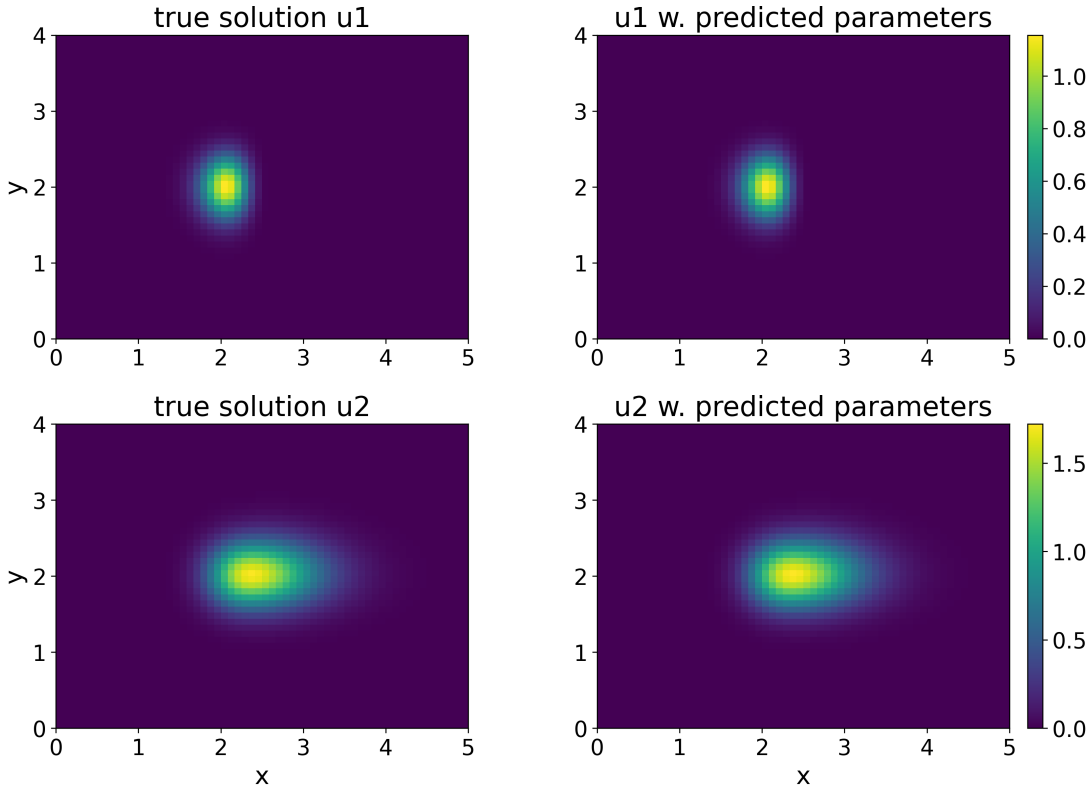


Figure 31: comparison of solutions at the final time $t = 1$

The true and predicted parameters for this example are shown in figure 32 and figure 33. The FNO is not able to reflect the modification in the velocity component $v^{1,x}$. However, since this change does not affect the PDE solutions, the remaining parameters are accurately predicted.

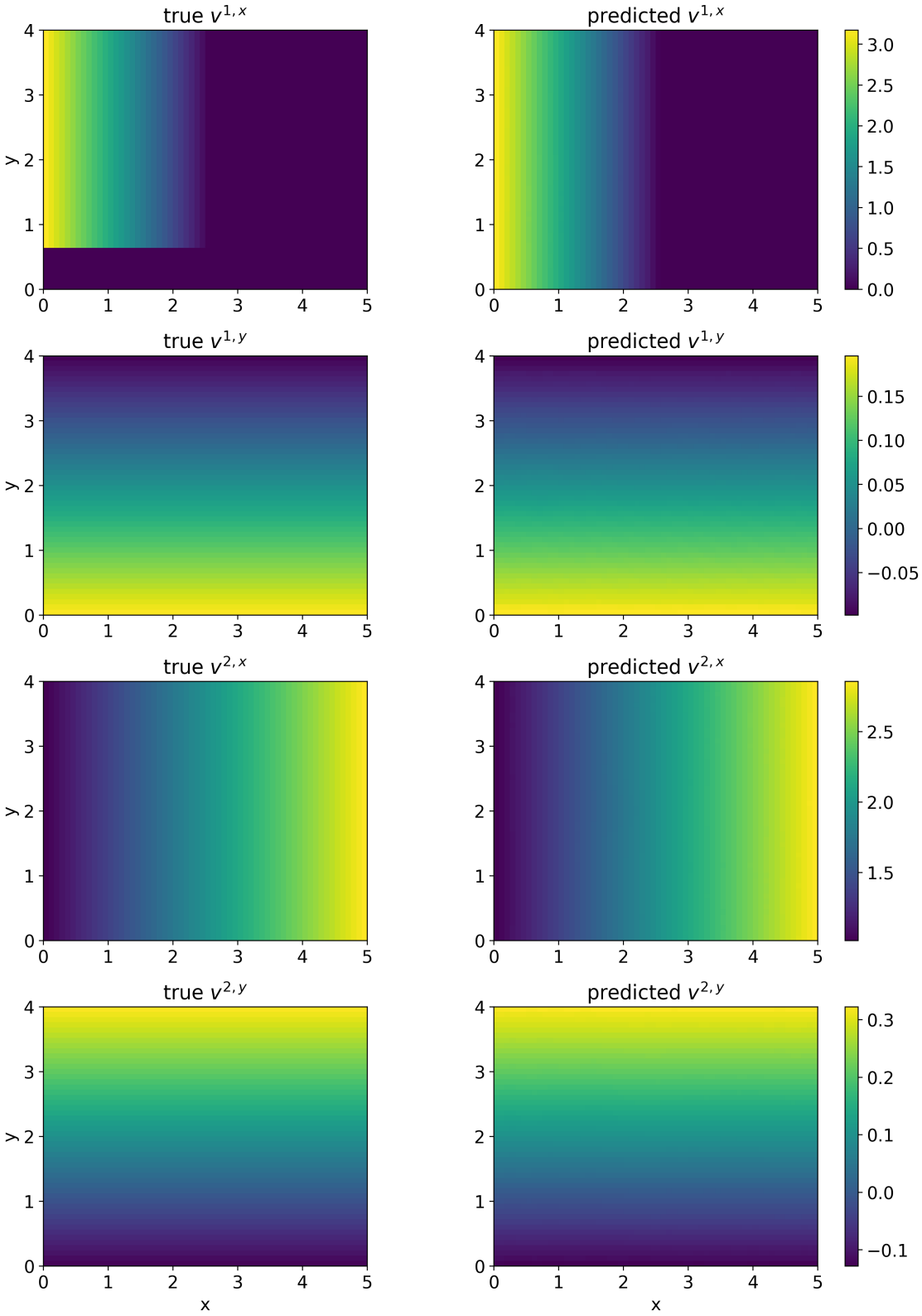


Figure 32: comparison of true and predicted velocities for example 4.2

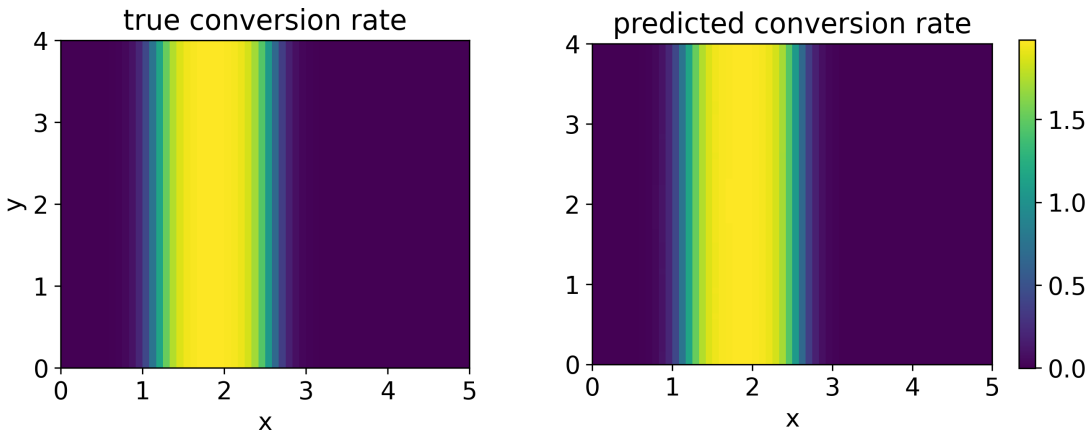


Figure 33: comparison of true and predicted conversion rate for example 4.2

Example 4.3. In this example, the velocity component $v^{1,x}$ is set to zero over a larger region. Unlike the previous case, this modification now affects the PDE solutions. The true solutions and the solutions obtained with the predicted parameters are presented in figure 34. Compared to the previous examples, the true solutions show clear differences, confirming that the alteration in the velocity field has an impact on the PDE solutions. In contrast, the solutions obtained with the predicted parameters look similar to previous examples, indicating that the FNO fails to capture this modification.

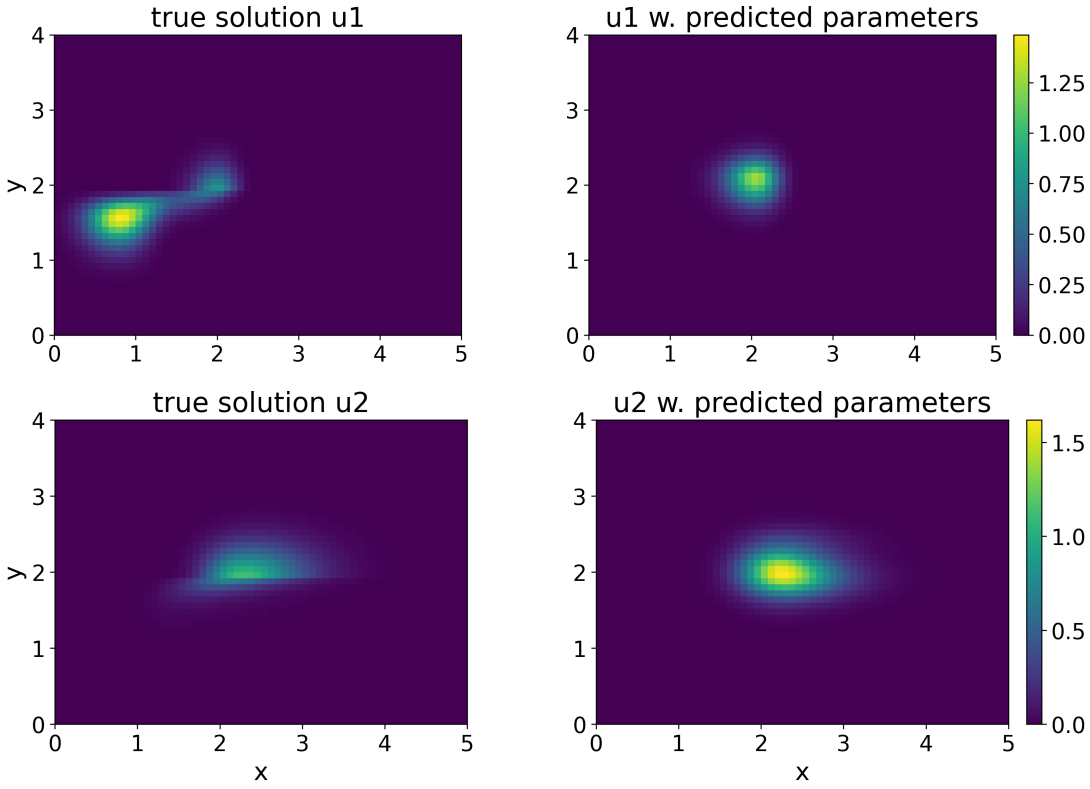


Figure 34: comparison of solutions at the final time $t = 1$

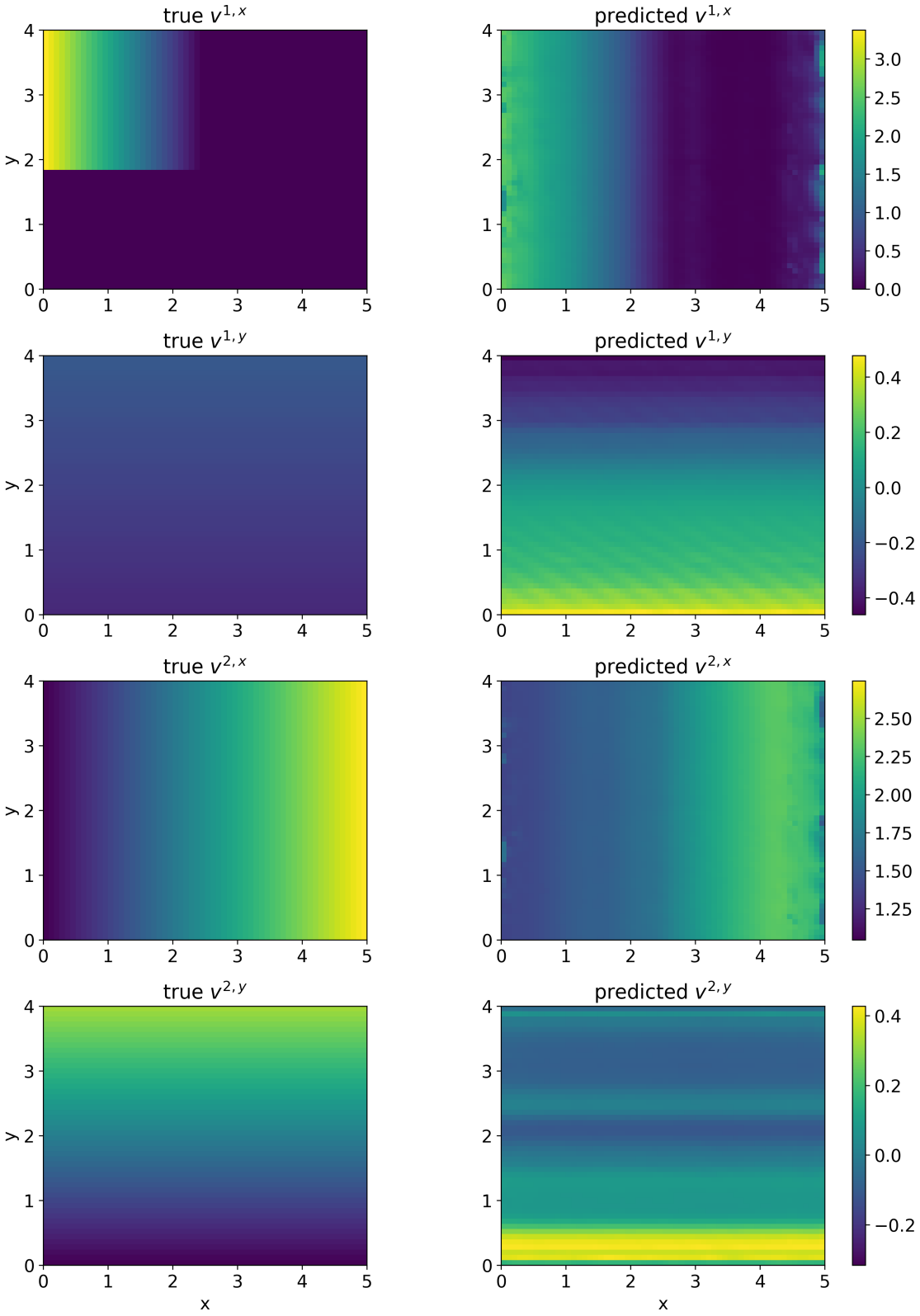


Figure 35: comparison of true and predicted velocities for example 4.3

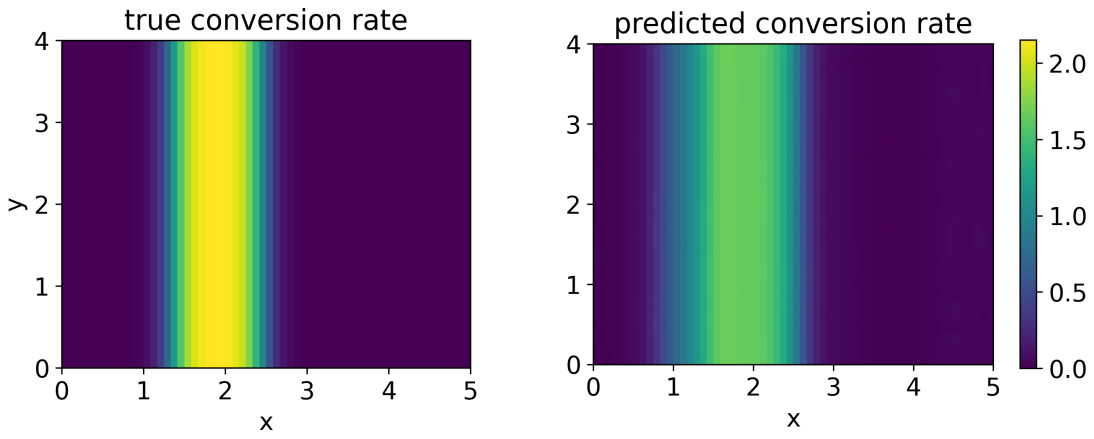


Figure 36: comparison of true and predicted conversion rate for example 4.3

Figure 35 and figure 36 illustrate that, in this example, the FNO is not able to provide accurate predictions for any of the parameter fields.

Example 4.4. For this experiment, a new FNO is trained. Whereas the solutions u^1 and u^2 were previously treated separately, the model is now trained on their sum $u(x, y) = u^1(x, y) + u^2(x, y)$. Consequently, the number of input channels is reduced to 18, representing the summed concentration $u(x, y)$ at each time step and the coordinates x, y. The output channels remain unchanged. Since the network only has access to the summed concentration, for which no governing PDE is available, the term Loss_{PDE} can not be computed in this setting. Hence, the loss function reduces to the mean squared error loss, Loss_{MSE}, while all other hyperparameters are not modified.

On the test set, the trained model achieves results comparable to the previously evaluated FNO, which handles u^1 and u^2 separately. To access whether the new model can perform comprehensive generalization, it is tested on advection-diffusion data. For consistency with the training resolution, the test sample is generated with randomly chosen parameters and a spatial resolution of 60×50. The advection-diffusion PDE parameters are provided in table 10, and the FNO's predictions for the two-component model parameters are shown in figure 37. While a direct comparison of the parameters is not meaningful due to the different PDEs, the solutions can be compared. Figure 38 presents the true advection-diffusion solution along with the summed two-component model solution computed using the predicted parameters. The solutions are depicted at the initial, one intermediate and the final time step. The clearly visible discrepancies in concentration levels, shapes and locations indicate that the FNO is incapable of generalizing to a new PDE.

V_1	V_2	D_1	D_2
0.3017	1.2458	0.0865	0.0584

Table 10: parameters for example 4.4

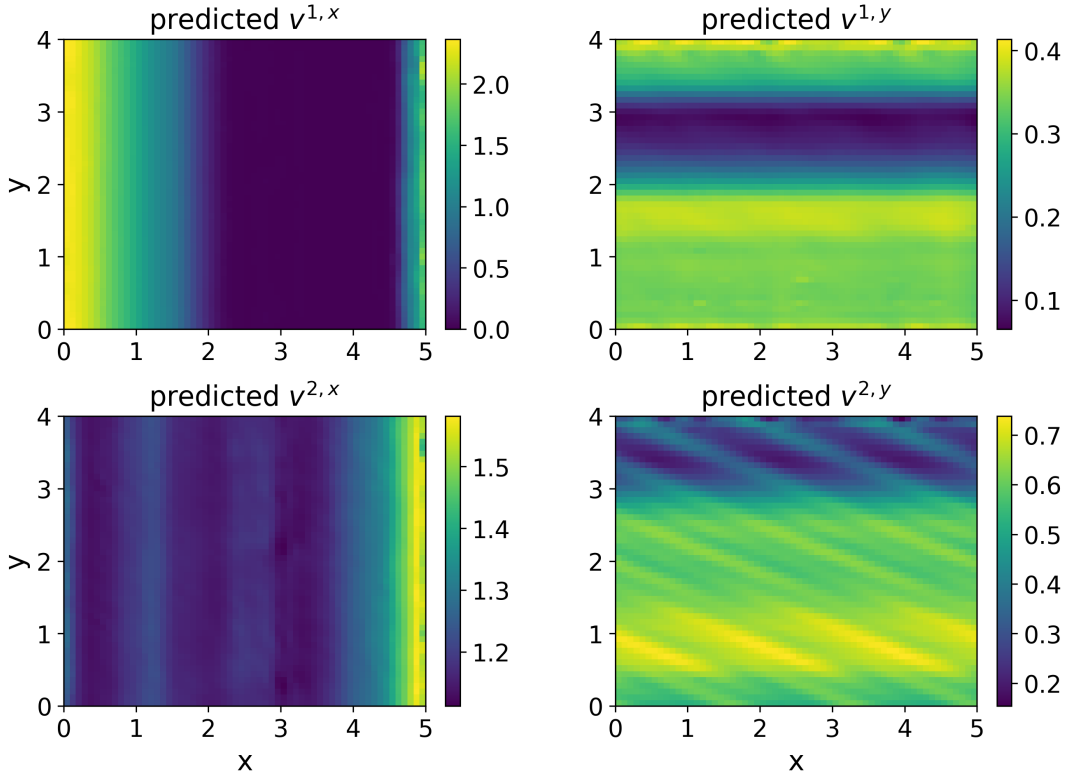


Figure 37: predicted parameters for example 4.4

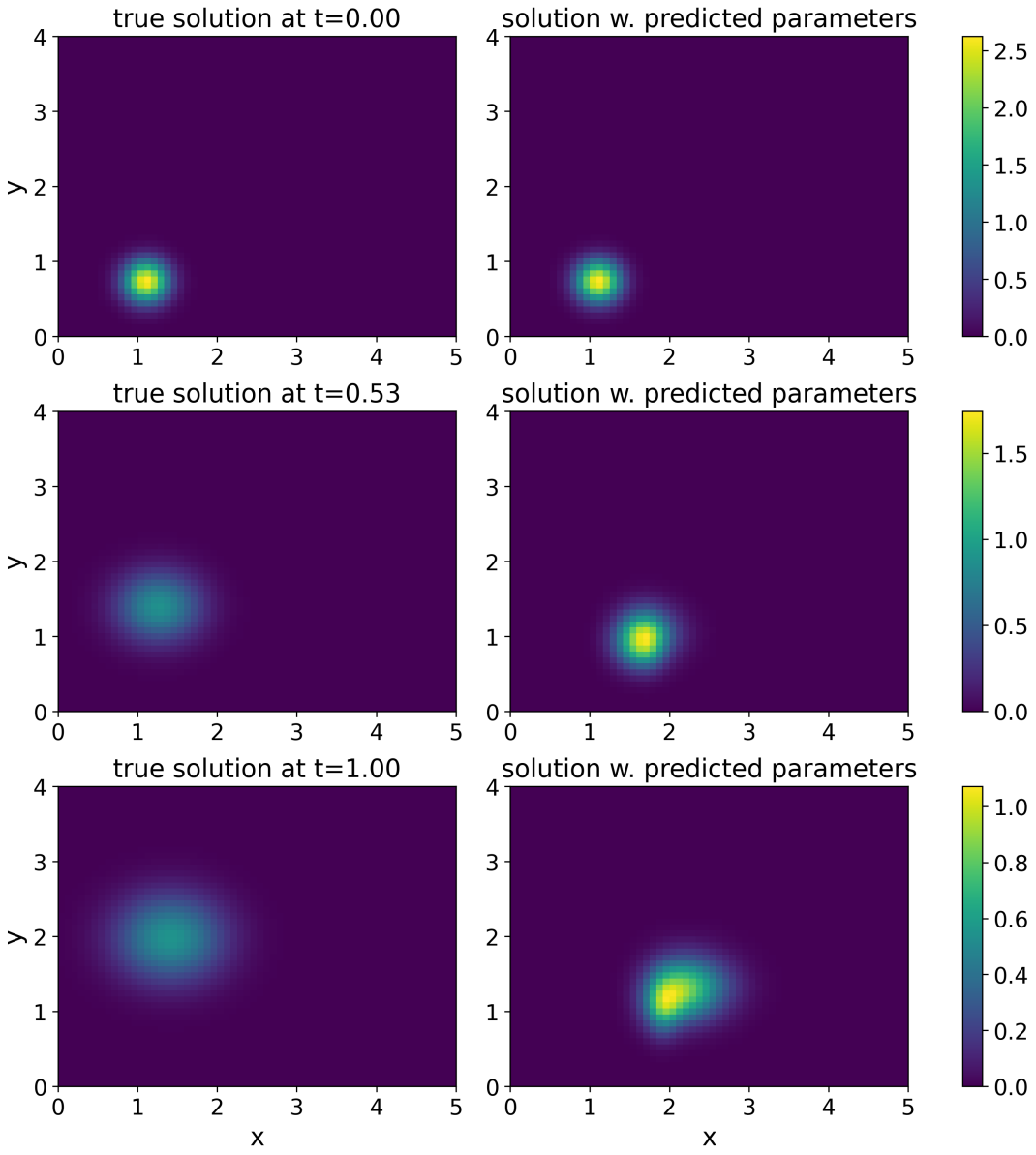


Figure 38: comparison of solutions for example 4.4

4.3 Two-Component Model in 3d

For the two-component model in three spatial dimensions, the transition from the forward to the inverse problem also corresponds to interchanging the input and output channels. Unless specified otherwise, the hyperparameters are chosen as for the forward problem. Compared to the FNO trained for the inverse problem in two spatial dimensions, the number of input channels remains unchanged, whereas the number of output channels increases by two, corresponding to the additional velocity components $v^{1,z}$ and $v^{2,z}$. The resulting model, as in the forward case, has a total of 3.54 million trainable parameters.

The model is trained for 500 epochs with an initial learning rate of 0.001, which is halved every 100 epochs. The same loss function as in the two-dimensional case is applied. In contrast to the forward problem, 5000 training samples are used here in order to achieve a higher accuracy.

The trained model achieves a relative ℓ^2 error of 0.04966 on the test set. Figure 39 and figure 40 present a visual comparison of the true and predicted parameters for the test samples with the lowest and highest errors, respectively. Since each parameter varies only along a single spatial dimension, it has been averaged over the other two dimensions to simplify the visualization.

In the best-case sample, although the predictions are not fully accurate, the model correctly captures the dynamics, i.e. whether the velocity components increase or decrease. In the worst-case sample, the model also mostly reproduces the dynamics accurately, except for the component $v^{2,z}$.

To evaluate how well the model captures that each parameter varies along only one dimension, table 11 presents the standard deviations of each parameter along the two dimensions in which it should remain constant. The values are consistently small for both test samples, indicating that the model efficiently learns that each parameter depends only on one spatial coordinate. Ideally, these standard deviations would be zero.

	$v^{1,x}$	$v^{1,y}$	$v^{1,z}$	$v^{2,x}$	$v^{2,y}$	$v^{2,z}$	c
best-case sample	0.0033	0.0119	0.0113	0.0081	0.0096	0.0165	0.0067
worst-case sample	0.0060	0.0260	0.0279	0.0155	0.0252	0.0376	0.0110

Table 11: standard deviations for each parameter for the two test samples

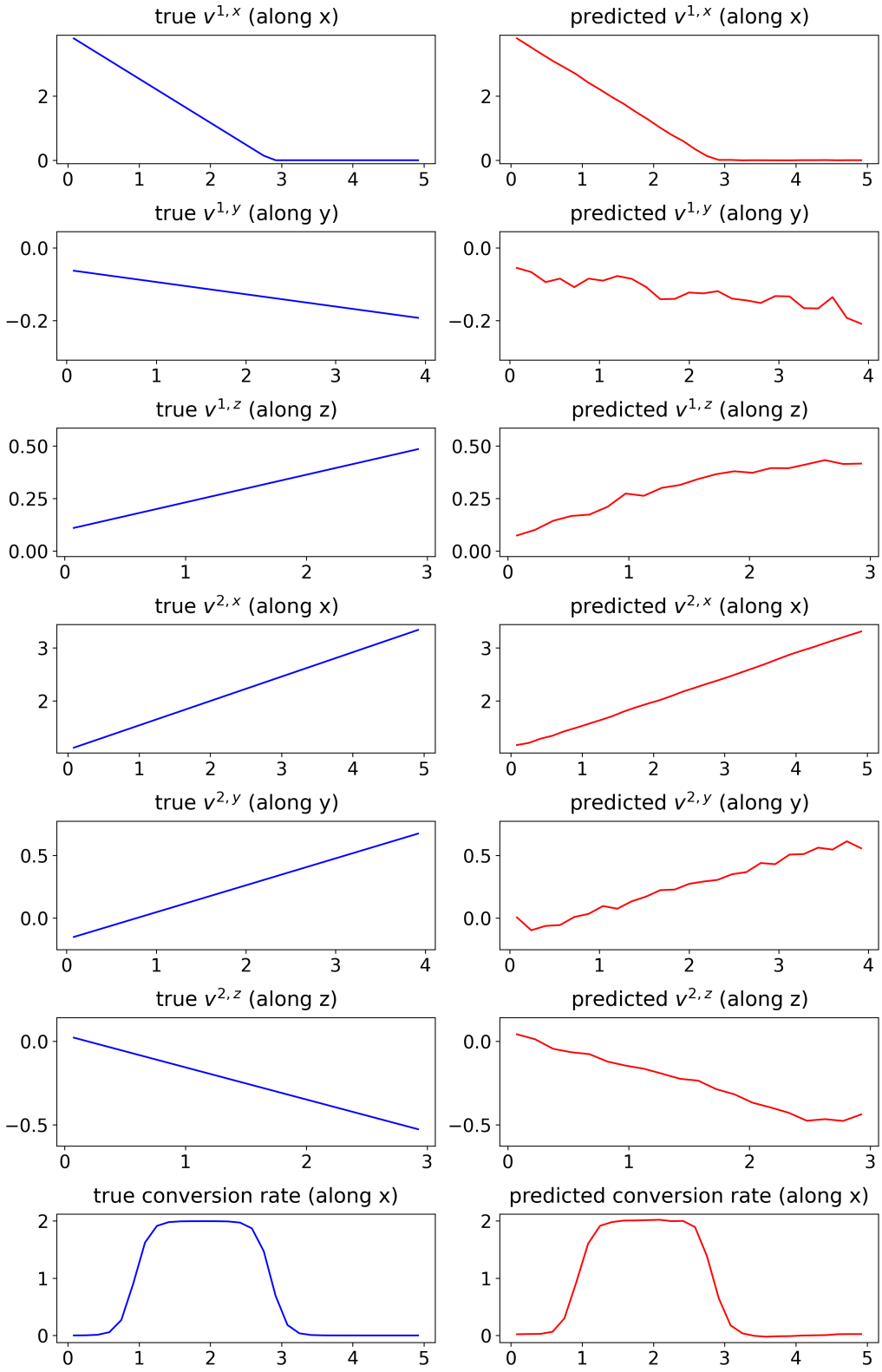


Figure 39: comparison of parameters for the test sample with the lowest error

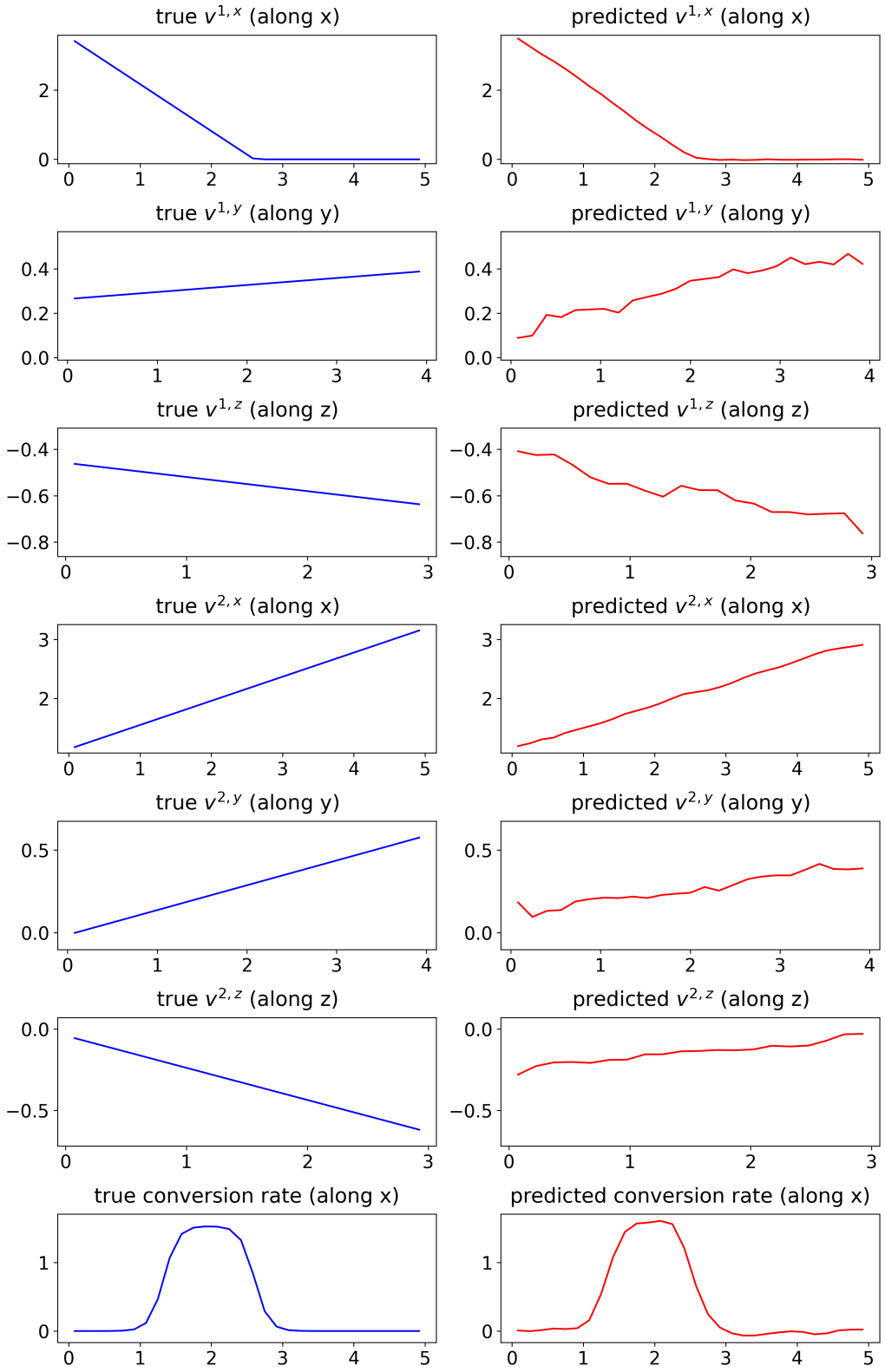


Figure 40: comparison of parameters for the test sample with the highest error

5 Conclusion

This thesis investigates the capabilities and limitations of Fourier neural operators through numerical experiments on two different partial differential equations. For both models, the Fourier neural operator successfully solves the forward as well as the inverse problem with high accuracy. Furthermore, the FNO is able to generalize across spatial resolutions different from those used during training without noteworthy loss of accuracy. However, certain limitations were identified: the FNO does not generalize well to parameters outside the range of the training data, nor does it generalize across different types of PDEs. The impact of incorporating a physics-informed loss term into the network is evaluated for the inverse advection-diffusion problem, showing a benefit only when the training set is very small. In general, future improvements, such as hyperparameter optimization or increasing the training dataset size, may further improve the predictive accuracy.

In conclusion, FNOs require a sufficiently large training dataset and computational resources for training. Once trained, they quickly provide accurate solutions for new parameter sets. Their good performance in solving inverse problems highlights the potential of FNOs and similar machine learning models for applications in medicine, such as reconstructing tumor perfusion from ultrasound data.

References

[1] Junghwan Cho, Kyewook Lee, Ellie Shin, Garry Choy, and Synho Do. *How Much Data Is Needed to Train a Medical Image Deep Learning System to Achieve Necessary High Accuracy?* 2016. DOI: 10.48550/arXiv.1511.06348.

[2] W.T. Cochran, J.W. Cooley, D.L. Favin, H.D. Helms, R.A. Kaenel, W.W. Lang, G.C. Maling, D.E. Nelson, C.M. Rader, and P.D. Welch. "What Is the Fast Fourier Transform?" In: *Proceedings of the IEEE* 55.10 (1967), pp. 1664–1674. DOI: 10.1109/PROC.1967.5957.

[3] David L. Colton, Richard E. Ewing, and William Rundell. *Inverse Problems in Partial Differential Equations*. Siam, 1990. ISBN: 978-0-89871-252-0.

[4] Sophie Externbrink. "Two-Component Model for Tracer Simulation". In: *TUHH Universitätsbibliothek* (2022). DOI: 10.15480/882.4765.

[5] Robert Eymard, Thierry Gallouët, and Raphaèle Herbin. "Finite Volume Methods". In: *Solution of Equation in Rn (Part 3), Techniques of Scientific Computing (Part 3)*. Vol. 7. Handbook of Numerical Analysis. Elsevier, 2000, pp. 713–1020. DOI: 10.1016/S1570-8659(00)07005-8.

[6] Gerald B. Folland. *Real Analysis: Modern Techniques and Their Applications*. 2nd edition. John Wiley & Sons, 1999. ISBN: 978-0-471-31716-6.

[7] Wenhan Gao, Ruichen Xu, Yuefan Deng, and Yi Liu. "Discretization-Invariance? On the Discretization Mismatch Errors in Neural Operators". In: *The Thirteenth International Conference on Learning Representations*. 2025.

[8] Ernst Hairer and Gerhard Wanner. "Stiff Differential Equations Solved by Radau Methods". In: *Journal of Computational and Applied Mathematics* 111.1-2 (1999), pp. 93–111. DOI: 10.1016/S0377-0427(99)00134-X.

[9] Alan C. Hindmarsh. "Odepack, a Systemized Collection of Ode Solvers". In: *Scientific Computing* (1982).

[10] Dimitre Hristov, Lauri Mustonen, Rie von Eyben, Sebastian Götschel, Michael Minion, and Ahmed El Kaffas. "Dynamic Contrast-Enhanced Ultrasound Modeling of an Analog to Pseudo-Diffusivity in Intravoxel Incoherent Motion Magnetic Resonance Imaging". In: *IEEE Transactions on Medical Imaging* 41.12 (2022), pp. 3824–3834. DOI: 10.1109/TMI.2022.3197363.

[11] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. "Physics-Informed Machine Learning". In: *Nature Reviews Physics* 3.6 (2021), pp. 422–440. DOI: 10.1038/s42254-021-00314-5.

[12] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. DOI: 10.48550/arXiv.1412.6980.

[13] Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. "Neural Operator: Learning Maps Between Function Spaces With Applications to PDEs". In: *Journal of Machine Learning Research* 24.89 (2023), pp. 1–97.

[14] Minhyeok Lee. "Mathematical Analysis and Performance Evaluation of the GELU Activation Function in Deep Learning". In: *Journal of Mathematics* (2023). DOI: 10.1155/2023/4229924.

[15] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. *Fourier Neural Operator for Parametric Partial Differential Equations*. 2021. DOI: 10.48550/arXiv.2010.08895.

[16] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. *Neural Operator: Graph Kernel Network for Partial Differential Equations*. 2020. DOI: 10.48550/arXiv.2003.03485.

[17] J. David Logan. *Applied Partial Differential Equations*. 3rd ed. Undergraduate Texts in Mathematics. Springer Cham, 2015. DOI: 10.1007/978-3-319-12493-3.

[18] Augustine T. Midhun. *A Survey on Universal Approximation Theorems*. 2024. DOI: 10.48550/arXiv.2407.12895.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. 2019. DOI: 10.48550/arXiv.1912.01703.

[20] Suhas V. Patankar. *Numerical Heat Transfer and Fluid Flow*. 1st ed. CRC Press, 1980. DOI: 10.1201/9781482234213.

[21] Alfio Quarteroni, Ricardo Sacco, and Fausto Saleri. *Numerical Mathematics*. 2nd ed. Vol. 37. Springer, 2007. ISBN: 978-3-540-34658-6.

[22] Ali Rezgui. "Précision et Convergence Des Méthodes de Volumes Finis". In: *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics* 327.4 (1998), pp. 421–426. DOI: 10.1016/S0764-4442(99)80060-3.

[23] Shawn G Rosofsky, Hani Al Majed, and E A Huerta. "Applications of Physics Informed Neural Operators". In: *Machine Learning: Science and Technology* 4.2 (2023). DOI: 10.1088/2632-2153/acd168.

[24] Jie Shen, Tao Tang, and Li-Lian Wang. *Spectral Methods: Algorithms, Analysis and Applications*. Vol. 41. Springer Series in Computational Mathematics. Springer, 2011. DOI: 10.1007/978-3-540-71041-7.

[25] Steven Sourbron. "A Tracer-Kinetic Field Theory for Medical Imaging". In: *IEEE Transactions on Medical Imaging* 33.4 (2014), pp. 935–946. DOI: 10.1109/TMI.2014.2300450.

[26] Elias M. Stein and Rami Shakarchi. *Fourier Analysis: An Introduction*. Princeton University Press, 2003. ISBN: 978-0-691-11384-5.

[27] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17.3 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

Appendix

A Forward Problem for Two-Component Model in 3d

Here, the test sample with the lowest relative ℓ^2 error is presented. The parameters, i.e. the velocities and the conversion rate, are shown in one-dimensional plots, since each varies only along one spatial dimension. The predicted solutions for u^1 and u^2, together with the corresponding absolute error compared to the true solutions, is illustrated using three representative slices in each spatial direction at the final time step.

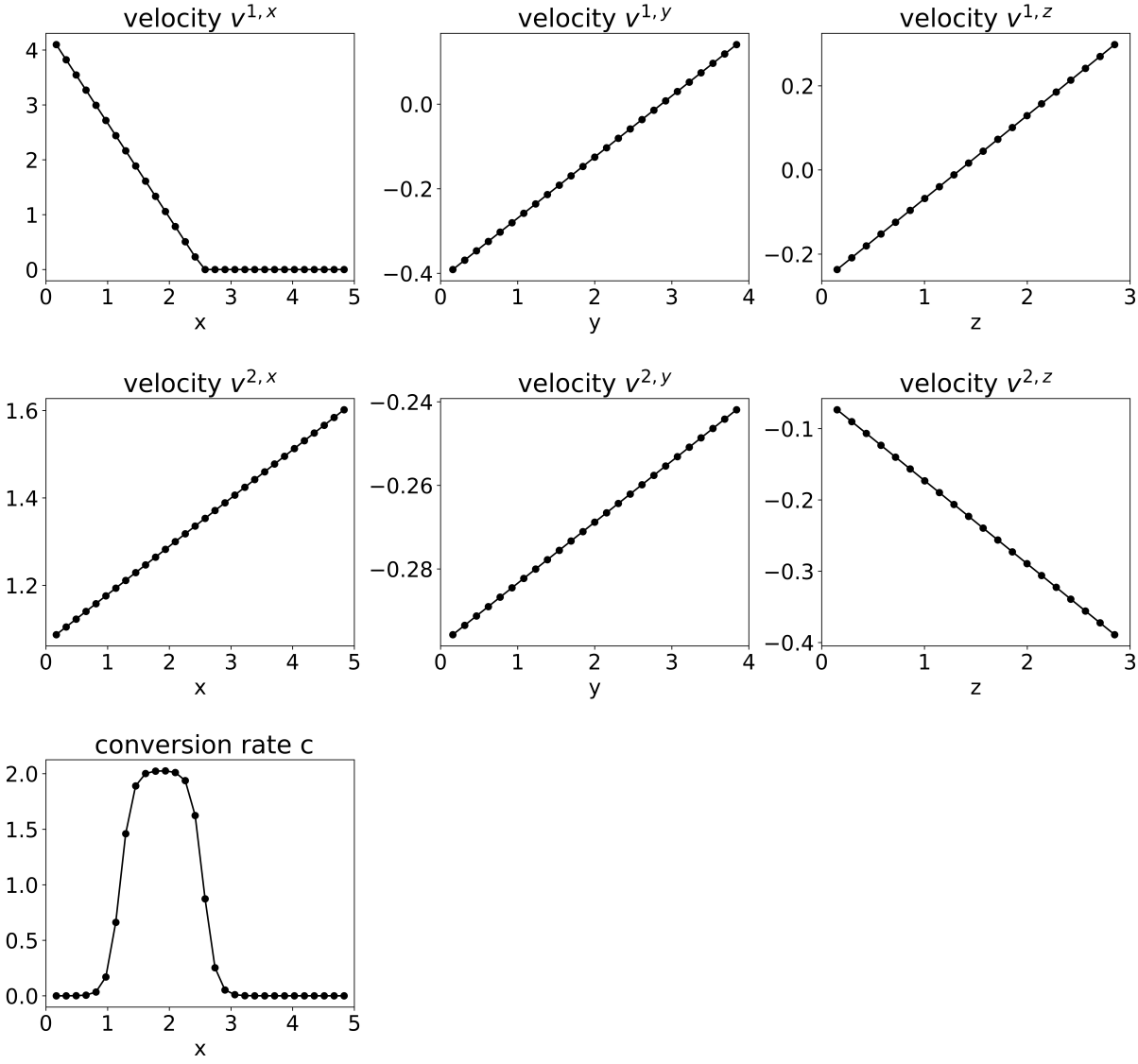


Figure 41: parameters for the test sample

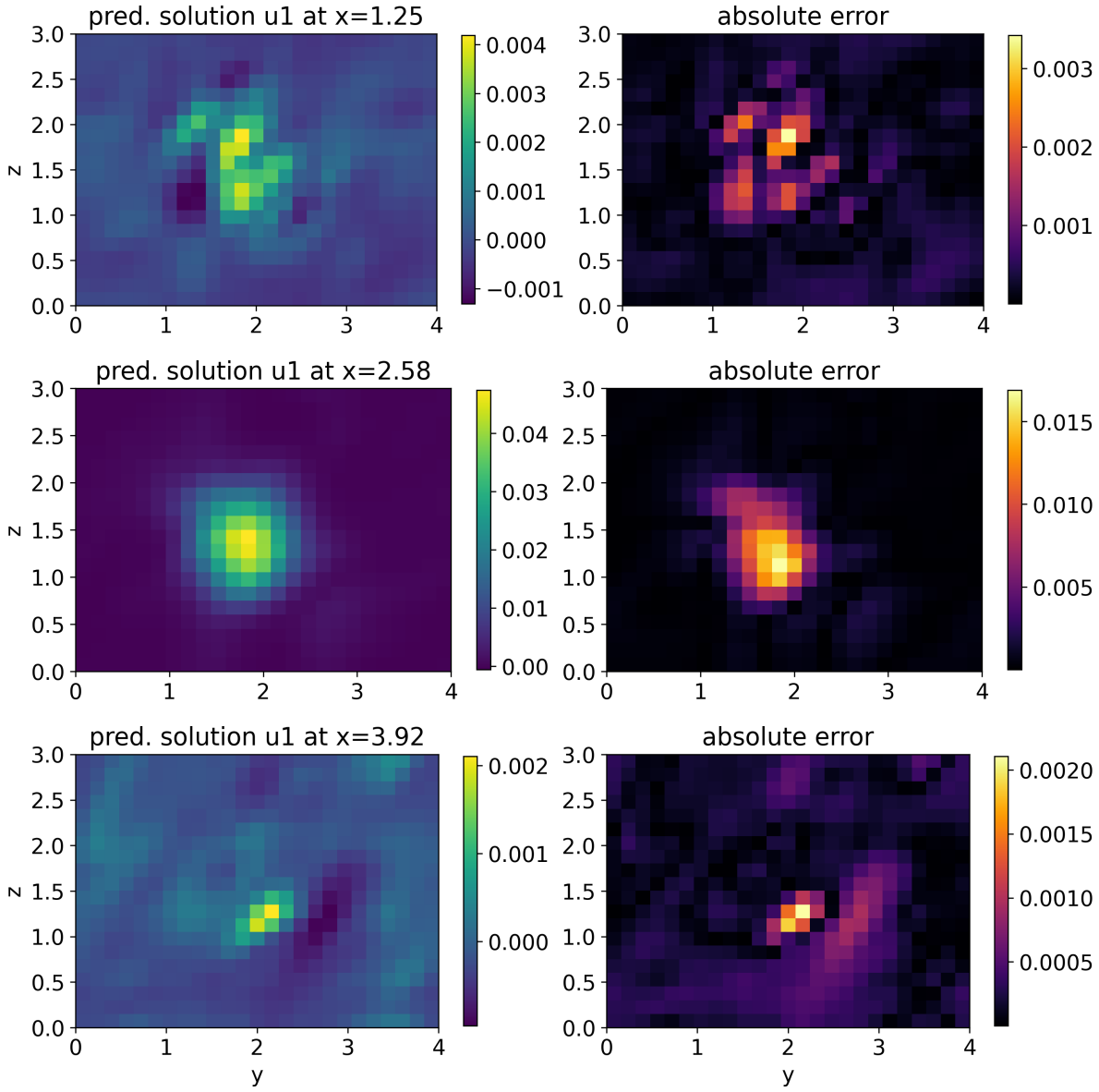


Figure 42: predicted solution u^1 and absolute error for three slices along the x-direction

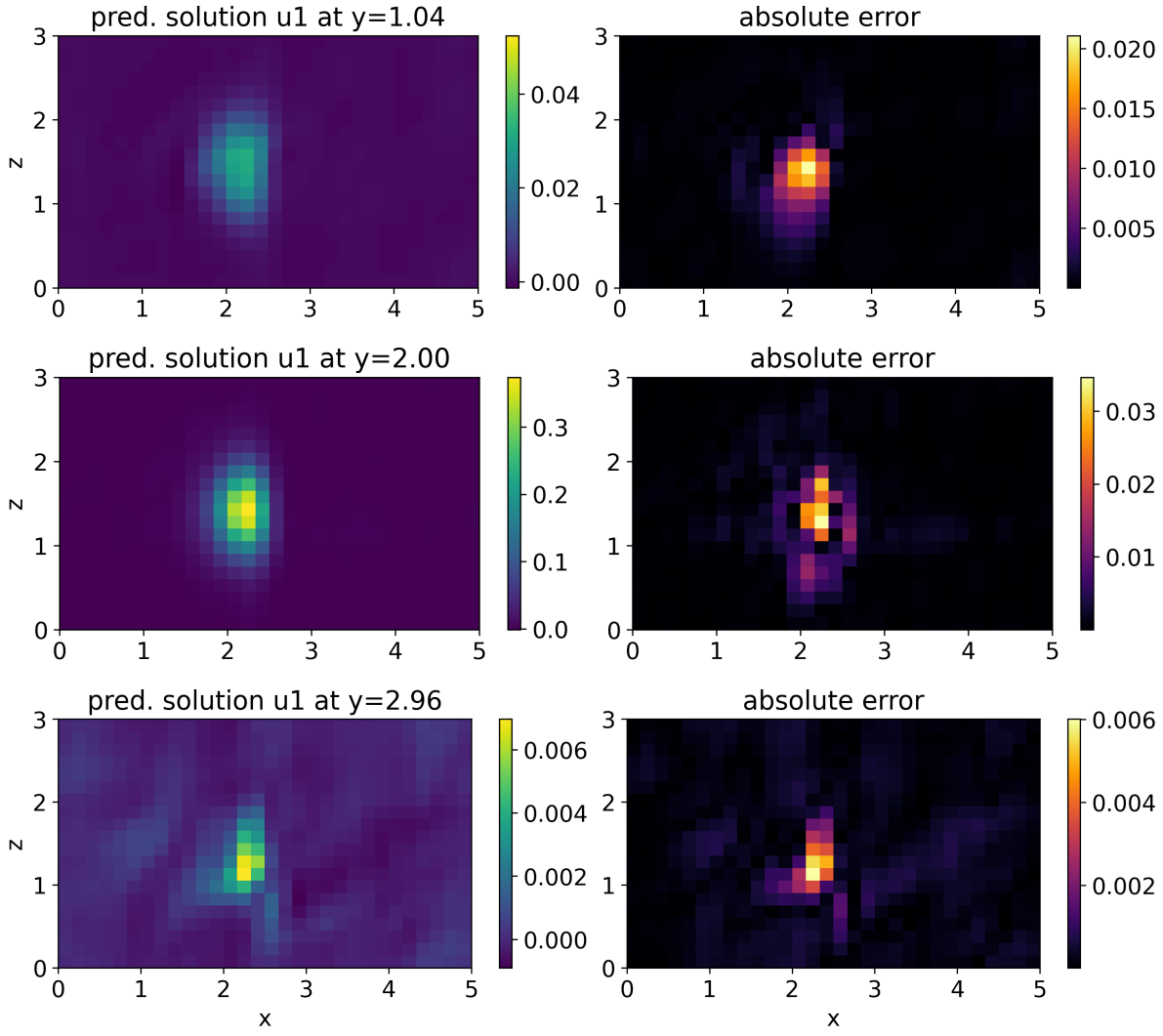


Figure 43: predicted solution u^1 and absolute error for three slices along the y-direction

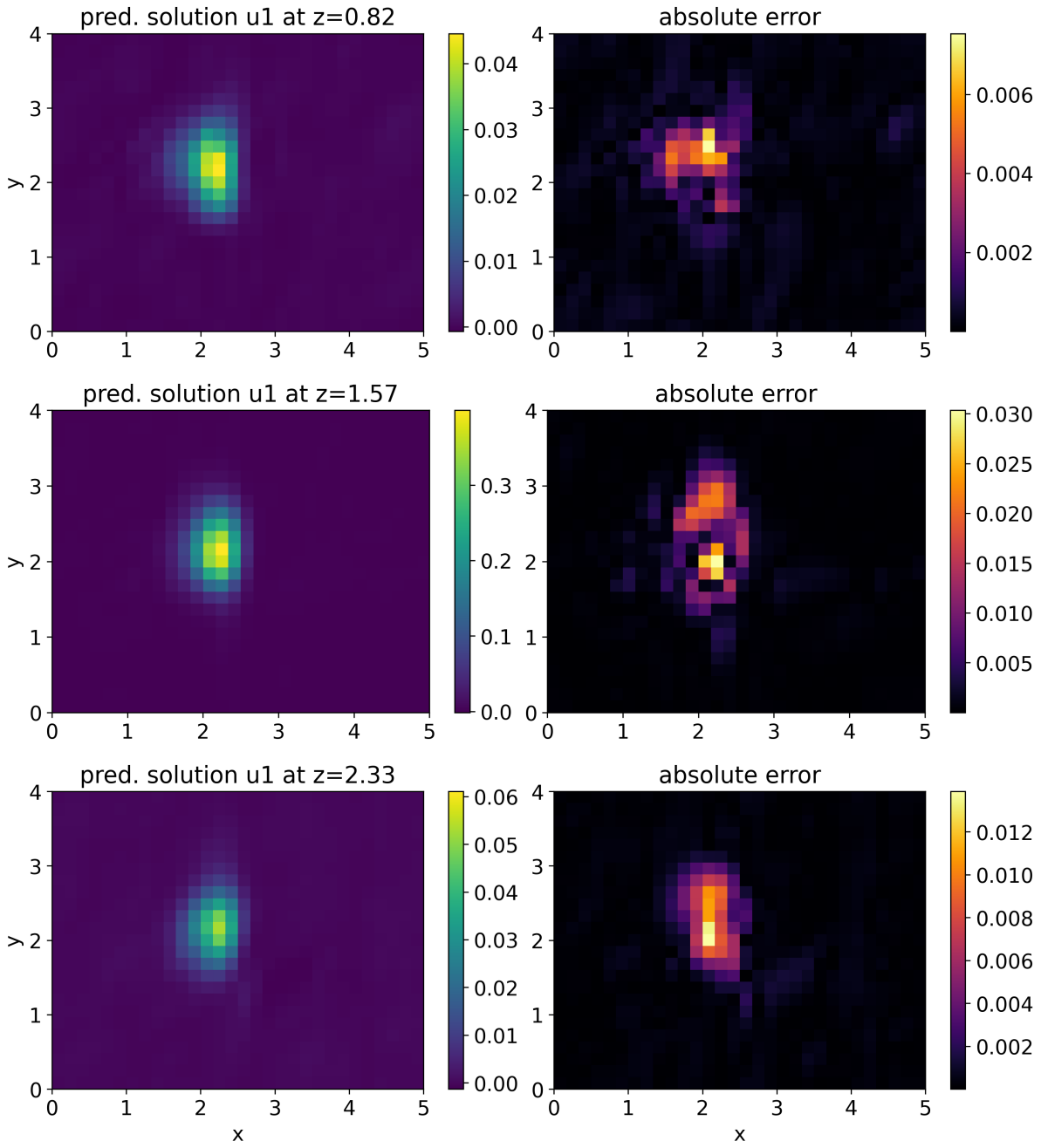


Figure 44: predicted solution u^1 and absolute error for three slices along the z-direction

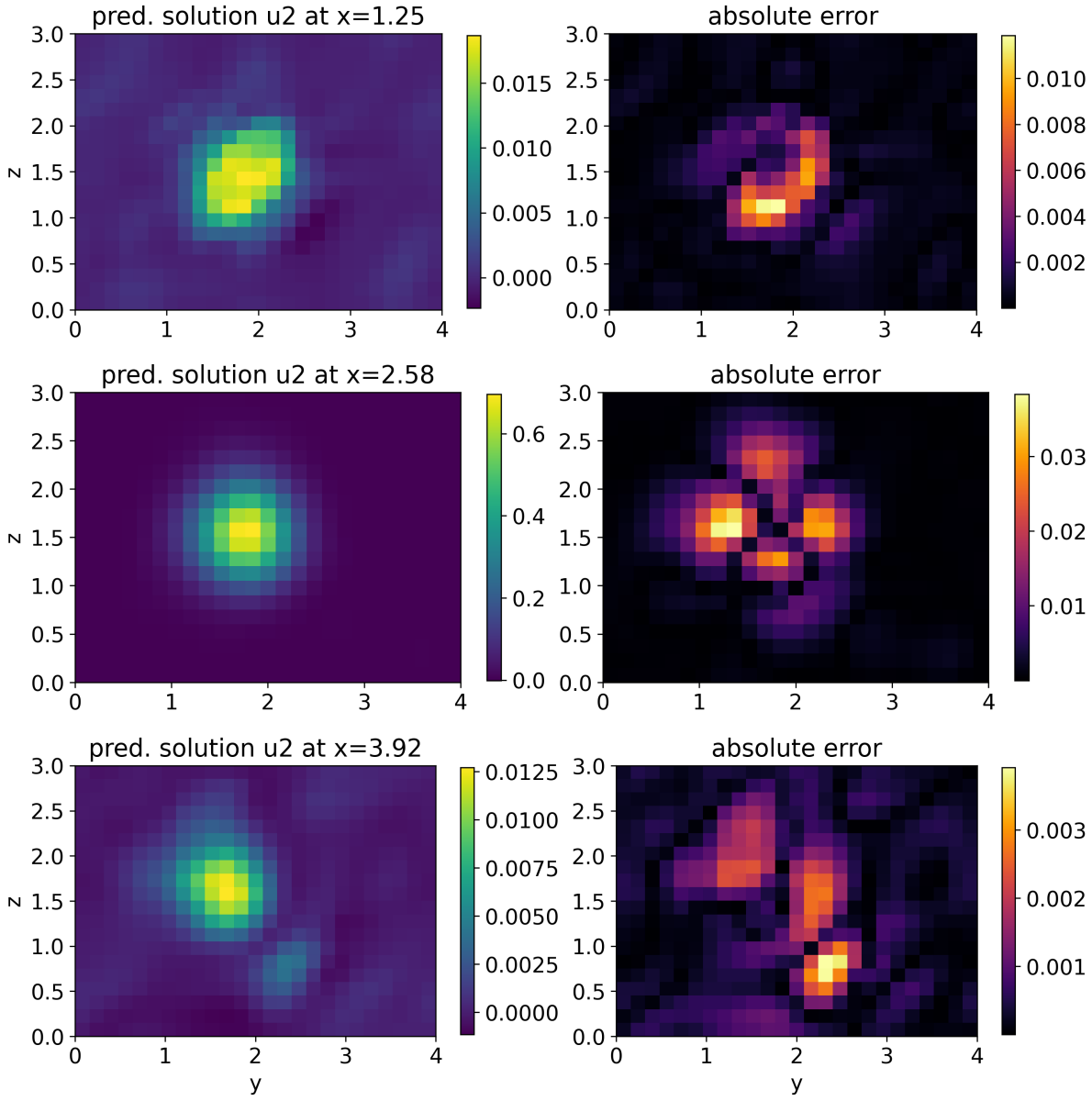


Figure 45: predicted solution u^2 and absolute error for three slices along the x-direction

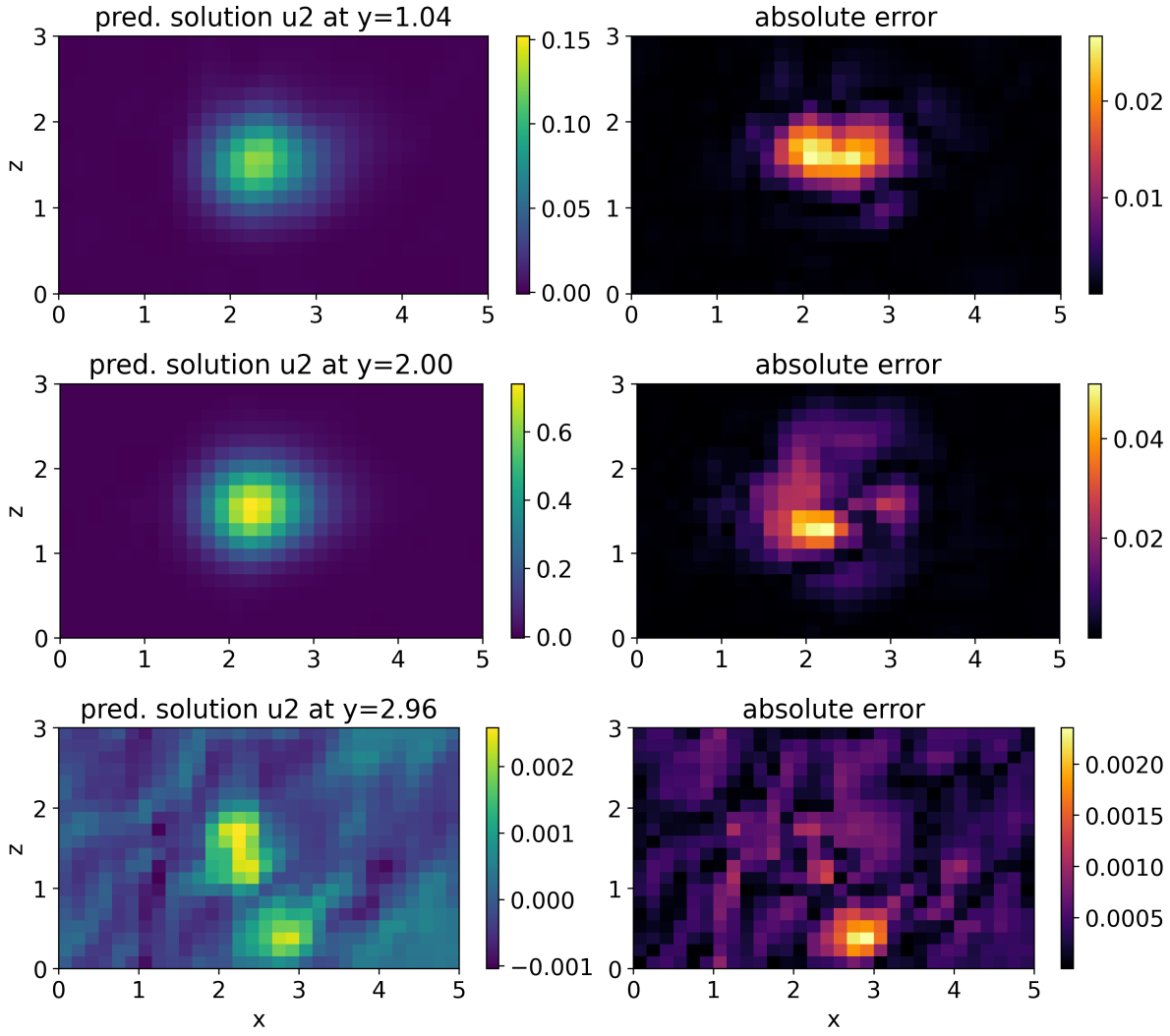


Figure 46: predicted solution u^2 and absolute error for three slices along the y-direction

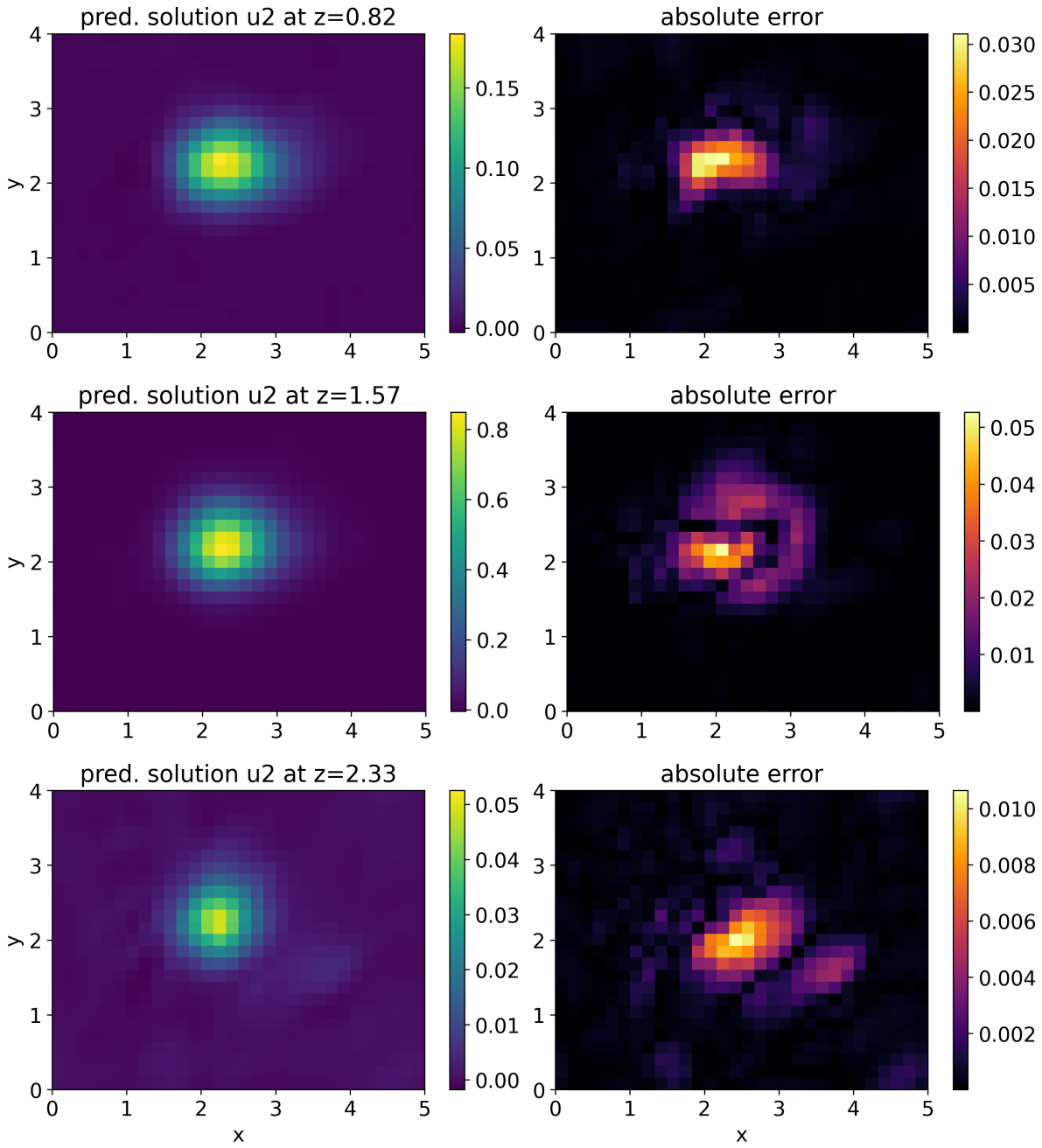


Figure 47: predicted solution u^2 and absolute error for three slices along the z-direction