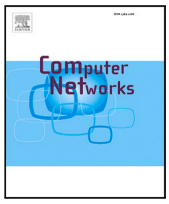
Computer Networks 269 (2025) 111437

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



EdgeBoost: Confidence boosting for resource constrained inference via selective offloading

Naina Said [a],*, Olaf Landsiedel [a,b]

[a] Department of Computer Science, Kiel University, Germany
[b] Institute for Networked Cyber-Physical Systems (NCPS), Hamburg University of Technology (TUHH), Germany

ARTICLE INFO

Keywords:
TinyML
EdgeAI
MCU
Model calibration
Temperature scaling
Inference offloading
Lightweight models

ABSTRACT

Deploying large Deep Neural Networks with state-of-the-art accuracy on edge devices is often impractical due to their limited resources. This paper introduces `EdgeBoost`, a selective input offloading system designed to overcome the challenges of limited computational resources on edge devices. `EdgeBoost` trains and calibrates a lightweight model for deployment on the edge and, in addition, deploys a large, complex model on the cloud. During inference, the edge model makes initial predictions for input samples, and if the confidence of the prediction is low, the sample is sent to the cloud model for further processing, otherwise, we accept the local prediction. Through careful calibration, `EdgeBoost` reduces the communication cost by 55%, 27% and 20% for the CIFAR-100, ImageNet-1k and Stanford Cars datasets, respectively, when compared to an cloud-only solution while achieving on-par classification accuracy. Furthermore, `EdgeBoost` reduces the total inference latency from 148 ms to 123.84 ms per inference compared to a cloud-only solution. Our evaluation also shows that calibrating the edge model for such a collaborative edge–cloud setup results in accuracy gains of up to 8 percent point, compared to an uncalibrated edge model. Additionally, EdgeBoost, when used as an abstaining classifier, can improve accuracy by up to 9 percent points over an uncalibrated model. Finally, `EdgeBoost` outperforms the Early Exit and Entropy thresholding baselines and achieves comparable accuracy to state-of-the-art routing-based methods without the need for hosting the router on the edge.

1. Introduction

In recent years, Deep Neural Networks (DNNs) have found widespread use. The deployment of these models on edge devices, such as smartphones, embedded systems, and IoT devices, poses significant challenges due to the resource constraints of edge devices and the high computational demands of DNNs.

Lightweight models [1–4], tailored to the constraints of edge device resources, however, often show reduced predictive performance. For example, for the popular ImageNet-1k [5] dataset, the lightweight MobileNet model [1], with 4.3 million parameters, suffers from an accuracy loss of approximately 7% compared to the state-of-the-art ResNet152 model with 60.4 million parameters [6]. Similarly, when considering modern vision transformers, this trade-off is even more pronounced. The Vision Transformer (ViT) Large model [7], for instance, achieves 85.8% top-1 accuracy on ImageNet-1k. However, its size of 307 million trainable parameters makes it impractical for deployment on resource-constrained edge devices. Some approaches use early exit techniques [8–10], where the initial layers of a powerful model run on the edge while the remaining layers run in the cloud. The initial network allows an early exit for the samples with sufficiently high prediction confidence. However, for such networks, splitting in the early layers results in high communication costs, while splitting in the later layers imposes high computational demands on the edge, making it ineffective for collaborative edge–cloud setup.

To address this challenge, today's approaches split the input data at runtime into easy and hard samples [11–13]: The edge device uses a lightweight model to classify the easier samples locally while sending the harder ones to the cloud for processing by a more powerful network. As lightweight models commonly show good performance when classifying easy inputs, this design improves overall classification performance, as we also show in our evaluation in Section 4. To separate easy and hard samples, many approaches directly use the output probabilities of the edge model [11]. However, due to the inherent miscalibration of models, this reduces the overall accuracy of the system, as our evaluation shows. Others train a pre-classifier as a router to split the input into easy and hard samples [12,13]. This pre-classifier, however, adds extra overhead and often needs to be designed and trained specifically for each dataset.

* Corresponding author.
E-mail address: nas@informatik.uni-kiel.de (N. Said).

https://doi.org/10.1016/j.comnet.2025.111437
Received 1 October 2024; Received in revised form 10 February 2025; Accepted 1 June 2025
Available online 16 June 2025

Addressing the above limitations, this paper introduces EdgeBoost, which instead of directly relying on the model probabilities for confidence calculation, calibrates these probabilities before using them for any decision-making. Thus, EdgeBoost utilizes a calibrated variant of a resource-efficient MobileNet or MCUNet [4] on the constrained edge device, to locally classify easy inputs while reliably identifying difficult inputs that exceed the local model's abilities, and offloads these to a large model in the cloud.

For the ImageNet-1k dataset, EdgeBoost, using MobileNetV3-Small as the edge model and EfficientNetV2-L as the cloud model, achieves an accuracy on par with the cloud by offloading 73% of the input samples to the cloud, thus saving 27% communication cost compared to an all-Cloud system where all the input samples need to be offloaded to the cloud. For this dataset, we reduce the total inference latency from 148 ms to 123.84 ms, demonstrating that EdgeBoost enables faster and more efficient inference compared to a cloud-only approach. Similarly, for CIFAR-100 [14] and Stanford Cars [15], we reduce the communication cost by 55% and 20% respectively, while achieving cloud accuracy. Our evaluation also demonstrates the effectiveness of edge model calibration for decision-making. We achieve accuracy gains up to 6, 8 and 1.5 percent points by calibrating the edge models trained on CIFAR-100, Stanford Cars, and ImageNet-1k datasets, respectively. A calibrated edge model also allows more accurate abstention decisions, leading to a 9 percent point accuracy improvement for the Stanford Cars dataset, compared to an uncalibrated model. Furthermore, we also show that high-confidence predictions from the calibrated model are significantly more reliable, resulting in a 10 percent point accuracy gain on easy samples. We further show in our evaluation that EdgeBoost achieves superior accuracy compared to early exit and entropy thresholding baselines. Moreover, we achieve comparable accuracy to state-of-the-art routing-based approaches while keeping the computational demands on the edge low.

Overall, this paper makes the following contributions:

- Design and development of EdgeBoost, a lightweight system for intelligent cloud offloading based on input hardness that achieves cloud accuracy with up to 55% reduced communication cost in CIFAR-100, 27% on ImageNet-1k and 20% on Stanford cars datasets.
- Calibration analysis of modern lightweight networks, which shows that these networks are not well calibrated and need calibration to effectively identify hard samples on the edge. Through Temperature Scaling, we increase the accuracy of our edge models up to 6, 8 and 1.5 percent points on CIFAR-100, Stanford Cars, and ImageNet-1k dataset, respectively.
- Performance evaluation of EdgeBoost, showing that EdgeBoost outperforms both early exit networks and entropy thresholding, achieving an average accuracy improvement of 8.47 and 1.7 percent points over both techniques, respectively. Furthermore, we also achieve accuracy comparable to state-of-the-art routing-based approaches without the need for an extra classifier on the edge.
- Evaluation of EdgeBoost's effectiveness, demonstrating its role as an abstaining classifier with a 9 percent point accuracy improvement, enhanced reliability of high-confidence predictions with a 10 percent point accuracy gain, and latency reduction—lowering inference time to 123.84 ms from 148 ms in an all-cloud setup.

The source code for EdgeBoost and for the experiments presented in this paper is available online.[1] The rest of the paper is organized as follows: Section 2 presents the necessary background on DNN confidence and calibration, Section 3 introduces the design of EdgeBoost, Section 4 evaluates EdgeBoost on three datasets and on the MobileNetV3-Small and MCUNet architectures, and Section 5 discusses related work. Finally, Section 6 concludes the paper.

[1] EdgeBoost GitHub repository: https://github.com/ds-kiel/EdgeBoost.

2. Background: Confidence and calibration

In this section, we provide the necessary background on DNN confidence and calibration. First, we introduce the basics of model confidence and discuss the need for calibration. Next, we introduce Expected Calibration Error (ECE), a metric for measuring the degree of calibration of a NN, and Temperate Scaling, a lightweight method for calibrating NNs. Finally, we discuss how we use the probabilities of the calibrated models to determine their confidence.

2.1. Model confidence and calibration

The confidence of a model is a measure of how confident a model is about its classification result for a given input. Thus, if it predicts that an input sample is of a particular class with a confidence of X, that prediction should have an X% probability of being correct. However, the literature shows that while modern DNNs are highly accurate, they tend to be uncalibrated [16,17]. As a result, uncalibrated DNNs often produce probability estimates that are not well aligned with the true probabilities. Furthermore, as we show in our evaluation in Section 4, many modern DNNs tend to be overconfident, which further increases the challenge of reliably separating easy and hard inputs.

For EdgeBoost, the calibration of the model at the edge is crucial, because it determines whether a local prediction is accepted or the sample is sent to the cloud for classification. Thus, our approach requires good calibration, and we introduce methods for calibration later in this section.

2.2. Expected calibration error

Expected Calibration Error (ECE) [18] is a metric that measures how well a model's estimated probabilities match the true (observed) probabilities. The metric is calculated by partitioning the probabilities into M equally spaced bins and taking a weighted average of the absolute difference between the accuracy *acc* and the confidence *conf* as shown in Eq. (1). Here B represents "bins", m the number of bins, and n the total number of samples. The difference between *acc* and *conf* for a given bin represents the calibration error. A low value of ECE indicates good model calibration, while a higher value means that the model is poorly calibrated. Throughout this paper, we use ECE to represent the calibration error of neural networks.

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} \left| \text{acc}\left(B_m\right) - \text{conf}\left(B_m\right) \right| \tag{1}$$

2.3. Temperature scaling for NN calibration

Temperature scaling [16] is a simple but effective method for recalibrating prediction probabilities. Temperature scaling uses a single scalar parameter $Temp > 0$ where *Temp* is the temperature, to rescale the logit scores of an NN before applying the softmax function as shown in Eq. (2). The optimal value of *Temp* is determined by minimizing the Negative Log Likelihood (NLL) on the validation set. While Temperature Scaling helps to calibrate a model, it does not affect the accuracy of the model since the maximum of softmax remains unchanged.

$$\text{softmax}_{\text{Temp}}\left(z_i\right) = \frac{e^{z_i/\text{Temp}}}{\sum_{j=1}^{K} e^{z_j/\text{Temp}}} \tag{2}$$

Besides Temperature Scaling, other methods such as Monte Carlo Dropout [19], Isotonic Regression [20] and Histogram Binning [21] allow NN calibration. However, we choose temperature scaling due to its simple implementation and superior performance over other techniques [16].

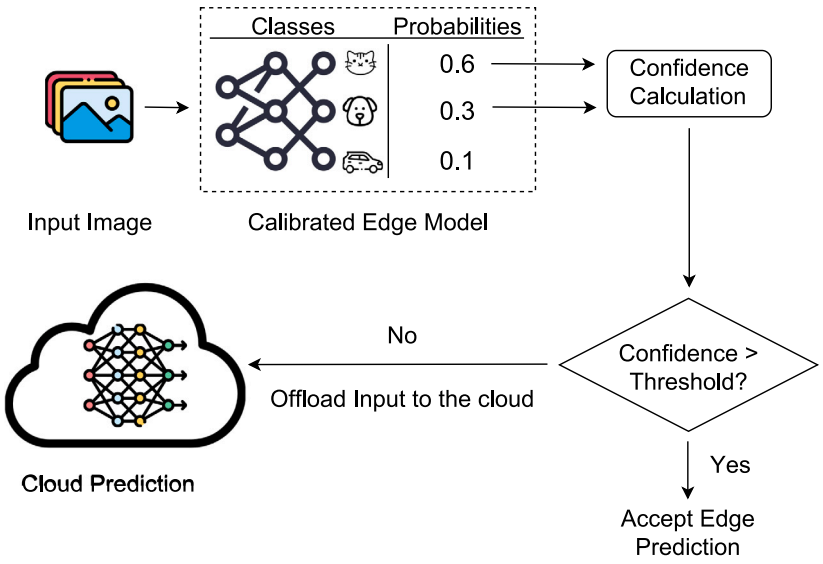


Fig. 1. Overview of EdgeBoost. First, EdgeBoost passes an input sample to a small, less powerful calibrated DNN on the edge. Next, we calculate the confidence of the prediction by using the highest and second highest prediction probability. If the confidence of the prediction is sufficiently high, we accept the local classification result. Otherwise, the sample is sent to the more powerful DNN deployed on the cloud. In our evaluation, we show that this simple design strongly improves overall classification accuracy when compared to an edge-only approach.

2.4. Calculation of prediction confidence

After calibrating a model, EdgeBoost utilizes the softmax margin [11] as a measure of the edge model's confidence which is the largest softmax value minus the second largest. This represents how much more confident the model is in its first prediction compared to the second one. If y_1 and y_2 are the top two most probable labels for a sample x under a model θ, we represent the softmax margin by Eq. (3). A larger difference represents more confidence, while a small difference value represents less confidence of the model in its prediction.

$$Confidence = p_\theta\left(y_1' \mid x\right) - p_\theta\left(y_2' \mid x\right) \tag{3}$$

3. Design

In this section, we introduce the design of EdgeBoost. First, we discuss the calibration of resource-efficient DNNs, highlighting model selection and temperature scaling. This is followed by an overview of the EdgeBoost architecture, focusing on its key components. We then examine the role of the confidence threshold (T) in balancing prediction accuracy and communication cost. Finally, we detail on the methodology for offloading samples to the cloud, emphasizing decision making based on calibrated confidence levels.

3.1. Calibration analysis of resource-efficient DNNs

Before diving into the design of EdgeBoost, we analyze the calibration of resource-efficient neural networks. While many works analyze the calibration of large neural networks [16,17], the calibration of models tailored for resource-constrained devices has received very little attention as of today. Thus, as a first step, in the design of EdgeBoost, we close this gap and analyze their calibration.

We analyze the calibration of state-of-the-art resource-efficient models such as MobileNet and MnasNet [3] for the edge and MCUNet for microcontrollers on common datasets such as CIFAR-100, Stanford Cars and ImageNet-1k, see Table 1. Despite the efficiency and good accuracy of resource-efficient models, our results show that the majority of these models show a miscalibration after training, as indicated by their respective ECE values. Only MCUNet shows very little miscalibration and does not require further calibration. Others, such as the MobileNetV3-Small [22] model show strong miscalibration and on the CIFAR-100 dataset, for example, show a reduction in ECE from 14.8% to 2.3% after applying Temperature Scaling. The main result of this analysis is that (a) many neural networks lack calibration, and (b) Temperature Scaling drastically improves their calibration.

We dive deeper into the calibration of models used on the edge in this study. Fig. 2 shows the calibration of MobileNetV3-Small for CIFAR100, Stanford Cars and ImageNet-1k and the calibration of MCUNet for ImageNet-1k. The results show that MobileNetV3-Small is overconfident for CIFAR100 and Stanford Cars datasets. For example, when it believes that an input is of a class with, for example, 90% confidence, the actual confidence should only be about 50% and 70% for CIFAR-100 (see Fig. 2(a)) and Stanford Cars (see Fig. 2(b)), respectively. Calibrating the models using Temperature Scaling removes this bias. For the ImageNet-1k dataset, we see similar effects but with much lower significance, see Figs. 2(c) and 2(d) for MobileNetV3-Small and MCUNet, respectively. Note that the optimal Temperature ($Temp$) values are determined separately for each task by minimizing the NLL on the validation set. This ensures that the calibration accounts for differences in the task-specific data distributions and behavior of the model across these tasks.

3.2. EdgeBoost: Design overview

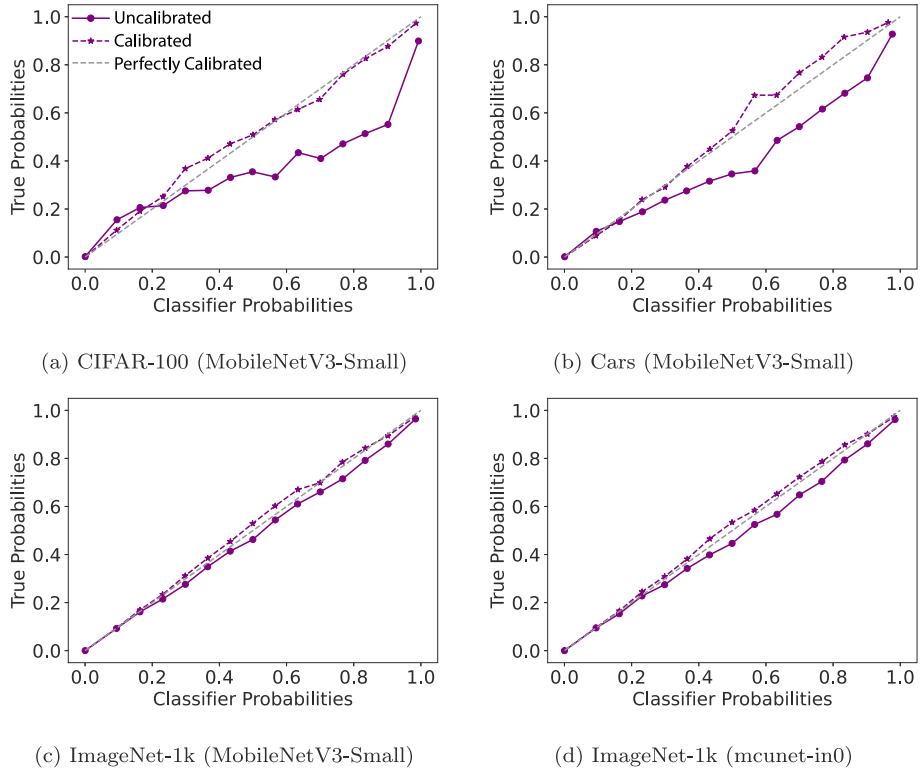
Based on the insights we derived from the analysis of model calibration in Section 3.1, we now introduce a simple and lightweight system design: EdgeBoost uses two models, one resource-efficient on the edge and a second large model on the cloud, see Fig. 1. Further, we calibrate the neural network on the edge using Temperature Scaling, as introduced in Section 2. For each input, we first use the edge model to make a prediction. Next, we use a confidence checker to assess the prediction confidence. If the confidence is high, we accept the local prediction; otherwise, we pass the input to the cloud model for prediction.

3.3. Assessing confidence & tuning the confidence threshold

We use the probability margin, which is the difference between the highest and second highest classification probabilities (see Section 2), as a metric to assess the confidence of the calibrated edge model. In EdgeBoost, a user-defined threshold parameter, denoted

Table 1
Calibration analysis of resource-efficient models such as MobileNet and MCUNet for common datasets such as CIFAR-100 and ImageNet-1k. As shown by the ECE value, most of these models are poorly calibrated after training and require Temperature Scaling for post-training calibration. Only MCUNet-in0 and MCUNet-in1 have a negligible calibration error and do not require further calibration.

Dataset	Model	Accuracy (%)	Parameters (M)	ECE (%)	
				Uncalibrated	Calibrated
CIFAR-100	Resnet20	68	0.28	11	2
	MobnetV3 Small	75	2.5	14.8	2.3
	MobnetV2	77	3.5	10.5	2.5
Stanford Cars	MobnetV3-Small	57.5	2.5	12.6	3.7
	MobnetV2	70.5	3.5	6.74	2.7
	MnasNet	71	6.2	8.92	3.16
ImageNet-1k	MCUNet-in0	41.5	0.75	0.4	–
	MCUNet-in2	60.9	0.73	0.7	–
	MCUNet-in4	68.41	1.73	3.66	1.4
	MobnetV3-Small	67.5	2.5	2.81	1.72
	MobileNetV2	71	3.5	2.73	1.87



(a) CIFAR-100 (MobileNetV3-Small) (b) Cars (MobileNetV3-Small)

(c) ImageNet-1k (MobileNetV3-Small) (d) ImageNet-1k (mcunet-in0)

Fig. 2. Reliability of MobileNetV3-Small and MCUNets, before and after calibration using Temperature Scaling. Each diagram compares the expected accuracy of a model's predictions with the observed accuracy at different levels of predicted confidence. The ideal model's predictions lie on the 'Perfectly Calibrated' line, indicating that the predicted probabilities match the empirical probabilities. The post calibration curves in these plots are close to the perfectly calibrated curve, indicating an improvement in the reliability of the models for calculating confidence. Thus, Temperature Scaling is effective for calibrating models with high calibration error.

T, acts as a critical decision boundary to determine whether a sample should be processed locally by the edge model or passed to the cloud model. Choosing a value for T resembles a trade-off between prediction accuracy and communication cost: A higher value of T causes the edge model to predict more conservatively, i.e., to accept fewer local classification results. As a result, `EdgeBoost` sends more samples to the cloud for processing, resulting in higher communication overhead. Conversely, a lower T value causes the edge model to process a larger fraction of samples, which reduces communication costs but compromises prediction accuracy, especially for challenging input samples. In Section 4, we evaluate the overall accuracy and communication overhead for different values of T to elaborate on this behavior. In practice, we should choose a value of T that satisfies the cost/accuracy requirements of a particular application. Since the metric acts only as a threshold, the confidence checker is designed to be lightweight and resource efficient.

3.4. Algorithmic details

After training the models on the edge and the cloud, we follow the process in algorithm 1 to upload samples to the cloud. First, we feed the input sample to the edge model without softmax to obtain the logit values. We then calibrate the logits for the input sample using the temperature value *Temp* precalculated using the validation set, in line 3 before applying softmax. This calibration provides reliable probability estimates. Next, in line 7 we calculate the difference between the highest and second highest probabilities. This difference, which serves as a measure of confidence, is compared to the threshold value T. A large difference indicates that the edge model has high confidence in its prediction, which leads us to accept the local prediction. If not, the sample is considered difficult to classify and we send the sample to the cloud model for further prediction.

Algorithm 1 Inference Using EdgeBoost

for all $\boldsymbol{x}$ in input samples **do**
  $\boldsymbol{z}_{\text{edge}} = \text{EdgeModel}(\boldsymbol{x})$ {Edge model's logits}
  $\hat{\boldsymbol{y}}_{\text{edge}} = \text{softmax}\left(\frac{\boldsymbol{z}_{\text{edge}}}{Temp}\right)$ {Logits scaling with temperature value *Temp*}
  Sort $\hat{\boldsymbol{y}}_{\text{edge}}$ in ascending order
  $y_{1_{\text{edge}}} = (\hat{\boldsymbol{y}}_{\text{edge}})_{(n)}$ {Highest value}
  $y_{2_{\text{edge}}} = (\hat{\boldsymbol{y}}_{\text{edge}})_{(n-1)}$ {Second highest value}
  if $(y_{1_{\text{edge}}} - y_{2_{\text{edge}}}) > T$ **then**
    Label : $\hat{\boldsymbol{y}} = \text{argmax}(\hat{\boldsymbol{y}}_{\text{edge}})$
  else
    $\hat{\boldsymbol{y}}_{\text{cloud}} = \text{CloudModel}(\boldsymbol{x})$
    Label: $\hat{\boldsymbol{y}} = \text{argmax}(\hat{\boldsymbol{y}}_{\text{cloud}})$
  end if
end for

4. Evaluation

In this section, we evaluate EdgeBoost. We begin by introducing the baselines and our experimental setup before presenting our evaluation results.

4.1. Datasets

We evaluate EdgeBoost on three public vision datasets: CIFAR-100, Stanford Cars, and ImageNet-1K. The CIFAR-100 dataset contains 50K training images and 10K testing images, categorized into 100 classes, with each image being 32×32 pixels in size. The Stanford Cars dataset focuses on fine-grained classification and includes 16.2K images of 196 car categories, with 8.1K images for training and 8K for testing. The ImageNet-1k dataset spans 1K object classes and contains 1.28M training images, 50K validation images, and 100K test images, with varying resolutions. We augment the training data using random crops, random horizontal flips, and normalization, while the testing data is processed with center crops and normalization to ensure consistency during evaluation.

4.2. Baselines

We benchmark **Edgeboost** against the following four baselines:

- **All-Edge Baseline:** This baseline represents the scenario where the edge device is solely responsible for the inference without using any cloud resources.
- **All-Cloud Baseline:** Unlike the all-edge approach, this baseline relies entirely on the cloud to perform inference.
- **Entropy-based Baseline:** The entropy-based approach uses the entropy of the output probability distribution obtained from a model to determine uncertainty. Entropy measures the degree of "confidence" or "uncertainty" in the prediction: higher entropy indicates greater uncertainty. Therefore, when entropy is used as a measure of prediction confidence, inputs with high entropy are offloaded to the cloud, while inputs with low entropy are handled locally by the edge model. We select this technique as a baseline based on the findings in Kag et al. [13], where entropy thresholding is found to be superior to methods such as AppealNet [12], BranchyNet [8], and other adaptive networks [23–25]. Entropy-based methods, however, do not take model calibration into account.
- **Routing-based Baseline:** This baseline involves designing and training a router to assess the difficulty of input samples. A lightweight neural network decides whether to process an input locally or offload it to the cloud. We select Kag et al. [13] as the routing-based baseline because of its superior performance compared to similar prior techniques. Kag et al. train the routing model as a two-layer feed-forward neural network and deploy it on the edge alongside the edge model. This hybrid system includes an edge model, a router, and a cloud model to maximize on-edge coverage without compromising accuracy. The router is trained separately for different tasks, and deploying it on the edge requires additional memory.
- **Early Exit Baseline:** Early exit neural networks add branches to a neural network and compute later layers only if the output of an early layer does not provide sufficient classification confidence [8,25]. These networks are typically trained using a joint optimization strategy, where the early exit branches are integrated into the training process. During training, each exit branch is trained with auxiliary classifiers, which are supervised using the same ground-truth labels as the final output. This ensures that each early exit branch learns meaningful intermediate representations and can make confident predictions. To compare with this baseline, we assume that the initial part of the cloud model executes on the edge, while the remaining layers execute on the cloud. Note that for this baseline, choosing the exit point is critical, since an exit at an early point in the network may not provide sufficient classification accuracy, while an exit at later layers imposes higher computational demands on the edge.

4.3. Experimental setup

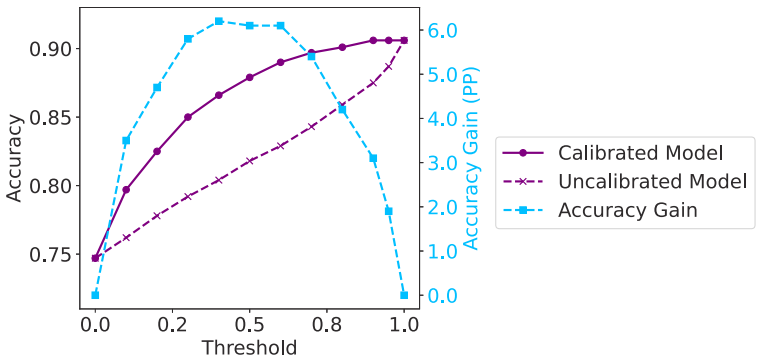
Next, we introduce our experimental setup for our comparison to (a) all-edge and all-cloud baselines and (b) to state-of-the-art baselines.

4.3.1. Comparison with all-edge and all-cloud baselines

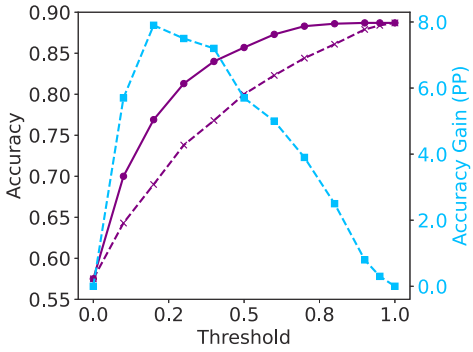
We evaluate two different settings for the constrained device: (a) an edge device utilizing MobileNetV3-Small and (b) an MCU utilizing MCUNet. MobileNetV3-Small comprises 2.5M parameters and has an accuracy of 67.5% on the ImageNet-1k dataset. For MCUNet we use multiple configurations (mcunet-in0, mcunet-in2, mcunet-in4) with 0.75, 0.73, and 1.73 Million parameters and achieve accuracies of 41.5%, 60.9% and 68.4% respectively, on the ImageNet-1k dataset. As cloud models, we deploy both EfficientNetV2-L (119M parameters) [26], a high-performing CNN model that achieves an accuracy of 86% on ImageNet-1k and a ViT, EVA02 (305M parameters) [27] which achieves a state-of-the-art accuracy of 89.5% on ImageNet-1k. Both of these models have a stark increase both in terms of accuracy and computational demands when compared to MobileNetV3-Small and the MCUNet variants we deploy on the constrained devices. We note that the exact choice of the edge and the cloud model is independent of our approach, and we argue that our design is applicable to any combination of models, as long as we have a significant performance gap between the edge and the cloud model. To demonstrate this, we use the small version of EfficientNetV2 (21.45M parameters and 85% accuracy on ImageNet-1k) for experiments on Stanford Cars dataset. Furthermore, our methodology does not rely on any specific hardware for edge devices or servers, as long as the selected devices are capable of running these models.

To ensure reproducible results, we use pre-trained networks EfficientNetV2 (large and small versions), EVA02, MobileNetV3-Small and MCUNet[2] where available, otherwise we fine tune the models. We achieve a base accuracy of 75% and 57% and cloud accuracy of 90% and 88% on CIFAR-100 and Stanford Cars, respectively. For all our experiments, we calibrate the edge models using Temperature Scaling, see Fig. 2.

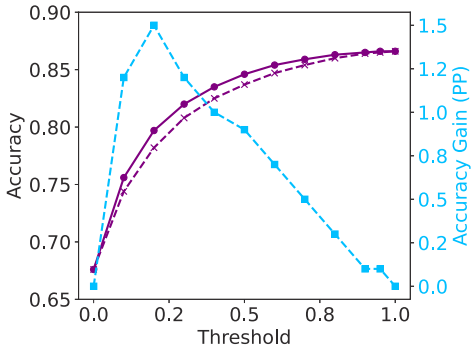
[2] EfficientNetV2-L, MobileNetV3: https://pytorch.org/vision/stable/models.html; MCUNet: https://github.com/mit-han-lab/mcunet.



(a) CIFAR-100 (MobileNetV3-S)



(b) Cars (MobileNetV3-S)



(c) ImageNet-1k (MobileNetV3-S)

Fig. 3. Accuracy comparison of calibrated and non-calibrated edge model, used as a confidence decider. At different threshold levels, the calibrated model is able to capture the uncertain samples more effectively than the poorly calibrated one, which is evident from the increase in the accuracy.

4.3.2. Comparison with state-of-the-art approaches

We compare EdgeBoost to (a) to entropy thresholding, (b) routing-based methods, and (c) early exit methods, as discussed in Section 4.2. To ensure a fair comparison with state-of-the-art methods, we select – where available – the same models at the edge and in the cloud that these papers use. For the entropy thresholding and routing-based baselines, we use the largest pre-trained model from the OFA space [28] which achieves 79.9% accuracy on ImageNet-1k as cloud model, while as edge model we experiment with two versions of MobileNetV3, one with 2.5M parameters and another with 5.4M parameters.

To compare EdgeBoost to early-exit approaches, we assume that the initial part of the cloud model (OFA in this case) executes on the edge while the remaining layers execute on the cloud. We test two configurations for this baseline. In the first configuration, we place the early exit at a point that results in approximately 6.7M parameters on the edge. In the second configuration, we position the early exit differently, reducing the number of parameters on the edge to approximately 1.5M. The training follows the BranchyNet [8] approach, optimizing all exit branches simultaneously with a weighted loss that combines both early and final classification losses.

4.4. Experimental results

In this section, we present the experimental results that demonstrate the effectiveness of EdgeBoost as an efficient cloud offloading system. First, we present the benefits of calibrating the edge model. Next, we compare EdgeBoost to an all-Edge and all-Cloud baseline and to state-of-the-art offloading techniques. We then perform the latency analysis of EdgeBoost. We also demonstrate the effectiveness of EdgeBoost as an abstaining classifier on the edge. Finally, we compare the performance of a calibrated and uncalibrated model on high confidence samples.

4.4.1. Comparison of calibrated and uncalibrated edge model

As noted in our previous evaluation, if our edge model is not calibrated, it tends to be overconfident and hence processes more samples locally than it actually should. Fig. 3 shows the accuracy of EdgeBoost when the edge model is calibrated and uncalibrated for different user-define thresholds. We see that as a result of calibration, only confident predictions are accepted by the edge model, thereby increasing its accuracy when compared to the uncalibrated model. For example, at a threshold value of 0.1, the uncalibrated model achieves an accuracy of 76% while the calibrated model achieves an accuracy of 80% for the CIFAR100 dataset when using MobileNetV3-Small as the edge model (see Fig. 3(a)). At a threshold value of 0.4, the accuracy of the calibrated model is 6 percent points better than the uncalibrated model. Furthermore, the calibrated model outperforms the uncalibrated model in terms of accuracy at every threshold. We observe the same trend for the in Stanford Cars dataset in Fig. 3(b). For the ImageNet-1k model (Fig. 3(c)), the accuracy gains are the smallest because the ECE value difference between the calibrated and uncalibrated value is small. Nevertheless, we see improvements of up to 1.5 percent points after calibration.

4.4.2. Comparison with all-edge and all-cloud baselines

Next, we compare EdgeBoost to all-Edge and all cloud method baselines. We report the results for both EfficientNetV2-L as well as a ViT used as cloud models.

(A) EfficientNetV2-L as cloud model. We evaluate three datasets (CIFAR-100, Stanford Cars and ImageNet-1k) and four edge networks (MobileNetV3-Small and three MCUNet variants). Figs. 4 and 5 show the relationship between the user-defined confidence threshold for the edge model and three key performance metrics: overall system accuracy, average communication cost per input image, and the percentage

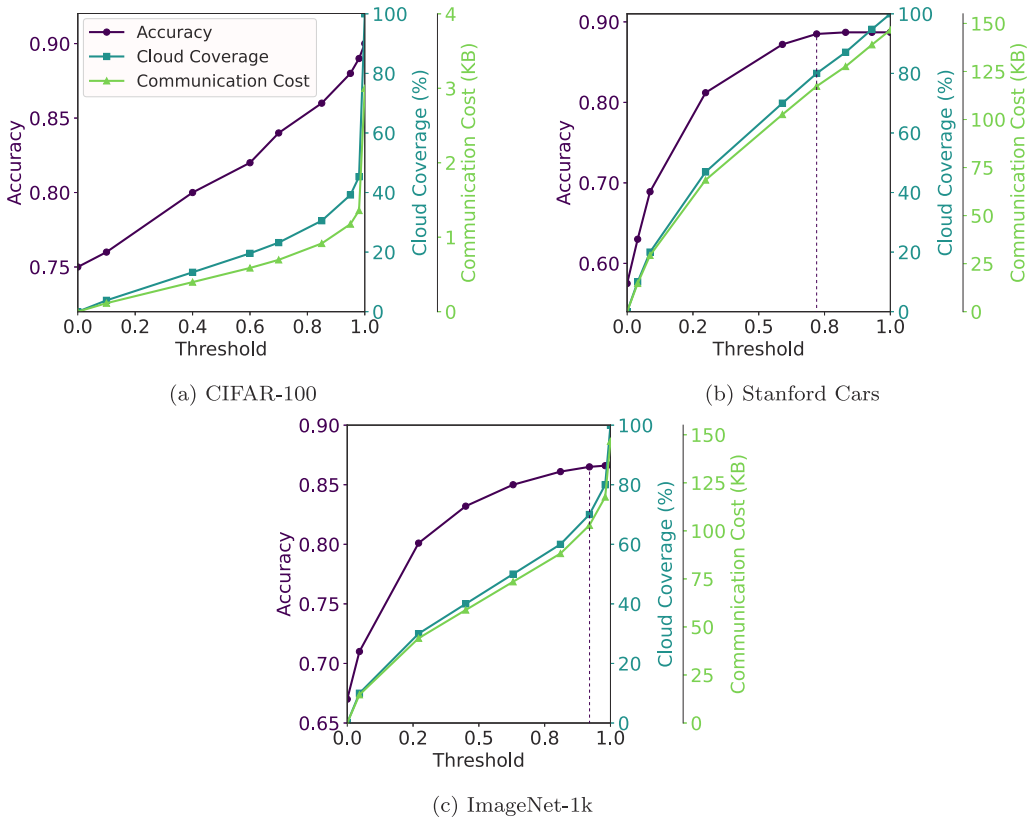


Fig. 4. EdgeBoost compared to an all-Cloud and all-Edge solution on three standard datasets (CIFAR-100, Stanford Cars, ImageNet-1k). The cloud model used for the CIFAR-100 and ImageNet-1k experiments is EfficientNetV2-L and for Stanford Cars is EfficientNetV2-S while the edge model used is MobileNetV3-Small. The accuracy improves for all three datasets when a small fraction of the samples are sent to the cloud. The dotted lines represent the threshold at which cloud accuracy is achieved. Thus, EdgeBoost improves the accuracy on the edge by sending less confident samples to the cloud.

of samples processed by the cloud. First, we find that as the threshold increases, the system accuracy also increases. This trend is consistent across all three datasets, suggesting that allowing more input samples to be processed by the cloud model improves the overall accuracy. The rate of increase varies across the datasets. However, as the accuracy approaches the cloud accuracy, all datasets exhibit a diminishing rate of accuracy improvement, suggesting that after a certain threshold, cloud offloading yields minimal gains. At a threshold of 0.1, the accuracy for CIFAR100 in MobileNetV3-Small is 76%, see Fig. 4(a). When we increase the threshold to 0.4, the accuracy increases by 4 percent points to 80%. At 0.8, the accuracy jumps to 85% and finally, at a threshold value of 0.98 when 45% of the samples are processed by the cloud, the system achieves cloud accuracy of 90%. Thus, we save 55% communication costs compared to an all-Cloud solution with equivalent accuracy. In comparison, the Stanford Cars already shows a significant improvement when the threshold is merely set at 0.1, see Fig. 4(b). At this value, the accuracy improves from 57% to 69% by offloading 21% of the samples to the cloud. We achieve cloud accuracy by sending 80% of the samples to the cloud, reducing the communication costs by 20% when compared to an all-Cloud solution with equivalent accuracy. For ImageNet-1k in Fig. 4(c), we approach cloud accuracy of 86% at a coverage level of 73%, i.e., saving 27% of communication cost.

In the case of the MCUNet variants, we see similar trends. For instance, for mcunet-in0 the system shows a significant accuracy improvement from 40% to 68% by only offloading 40% of the input samples to the cloud, see Fig. 5(a). At 83% cloud coverage, cloud accuracy is achieved. Similarly, mcunet-in4 (Fig. 5(c)) and mcunet-in2 (Fig. 5(b)) achieve cloud accuracy at 60% and 73% cloud coverage, respectively. Overall, we find EdgeBoost to be an effective system for achieving high inference performance while sending only a subset of the samples to the cloud.

(B) ViT as cloud model. While CNNs have historically dominated computer vision tasks, ViT based models have recently achieved state-of-the-art or competitive performance across various computer vision challenges [29]. Therefore, to leverage the performance of ViTs, we replace the EfficientNetV2-L model on the cloud with the EVA02 ViT trained on ImageNet-1k while using the MobileNetV3-Small at the edge. We show the results in Fig. 6. As Fig. 6 shows that offloading 10% of the samples to the cloud improves the accuracy from a base accuracy of 67% to 72%. Similarly, when the threshold value is set to 0.27, the accuracy of the system improves significantly to 83% by sending 32% of the samples to the cloud. Finally, we achieve cloud accuracy by sending 73% of the samples to the cloud thus reducing the communication costs by 27% compared to an all-Cloud solution. By using a more powerful ViT on the cloud, we achieve a better overall accuracy of 89.5% compared to 86% achieved by using EfficientNetV2-L.

4.4.3. Comparison with state-of-the-art approaches

Next, we compare EdgeBoost with state-of-the-art methods, namely early exit, entropy-based thresholding and a routing-based model as discussed in Section 4.2., see Table 2. In the routing-based model, which we use as a baseline, the decision router consists of a two layer DNN with 256 neurons in the first layer and 64 in the second. Therefore, in this setting, the edge device must host both the prediction model and this router. Furthermore, the router is trained together with the edge model.

First, we see that EdgeBoost achieves near-cloud accuracy using the 5.3M MobileNetV3-Large at 30% coverage. This reduces the communication costs by 70%. Also note that at 30% coverage, the smaller MobileNetV3 with 2.5M parameters exceeds the base accuracy of the larger version. Thus, we achieve the accuracy of the larger version by deploying the smaller model with approximately half the parameters.

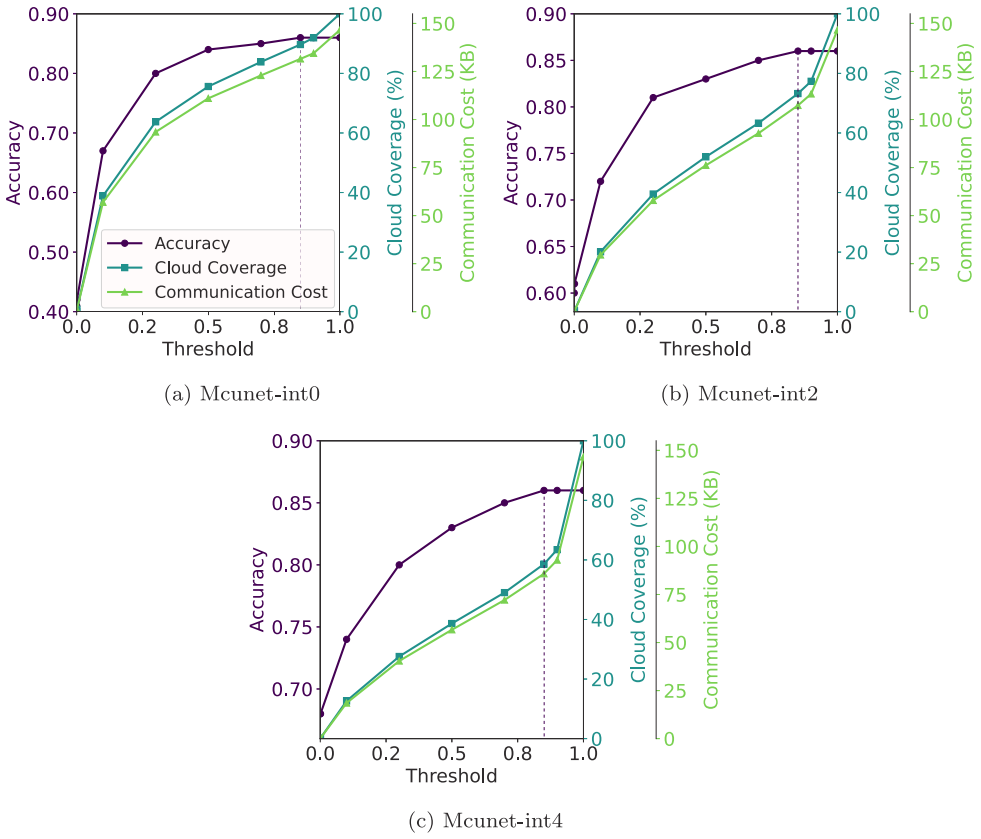


Fig. 5. EdgeBoost compared to an all-Cloud and all-Edge solution when the edge is a resource-constrained micro-controller. The cloud model used is EfficientNetV2-L while at the edge, we use three versions of MCUNet (in0,in2,in4) trained on ImageNet-1k. Similar to previous results, the accuracy improves when a small fraction of the samples are sent to the cloud. The dotted lines represent the threshold at which cloud accuracy is achieved. Thus, EdgeBoost improves the accuracy on the edge by sending less confident samples to the cloud.

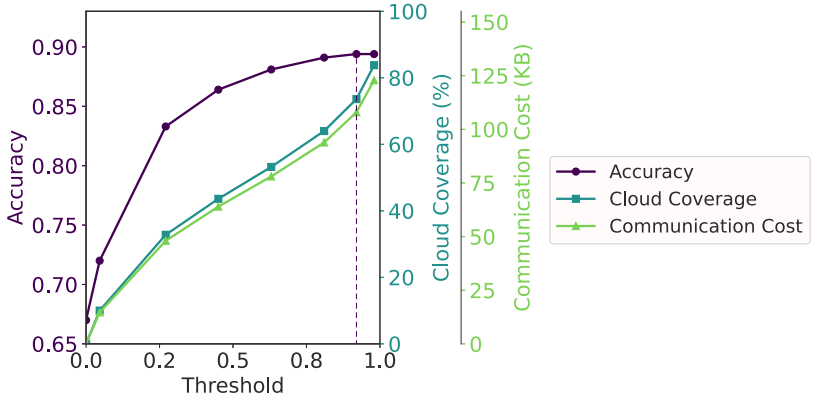


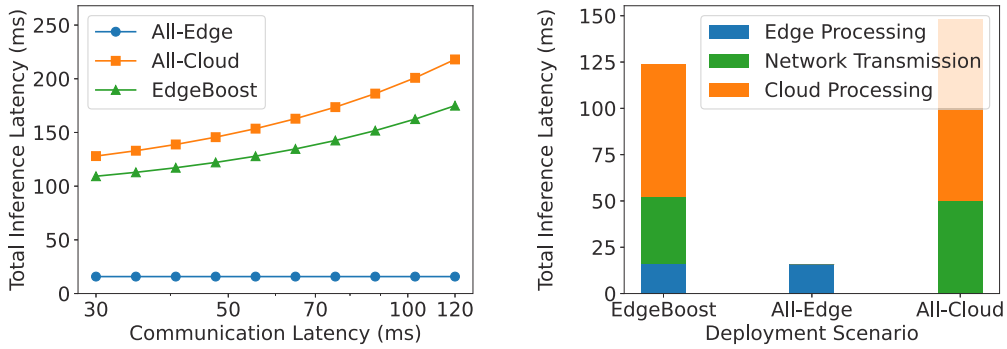
Fig. 6. EdgeBoost compared to an all-Cloud and all-Edge solution when we use ViT as the cloud model. The powerful ViT results in an overall accuracy of 89.5% by offloading 73% of the samples to the cloud.

In addition, EdgeBoost outperforms entropy thresholding by achieving nearly 1.7 percent points higher accuracy with MBV3-2.5M at 30% cloud coverage. Compared to the routing-based model, we show that while the accuracy of EdgeBoost is slightly lower, we can achieve comparable accuracy to this method without requiring a router, i.e., which (a) requires additional memory to host the router on the resource-constrained edge device, (b) is trained in a complex manner, (c) is often customized for different datasets and tasks.

Finally, training the OFA model with an early exit layer results in an edge accuracy of 68.15% (6.7M parameters at the edge) and 60.2% (1.5M parameters at the edge), see Table 2. This baseline also performs worse than all other methods at all levels of cloud coverage. For instance, for the 6.7M configuration, when 90% of samples exit early(leaving only 10% to be handled by the full network on the cloud), we achieve an accuracy of 69.1%. This is 7.6 percent points lower than the accuracy achieved with 10% cloud coverage by EdgeBoost using MobileNetV3 Large at the edge. Overall, using this model at the edge, EdgeBoost achieves an average accuracy gain up to 8.47 percent point compared to the early exit network. Furthermore, while the highest accuracy the early exit network can achieve is limited to the accuracy of the network itself (70% in this case), achieving higher accuracy would require retraining the entire model with a significantly more powerful network. In contrast, with EdgeBoost, we can simply replace the cloud model with a more powerful one (see Section 4.4.2) to

Table 2
Comparison of EdgeBoost with baselines. The accuracy of the cloud model, for comparison with Entropy and Routing based model, is 79.9%. EdgeBoost provides nearly the same accuracy as the routing-based model at different coverage levels while using less memory at the edge. In case of early exit, the final cloud model is trained jointly with an early exit layer which results in cloud accuracy of 70%. At all coverage levels, EdgeBoost achieves higher accuracy than early exit baseline.

Edge Model	Edge Accuracy (%)	Method	Cloud Coverage		
			10%	20%	30%
MobnetV3 Small (2.5M)	67.6	Entropy	70.7	73.3	74.9
		Routing based Model	71.6	74.6	76.8
		EdgeBoost	70.9	74.2	76.6
MobnetV3 Large (5.3M)	75.7	Entropy	77.1	78.3	78.9
		Routing based Model	77.6	79.0	79.6
		EdgeBoost	76.7	78.5	79.7
OFA Early Exit (6.7M)	68.15	–	69.1	69.9	70.5
OFA Early Exit (1.7M)	60.02	–	62.4	64.4	66.2



(a) Communication Latency between [30,120]ms (b) Constant Communication Latency (50 ms)

Fig. 7. Latency of EdgeBoost compared to an all-Edge and all-Cloud solution. EdgeBoost achieves cloud accuracy with lower latency compared to an all-Cloud deployment scenario.

achieve higher accuracy without the need for any significant change to the system architecture, making it a more scalable and flexible solution for edge–cloud setups.

4.4.4. *EdgeBoost latency analysis*

In the results presented above, we show that EdgeBoost acts as an effective offloading system to achieve near-cloud accuracy while sending only a fraction of the input samples to the cloud. In this section, we analyze the latency implications of various deployment scenarios on the ImageNet-1k dataset using MobileNetV3-Small as the edge model and EfficientNetV2-L in the cloud. In particular, we consider three scenarios:

1. EdgeBoost: MobileNetV3 processes images locally and selectively offloads to the cloud based on confidence thresholds. In this scenario, we consider the threshold value at which we achieve cloud accuracy.
2. All-Edge: All processing is done locally on MobileNetV3-Small.
3. All-Cloud: All images are processed by EfficientNetV2-L in the cloud.

The latency of MobileNetV3-Small is 15.8 ms on a Pixel 1 device [22] while that of EfficientNetV2-L large is 98 ms on a V100 GPU [26]. We assume the network transmission latency to be between 30–120 ms, based on real-world 4G LTE uplink speeds and ImageNet-1k image sizes. Fig. 7(a) shows the total inference latency as a function of communication latency (in log scale) for all three deployment scenarios. As shown, the all-Cloud scenario shows a linear increase in total inference latency with increasing communication latency, due to the added delay in transmitting data to the cloud. In comparison, while the latency of EdgeBoost also increases with increasing communication latency, it remains lower than the all-Cloud scenario since the cloud accuracy is achieved by only offloading 73% of the samples to the cloud. The all-Edge scenario maintains a constant and the lowest total inference latency, regardless of the communication latency, because all processing occurs locally on the edge device. In Fig. 7(b), we show the latency composition for a communication latency of 50 ms. For EdgeBoost, we observe an average latency of 123.84 ms per inference. This includes a local latency of 15.8 ms, an average network transmission delay of 36.5 ms (as 73% of the samples are offloaded to achieve cloud accuracy), and a cloud processing time averaging 71.54 ms for the offloaded samples. In comparison, the All-Edge scenario has a latency of 15.8 ms. Finally, when we send all the samples to the cloud for processing, we achieve a latency of 148 ms per inference, due to the combined latency of the network transmission time (50 ms) and the cloud computation time (98 ms). Overall, EdgeBoost achieves cloud accuracy with lower latency compared to the All-Cloud scenario.

4.4.5. *EdgeBoost as an abstaining classifier*

In this section, we explore the effectiveness of using the edge model as an abstaining classifier within the EdgeBoost framework. We remove the cloud model from the experimental setup and rely only on the edge model for predictions. The edge model withholds predictions for the samples for which it lacks sufficient confidence ensuring that only confident predictions are locally accepted when the cloud model is unavailable. Thus, the accuracy reported for this experiment is the accuracy of the edge model only and the cloud model is not queried for any input samples, compared to the results reported in Fig. 4, where the accuracy is the combined accuracy obtained from both edge and cloud.

To evaluate the impact of calibration on abstention, we conduct experiments using both the calibrated and uncalibrated MobileNetV3-Small models trained on CIFAR-100, Stanford Cars and ImageNet-1k. We test both these models for different thresholds, such that the predictions of the samples with confidence falling below the threshold are

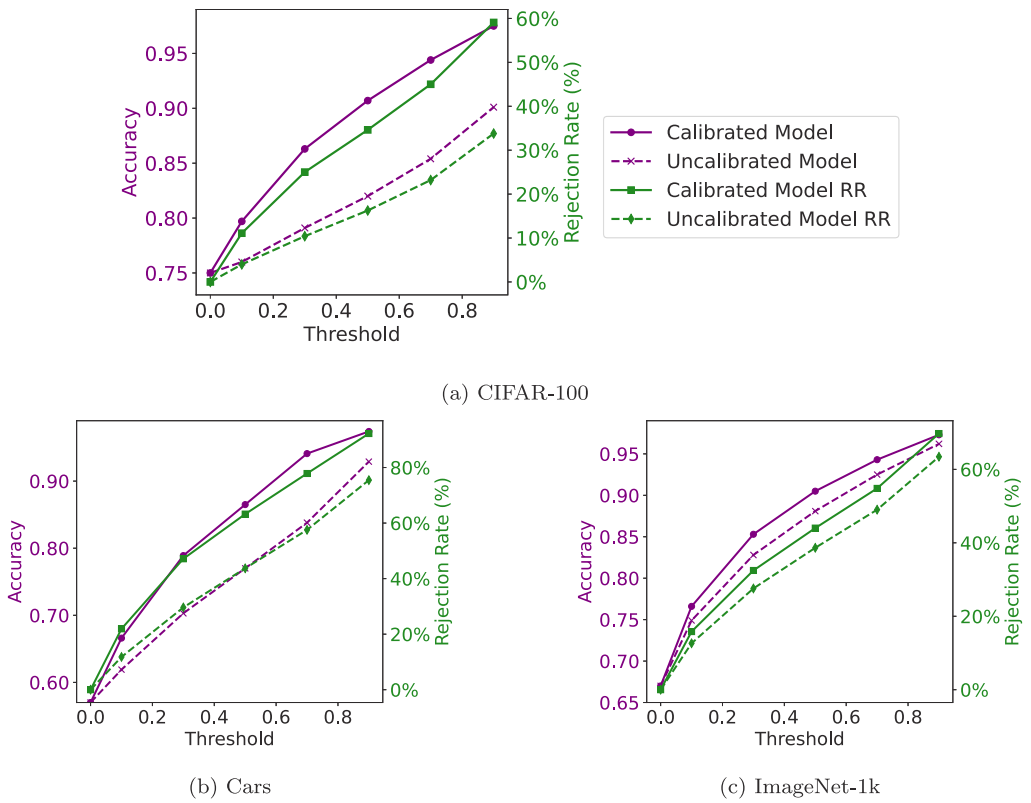


Fig. 8. Accuracy comparison of calibrated and non-calibrated edge models, used as an abstaining classifier. At different threshold levels, the calibrated model is able to capture the uncertain samples more effectively than the uncalibrated one, as shown by the increase in accuracy.

rejected. Fig. 8 shows that the rejection rate for the calibrated and uncalibrated model differs for similar threshold values. For CIFAR-100, 8(a), at a threshold value of 0.3, the calibrated model rejects predictions for 25% of the samples, resulting in an accuracy of 86% while the uncalibrated model rejects 11% of the samples as uncertain resulting in an accuracy of 78%. Similarly, at a threshold value of 0.7, the calibrated model rejects predictions for 45% of the samples compared to only 23% by the uncalibrated model resulting in an accuracy increase of 9 percent points. We observe similar trends for the Stanford Cars 8(b), and ImageNet-1k 8(c) datasets, where the uncalibrated model has a lower rejection rate than the calibrated model at all thresholds. This lower rejection rate of the uncalibrated model indicates that this model cannot effectively capture samples with uncertain prediction compared to the calibrated model, resulting in lower accuracy.

4.4.6. Performance of edge model on the high confidence samples

To further demonstrate the effectiveness of model calibration, we now focus on the samples for which the edge model has high confidence. Our goal is to illustrate that while an uncalibrated edge model may have high confidence in its predictions, its accuracy on these high-confidence samples is significantly lower compared to the calibrated model. We choose the edge model trained on the Stanford Cars dataset because it has the highest calibration error among the models used in this study. By setting different confidence thresholds, we measure the accuracy of both models on samples they classify as easy. Fig. 9 shows that the accuracy of the calibrated model is consistently higher than that of the uncalibrated model across all thresholds. For example, when we set the decision threshold to 0.5, the uncalibrated model has an accuracy of 68%, while the calibrated model achieves 78%. These results highlight the effectiveness of the calibration process, which ensures that high-confidence predictions are more reliable and accurate, thereby improving the trustworthiness of the edge model.

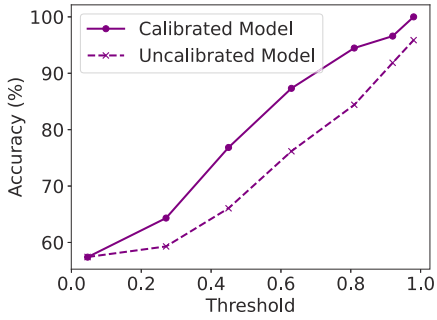


Fig. 9. Accuracy comparison of calibrated and uncalibrated edge models for high confidence samples. At different threshold levels, the calibrated model achieves higher accuracy than the uncalibrated model on these samples.

4.4.7. Analysis of easy and hard samples

To gain further insights into EdgeBoost's performance, we analyze the samples that are classified as easy and hard at the edge. The goal is to illustrate how the system effectively distinguishes between these cases. To demonstrate this, we set the decision threshold at the edge to 0.6. Thus, any sample with confidence equal to or greater than the threshold is processed locally at the edge (easy sample), otherwise, the cloud prediction is requested (hard sample). In Fig. 10, we show randomly selected hard and easy images from both the Cars and ImageNet-1k datasets. Below each image, the probability score of the edge model (MobileNetV3-Small) is shown, along with symbols indicating whether the edge and cloud model (EfficientNetV2-L) correctly classified the sample. As the Figure shows, the edge model often classifies hard samples incorrectly in both datasets. However, the cloud model frequently classifies these samples correctly. Overall, the figures visualize that low-confidence predictions tend to be incorrect

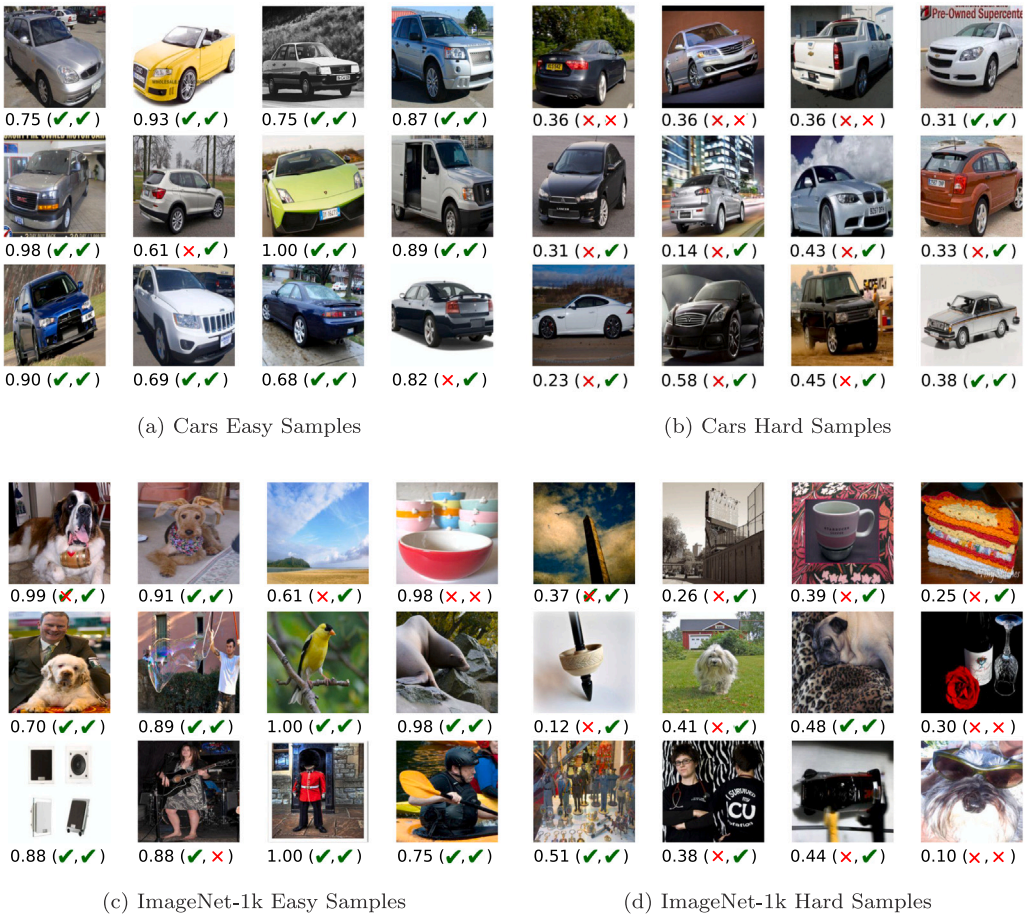


Fig. 10. Samples identified as easy and hard by `EdgeBoost` in the Stanford Cars and ImageNet-1k dataset. The samples identified as easy are mostly correctly classified by the edge model. On the other hand, the edge model is less accurate for the hard samples. Therefore, it is useful to offload these samples to the cloud to get more accurate predictions from a powerful cloud model.

when handled by the edge model only, but offloading these hard cases to the cloud results in improved accuracy. We summarize the accuracy of the edge model on the easy and hard samples for all three datasets in Table 3.

5. Related work

The high computational cost associated with modern DNNs has motivated researchers to explore methods for efficient inference, especially for devices with limited memory and compute power. Several approaches exist, including lightweight models, model compression, split computing, early exit strategies, model partitioning, and hybrid models that combine edge and cloud inference. In the following subsections, we discuss key related work in each category.

5.1. Lightweight models

One approach to reduce inference complexity is to design lightweight neural networks. MobileNetV1 [1] uses depth-wise and point-wise convolutions for size reduction. Others such as MobileNetV2 [30] and MobileNetV3 [22] reduce the model size and improve accuracy. For extremely resource-constrained devices such as microcontrollers, MCUNet [4] achieves ImageNet-1k-scale inference with up to 63.5% accuracy. While most of these small models are often designed manually, many studies in the literature also propose automating the process by using Neural Architecture Search (NAS) [31]. For example, [2] proposes NASNet, a family of Convolutional Neural Networks designed using NAS. This study uses a small dataset to search for a good architecture and then transfers the learned architecture to ImageNet-1k. Some studies [3,32] also use the concepts of NAS to design architectures tailored to the resources available on specific devices. Although designing lightweight models offers a straightforward solution, it often involves a trade-off between complexity and accuracy, making high-performance goals challenging. In contrast, `EdgeBoost` effectively utilizes a lightweight edge model and a powerful cloud model to achieve high accuracy while maintaining communication efficiency.

5.2. Model compression

Network compression is another approach to “compress” a large model into a smaller one. Two notable compression are pruning and quantization [33–35]. Pruning removes parameters from a network while quantization uses fewer bits to represent them. [36] achieves an improved tradeoff between accuracy and inference time by quantizing MobileNet, compared to the original floating point version. On the other hand, [37] reduces the computational cost of CNN inference by pruning filters. While pruning and quantization are promising techniques to reduce the model size, they are often not sufficient on their own and also degrade the accuracy of DNN models. Knowledge Distillation [38] is another approach used for model compression. While pruning and quantization reduce the size of trained models, Knowledge Distillation uses a large model (called the teacher) to guide the training process of a smaller model (called the student). Knowledge from large model is distilled into a small model thereby increasing its accuracy without increasing the model complexity. Similar to lightweight models, aggressive pruning and quantization can also degrade model

Table 3
Accuracy of MobileNetV3-Small on samples identified as hard and easy at a threshold value of 0.6. For all three datasets, the accuracy of model is low on the hard samples hence requiring cloud offload.

Model	Dataset	Accuracy (%)	
		Hard samples	Easy samples
MobnetV3 Small	CIFAR100	45.6	89.8
	Cars	36.9	85.2
	ImageNet-1k	39.0	88.6

performance. EdgeBoost, on the other hand, improves the accuracy by leveraging both a lightweight edge model and a powerful cloud model.

5.3. Split computing

To leverage the computational power of the cloud for edge-based applications, split computing executes a model partially, typically the first layers of a DNN, on a resource-constrained edge device and then transmits intermediate results, i.e., feature maps, to the cloud for further processing. [23,39] searches for the best partitioning layer in a DNN to reduce latency and energy consumption. Compressive offloading [40–42] extends this approach, by compressing the intermediate output (or the raw input signal) using a lightweight neural network to further reduce communication requirements. [43,44] use lossy compression techniques for intermediate DNN outputs. While these approaches provide high classification accuracy, they have a significant drawback: For each input, communication between an edge and a cloud device is needed. This communication is often wireless, i.e., via WiFi or cellular technologies, and thus time and energy consuming [40]. Another study [45] proposes Edgent, which dynamically allocates a part of the DNN computation to the edge server, instead of the cloud. This approach reduces the latency by moving the computation closer to the source. However, it still requires communication with the edge server for each input sample. In addition, because edge servers have fewer computational resources than the cloud, they may struggle to efficiently handle computationally intensive tasks, limiting scalability in high-demand scenarios. EdgeBoost, on the other hand, processes confident predictions locally and offloads only the most challenging samples to a powerful model deployed on the cloud, thereby reducing both latency and communication costs while maintaining high accuracy.

5.4. Early exit networks

Early exit [8–10] adds branches to a neural network, computes later layers only if the output of an early layer does not provide sufficient classification confidence. [46] divides a DNN inference pipeline into multiple stages and incorporates early exit points to enable energy-efficient inference, while the authors in [47] propose a strategy to dynamically adjust the losses of early exit points during training. [48,49] combine Knowledge Distillation with early exit where all early exits act as students and their final classifier acts as the teacher model. Combined with split computating, early exits reduce communication overhead in DNNs [50]. However, this approach often suffers from the problem that the early layers, i.e., those deployed on an edge device, typically do not provide sufficient classification accuracy [13]. Moreover, the accuracy of an early exit network is limited by the performance of the model itself, and improving it requires retraining the entire network with a more powerful architecture. On the other hand, the design of EdgeBoost allows accuracy improvement by simply replacing the cloud model with a more powerful one, without requiring significant changes to the system architecture.

5.5. Model partitioning

Model partitioning techniques allow IoT devices to jointly execute a DNN by distributing the computational workload among them [51]. Many approaches in the literature, such as [52,53], and [54], partition the model vertically by assigning layer-by-layer computations to different devices in the IoT environment. Although these methods assign different layers to different devices, they fail to fully utilize the available computational resources due to the lack of intra-layer parallelism. To address this limitation, many recent studies, including [55–57], propose horizontal partitioning. These methods allow devices to execute a model's layers in parallel and distribute the input data. However, these approaches largely ignore the varying computational and network resources of IoT networks. DISNET [58] addresses this gap by combining horizontal and vertical partitioning while taking into account the IoT devices' computational and communication resources. This work enables resource-aware cooperative DNN inference. Although distributed ML effectively utilizes the collective computational resources of IoT devices, it introduces high communication overhead and synchronization complexity due to frequent inter-device data exchanges. In contrast, EdgeBoost eliminates the need for complex inter-device synchronization by leveraging a lightweight edge model for confident predictions and selectively offloading only challenging samples to the cloud, thereby reducing communication overhead while maintaining high inference accuracy.

5.6. Neural network routers

Some works [12,13] propose techniques where an edge and a cloud model are trained together with a so-called "router". The router decides whether an input sample should be processed locally or should be sent to the cloud for prediction. While such methods achieve good results, deploying the router along with the model at the edge requires a custom router design, additional memory at the edge and offer custom training procedures. EdgeBoost, in contrast, achieves near-cloud accuracy by sending only a fraction of the input samples to the cloud depending on the hardness. It achieves a performance comparable to router-based approaches, while relying only on the output probability of the calibrated edge model to determine the confidence.

6. Conclusion

In this paper, we introduce EdgeBoost, a selective input offloading system for resource-constrained edge devices. In contrast to an all-Cloud solution where all samples must be sent to the cloud, EdgeBoost offloads only those samples to the cloud for which an edge model is not confident in its prediction. On the large-scale ImageNet-1k dataset, for example, we reduce the communication costs by 27% and achieve comparable accuracy to an all-Cloud solution with lower latency. We also show that calibrating the neural network at the edge results in capturing the less confident prediction more effectively. For example, it increases the overall accuracy of the system by up to 8 percent points. By using a calibrated model at the edge as an abstention classifier, we achieve an accuracy gain of 9 percent points over an uncalibrated model. EdgeBoost also outperforms the early exit and entropy thresholding methods and achieves comparable accuracy to routing based offloading methods without the need to host the router on the resource-constrained edge devices.

CRediT authorship contribution statement

Naina Said: Writing – original draft, Investigation, Conceptualization. **Olaf Landsiedel:** Supervision.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Naina Said reports financial support was provided by Kiel University. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This project has received funding from the Federal Ministry for Economic Affairs and Climate Action under the Marispace-X project grant no. 68GX21002E and the Federal State of Schleswig-Holstein under the DataCampus grant no. 22021016 and INSYST grant no. 22023008.

Data availability

I have shared the code.

References

[1] A.G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017, arXiv preprint arXiv:1704.04861.

[2] B. Zoph, V. Vasudevan, J. Shlens, Q.V. Le, Learning transferable architectures for scalable image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 8697–8710.

[3] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, Q.V. Le, Mnasnet: Platform-aware neural architecture search for mobile, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 2820–2828.

[4] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han, et al., Mcunet: Tiny deep learning on iot devices, Adv. Neural Inf. Process. Syst. 33 (2020) 11711–11722.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2009, pp. 248–255.

[6] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., An image is worth 16x16 words: Transformers for image recognition at scale, 2020, arXiv preprint arXiv:2010.11929.

[8] S. Teerapittayanon, B. McDanel, H.-T. Kung, Branchynet: Fast inference via early exiting from deep neural networks, in: 2016 23rd International Conference on Pattern Recognition, ICPR, IEEE, 2016, pp. 2464–2469.

[9] Y. Matsubara, M. Levorato, F. Restuccia, Split computing and early exiting for deep learning applications: Survey and research challenges, ACM Comput. Surv. 55 (5) (2022) 1–30.

[10] L. Lebovitz, L. Cavigelli, M. Magno, L.K. Muller, Efficient inference with model cascades, Trans. Mach. Learn. Res. (2023).

[11] E. Park, D. Kim, S. Kim, Y.-D. Kim, G. Kim, S. Yoon, S. Yoo, Big/little deep neural network for ultra low power inference, in: 2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS), IEEE, 2015, pp. 124–132.

[12] M. Li, Y. Li, Y. Tian, L. Jiang, Q. Xu, AppealNet: An efficient and highly-accurate edge/cloud collaborative architecture for DNN inference, in: 2021 58th ACM/IEEE Design Automation Conference, DAC, IEEE, 2021, pp. 409–414.

[13] A. Kag, I. Fedorov, A. Gangrade, P. Whatmough, V. Saligrama, Efficient edge inference by selective query, in: The Eleventh International Conference on Learning Representations, 2022.

[14] A. Krizhevsky, G. Hinton, et al., Learning multiple layers of features from tiny images, 2009.

[15] J. Krause, M. Stark, J. Deng, L. Fei-Fei, 3D object representations for fine-grained categorization, in: Proceedings of the IEEE International Conference on Computer Vision Workshops, 2013, pp. 554–561.

[16] C. Guo, G. Pleiss, Y. Sun, K.Q. Weinberger, On calibration of modern neural networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 1321–1330.

[17] T. Pearce, A. Brintrup, J. Zhu, Understanding softmax confidence and uncertainty, 2021, arXiv preprint arXiv:2106.04972.

[18] M.P. Naeini, G. Cooper, M. Hauskrecht, Obtaining well calibrated probabilities using bayesian binning, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 29, (1) 2015.

[19] Y. Gal, Z. Ghahramani, Dropout as a bayesian approximation: Representing model uncertainty in deep learning, in: International Conference on Machine Learning, PMLR, 2016, pp. 1050–1059.

[20] B. Zadrozny, C. Elkan, Transforming classifier scores into accurate multiclass probability estimates, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002, pp. 694–699.

[21] B. Zadrozny, C. Elkan, Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers, in: Icml, vol. 1, 2001, pp. 609–616.

[22] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, et al., Searching for mobilenetv3, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1314–1324.

[23] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, L. Tang, Neurosurgeon: Collaborative intelligence between the cloud and mobile edge, ACM SIGARCH Comput. Archit. News 45 (1) (2017) 615–629.

[24] F. Nan, V. Saligrama, Adaptive classification for prediction under a budget, Adv. Neural Inf. Process. Syst. 30 (2017).

[25] T. Bolukbasi, J. Wang, O. Dekel, V. Saligrama, Adaptive neural networks for efficient inference, in: International Conference on Machine Learning, PMLR, 2017, pp. 527–536.

[26] M. Tan, Q. Le, Efficientnetv2: Smaller models and faster training, in: International Conference on Machine Learning, PMLR, 2021, pp. 10096–10106.

[27] Y. Fang, Q. Sun, X. Wang, T. Huang, X. Wang, Y. Cao, EVA-02: A visual representation for neon genesis, 2023, arXiv preprint arXiv:2303.11331.

[28] H. Cai, C. Gan, T. Wang, Z. Zhang, S. Han, Once-for-all: Train one network and specialize it for efficient deployment, 2019, arXiv preprint arXiv:1908.09791.

[29] A. Hatamizadeh, H. Yin, G. Heinrich, J. Kautz, P. Molchanov, Global context vision transformers, in: International Conference on Machine Learning, PMLR, 2023, pp. 12633–12646.

[30] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 4510–4520.

[31] B. Zoph, Neural architecture search with reinforcement learning, 2016, arXiv preprint arXiv:1611.01578.

[32] J.-D. Dong, A.-C. Cheng, D.-C. Juan, W. Wei, M. Sun, Dpp-net: Device-aware progressive search for pareto-optimal neural architectures, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 517–531.

[33] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, Adv. Neural Inf. Process. Syst. 28 (2015).

[34] C. Buciluǎ, R. Caruana, A. Niculescu-Mizil, Model compression, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006, pp. 535–541.

[35] S. Han, H. Mao, W.J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015, arXiv preprint arXiv:1510.00149.

[36] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 2704–2713.

[37] D. Filters'Importance, Pruning filters for efficient ConvNets, 2016.

[38] J. Ba, R. Caruana, Do deep nets really need to be deep? Adv. Neural Inf. Process. Syst. 27 (2014).

[39] G. Li, L. Liu, X. Wang, X. Dong, P. Zhao, X. Feng, Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge, in: Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part I 27, Springer, 2018, pp. 402–411.

[40] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, T. Abdelzaher, Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency, in: Proceedings of the 18th Conference on Embedded Networked Sensor Systems, 2020, pp. 476–488.

[41] B. Chen, Z. Yan, H. Guo, Z. Yang, A. Ali-Eldin, P. Shenoy, K. Nahrstedt, Deep contextualized compressive offloading for images, in: Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems, 2021, pp. 467–473.

[42] B. Chen, Z. Yan, K. Nahrstedt, Context-aware image compression optimization for visual analytics offloading, in: Proceedings of the 13th ACM Multimedia Systems Conference, 2022, pp. 27–38.

[43] H. Choi, I.V. Bajić, Deep feature compression for collaborative object detection, in: 2018 25th IEEE International Conference on Image Processing, ICIP, IEEE, 2018, pp. 3743–3747.

[44] R.A. Cohen, H. Choi, I.V. Bajić, Lightweight compression of neural network feature tensors for collaborative intelligence, in: 2020 IEEE International Conference on Multimedia and Expo, ICME, IEEE, 2020, pp. 1–6.

[45] E. Li, L. Zeng, Z. Zhou, X. Chen, Edge AI: On-demand accelerating deep neural network inference via edge computing, IEEE Trans. Wirel. Commun. 19 (1) (2019) 447–457.

[46] K. Neshatpour, F. Behnia, H. Homayoun, A. Sasan, Exploiting energy-accuracy trade-off through contextual awareness in multi-stage convolutional neural networks, in: 20th International Symposium on Quality Electronic Design, ISQED, IEEE, 2019, pp. 265–270.

[47] M. Wang, J. Mo, J. Lin, Z. Wang, L. Du, Dynexit: A dynamic early-exit strategy for deep residual networks, in: 2019 IEEE International Workshop on Signal Processing Systems, SiPS, IEEE, 2019, pp. 178–183.
[48] H. Li, H. Zhang, X. Qi, R. Yang, G. Huang, Improved techniques for training adaptive deep networks, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1891–1900.
[49] M. Phuong, C.H. Lampert, Distillation-based training for multi-exit architectures, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 1355–1364.
[50] D.J. Bajpai, V.K. Trivedi, S.L. Yadav, M.K. Hanawal, SplitEE: Early exit in deep neural networks with split computing, 2023, arXiv preprint arXiv:2309.09195.
[51] A.F. Rocha Neto, F.C. Delicato, T.V. Batista, P.F. Pires, Distributed machine learning for iot applications in the fog, Fog Comput.: Theory Pr. (2020) 309–345.
[52] Y. Gao, M. Kim, C. Thapa, A. Abuadbba, Z. Zhang, S. Camtepe, H. Kim, S. Nepal, Evaluation and optimization of distributed machine learning techniques for internet of things, IEEE Trans. Comput. 71 (10) (2021) 2538–2552.
[53] W. He, S. Guo, S. Guo, X. Qiu, F. Qi, Joint DNN partition deployment and resource allocation for delay-sensitive deep learning inference in IoT, IEEE Internet Things J. 7 (10) (2020) 9241–9254.
[54] C. Hu, W. Bao, D. Wang, F. Liu, Dynamic adaptive DNN surgery for inference acceleration on the edge, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 1423–1431.
[55] L. Zeng, X. Chen, Z. Zhou, L. Yang, J. Zhang, Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices, IEEE/ACM Trans. Netw. 29 (2) (2020) 595–608.
[56] J. Mao, X. Chen, K.W. Nixon, C. Krieger, Y. Chen, Modnn: Local distributed mobile computing system for deep neural network, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017, IEEE, 2017, pp. 1396–1401.
[57] Z. Zhao, K.M. Barijough, A. Gerstlauer, Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 37 (11) (2018) 2348–2359.
[58] E. Samikwa, A. Di Maio, T. Braun, Disnet: Distributed micro-split deep learning in heterogeneous dynamic IoT, IEEE Internet Things J. (2023).

Naina Said is a doctoral researcher in the Department of Computer Science at Kiel University, Germany. Her research focuses on resource-efficient artificial intelligence, edge computing, and distributed systems, with a particular interest in optimizing inference under constrained conditions.

Before joining Kiel University in 2022, she worked as a research assistant and instructor at the University of Engineering and Technology (UET) Peshawar, Pakistan. From 2017 to 2019, she was affiliated with the STAMP research lab and later with the National Center of Artificial Intelligence (NCAI), both research initiatives funded by the Higher Education Commission (HEC) of Pakistan at UET Peshawar. Her work there focused on multimedia analytics and disaster response using social media data.

Olaf Landsiedel is head of the Institute for Networked Cyber Physical Systems (NCPS) at Hamburg University of Technology (TUHH), Germany. His research focuses on Distributed and Networked Systems, and, in particular, the Internet of Things (IoT), Cyber-Physical Systems (CPS), TinyML, Edge AI, and Edge & Fog Computing. He is also an Adjunct Professor at Kiel University, Germany.

Before joining TUHH, he was Professor for Computer Science at Kiel University, leading the Distributed Systems Group from 2018 to 2025. 2012 to 2018, he was an Assistant and later tenured Associate Professor at Chalmers University of Technology, Sweden. And from 2010 to 2012 he spent two years as a Postdoctoral fellow at the Royal Institute of Technology (KTH), Sweden, and the Swedish Institute of Computer Science (SICS, now RISE), Sweden. He received his PhD from RWTH Aachen, Germany, in 2010.