EdgeBoost: Confidence Boosting for Resource Constrained Inference via Selective Offloading

Naina Said, Olaf Landsiedel
Kiel University, Germany
{nas,ol}@informatik.uni-kiel.de

***Abstract*—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. Due to careful calibration, `EdgeBoost` reduces the communication cost by 55%, 29% and 20% for CIFAR-100, ImageNet and Stanford Cars datasets, respectively, when compared to an cloud-only solution while achieving on par classification accuracy. Finally, `EdgeBoost` also achieves comparable accuracy to the state-of-the-art routing-based methods without the need for hosting the router on the edge.**

***Index Terms*—TinyML, MCU, Model Calibration, Temperature Scaling, Inference Offloading, Lightweight Models**

I. Introduction

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

Lightweight models [1]–[4], tailored to the resource-constraints of edge devices, however, often show reduced predictive performance. For example, for the popular ImageNet [5] dataset, the lightweight MobileNet model [1] with 4.3M parameters suffers from accuracy loss of about 7% compared to the state-of-the-art the ResNet152 model with 60.4M 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. However, its size of 307M trainable parameters makes it impractical for deployment on resource-constrained edge devices.

To address this challenge, some approaches split the input data at runtime into easy and hard samples [8]–[10]: The edge device uses a lightweight model to classify the easier samples locally while sending the harder ones to the cloud for processing with a more powerful network. As lightweight models commonly show a good performance when classifying easy inputs, this design improves the overall classification performance, as we also show in our evaluation in Section IV. To separate easy and hard samples, some approaches directly use output probabilities of the edge model [8]. 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 [9], [10]. This pre-classifier, however, adds extra overhead and often needs to be designed and trained specifically for each dataset.

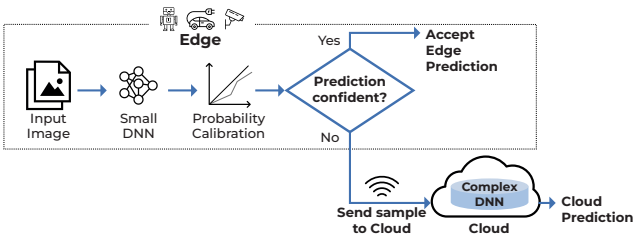


Fig. 1: Overview of `EdgeBoost`. First, `EdgeBoost` passes an input sample to a small, less powerful DNN on the edge. Next, we analyze the confidence of the prediction made by the edge. If the confidence of the prediction is sufficiently high, we accept the local classification result. Otherwise, the 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.

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, which exceed the local model's abilities, and offloads these to a large model in the cloud.

For the ImageNet dataset, `EdgeBoost` achieves an accuracy on par to the cloud by offloading 71% of the input samples to the cloud, hence saving 29% communication cost compared to an all-cloud system where all the input samples need to be offloaded to the cloud. For CIFAR-100 [11] and Stanford Cars [12], we reduce communication cost by 55% and 20% respectively, while achieving cloud accuracy. Our evaluation also demonstrates the effectiveness of 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 datasets, respectively.

Overall, this paper makes the following contributions:

- Design and development of `EdgeBoost`, a lightweight system for intelligent cloud offloading based on the input hardness that achieves cloud accuracy with upto 55% reduced communication cost on CIFAR-100, 29% on ImageNet 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 dataset, respectively.
- Demonstration that `EdgeBoost` achieves comparable accuracy to state-of-the-art routing-based approaches without the need for an extra classifier on the edge. Our system also outperforms entropy thresholding by achieving a nearly 1.7 percent points increase in accuracy.

The source code for EdgeBoost and the experiments presented in this paper is available online[1]. The remainder of the paper is organized as follows: Section II presents the required background on DNN confidence and calibration, Section III introduces the design of `EdgeBoost`, Section IV evaluates `EdgeBoost` on three datasets and on both MobileNetV3 Small and MCUNet architectures and Section V discusses related work. Section VI concludes the paper.

II. Background: Confidence and Calibration

In this section, we introduce the required background on DNN confidence and calibration. We begin by introducing the basics of model confidence and discussing the need for calibration. Next, we introduce Expected Calibration Error (ECE), a metric to measure the degree of calibration of a NN, and Temperate Scaling, a lightweight method for calibration of NNs. Finally, we discuss how we utilize the probabilities of the calibrated models to determine their confidence.

A. Model Confidence and Calibration

The confidence of a model is a metric for how confident a model is about its classification result for a specific input. Thus, if it predicts that an input sample is of a specific class with a confidence of X, this 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 [13], [14]. As a result, uncalibrated DNNs often produce probability estimates that are not well-aligned with the true probabilities. Further, as we show in our evaluation in Section IV, 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 on the edge is crucial because it dictates whether a local prediction is accepted or the sample is sent to the cloud for classification. Thus, our approach demands a good calibration, and we introduce methods for calibration later in this section.

B. Expected Calibration Error

Expected Calibration Error (ECE) [15] is a metric to measure 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 over the absolute difference between the accuracy *acc* and confidence *conf* as shown in Equation 1. Here, *B* represents "bins", *m* the bin number and *n* represents the total number of samples. The difference between *acc* and *conf* for a particular bin represents the calibration gap. The range of ECE is between 0 and 1 where 0 represents poor calibration and 1 represents perfect calibration. Throughout this research, we use ECE as a measure to represent the calibration error of neural networks.

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

C. Temperature Scaling for NN Calibration

Temperature scaling [13] is a simple yet effective method for re-calibration of prediction probabilities. Temperature scaling uses a single scalar parameter $Temp > 0$, where *Temp* is the temperature, to rescale logit scores of a NN before applying the softmax function as shown in Equation 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 impact the accuracy of the model since the maximum of the softmax remains unchanged.

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

Next to Temperature Scaling, further methods such as Monte Carlo Dropout [16], extension of binning methods to Isotonic Regression [17] and Histogram binning [18] enable NN calibration. However, we select Temperature Scaling due to its simple implementation and superior performance over other techniques [13]

D. Calculation of Prediction Confidence

After calibrating a model, `EdgeBoost` utilizes the softmax margin [8] 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. 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 Equation 3. A higher difference represents more confidence, whereas a small difference value represents less confidence of the model on its prediction.

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

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

III. 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 costs. Finally, we detail on the methodology for offloading samples to the cloud, emphasizing decision-making based on calibrated confidence levels.

A. 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 [13], [14], the calibration of models tailored for resource constrained devices has received very little attention as of today. Thus, as 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, see Table I. Despite the efficiency and good accuracy of resource-efficient models, our results show that the majority of these models show a miscalibration after training, evident from their respective ECE values. Just MCUNet shows very little miscalibration and does not require further calibration. Others, such as the MobileNetV3 Small [19] model show strong miscalibration and on the CIFAR-100 dataset, exhibits a reduction in ECE from 14.8% to 2.3% after applying Temperature Scaling for post-training calibration. Overall, the results underline that Temperature Scaling effectively boosts model calibration.

B. `EdgeBoost`: Design Overview

Based on the insights we derived from the analysis of model calibration in Section III-A, we can now introduce a simple and lightweight system design: `EdgeBoost` employs two models, one resource-efficient on the edge and a second large model on the cloud, see Figure 1. Further, we calibrate the neural network on the edge using Temperature Scaling, as introduced in Section II). For each input, we first use the edge model to make a prediction. Next, we employ 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.

C. Assessing Confidence & Tuning the Confidence Threshold

We utilize the Probability Margin, i.e., the difference between the highest and second-highest classification probabilities (see Section II), as a metric to assess the confidence of the calibrated edge model. In `EdgeBoost`, a user-defined threshold parameter, denoted as *T*, acts as a critical decision boundary to determine whether to process a sample locally by the edge model or forward it to the cloud model. Choosing a value for *T* resembles a trade-off between prediction accuracy and communication cost: A higher *T* value results in the edge model predicting more conservatively, i.e., accepting fewer of the local classification results. Consequently, `EdgeBoost` sends more samples to the cloud for processing, which incurs higher communication overhead. Conversely, a lower *T* value leads to the edge model handling a greater proportion of samples, reducing communication costs but compromising prediction accuracy, especially for challenging samples. In Section IV), we evaluate overall accuracy and communication overhead for different values of *T* to elaborate this behavior. Practically, we should choose a *T* value that satisfies the cost/accuracy requirements of a specific application. As the metric merely acts as a threshold, the confidence checker is designed lightweight and resource-efficient.

D. Algorithmic Details

After training the models on the edge and cloud, we follow the process in algorithm 1 to upload samples to the cloud. First, we pass the input sample to the edge model without the softmax to obtain the logit values. We then calibrate the logits for the input sample using the temperature value *Temp*, pre-calculated using the validation set in line 3 before applying the softmax. This calibration yields reliable probability estimates. Next in line 7, we calculate the difference between the highest and second-highest probabilities. This difference, serving as a measure of confidence, is compared with the threshold value *T*. A large difference indicates the edge model's high confidence in its prediction, leading us to accept the local prediction. If not, we send the sample to the cloud model for further prediction.

Algorithm 1 Inference Using `EdgeBoost`

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

IV. Evaluation

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

A. Baselines

We benchmark Edgeboost against four baselines:

Dataset	Model	Accuracy (%)	# of Parameters (M)	ECE Uncalibrated (%)	ECE 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	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

TABLE I: Calibration of resource-efficient models such as MobileNet and MCUNet for common datasets including CIFAR-100 and ImageNet. As evident from the ECE value, most of these models are poorly calibrated after training and require Temperature Scaling for post training calibration. Merely MCUNet-in0 and MCUNet-in1 have a negligible calibration error and do not require further calibration.

1) All-Edge Baseline: A baseline representing the scenario when the edge device solely carries out the inference without utilizing the capability of the cloud.

2) All-Cloud Baseline: In contrast to an all edge approach, this baseline entirely relies on cloud for inference.

3) Entropy-based Baseline: This approach uses the entropy-based decision mechanism for deciding between edge and cloud. We select this baseline based on the findings of authors in Kag et al. [10], where they find entropy thresholding superior to others such as AppealNet [9], BranchyNet [20] and other adaptive networks [21]–[23].

4) Routing-based Baseline: This baseline involves designing and training a router to determine the difficulty of the input samples, which is then deployed on the edge in addition to the base model. We select Kag et al. [10] as a baseline among the routing-based solutions due to its superior performance to other similar prior techniques. It is a hybrid system that trains a base, a router, and a cloud model such that, on average, coverage on-device is maximized without sacrificing accuracy.

B. Experimental Setup

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 dataset. For MCUNet we use multiple configurations (mcunet-in0, mcunet-in2, mcunet-in4) with 0.75, 0.73, and 1.73 Million parameters reaching an accuracy of 41.5%, 60.9% and 68.4% on the ImageNet dataset, respectively. As cloud model, we employ EfficientNetv2L [24], a state-of-the-art model for image classification which achieves on par-performance to modern vision transformers. It has 118.5M parameters and accuracy of 86%, 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 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 utilize the small version of EfficientNetv2 (21.45M parameters and 85% accuracy on ImageNet) for the experiments on Stanford Cars dataset.

To ensure reproducible results, we utilize pre-trained networks EfficientNetv2 (large and small version), 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 Figure 2.

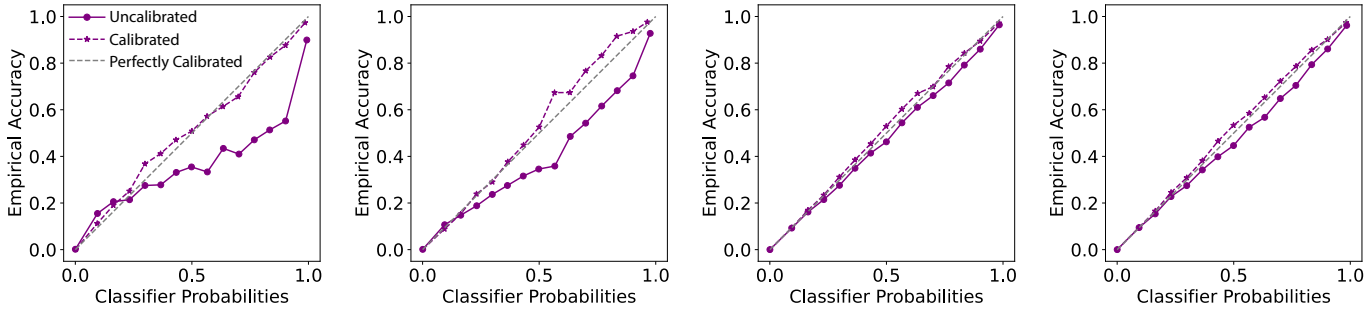
2) For Comparison with State-of-the-art: We compare our results to entropy thresholding and routing based method as discussed in Section IV-A. We utilize the largest pre-trained model from the OFA space [25] which achieves 79.9% accuracy on ImageNet as a cloud model while as our edge model, we experiment with two versions of MobileNetV3 Small, one with 2.5M parameters and another one having 5.4M parameters. We select these models for fair comparison with state-of-the-art as they use these models, too.

C. Experimental Results

1) Model Calibration & Temperate Scaling: At the start of the design section, we analyze the calibration of modern, resource-efficient neural networks, see Section III and Table I. The main result of this analysis is that (a) many neural networks lack calibration, and (b) Temperature Scaling drastically improves their calibration. These results motivate the design of `EdgeBoost`.

In our evaluation, we now dive deeper into the model calibration. Figure 2 shows the calibration of MobileNetV3 for CIFAR100, Stanford Cars and ImageNet and the calibration of MCUNet for ImageNet. The results show, that for CIFAR100 and Stanford Cars, MobileNetV3 is overconfident. 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 Figure 2a) and Stanford Cars (see Figure 2b), respectively. Temperate Scaling practically removes this bias. For the ImageNet dataset, we do

[2]EfficientNetv2L, 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 (MobileNetV3 S) (d) ImageNet (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 across different levels of predicted confidence. The ideal model's predictions would lie on the 'Perfectly Calibrated' line, indicating that the predicted probabilities match the empirical probabilities. The curves after calibration in these diagrams are close to the perfectly calibrated curve, indicating an improvement in the models' trustworthiness for confidence calculation. Thus, Temperature Scaling proves to be effective to calibrate models with high calibration error.

see similar effects but at a much lower significance, see Figure 2c and Figure 2d for MobileNetV3 and MCUNet, respectively.

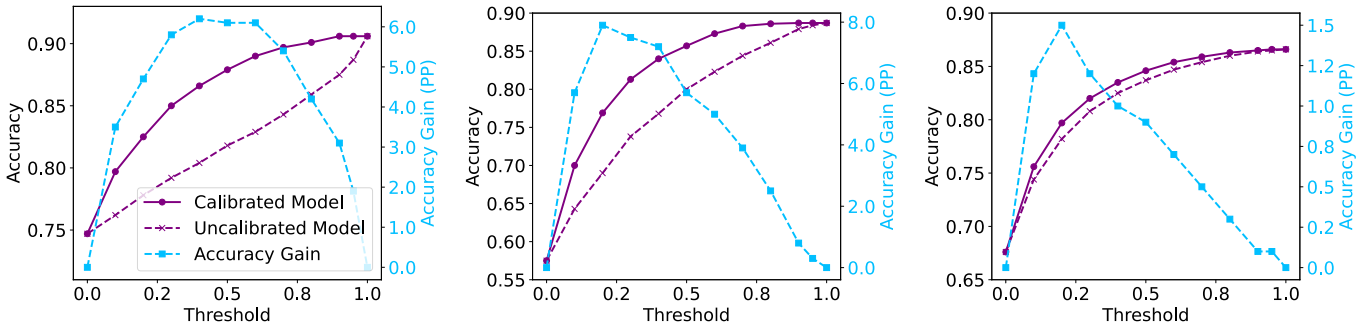
For all our experiments, we calibrate the edge models using Temperature Scaling. In Figure 2 we omit the plots for mcunet-in0 and mcunet-in2 as these models show negligible calibration error making calibration unnecessary, We only show mcunet-in0 in Figure 2d as an example.

2) Benefits of Calibration: 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. Figure 3 shows the accuracy of `EdgeBoost` when the edge model is calibrated and uncalibrated for different user-define thresholds values. We find 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 instance, at a threshold value of 0.1, the uncalibrated model achieves accuracy of 76% whereas the calibrated model achieves accuracy of 80% for the CIFAR100 dataset when using MobileNetV3 Small as edge model (see Figure 3a). At a threshold value of 0.4, the accuracy of the calibrated model is 6 percent points more than the uncalibrated model. Further, at any threshold the calibrated model outperforms the uncalibrated one in terms of accuracy. We observe the same trend for the in Stanford Cars dataset in Figure 3b. For ImageNet model (Figure 3c), the accuracy gains are the smallest because the ECE value difference between the calibrated and uncalibrated value is small. Nonetheless, we see improvements of upto 1.5 percent points after calibration.

3) Comparison with All-edge and All-Cloud Baselines: Next, we compare EdgeBoost to all edge and all cloud method baselines, see Figure 4. We evaluate three datasets (CIFAR-100, Stanford Cars and ImageNet) and four networks (MobileNetV3 Small and three MCUNet variants). Figure 4 shows the relationship between the user-defined confidence threshold for the edge model and three key performance indicators: overall system accuracy, average communication cost per input image, and the percentage of samples processed by the cloud. Firstly, 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 different datasets. However, as accuracy approaches 100%, 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 Figure 4a. 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 cost 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 Figure 4b. 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, thus decreasing the communication cost by 20% when compared to an all-cloud solution of equivalent accuracy. For ImageNet in Figure 4c, we approach cloud accuracy of 86% at a coverage level of 71%, i.e., saving 29% 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 Figure 4d. At 83% cloud coverage, cloud accuracy is achieved. Similarly, mcunet-in4 (Figure 4f) and mcunet-in2 (Figure 4e) achieve cloud accuracy at 60% and 73% cloud coverage, respectively. Overall, we find EdgeBoost to be an effective system to achieve high inference performance while sending only subset of the samples to the cloud.

4) Comparison with State-of-the-Art Approaches: Next, we compare `EdgeBoost` with state-of-the-art methods, namely



(a) CIFAR-100 (MobileNetV3 S) (b) Stanford Cars (MobileNetV3 S) (c) ImageNet (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.

Edge Model	Edge Accuracy (%)	Method	10% coverage Acc	20% coverage Acc	30% Coverage Acc
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

TABLE II: Comparison of `EdgeBoost` with baselines. Cloud model accuracy is 79.9%. `EdgeBoost` offers almost the same accuracy as routing based model at different coverage levels while keeping the memory requirements on the edge lower.

entropy-based thresholding and a routing-based model as we discussed in Section IV-A., see Table II. In the routing-based model, which we use as a baseline, the decision router comprises a two layer DNN with 256 neurons in the first layer and 64 in the second. Hence, in this setting, the edge device needs to host both the prediction model and this router. Further, the router is trained together with the edge model.

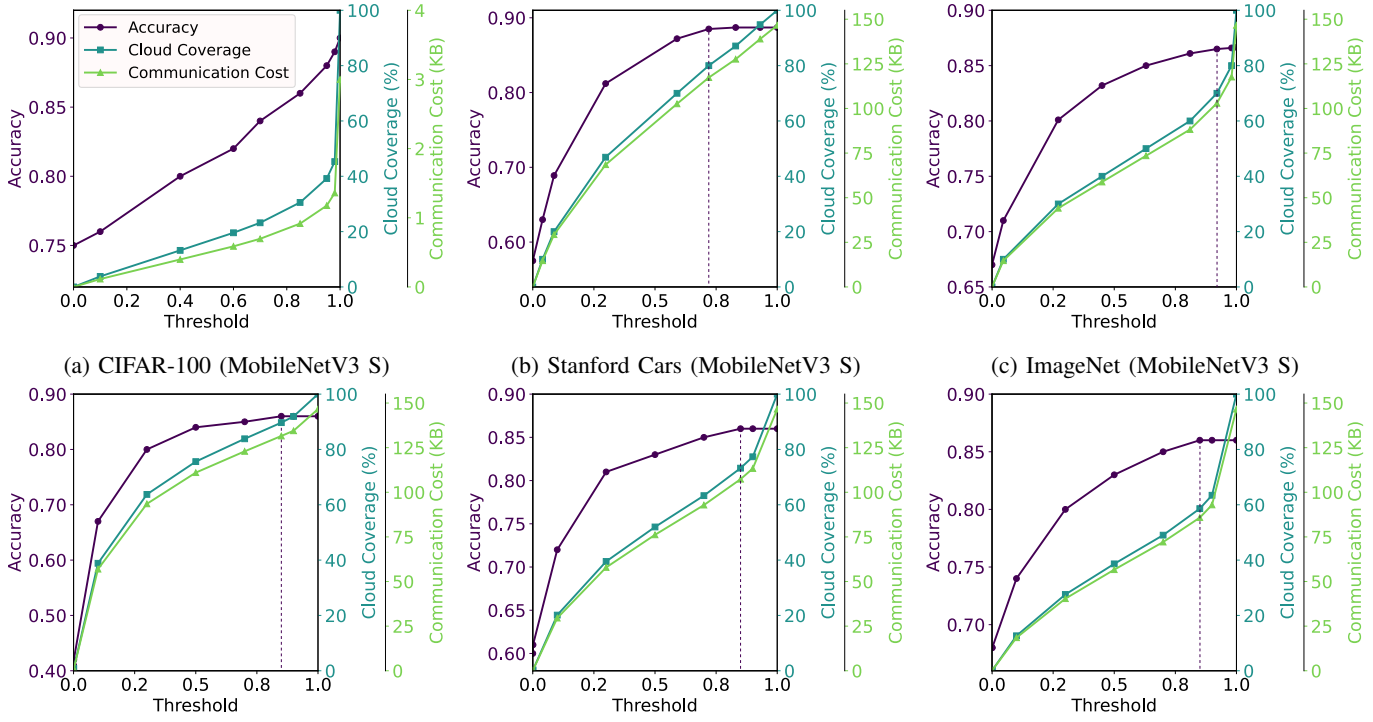
First, we observe that the `EdgeBoost` achieves near-cloud accuracy using the 5.3M MobileNetV3 at 30% coverage. Hence, reducing the communication cost by 70%. Also note that at 30% coverage, the smaller MobileNetV3 with 2.5M parameters exceeds the base accuracy of the larger version. Thus, we achieve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

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) needs additional memory to host the router on the resource-constrained edge device, (b) is trained in a complex manner, (c) often is customized for different datasets and tasks.

V. Related Work

The high computation cost associated with modern DNNs has motivated researchers to explore methods for efficient inference, particularly for devices with limited memory. Although designing lightweight models offers a straightforward solution, it often involves a trade-off between complexity and accuracy, challenging high-performance goals. MobileNetV1 [1] employs depth-wise and point-wise convolutions for size reduction. Others such MobileNetV2 [26] and MobileNetV3 [19] reduce the model size and improve accuracy. For extremely resource constrained devices like microcontrollers, MCUNet [4] achieves ImageNet-scale inference with up to 63.5% accuracy. Additionally, network compression techniques [27], [28] minimize DNN parameters and computational cost.

To leverage the computational power of the cloud for edge-based applications, split computing executes a model partially, usually 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. Compressive offloading [29]–[31] extends this approach, by compressing the intermediate output (or the raw input signal) with the help of a lightweight neural network to further reduce communication demands. While providing high classification accuracy, these approaches have an essential 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 thereby time and energy-demanding [29]. Early exit [20], [32], [33] adds branches to a neural network, and only computes later layers if the output of an early layer does not provide sufficient classification certainty. Combined with split computing, early exits in DNNs reduce the communication overhead [34]. However, this approach often suffers from the problem that the early layers, i.e., the ones deployed on an edge device, commonly



(a) CIFAR-100 (MobileNetV3 S) (b) Stanford Cars (MobileNetV3 S) (c) ImageNet (MobileNetV3 S)

(d) ImageNet (mcunet-int0) (e) ImageNet (mcunet-int2) (f) ImageNet (mcunet-int4)

Fig. 4: `EdgeBoost` compared to an all cloud and all edge solution on three standard datasets (CIFAR-100, Stanford Cars, ImageNet). The cloud model for CIFAR-100 and ImageNet experiments is EfficientNetv2 large and for Stanford Cars is EfficientNetv2 small model while the edge model used is MobileNetV3 small and three versions of MCUNet (in0,in2,in4). The accuracy improves for all three datasets when a small fraction of samples is sent to the cloud. The dotted lines represent the threshold at which cloud accuracy is achieved. Hence, `EdgeBoost` improves the accuracy on the edge by sending less confident samples to the cloud.

do not provide sufficient classification accuracy [10].

Some works [9], [10] propose hybrid models where an edge and cloud model is trained along 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 on the edge requires a custom router design, additional memory on the edge and offer custom training procedures.

`EdgeBoost`, in contrast, achieves near cloud accuracy by only sending a fraction of input samples to the cloud depending on the hardness. It achieves a performance comparable to router-based approaches, while merely relying on the output probability of the calibrated edge model to determine the confidence.

VI. Conclusion

In this paper, we propose `EdgeBoost`, a selective input offloading system for resource constrained edge devices. In contrast to an all-cloud solution where all samples need to be sent to the cloud to achieve cloud accuracy, `EdgeBoost` only offloads those samples to the cloud for which an edge model is not confident on its prediction. On the large-scale ImageNet dataset, for example, we reduce the communication cost by 29% in comparison to an all-cloud solution. We further show that calibrating the neural network on the edge results in capturing the less confident prediction more effectively. For example, it increases the overall accuracy of the system upto 8 percent points. `EdgeBoost` also achieves comparable accuracy to routing based offloading methods without the need to host the router on the resource constrained edge devices.

Acknowledgment

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.

References

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

[2] B. Zoph, V. Vasudevan, J. Shlens, and 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, and 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," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 711–11 722, 2020.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and 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, and 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," *arXiv preprint arXiv:2010.11929*, 2020.

[8] E. Park, D. Kim, S. Kim, Y.-D. Kim, G. Kim, S. Yoon, and 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.

[9] M. Li, Y. Li, Y. Tian, L. Jiang, and 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.

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

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

[12] J. Krause, M. Stark, J. Deng, and 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.

[13] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International conference on machine learning*. PMLR, 2017, pp. 1321–1330.

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

[15] M. P. Naeini, G. Cooper, and M. Hauskrecht, "Obtaining well calibrated probabilities using bayesian binning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 29, no. 1, 2015.

[16] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*. PMLR, 2016, pp. 1050–1059.

[17] B. Zadrozny and 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.

[18] ——, "Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers," in *Icml*, vol. 1, 2001, pp. 609–616.

[19] 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.

[20] S. Teerapittayanon, B. McDanel, and 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.

[21] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[22] F. Nan and V. Saligrama, "Adaptive classification for prediction under a budget," *Advances in neural information processing systems*, vol. 30, 2017.

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

[24] M. Tan and Q. Le, "Efficientnetv2: Smaller models and faster training," in *International conference on machine learning*. PMLR, 2021, pp. 10 096–10 106.

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

[26] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and 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.

[27] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 535–541.

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

[29] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and 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.

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

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

[32] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–30, 2022.

[33] L. Lebovitz, L. Cavigelli, M. Magno, and L. K. Muller, "Efficient inference with model cascades," *Transactions on Machine Learning Research*, 2023.

[34] D. J. Bajpai, V. K. Trivedi, S. L. Yadav, and M. K. Hanawal, "Splitee: Early exit in deep neural networks with split computing," *arXiv preprint arXiv:2309.09195*, 2023.