On the impact of deep neural network calibration on adaptive edge offloading for image classification

Roberto G. Pacheco[a,*], Rodrigo S. Couto[a], Osvaldo Simeone[b]

[a] *Universidade Federal do Rio de Janeiro - PEE/COPPE/GTA*
[b] *King's College London, KCLIP lab, Department of Engineering, London, United Kingdom*

Abstract

Edge devices can offload deep neural network (DNN) inference to the cloud to overcome energy or processing constraints. Nevertheless, offloading adds communication delay, which increases the overall inference time. An alternative is to use adaptive offloading based on early-exit DNNs. Early-exit DNNs have branches inserted at the output of given intermediate layers. These side branches provide confidence estimates. If the confidence level of the decision produced is sufficient, the inference is made by the side branch. Otherwise, the edge offloads the inference decision to the cloud, which implements the remaining DNN layers. Thus, the offloading decision depends on reliable confidence levels provided by the side branches at the device. This article provides an extensive calibration study on different datasets and early-exit DNNs for the image classification task. Our study shows that early-exit DNNs are often miscalibrated, overestimating their prediction confidence and making unreliable offloading decisions. To evaluate the impact of calibration on accuracy and latency, we introduce two novel application-level metrics and evaluate well-known DNN models in a realistic edge computing scenario. The results demonstrated that calibrating early-exit DNNs improves the probabilities of meeting accuracy and latency requirements.[1]

*Corresponding author.
Email addresses: pacheco@gta.ufrj.br (Roberto G. Pacheco), rodrigo@gta.ufrj.br (Rodrigo S. Couto), osvaldo.simeone@kcl.ac.uk (Osvaldo Simeone)

 DOI: https://doi.org/10.1016/j.jnca.2023.103679

Keywords: early-exit deep neural networks; edge computing; deep neural network calibration; edge offloading

1. Introduction

Deep Neural Networks (DNNs) have presented an astonishing improvement in their accuracy over the years [1, 2, 3, 4]. To achieve the accuracy required for intelligent applications, DNN models have become increasingly deeper, requiring high processing power to make inferences. For this reason, mobile devices generally gather raw data and offload DNN inference tasks to a cloud server. The server can be equipped with the required computational resources, such as Graphics Processing Units (GPUs) [5, 6]. However, cloud offloading may introduce communication delays due to the network conditions between the mobile device and the cloud server. Edge computing has emerged as a solution to circumvent excessive delays and communication overhead [5].

Edge computing moves computing resources closer to the end device, e.g., by deploying computational resources on base stations [7]. This way, edge computing can reduce the communication delay inherent in a cloud-based deployment. However, edge devices may have significantly less processing power compared to a cloud server and may not be able to support the desired reliability levels. DNN model partitioning has been introduced to alleviate this problem by performing an edge-cloud co-inference [6, 8, 9].

DNN partitioning chooses a partitioning layer to split the DNN model into two parts. A mobile device gathers an input (e.g., an image) and sends it to the edge device, which processes the input layer-by-layer until the partitioning layer. Next, the edge device offloads the partitioning layer's output data to the cloud server, which processes the remaining layers. The partitioning layer is chosen by employing an optimization problem for a given objective, such as reducing the inference time [9] or saving energy on the edge device [10]. This approach always offloads data to the cloud server, and its performance in terms of latency is hence dependent on network conditions. Early-exit DNNs have emerged as a

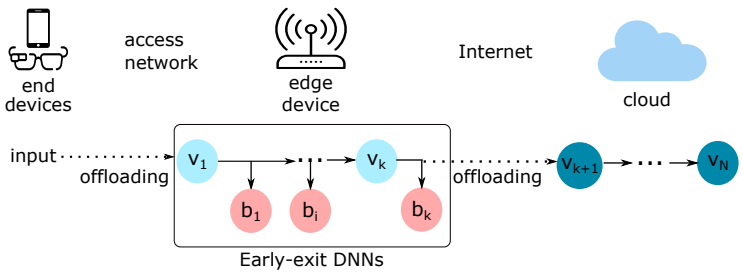


Figure 1: Illustration of adaptive model partitioning between the edge device and the cloud via early-exit DNNs.

more flexible alternative to DNN partitioning that can strike a more desirable balance between edge and cloud processing [11, 12, 13].

Early-exit DNNs leverage the fact that not all inputs are equally difficult to classify. Therefore, distinct inputs may require a different number of neural layers to achieve a sufficiently reliable confident prediction. Early-exit DNNs consist of a DNN backbone into which side branches are inserted along intermediate layers, as shown in Figure 1. Given an input, side branches estimate the accuracy of their inference outputs based on a confidence metric. Then, they compare the obtained confidence with an application-defined threshold to decide if the prediction is sufficiently accurate. Hence, early-exit DNNs can accelerate inference time by avoiding the need to run all layers since a significant portion of inputs may be classified at side branches. BranchyNet [11] and SPINN [12] allows the classification of a sample at side branches when the estimated accuracy is larger than a tunable threshold. Early-exit DNNs and DNN partitioning can be combined to implement an adaptive offloading strategy, illustrated in Figure 1, whereby classification at the edge occurs if a side branch provides a sufficiently accurate prediction. Otherwise, offloading to the cloud is carried out to process the remaining layers and perform remote classification [14, 15].

A successful adaptive offloading via early-exit DNNs requires that the side branches provide an accurate measure of uncertainty. This measure should gauge whether the inference decisions are sufficiently confident to be terminated on a side branch [16]. To address this problem, DNN calibration emerges as an

alternative approach.

DNN calibration refers to the process of adjusting the confidence estimates produced by the DNN to align with the actual accuracy of its predictions. In other words, calibration aims to ensure that the confidence estimates accurately reflect its actual prediction accuracy. Guo *et al.* [17] experimentally demonstrated that DNNs are typically miscalibrated, making overconfident predictions. An overconfident side branch would erroneously decide to classify an input at the edge device, decreasing the accuracy.

While reference [17] did not address the calibration problem in early-exit DNNs, this work fills this gap by analyzing the impact of calibration for adaptive offloading via early-exit DNNs[2]. SPINN [12] also applied a calibration method in an offloading scenario. Nevertheless, reference [12] does not discuss or analyze the impact of calibration on adaptive offloading, which is the focus of our work. In the context of adaptive offloading, early-exit DNN calibration is particularly crucial because the offloading decision depends on reliable confidence levels provided by the side branches at the edge device. If the confidence estimates are miscalibrated and the prediction accuracy is overestimated, the offloading decisions may be unreliable, decreasing the overall accuracy. First, we show via experiments that miscalibration significantly affects the offloading decision's reliability. Next, we calibrate early-exit DNN models by applying the state-of-the-art temperature-scaling calibration method introduced in [17].

We evaluate the impact of calibration using state-of-the-art models, demonstrating an improvement in terms of two novel metrics, namely the edge inference outage probability and the missed deadline probability. The edge outage probability quantifies the ability of an early-exit DNN model to meet a required accuracy level for inference at the edge device. In other words, this metric quantifies the probability that an edge device fails to meet the accuracy requirement when performing the inference locally at the edge device. Therefore, this metric is a key indicator of the reliability and effectiveness of edge computing systems,

[2]A preliminary version of this work appeared in [14].

particularly in resource-constrained devices. On the other hand, the missed deadline probability accounts for the probability of meeting accuracy and inference latency requirements when performing the inference at the edge or the cloud. This metric is particularly relevant in applications that demand a critical deadline. A high missed deadline probability indicates that the system cannot meet the required performance objectives and may need to adjust the offloading decisions or allocate additional resources to meet the requirements.

We evaluate the proposed metrics by implementing a realistic edge computing scenario encompassing an NVIDIA Jetson Nano board as the edge computing and an Amazon EC2 (Elastic Compute Cloud) instance as the cloud server. We instantiate the cloud servers in two countries to evaluate network conditions. Our evaluations experimentally demonstrate that early-exit DNN calibration can improve accuracy due to more reliable offloading decisions, leading to better overall performance. However, it comes at the cost of additional communication delay due to more offloading between the edge device and the cloud. This delay can impact overall inference time and should be carefully considered in designing an adaptive offloading system.

In summary, we list this article's contributions as follows:

- We evaluate the calibration impact on several state-of-the-art early-exit DNNs and experimentally demonstrate that the side branches are *over-confident*. Hence, they can wrongly decide to classify samples at the edge device.

- We adopt the state-of-the-art temperature-scaling calibration method from [17] to address the outlined calibration problem. We experimentally evaluate the impact of calibration in terms of two novel metrics, namely the edge inference outage probability and the missed deadline probability, providing extensive experimental insights.

This article is organized as follows. Section 2 reviews related work. Section 3 describes the adaptive offloading scenario via early-exit DNNs. We present the adopted concepts and notation in Section 4. Section 5 presents the proposed

metrics that evaluate the calibration impact considering application-defined requirements. Then, Section 6 analyzes the impact of calibration on several early-exit DNN models. Next, Section 7 uses these metrics to evaluate the calibration impact on a realistic adaptive edge offloading scenario. Finally, Section 8 concludes this article and presents the next research steps.

2. Related Work

In the conference version of this work [14], we analyze calibration for early-exit DNNs by considering AlexNet. This article extends the analysis to a variety of early-exit DNN models. Furthermore, we implement a realistic edge computing scenario, while our former work uses simulation-based estimates of processing and communications delays. The following review of works focuses on early-exit DNNs, DNN model partitioning, and DNN model calibration.

2.1. Early-Exit DNNs

Early-exit DNNs have been applied to different tasks, such as image classification [11, 12, 15], image segmentation [18, 19], and natural language processing [20, 21].

Many works have proposed advances in early-exit DNN design. BranchyNet [11] allocates branches at the beginning of the DNN architecture to classify the input as soon as possible. SPINN [12] inserts the side branches equidistantly along the DNN backbone. HAPI [22] co-optimizes the early-exit placement and threshold fine-tuning to meet application requirements. Similarly, FlexDNN [23] optimizes the side branches placement along the DNN backbone to find an optimal early-exit DNN architecture, aiming to maximize the benefits of the early-exit strategy. Our article follows SPINN's methodology on how to place the side branches to simplify the analysis, focusing on the calibration aspect.

Other works, such as Li*et al.* [24] and Farhadi *et al.* [25], adapt DNN model complexity according to the input to perform a device-edge co-inference scenario. However, they do not calibrate the DNN. Specifically, Li*et al.* [24] present

the framework Edgent, which enables device-edge co-inference via an early-exit strategy, adjusting the appropriate DNN depth to balance the benefits of using powerful cloud resources and nearby edge resources. Meanwhile, Farhadi *et al.* [25] terminate the inference using a lightweight DNN model at the device or send the inference to a deeper one at the edge. Similarly, Dong *et al.* [26] developed an early-exit module that guides difficult samples through the DNN backbone, bypassing the computation of side branches. This approach reduces on-device computation required at the edge device. Samikwa *et al.* [27] explore the benefits of employing the early-exit strategy in IoT environments. Their work analyzes the reduction in inference time and energy consumption using an early-exit strategy in IoT environments. Samikwa *et al.* [27] conclude that adopting an early-exit strategy in an IoT environment can reduce inference time and energy consumption under varying network conditions and computing resources.

Other articles focus on accelerating the early-exit inference by hardware, using Field-Programmable Gate Array (FPGAs) or hardware for edge computing, such as Nvidia Jetson Nano board [28, 29]. Our work employs an Nvidia Jetson Nano board as the edge device to follow this direction.

All the works mentioned previously, except for SPINN [12], do not calibrate early-exit DNN models. However, SPINN does not analyze the impact of calibration. In contrast, this article conducts a comprehensive study of the calibration impact on offloading decisions provided by early-exit DNNs.

2.2. DNN Model Partitioning

DNN model partitioning splits a DNN model in a fixed manner for a given offloading objective. Most existing works choose the partitioning layer, at which the offloading occurs, using an algorithm that minimizes different metrics, such as inference time and energy consumption [6, 8, 9, 12, 30]. Our work considers the partitioning layer at the last side branch.

2.3. DNN Model Calibration

The calibration of a DNN model refers to the extent to which the model's predictive confidence reflects the accuracy of its decisions [31]. DNNs output confidence levels along with their decisions through uncertainty metrics such as the top-1 value of a softmax output layer [12]. For a perfectly calibrated DNN, a confidence value of $\alpha \in [0, 1]$ for a given subset of inputs implies that the model correctly classifies $\alpha \cdot 100\%$ of such inputs.

Several works have reported that DNNs, especially Convolutional Neural Networks (CNNs) such as ResNets [3] and MobileNet [2], are poorly calibrated, despite the growing improvements in terms of accuracy [17, 32, 31]. More specifically, Guo *et al.* [17] demonstrate that more accurate DNNs often imply a worse calibration, typically providing overconfident predictions. To address the calibration problem, reference [17] evaluates several calibration methods for different datasets. Their work concludes that *temperature scaling* [17] is the most effective calibration method, especially for computer vision applications. However, Guo *et al.* do not address the calibration problem in early-exit DNNs and its impact on edge inference offloading. Minderer *et al.* [31] suggest that recent non-convolutional DNN models, such as MLP-Mixer [33] and Vision Transformers [34], are well calibrated as compared to CNN models. Bayesian learning offers a general framework to ensure the calibration of machine learning models [35]. Our work focuses on early-exit DNNs that use CNNs as the backbone architecture.

Table 1 summarizes the related work and highlights their differences compared to this study. In Table 1, we consider that a paper employs a realistic adaptive offloading scenario if the work implements edge-cloud co-inference by deploying an early-exit DNN at the edge and evaluates it using a real network infrastructure.

Table 1: Comparison with related works.

Reference	Classifies images	Employs early-exit DNNs	Calibrates the model	Analyzes the impact of calibration	Considers a realistic adaptive offloading scenario
[11]	Yes	No	No	No	No
[12]	Yes	Yes	Yes	No	Yes
[15]	Yes	Yes	Yes	No	Yes
[18, 19]	No	Yes	No	No	No
[20, 21]	No	Yes	No	No	No
[25]	No	No	No	No	Yes
[26]	Yes	Yes	No	No	No
[24]	Yes	No	No	No	Yes
[28, 29]	Yes	Yes	No	No	No
[27]	Yes	Yes	No	No	No
[6, 30]	Yes	No	No	No	No
[8]	Yes	No	No	No	No
[9]	Yes	Yes	No	No	Yes
[17, 32, 31]	Yes	No	Yes	Yes	No
[14]	Yes	Yes	Yes	Yes	No
This work	Yes	Yes	Yes	Yes	Yes

3. Adaptive Offloading Via Early-exit DNNs

DNNs can be described as a sequence of neural layers that can extract features from inputs. In general, shallow DNNs (i.e., DNNs with few neural layers) can extract simple features, while deeper DNN models can extract more complex features and obtain more accurate predictions. Early-exit DNNs classify some inputs based on feature representation obtained by shallow neural layers, while other inputs rely on features provided by deeper layers to be classified. The intuition behind this approach is that distinct samples may not require

features of equal complexity to be classified [11].

Figure 1 illustrates an early-exit DNN with multiple side branches inserted into its intermediary layers. The vertices $v_1, \cdots, v_N$ represent the DNN backbone's layers. The vertices $b_1, \cdots, b_k$ are the side branches, each of which contains a fully-connected layer capable of classifying inputs based on the features extracted in the previous layers $v_1, \cdots, v_i$. Following SPINN's methodology, we insert side branches equidistantly after intermediate layers along the DNN backbone. In the experimental results, we insert five side branches as [12]. Thus, we have six exits, including the DNN backbone's output layer.

After placing the side branches, we split the image dataset into training, validation, and testing sets. We train the early-exit DNN following the methodology presented by BranchyNet [11] for an image classification task. We employ the softmax cross-entropy loss function as the objective function for each exit.

Let $(\boldsymbol{x}, \boldsymbol{y})$ be an example, with input $\boldsymbol{x}$ and one-hot class vector $\boldsymbol{y}$; C be the set of possible classes, and $\hat{\boldsymbol{p}}_i(\boldsymbol{x}|\boldsymbol{\theta})$ be the probability vector provided by the i-th exit for model parameter vector $\boldsymbol{\theta}$. The probability vector $\hat{\boldsymbol{p}}_i(\boldsymbol{x}|\boldsymbol{\theta})$ provides an estimate of the probability that an input $\boldsymbol{x}$ belongs to each of the predefined classes. The loss function for example $(\boldsymbol{x}, \boldsymbol{y})$, at the i-th exit, is defined as

$$\mathcal{L}_i(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta}) = -\frac{1}{|C|} \sum_{c \in C} y_c \cdot \log(\hat{p}_{i,c}(\boldsymbol{x}|\boldsymbol{\theta})), \tag{1}$$

where y_c and $\hat{p}_{i,c}(\boldsymbol{x}|\boldsymbol{\theta})$ are respectively the c-th elements of vectors $\boldsymbol{y}$ and $\hat{\boldsymbol{p}}_i(\boldsymbol{x}|\boldsymbol{\theta})$, which correspond to class $c \in C$. The probability vector $\hat{\boldsymbol{p}}_i(\boldsymbol{x}|\boldsymbol{\theta})$ is computed through the softmax layer as in

$$\hat{\boldsymbol{p}}_i(\boldsymbol{x}|\boldsymbol{\theta}) = \text{softmax}(\boldsymbol{z}_i(\boldsymbol{x}|\boldsymbol{\theta})) \propto \exp(\boldsymbol{z}_i(\boldsymbol{x}|\boldsymbol{\theta})), \tag{2}$$

where $\boldsymbol{z}_i(\boldsymbol{x}|\boldsymbol{\theta})$ is the logit vector provided by i-th exit's fully-connected layer, and the exponential function in (2) is applied element-wise.

To optimize the early-exit DNN model, we jointly train all the exits by adopting an overall loss function given by the weighted sum of the loss function of each exit as $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta}) = \sum_{i=1}^{N} \omega_i \cdot \mathcal{L}_i(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta})$, where N is the number of exits,

and the weights ω_i increase linearly according to the exit position in the early-exit DNN model [12]. The loss function $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{\theta})$ is averaged over all examples $(\boldsymbol{x}, \boldsymbol{y})$ in the training set.

As illustrated in Figure 1, the early-exit DNN is split into two parts: the first part being implemented on the edge device and the second remotely at the cloud server. Side branches are included only in the first part. At inference time, given an input $\boldsymbol{x}$, the device estimates the prediction confidence for the i-th side branch as the probability of the most likely class

$$f_i(\boldsymbol{x}|\boldsymbol{\theta}) = \max_{c \in C} \hat{p}_{i,c}(\boldsymbol{x}|\boldsymbol{\theta}). \tag{3}$$

If the confidence value is greater or equal to a predefined target threshold p_{tar}, the device concludes that the i-th side branch can classify the input and inference terminates by classifying the input as the class with the largest probability in vector $\hat{\boldsymbol{p}}_i(\boldsymbol{x}|\boldsymbol{\theta})$, i.e., the class

$$\hat{y}_i(\boldsymbol{x}|\boldsymbol{\theta}) = \arg\max_{c \in C} \hat{p}_{i,c}(\boldsymbol{x}|\boldsymbol{\theta}). \tag{4}$$

Otherwise, the input is processed by the subsequent layers until it reaches the next side branch, following the same procedure described above.

If no side branches reach the desired accuracy level p_{tar}, the edge device offloads data to the cloud, which processes the remaining DNN backbone's layers until the output layer. The cloud server then sends the prediction and its confidence level back to the edge device. If the confidence level does not reach the target p_{tar}, the classification uses the most confident predicted class among all exits.

We next describe early-exit DNN calibration, which is the focus of this work and a key step in performing reliable offloading.

4. Early-Exit DNN Calibration

An early-exit DNN model is well-calibrated if it outputs a prediction confidence value $f_i(\boldsymbol{x}|\boldsymbol{\theta})$ that reflects the true probability of correct classification, i.e.,

the model's accuracy. Formally, an early-exit DNN model is perfectly calibrated at side branch i if, for all inputs $\boldsymbol{x}$, we have the equality

$$P[\hat{y}_i(\boldsymbol{x}|\boldsymbol{\theta}) = y \mid f_i(\boldsymbol{x}|\boldsymbol{\theta}) = p] = p, \quad \text{for all } p \in [0,1], \tag{5}$$

where $\hat{y}_i(\boldsymbol{x}|\boldsymbol{\theta})$ is the predicted class in (4), $f_i(\boldsymbol{x}|\boldsymbol{\theta})$ is the confidence level (3), and y is the ground-truth label [17]. The probability in (5) is taken with respect to the true distribution of the data when conditioned on the confidence level (3) being equal to p. Equation (5) states that the confidence $f_i(\boldsymbol{x}|\boldsymbol{\theta})$ in (3) provided by the early-exit DNN model perfectly reflects the actual probability of correct classification.

4.1. Temperature Scaling

Temperature scaling (TS) is a post-processing calibration method that scales the early-exit DNN's logit vector $\boldsymbol{z}_i(\boldsymbol{x}|\boldsymbol{\theta})$ for the i-th exit to enhance calibration [17, 31]. With a temperature parameter T, the calibrated probability vector is given by

$$\hat{\boldsymbol{p}}_i^{TS}(\boldsymbol{x}|\boldsymbol{\theta}) = \text{softmax}\left(\frac{\boldsymbol{z}_i(\boldsymbol{x}|\boldsymbol{\theta})}{T}\right), \tag{6}$$

where the scaling is applied element-wise. The parameter T accordingly modifies the estimated confidence value in (3) as

$$f_i^{\text{TS}}(\boldsymbol{x}|\boldsymbol{\theta}) = \max_{c \in C} \hat{\boldsymbol{p}}_{i,c}^{TS}(\boldsymbol{x}|\boldsymbol{\theta}). \tag{7}$$

Note that TS does not impact the accuracy since it scales all elements of $\boldsymbol{z}_i(\boldsymbol{x}|\boldsymbol{\theta})$ by the same constant.

4.2. Optimizing the Temperature

To optimize the temperature T, following [17, 31], we gather the confidences values $f_i(\boldsymbol{x}|\boldsymbol{\theta})$ and the predicted class $\hat{y}_i(\boldsymbol{x}|\boldsymbol{\theta})$ provided by all the exit branches i for each input $\boldsymbol{x}$ in the validation dataset. With this information, we implement TS in DNNs in one of two ways, which are described next.

Global TS: Global TS utilizes the collected confidences levels $\{f_i(\boldsymbol{x}|\boldsymbol{\theta})\}$ and corresponding predicted classes $\{\hat{y}_i(\boldsymbol{x}|\boldsymbol{\theta})\}$ to optimize a single parameter T for all exits i. This is done by minimizing over T the validation loss

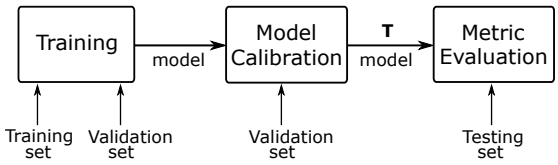


Figure 2: Methodology for calibrating an early-exit DNN model.

$\sum_i \sum_{\boldsymbol{x},\boldsymbol{y}} \mathcal{L}_i^{TS}(\boldsymbol{x},\boldsymbol{y}|\boldsymbol{\theta})$, where $\mathcal{L}_i^{TS}(\boldsymbol{x},\boldsymbol{y}|\boldsymbol{\theta}) = -\frac{1}{|C|}\sum_{c\in C} y_c \cdot \log(\hat{p}_{i,c}^{TS}(\boldsymbol{x}|\boldsymbol{\theta}))$ and the sum is over all validation examples.

Per-branch TS: Per-branch TS uses only the collected confidence levels $\{f_i(\boldsymbol{x}|\boldsymbol{\theta})\}$ and predicted classes $\{\hat{y}_i(\boldsymbol{x}|\boldsymbol{\theta})\}$ provided by the i-th exit to optimize a distinct temperature parameter T_i at each branch. The parameter T_i is optimized by minimizing the validation loss $\sum_{\boldsymbol{x},\boldsymbol{y}} \mathcal{L}_i^{TS}(\boldsymbol{x},\boldsymbol{y}|\boldsymbol{\theta})$, where the sum runs over all validation examples.

Figure 2 presents a diagram summarizing the methodology for calibrating an early-exit DNN model. The "Training" box illustrates the essential procedures required to train and validate the early-exit DNN model, as explained in Section 3. Then, the trained model is saved to the next stage, named "Model calibration", in which calibration is performed using the validation set as explained in Section 4. This stage employs the global TS and per-branch TS methods presented previously to find the optimal temperature vector $\boldsymbol{T}$. This section uses the global TS and per-branch TS methods mentioned earlier to determine a temperature vector $\boldsymbol{T}$. Once the temperature vector is derived, the "Metric evaluation" stage uses the trained model and $\boldsymbol{T}$ to evaluate the performance of calibration under the metrics presented next.

4.3. Calibration Metrics

Reliability diagrams plot the expected accuracy as a function of prediction confidence, providing a common way to visualize the calibration of a model. For each i-th exit, the reliability diagram is obtained by grouping confidence levels $\{f_i(\boldsymbol{x}|\boldsymbol{\theta})\}$ evaluated on the validation data into m equal-sized interval bins $B_{i,1}, \cdots, B_{i,m}$. Each bin $B_{i,j}$ contains inputs $\boldsymbol{x}$ whose confidence $f_i(\boldsymbol{x}|\boldsymbol{\theta})$

lies in the interval $\left(\frac{j-1}{m}, \frac{j}{m}\right]$. The expected accuracy of the i-th exit for each bin $B_{i,j}$ is computed as

$$\mathrm{Acc}(B_{i,j}|\boldsymbol{\theta}) = \frac{1}{|B_{i,j}|} \sum_{\boldsymbol{x} \in B_{i,j}} \mathbb{1}[\hat{y}_i(\boldsymbol{x}|\boldsymbol{\theta}) = y], \tag{8}$$

where the sum is over validation examples $(\boldsymbol{x}, \boldsymbol{y})$ in bin $B_{i,j}$. We similarly evaluate the average confidence of i-th exit within a bin $B_{i,j}$ as

$$\mathrm{Conf}(B_{i,j}|\boldsymbol{\theta}) = \frac{1}{|B_{i,j}|} \sum_{\boldsymbol{x} \in B_{i,j}} f_i(\boldsymbol{x}|\boldsymbol{\theta}). \tag{9}$$

To build a reliability diagram for the i-th side branch, we gather the confidence levels $f_i(\boldsymbol{x}|\boldsymbol{\theta})$ and the predicted class $\hat{y}_i(\boldsymbol{x}|\boldsymbol{\theta})$ for all validation examples. If a DNN model is well-calibrated at branch i, then its reliability diagram should approximate an identity function, as defined in (5).

While reliability diagrams provide a fine-grained visual representation of calibration, the Expected Calibration Error (ECE) returns a single numerical value that quantifies calibration [17]. This metric is computed as

$$\mathrm{ECE} = \sum_{j=1}^{m} \frac{|B_{i,j}|}{n} \cdot |\mathrm{Acc}(B_{i,j}|\boldsymbol{\theta}) - \mathrm{Conf}(B_{i,j}|\boldsymbol{\theta})|, \tag{10}$$

where n is the total number of inputs in the validation dataset. By (10), for each i-th exit, the ECE computes the average per-bin accuracy $\mathrm{Acc}(B_{i,j}|\boldsymbol{\theta})$ and confidence level $\mathrm{Conf}(B_{i,j}|\boldsymbol{\theta})$ weighted by the number of samples in each bin $B_{i,j}$.

5. Application-Level Metrics

In order to evaluate the performance at the application level, we consider the average accuracy of the decisions made at the edge and at the cloud as well as two novel metrics, first introduced in the conference version [14] of this work, which accounts for the interplay of accuracy and offloading latency. The first metric, referred to as edge inference outage probability, measures the model's ability to guarantee an accuracy requirement via inference at the edge. The second

metric, referred to as missed deadline probability, accounts for the probability in meeting both accuracy and inference latency requirements when allowing for inference at the edge or at the cloud.

5.1. Average Accuracy Metrics

We define the average edge accuracy $\mathrm{Acc}_{\mathrm{edge}}(\boldsymbol{\theta})$ as the fraction of test examples classified at any of the edge side branches that are correctly classified. The average total accuracy $\mathrm{Acc}_{\mathrm{total}}(\boldsymbol{\theta})$ is similarly defined as the fraction of test examples that are correctly classified, irrespective of whether detection is done at the edge or at the cloud.

5.2. Edge Inference Outage Probability

In practice, as discussed in Section 3, one is interested in attaining the desired level of accuracy p_{tar}. Therefore, a more suitable measure of performance in terms of accuracy is the fraction of inputs that are classified with an accuracy level larger than p_{tar}. Following a similar approach as that used in Section 4.3 to introduce reliability diagrams, we divide test inputs into batches and evaluate the average accuracy $\mathrm{Acc}_{\mathrm{edge}}(B_b|\boldsymbol{\theta})$ for each batch B_b. Note that only inputs classified at the edge are included in the average. The edge inference outage probability is then defined as

$$P_{\mathrm{edge}}^{\mathrm{out}} = \frac{\sum_{b=1}^{N_B} \mathbb{1}[\mathrm{Acc}_{\mathrm{edge}}(B_b) < p_{\mathrm{tar}}]}{N_B}, \tag{11}$$

where N_B is the number of batches $B_1, \cdots, B_{N_B}$.

5.3. Missed Deadline Probability

A missed deadline is defined as the event that occurs when either the average inference time is larger than an application-defined latency deadline t_{tar} *or* when the average total accuracy is smaller than an accuracy requirement p_{tar}. The missed deadline probability is thus a measure of the fraction of inputs that are reliably classified within an acceptable maximal latency. Unlike the edge

inference outage, which is edge-focused, the missed deadline probability is an end-to-end application performance metric.

To evaluate the missed deadline probability, we again divide the test inputs $\boldsymbol{x}$ into image batches $B_1, \cdots, B_{N_B}$. For each batch, we measure the average inference time $T_{\text{inf}}(B_b|\boldsymbol{\theta})$ and the average total accuracy $\text{Acc}_{\text{total}}(B_b|\boldsymbol{\theta})$. The missed deadline probability is defined as

$$P_{md}(\boldsymbol{\theta}) = \sum_{b=1}^{N_B} \frac{(1 - \mathbb{1}[T_{\text{inf}}(B_b|\boldsymbol{\theta}) \leq t_{\text{tar}}] \cdot \mathbb{1}[\text{Acc}_{\text{total}}(B_b|\boldsymbol{\theta}) \geq p_{\text{tar}}])}{N_B} \tag{12}$$

6. Numerical Results on Calibration

In this section, we present numerical results with the aim of quantifying the calibration performance in terms of reliability diagrams, ECE (see Section 4.3), and probability of offloading with and without TS. The next section will focus on application-level performance metrics. We provide the code used in this article in an open repository[3].

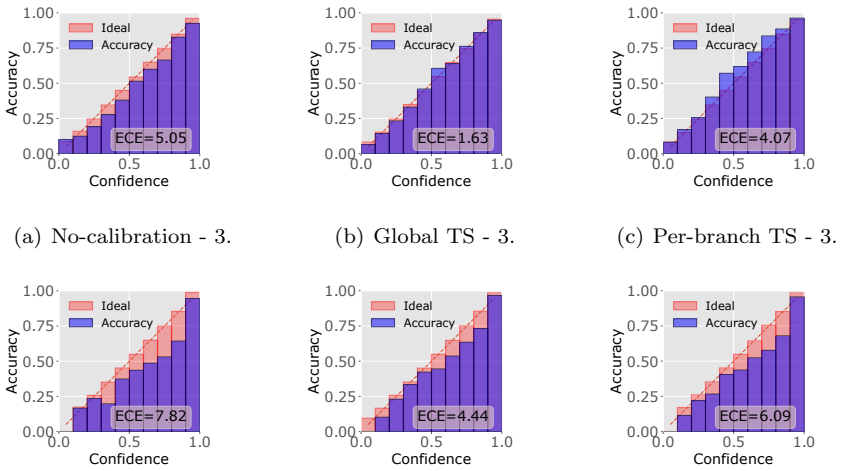
6.1. DNN Models and Datasets

The evaluations in this section use, as DNN backbones, MobileNetV2 [2], ResNet18 [3], VGG16 [36], and ResNet152 [3]. Many papers often employ these well-known DNN models for image classification tasks [12, 37, 38, 39]. We use the Caltech-256 [40] and Cifar-100 [41] datasets, which are well-known benchmarks to evaluate proposals in image classification [42, 12, 22, 15]. We split all datasets into 80%/10%/10% for train/validation/test. Other implementation details can be found in Appendix A.

6.2. Reliability Diagrams

This subsection evaluates the benefits of TS in calibrating early-exit DNNs based on reliability diagrams and the ECE metric. For conciseness, we only

[3] `https://github.com/pachecobeto95/early_exit_calibration`



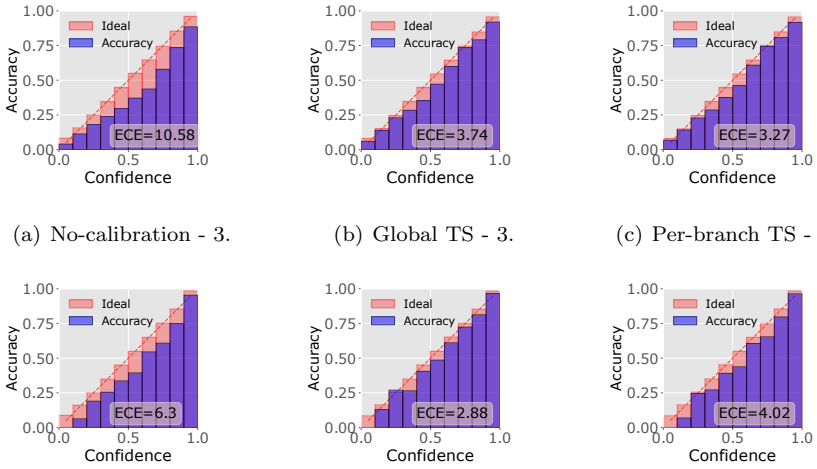
(a) No-calibration - 3. (b) Global TS - 3. (c) Per-branch TS - 3.

(d) No-calibration - 5. (e) Global TS - 5. (f) Per-branch TS - 5.

Figure 3: Reliability diagrams for early-exit MobileNetV2 models on Caltech-256 before (leftmost) and after (middle and rightmost) calibration. Each row shows the reliability diagram for a given side branch and the corresponding ECE (Expected Calibration Error).

present results for Caltech-256 since Cifar-100 leads to similar conclusions (see our open repository).

Figures 3 and 4 present reliability diagrams for each of the considered early-exit DNN models with no calibration as well as with global and per-branch TS. We focus on the third and fifth side branches. The reliability diagrams of early-exit VGG16 and Resnet152 are in Appendix B. The dashed line in the diagrams represents the ideal case, in which the model is perfectly calibrated (i.e., $\text{Acc}(B_{i,j}|\boldsymbol{\theta}) = \text{Conf}(B_{i,j}|\boldsymbol{\theta})$). Each plot also shows the ECE value, which is zero only for an ideally calibrated DNN. These results indicate that non-calibrated side branches are miscalibrated, providing overconfident predictions. In contrast, TS obtains a significantly lower ECE, with global TS outperforming per-branch TS.



(a) No-calibration - 3. (b) Global TS - 3. (c) Per-branch TS - 3.

(d) No-calibration - 5. (e) Global TS - 5. (f) Per-branch TS - 5.

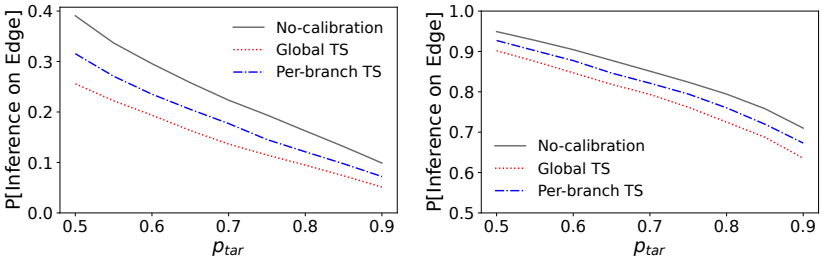
Figure 4: Reliability diagrams for early-exit ResNet18 on Caltech-256 before (leftmost) and after (middle and rightmost) calibration. Each row shows the reliability diagram for a given side branch and the corresponding ECE (Expected Calibration Error).

6.3. Offloading Probability

Next, we evaluate the impact of calibrating an early-exit DNN on the offloading probability. Figures 5 and 6 show the probability of classifying at the edge considering a given number k of side branches. The results show that calibration reduces the probability of classifying inputs at the edge device as compared to a non-calibrated model. Furthermore, Figure 5 shows that global TS reduces the probability of classifying inputs at the edge more as compared to per-branch TS. This is expected since, for this model, per-branch TS is more overconfident than global TS, as shown in Figure 3. For ResNet18, Figure 6 shows that global TS and per-branch TS obtain similar results.

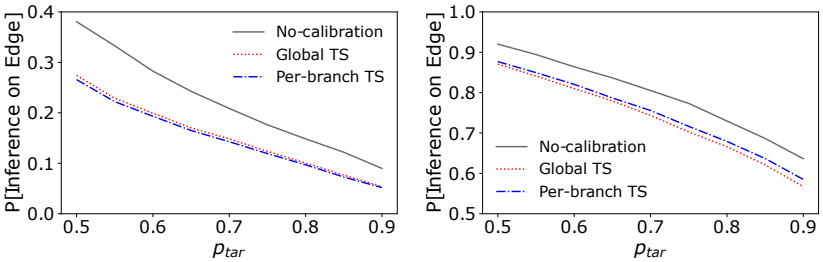
7. Experimental Results on Application-Level Performance

This section first evaluates the impact of calibration in terms of average accuracy metrics and then turns to the probability metrics introduced in Section 5.



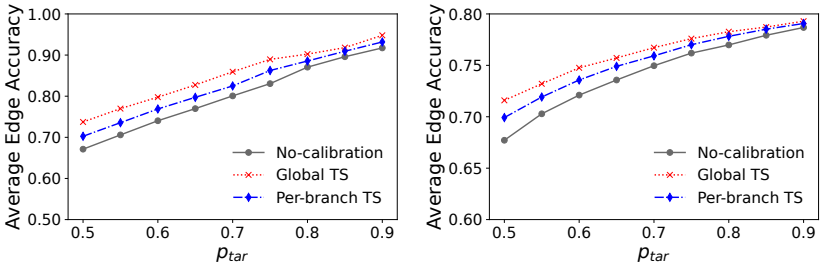
(a) Three side branches ($k = 3$). (b) Five side branches ($k = 5$).

Figure 5: Probability of classifying inputs at the edge for early-exit MobileNetV2 on Caltech-256.



(a) Three side branches ($k = 3$). (b) Five side branches ($k = 5$).

Figure 6: Probability of classifying inputs at the edge for early-exit ResNet18 on Caltech-256.



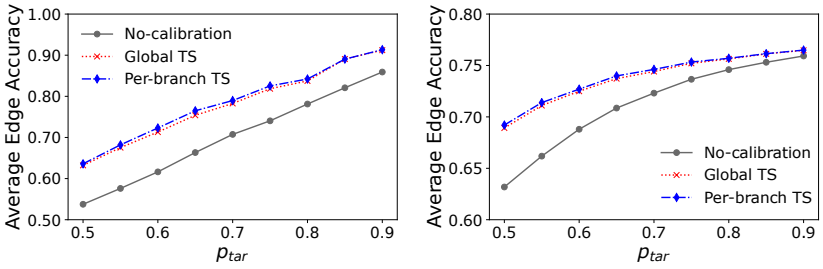
(a) Three side branches ($k = 3$). (b) Five side branches($k = 5$).

Figure 7: Accuracy on the edge using an early-exit MobileNetV2 on Caltech-256.

For the latter, we rely on an experimental setup involving an actual edge device and cloud server, as detailed in Section 7.2.

7.1. Average Accuracy Before and After Calibration

We start by evaluating the impact of calibration on the average accuracy in the same numerical setting considered in the previous section. We refer to Section 5 for a definition of the average accuracy metrics. First, we focus on the average accuracy at the edge.



(a) Three side branches ($k = 3$). (b) Five side branches ($k = 5$).

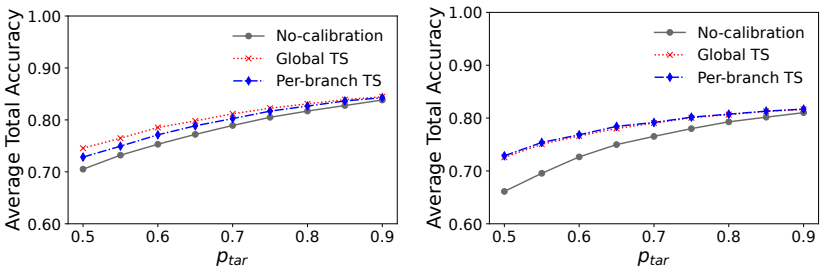
Figure 8: Accuracy on the edge considering an early-exit ResNet18 on Caltech-256.

Figures 7 and 8 show the accuracy at the edge as a function of the target reliability level p_{tar}. These results indicate that calibrating the side branches significantly improves their accuracy at the edge over non-calibrated side branches, with global TS outperforming per-branch TS. Furthermore, Figure 9 evaluates the average total accuracy versus p_{tar}. For brevity, we assess only early-exit DNNs with five side branches. This figure shows that calibrated early-exit DNNs can outperform non-calibrated ones for any p_{tar} value.

7.2. Experimental Setup

To evaluate the other application-level metrics introduced in Section 5, we use an Nvidia Jetson Nano[4] board as the edge device and an Amazon EC2[5] vir-

[4]https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/
[5]https://aws.amazon.com/ec2/



(a) MobileNetV2. (b) ResNet18.

Figure 9: Average total accuracy for early-exit DNNs with five side branches on Caltech-256.

tual machine as the cloud server. The edge device is located in Rio de Janeiro, Brazil, and runs an Ubuntu 18.04 operating system. This device has a 128-core Nvidia Maxwell GPU and an ARM A57 CPU with four cores at 1.43 GHz. The cloud server is an Amazon EC2 `4dn.xlarge` instance running Ubuntu 20.04 LTS with four vCPUs from an Intel Cascade Lake CPU and an NVIDIA Tesla T4 GPU. We use Python Flask[6] to implement the cloud server application. The edge device is connected to the Internet using a Gigabit Ethernet network interface. The edge and the cloud server communicate through the Internet using HTTP. We instantiate the cloud server in two AWS regions (i.e., geographical locations) to analyze our proposal using different network conditions.

Before running the experiment, we analyze the network conditions between the edge and the Amazon EC2 instances. We employ `ping` and `iPerf3` to measure, respectively, the RTT (Round Trip Time) and the throughput, shown in Table 2. The latency to the cloud instantiated in São Paulo is significantly lower than the cloud located in Ohio, USA. It happens because São Paulo is closer to the edge device located in Rio de Janeiro. We report these values only for illustration since the network conditions may vary during the experiment.

Figure 10 illustrates our experimental setup to evaluate the calibration impact. As Section 5.3 explains, we compute the missed deadline probability and

[6]https://flask.palletsprojects.com/en/2.2.x/

Table 2: Network Conditions For Each Cloud Region.

AWS Region	Location	Throughput	RTT
us-east-2	Ohio	73.1 Mbit/sec	177.76 ms
sa-east-1	São Paulo	93.4 Mbit/sec	38.98 ms

edge inference outage probability using a batch of images. Therefore, we divide the test set into equally-sized batches with 512 images. Each experimental round proceeds as follows. For a given image $\boldsymbol{x}$ in a batch, the edge device receives a given input from a batch, starts a time counter, and runs the inference, as described in Section 3. During this inference process, if the i-th side branch at the edge device provides a confidence estimate $f_i(\boldsymbol{x}|\boldsymbol{\theta})$ that exceeds our confidence threshold p_{tar} (i.e., $f_i(\boldsymbol{x}|\boldsymbol{\theta}) \geq p_{\text{tar}}$), the inference and the counter ends (step (1) in Figure 10). Consequently, the inference time corresponds to the processing delay of running inference up to the i-th side branch at the edge device. Otherwise, $f_i(\boldsymbol{x}|\boldsymbol{\theta}) < p_{\text{tar}}$, the edge device offloads the inference task via HTTP POST to the cloud, which processes the remaining layers (step (2) in Figure 10). Next, The edge device waits for an HTTP response from the cloud to terminate the time counter (step (3) in Figure 10). In this case, the inference time includes the processing delay at the edge device, the communication time required to offload data from the edge to the cloud, and the processing delay at the cloud. Therefore, we can compute the edge inference outage probability and the missed deadline probability according to Equations (11) and (12). To obtain reliable results, we repeat this procedure for fifty rounds to compute their average missed deadline probability and edge inference outage probability with a confidence interval of 95%.

7.3. Edge Inference Outage Probability

Figures 11 and 12 show the edge inference outage probability versus the desired reliability level p_{tar} for MobileNetV2 and ResNet18, respectively. As the edge inference outage does not depend on inference time, these results are

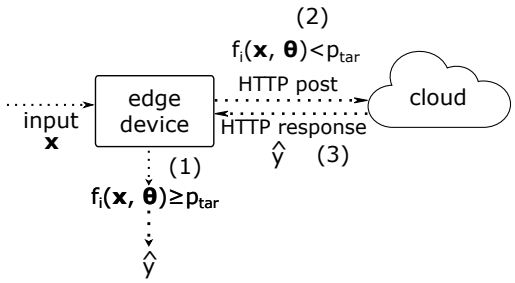
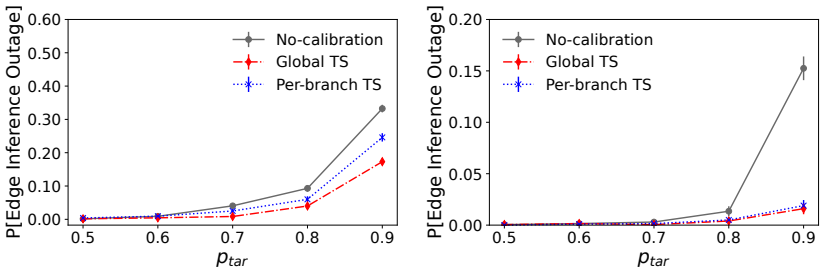


Figure 10: Illustration of the experimental setup for adaptive model partitioning between the edge device and cloud via early-exit DNNs.

agnostic to the cloud region or edge hardware. The figures show that calibration improves the edge inference outage probability for all values of p_{tar}. Hence, calibrated side branches are more likely to meet the accuracy target p_{tar} than non-calibrated ones. Thus, calibration improves the reliability of early-exit DNNs to meet application-defined accuracy requirements.

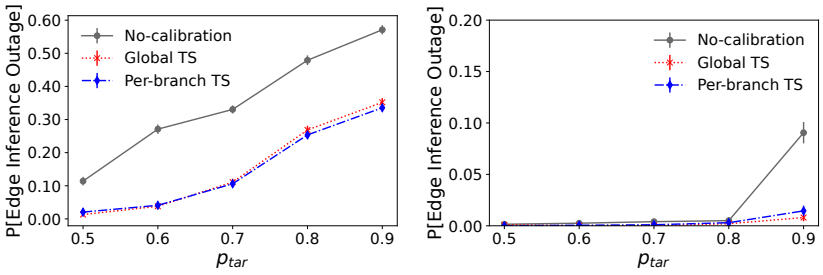


(a) Three side branches ($k = 3$). (b) Five side branches ($k = 5$).

Figure 11: Edge Inference outage probability, using an early-exit MobileNetV2 on Caltech-256.

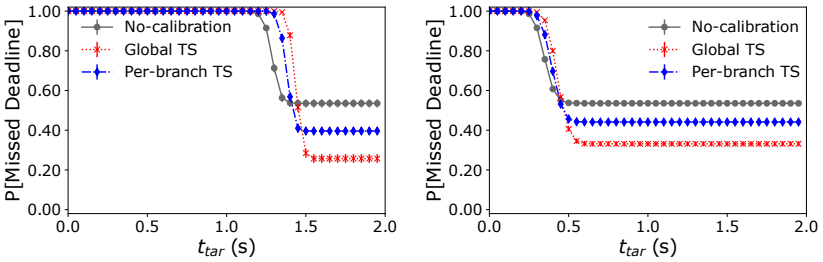
7.4. Missed Deadline Probability

Figures 13 and 14 show the missed deadline probability as a function of the latency deadline t_{tar}. We choose $p_{\text{tar}} = 0.8$ because, for calibrated DNNs, it is closer to the average total accuracy than the other values, as shown in Figure 9. Figures 13 and 14 report the results for a cloud server in Ohio, USA. This



(a) Three side branches ($k = 3$). (b) Five side branches ($k = 5$).

Figure 12: Edge Inference outage probability, using an early-exit ResNet18 on Caltech-256.

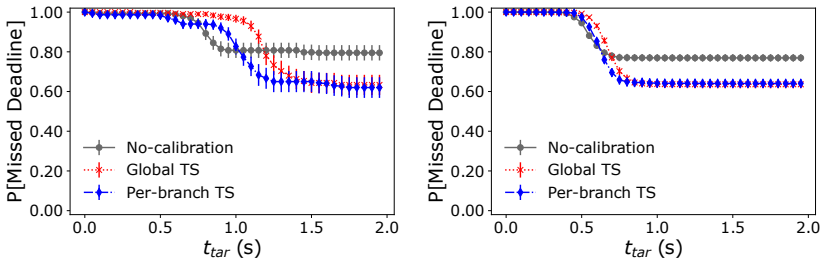


(a) Three side branches ($k = 3$). (b) Five side branches ($k = 5$).

Figure 13: Missed deadline probability for non-calibrated and calibrated early-exit MobileNetV2 models using $p_{\text{tar}} = 0.8$. The cloud server is located in Ohio, USA.

location represents a scenario where the cloud server is far away from the edge device. Hence, this case has the worst network conditions, according to Table 2.

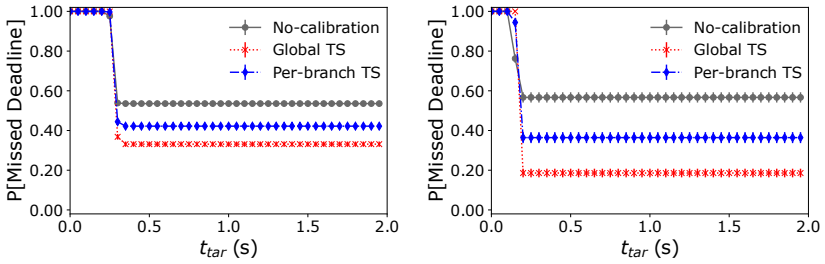
Figure 13 and 14 shows that calibrated early-exit DNNs and the non-calibrated one have a missed deadline probability close to one for low t_{tar} values. This happens because the models cannot meet strict latency deadlines. As t_{tar} increases, the calibrated early-exit DNNs outperform the non-calibrated ones because, as t_{tar} increases, it is easier to meet the latency deadline t_{tar}. In this case, all the models meet the latency deadline, and the missed deadline event only occurs because of the accuracy requirement p_{tar}, which is more likely to be met by calibrated models.



(a) Three side branches ($k = 3$). (b) Five side branches ($k = 5$).

Figure 14: Missed deadline probability for non-calibrated and calibrated early-exit ResNet18 models using $p_{\text{tar}} = 0.8$. The cloud server is located in Ohio, USA.

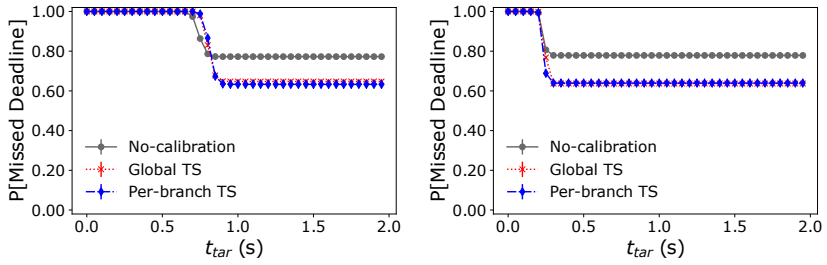
Figures 13(a) and 14(a) depict the results for early-exit DNNs with three side branches. For a short range of t_{tar} values, non-calibrated models outperform calibrated ones because, as seen in Figure 5, calibrated side branches offload more inputs to the cloud server, which introduces more communication delay. Thus, on the one hand, calibrating the models restricts meeting tight deadlines. On the other hand, calibration is required to ensure meeting the accuracy target for high p_{tar} values, as shown in Figure 9(a). In contrast, when early-exit DNNs have five side branches, Figures 13(b) and 14(b) show that the calibrated models outperform the non-calibrated ones.



(a) Three side branches ($k = 3$). (b) Five side branches ($k = 5$).

Figure 15: Missed deadline probability for non-calibrated and calibrated early-exit MobileNetV2 models using $p_{\text{tar}} = 0.8$. The cloud location is in São Paulo, Brazil.

Figures 15 and 16 present the missed deadline probability using the same previous methodology but with the cloud now hosted in São Paulo (i.e., the best network conditions according to Table 2). The calibrated side branches always outperform the non-calibrated ones for any t_{tar}, reducing the missed deadline probability. This happens since the cloud server is closer to the edge device, and thus the communication delay is reduced. Hence, the missed deadline probability depends more on meeting the accuracy requirement p_{tar} than meeting the deadline target t_{tar}.



(a) Three side branches ($k = 3$). (b) Five side branches ($k = 5$).

Figure 16: Missed deadline probability for non-calibrated and calibrated early-exit ResNet18 models using $p_{\text{tar}} = 0.8$. The cloud location is in São Paulo, Brazil.

8. Conclusions and Future Work

Adaptive offloading via early-exit DNNs is an effective way to reduce inference time on edge computing scenarios. In an early-exit DNN, side branches estimate the prediction confidence to decide whether to end the inference on the edge device or offload it to the cloud. For this decision to be effective, the side branches must provide reliable confidence estimates to make effective offloading decisions. This article has provided an extensive calibration study on different datasets and early-exit DNN models for image classification. We have provided empirical evidence that miscalibrated early-exit DNNs overestimate their prediction confidence, providing unreliable offloading decisions. Hence, they

classify more inputs earlier than they should. Our experiments, under a realistic edge-cloud computing scenario, demonstrate that temperature scaling can help to solve the miscalibration problem. Calibrated models can thus provide reliable confidence estimates, improving the offloading decisions and increasing their accuracy.

Future work may evaluate the impact of calibration considering user experience for specific image classification applications, such as augmented reality and cognitive assistance. As this study is primarily concerned with image classification tasks, future work will also investigate the effect of calibration on other tasks, such as natural language processing (NLP). As tasks become more complex, the impact of classification at the edge device can imply greater consequences than those evaluated in this study. Therefore, calibration becomes even more critical in such scenarios. Another future work is to develop calibration methods that explore the trade-off between accuracy and latency in adaptive offloading.

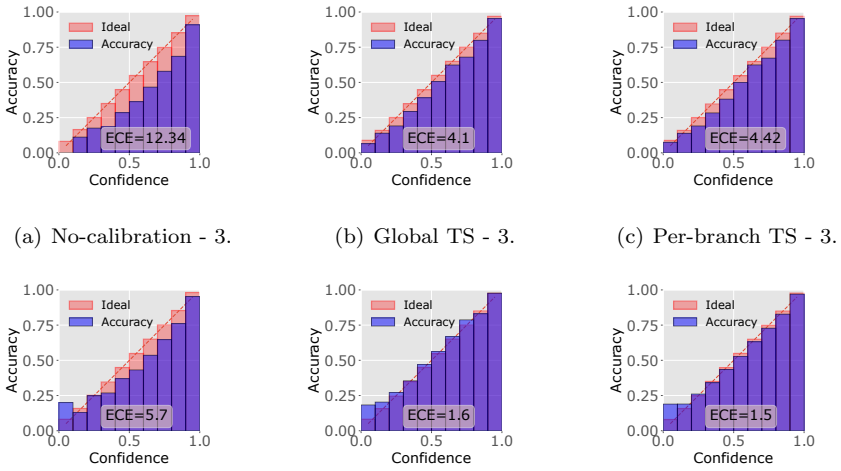
Appendix A. Early-exit DNN Training

Before training the early-exit DNNs, we apply preprocessing procedures on dataset inputs. We apply image scaling and cropping, stochastic image flip with a 0.25 flipping probability, and color channel normalization. We initialize the fully-connected layers through Xavier initialization [43]. Next, we initialize the DNN backbone's layers using weights trained on the ImageNet dataset [44], provided by the torchvision[7] 0.8.2 framework. Then, we train the model following the methodology described in Section 3. We employ the same hyperparameters (e.g., batch size, learning rate, and weight decay) as in the original paper. For example, we train an early-exit ResNet18 using the hyperparameters employed on the original ResNet paper [3]. We train the early-exit DNNs until the validation loss stops decreasing for ten epochs in a row.

[7] `https://pytorch.org`

Appendix B. Reliability Diagram

Figure B.17 and B.18 complements the results of Section 6.2, presenting additional reliability diagrams for early-exit VGG16 and ResNet152.



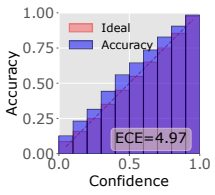
(a) No-calibration - 3. (b) Global TS - 3. (c) Per-branch TS - 3.

(d) No-calibration - 5. (e) Global TS - 5. (f) Per-branch TS - 5.

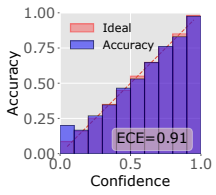
Figure B.17: Reliability diagrams for early-exit VGG16 on Caltech-256 before (leftmost) and after (middle and rightmost) calibration. Each row shows the reliability diagram for a given side branch and the corresponding ECE (Expected Calibration Error).

Appendix C. Offloading Probability

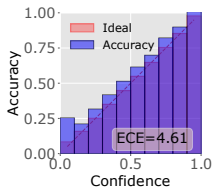
Figure C.19 complements Section 6.3, showing the probability of classifying at the edge device for early-exit DNNs models using the Caltech-256. Figure C.20 shows the results obtained on the Cifar-100 dataset. These figures show that calibrating the side branches reduces the number of inputs classified at the edge, corroborating the results presented in Section 6.3.



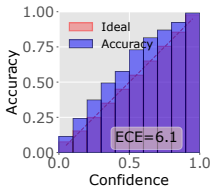
(a) No-calibration - 3.



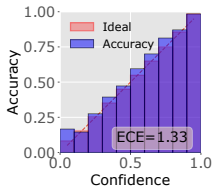
(b) Global TS - 3.



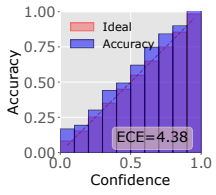
(c) Per-branch TS - 3.



(d) No-calibration - 5.

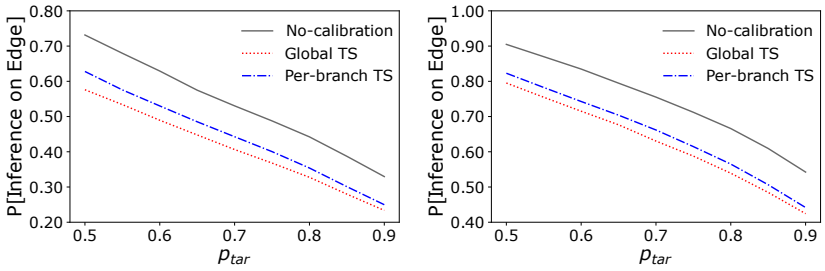


(e) Global TS - 5.



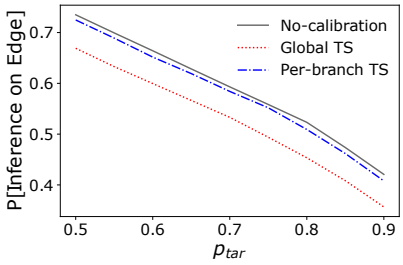
(f) Per-branch TS - 5.

Figure B.18: Reliability diagrams for early-exit ResNet152 on Caltech-256 before (leftmost) and after (middle and rightmost) calibration. Each row shows the reliability diagram for a given side branch and the corresponding ECE (Expected Calibration Error).

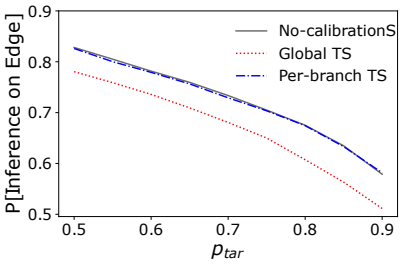


(a) Three side branches - VGG16.

(b) Five side branches - VGG16.

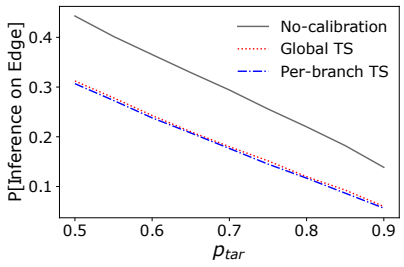


(c) Three side branches - ResNet152.

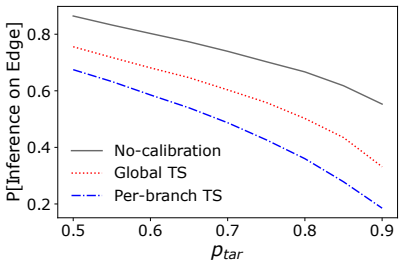


(d) Five side branches - ResNet152.

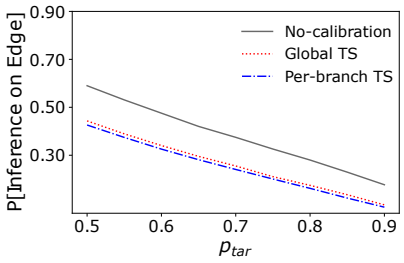
Figure C.19: Probability of classifying input at the edge on Caltech-256 for early-exit VGG16 and ResNet152.



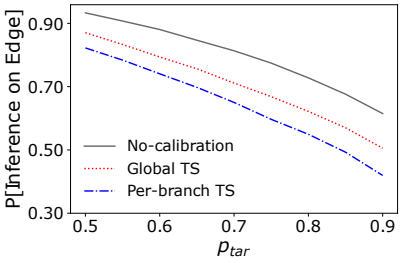
(a) Three side branches - MobileNetV2.



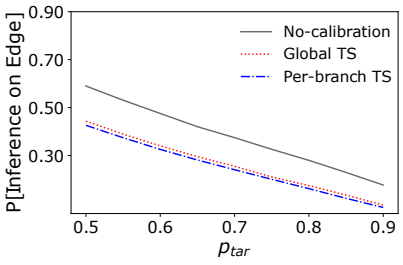
(b) Five side branches - MobileNetV2.



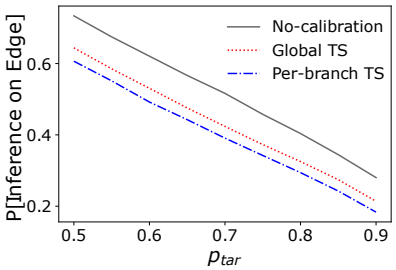
(c) Three side branches - ResNet18.



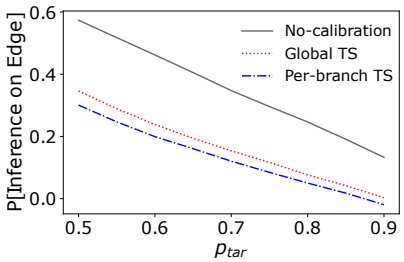
(d) Five side branches - ResNet18.



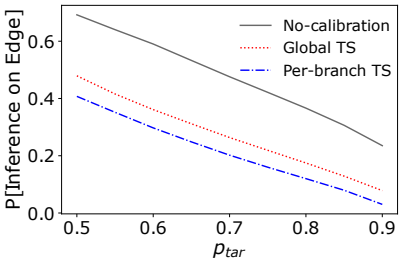
(e) Three side branches - VGG16.



(f) Five side branches - VGG16.



(g) Three side branches - ResNet152.

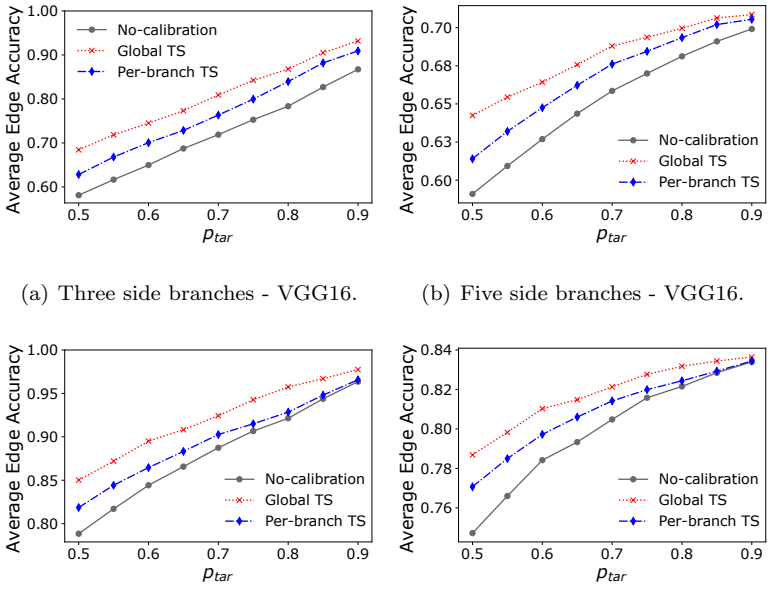


(h) Five side branches - ResNet152.

Figure C.20: Probability of classifying input at the edge for early-exit DNNs on Cifar-100 for different early-exit DNNs.

Appendix D. Average Accuracy Before and After Calibration

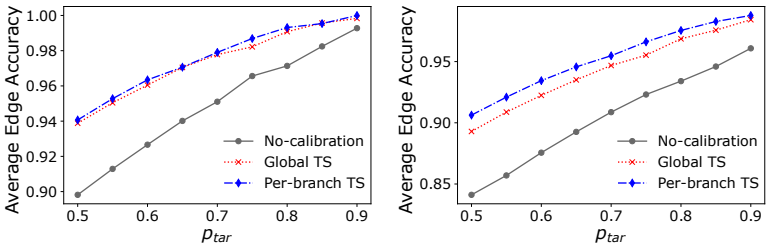
Figure D.21 presents additional plots for comparing accuracy before and after calibration using early-exit VGG16 and early-exit ResNet152 using Caltech-256. Figure D.22 shows these same results for several early-exit DNNs using Cifar-100. These figures show that calibrated early-exit DNNs achieve higher accuracy on edge than non-calibrated models. We notice that applying a simple calibration method on the side branches improves their accuracy. Therefore, the conclusions presented in Section 7.1 are also valid for other image classification datasets.



(a) Three side branches - VGG16. (b) Five side branches - VGG16.

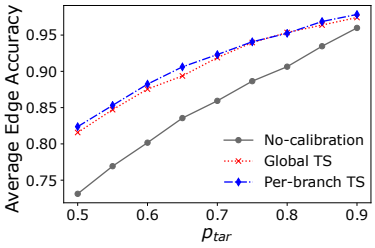
(c) Three side branches - ResNet152. (d) Five side branches - ResNet152.

Figure D.21: Accuracy on the edge for early-exit DNNs on Caltech-256 for an early-exit VGG16 and ResNet152.

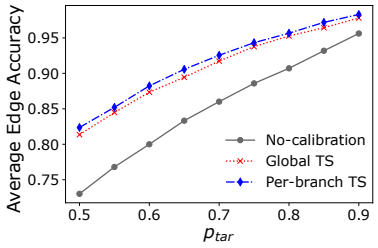


(a) Three side branches - MobileNetV2.

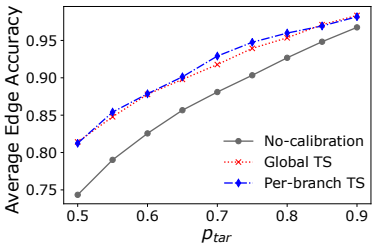
(b) Five side branches - MobileNetV2.



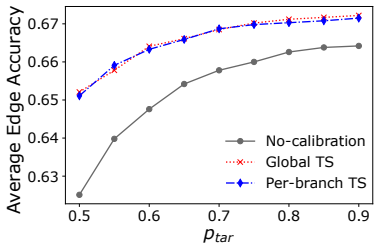
(c) Three side branches - ResNet18.



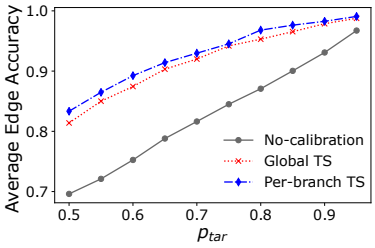
(d) Five side branches - ResNet18.



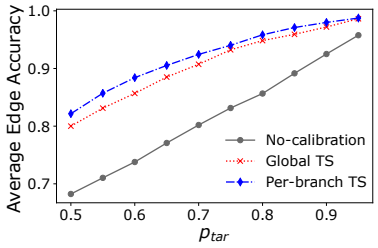
(e) Three side branches - VGG16.



(f) Five side branches - VGG16.



(g) Three side branches - ResNet152.

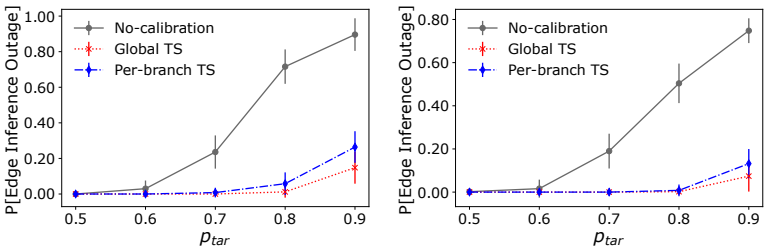


(h) Five side branches - ResNet152.

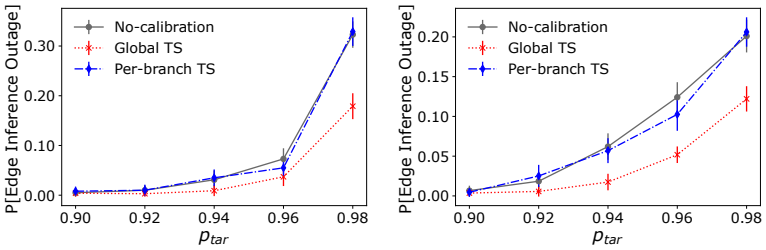
Figure D.22: Accuracy on the edge for several early-exit DNNs using Cifar-100 for several early-exit DNNs.

Appendix E. Edge Inference Outage Probability

Figure E.23 complements the results of edge inference outage probability for Caltech-256 presented in Section 7.3. Figure E.24 presents these same results for multiple early-exit DNNs using Cifar-100. These figures show that calibrated models are more likely to meet accuracy requirements than non-calibrated ones. Thus, these figures corroborate the conclusions of Section 7.3.

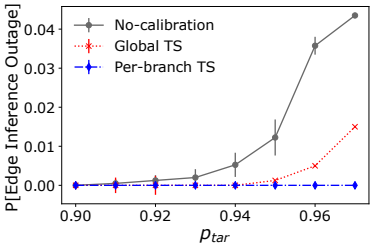


(a) Three side branches - VGG16. (b) Five side branches - VGG16.

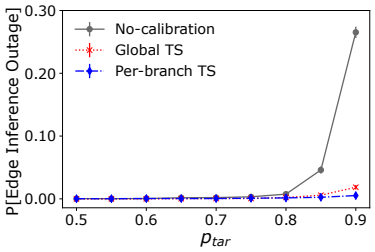


(c) Three side branches - ResNet152. (d) Five side branches - ResNet152.

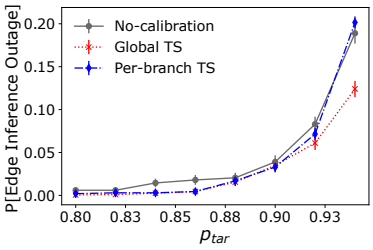
Figure E.23: Edge Inference outage probability on Caltech-256 for an early-exit VGG16 and ResNet152.



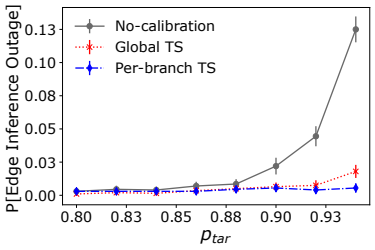
(a) Three side branches - MobileNetV2.



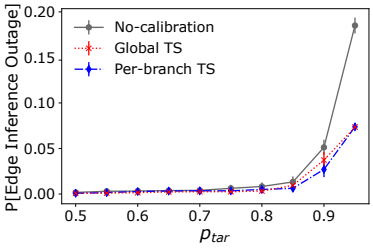
(b) Five side branches - MobileNetV2.



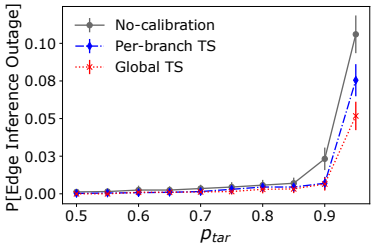
(c) Three side branches - ResNet18.



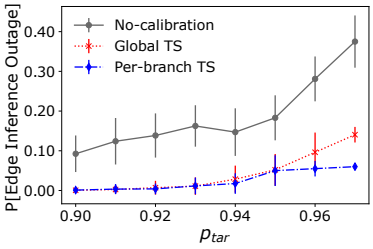
(d) Five side branches - ResNet18.



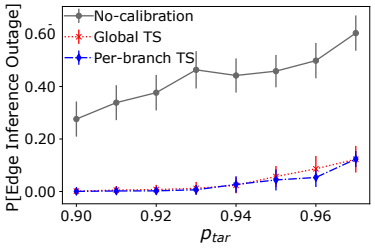
(e) Three side branches - VGG16.



(f) Five side branches - VGG16.



(g) Three side branches - ResNet152.



(h) Five side branches - ResNet152.

Figure E.24: Edge Inference outage probability for several early-exit DNNs on Cifar-100 for several early-exit DNNs.

Data Availability

The research data developed for this article is available in the following open repository: `https://github.com/pachecobeto95/early_exit_calibration`.

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. It was also supported by CNPq, PR2/UFRJ, FAPERJ Grants E-26/203.211/2017, E-26/010.002174/2019, and E-26/201.300/2021, and FAPESP Grant 15/24494-8. The work of O. Simeone was supported by the European Research Council (ERC) through European Union's Horizon 2020 Research and Innovation Programme under Grant 725731, by an Open Fellowship of the EPSRC with reference EP/W024101/1, by the European Union's Horizon Europe project CENTRIC (101096379), and by Project REASON, a UK Government funded project under the Future Open Networks Research Challenge (FONRC) sponsored by the Department of Science Innovation and Technology (DSIT).

References

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Neural Information Processing Systems (NIPS), 2012, pp. 1097–1105.

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

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

[4] K. Bochie, M. S. Gilbert, L. Gantert, M. S. Barbosa, D. S. Medeiros, M. E. M. Campista, A survey on deep learning for challenged networks: Applications and trends, Journal of Network and Computer Applications 194 (2021) 103213.

[5] M. Satyanarayanan, The emergence of edge computing, Computer 50 (1) (2017) 30–39.

[6] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, L. Tang, Neurosurgeon: Collaborative intelligence between the cloud and mobile edge, in: ACM Computer Architecture News, Vol. 45, 2017, pp. 615–629.

[7] P. Cruz, N. Achir, A. C. Viana, On the edge of the deployment: A survey on multi-access edge computing, ACM Computing Surveys.

[8] C. Hu, W. Bao, D. Wang, F. Liu, Dynamic adaptive DNN surgery for inference acceleration on the edge, in: IEEE Conference on Computer Communications, 2019, pp. 1423–1431.

[9] R. G. Pacheco, R. S. Couto, Inference time optimization using branchynet partitioning, in: IEEE Symposium on Computers and Communications, 2020, pp. 1–7.

[10] M. Xu, F. Qian, M. Zhu, F. Huang, S. Pushp, X. Liu, Deepwear: Adaptive local offloading for on-wearable deep learning, IEEE Transactions on Mobile Computing 19 (2) (2019) 314–330.

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

[12] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, N. D. Lane, SPINN: synergistic progressive inference of neural networks over device and cloud, in: Conference on Mobile Computing and Networking, 2020, pp. 1–15.

[13] R. G. Pacheco, K. Bochie, M. S. Gilbert, R. S. Couto, M. E. M. Campista, Towards edge computing using early-exit convolutional neural networks, Information 12 (10).

[14] R. G. Pacheco, R. S. Couto, O. Simeone, Calibration-aided edge inference offloading via adaptive model partitioning of deep neural networks, in: IEEE International Conference on Communications, 2021, pp. 1–6.

[15] R. G. Pacheco, F. D. Oliveira, R. S. Couto, Early-exit deep neural networks for distorted images: providing an efficient edge offloading, in: IEEE Global Communications Conference, 2021, pp. 1–6.

[16] A. Kendall, Y. Gal, What uncertainties do we need in bayesian deep learning for computer vision?, in: Advances in neural information processing systems, 2017, pp. 5574–5584.

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

[18] X. Li, Z. Liu, P. Luo, C. Change Loy, X. Tang, Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade, in: IEEE conference on computer vision and pattern recognition, 2017, pp. 3193–3202.

[19] A. Kouris, S. I. Venieris, S. Laskaridis, N. D. Lane, Multi-exit semantic segmentation networks, arXiv preprint arXiv:2106.03527.

[20] J. Xin, R. Tang, J. Lee, Y. Yu, J. Lin, Deebert: Dynamic early exiting for accelerating bert inference, arXiv preprint arXiv:2004.12993.

[21] W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, F. Wei, Bert loses patience: Fast and robust inference with early exit, Neural Information Processing Systems 33 (2020) 18330–18341.

[22] S. Laskaridis, S. I. Venieris, H. Kim, N. D. Lane, Hapi: Hardware-aware progressive inference, in: IEEE/ACM International Conference On Computer Aided Design, 2020, pp. 1–9.

[23] B. Fang, X. Zeng, F. Zhang, H. Xu, M. Zhang, Flexdnn: Input-adaptive on-device deep learning for efficient mobile vision, in: IEEE/ACM Symposium on Edge Computing, 2020, pp. 84–95.

[24] E. Li, L. Zeng, Z. Zhou, X. Chen, Edge ai: On-demand accelerating deep neural network inference via edge computing, IEEE Transactions on Wireless Communications 19 (1) (2019) 447–457.

[25] M. Farhadi, M. Ghasemi, Y. Yang, A novel design of adaptive and hierarchical convolutional neural networks using partial reconfiguration on fpga, in: IEEE High Performance Extreme Computing Conference, 2019, pp. 1–7.

[26] R. Dong, Y. Mao, J. Zhang, Resource-constrained edge ai with early exit prediction, Journal of Communications and Information Networks 7 (2) (2022) 122–134.

[27] E. Samikwa, A. Di Maio, T. Braun, Adaptive early exit of computation for energy-efficient and low-latency machine learning over iot networks, in: 2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC), IEEE, 2022, pp. 200–206.

[28] M. Wang, J. Mo, J. Lin, Z. Wang, L. Du, Dynexit: A dynamic early-exit strategy for deep residual networks, in: IEEE International Workshop on Signal Processing Systems, 2019, pp. 178–183.

[29] G. Kim, J. Park, Low cost early exit decision unit design for cnn accelerator, in: IEEE International SoC Design Conference, 2020, pp. 127–128.

[30] L. Zhang, L. Chen, J. Xu, Autodidactic neurosurgeon: Collaborative deep inference for mobile edge intelligence via online learning, in: Proceedings of the Web Conference 2021, 2021, pp. 3111–3123.

[31] M. Minderer, J. Djolonga, R. Romijnders, F. Hubis, X. Zhai, N. Houlsby, D. Tran, M. Lucic, Revisiting the calibration of modern neural networks, Advances in Neural Information Processing Systems 34.

[32] B. Lakshminarayanan, A. Pritzel, C. Blundell, Simple and scalable predictive uncertainty estimation using deep ensembles, Advances in neural information processing systems 30.

[33] I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, J. Uszkoreit, et al., Mlp-mixer: An all-mlp architecture for vision, Neural Information Processing Systems 34.

[34] 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.

[35] O. Simeone, Machine Learning for Engineers, Cambridge University Press, 2022.

[36] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556.

[37] Z. Liu, G. Lan, J. Stojkovic, Y. Zhang, C. Joe-Wong, M. Gorlatova, Collabar: Edge-assisted collaborative image recognition for mobile augmented reality, in: ACM/IEEE International Conference on Information Processing in Sensor Networks, 2020, pp. 301–312.

[38] S. Dodge, L. Karam, Quality resilient deep neural networks, arXiv preprint arXiv:1703.08119.

[39] I. Leontiadis, S. Laskaridis, S. I. Venieris, N. D. Lane, It's always personal: Using early exits for efficient on-device cnn personalisation, in: Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications, 2021, pp. 15–21.

[40] G. Griffin, A. Holub, P. Perona, Caltech-256 object category dataset.

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

[42] Z. Liu, G. Lan, J. Stojkovic, Y. Zhang, C. Joe-Wong, M. Gorlatova, Collabar: Edge-assisted collaborative image recognition for mobile augmented reality, in: ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), 2020, pp. 301–312.

[43] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.

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