PORTEND: A Joint Performance Model for Partitioned Early-Exiting DNNs

Maryam Ebrahimi
University of Toronto
maryamebr@cs.toronto.edu

Alexandre da Silva Veith
Nokia Bell Labs
alexandre.da_silva_veith@nokia-bell-labs.com

Moshe Gabel
York University
mgabel@yorku.ca

Eyal de Lara
University of Toronto
delara@cs.toronto.edu

***Abstract*—The computation and storage requirements of Deep Neural Networks (DNNs) make them challenging to deploy on edge devices, which often have limited resources. Conversely, offloading DNNs to cloud servers incurs high communication overheads. Partitioning and early exiting are attractive solutions for reducing computational costs and improving inference speed. However, current work often addresses these approaches separately and/or ignores common communication intricacies on edge networks such as de(serialization) and data transmission overheads. We present PORTEND, a novel performance model that jointly optimizes partitioning, early exiting, and multi-tier network placement. PORTEND'S novel approach outperforms the state-of-the-art solutions in edge computing setups, reducing the DNN inference latency by 29%.**

***Index Terms*—DNN partitioning, early exit, performance model**

I. Introduction

Next-generation applications, such as pervasive health monitoring [1] and augmented reality [2], require fast and accurate inference over rich and complex data. Unfortunately, edge devices, such as smartphones and other Internet of Things (IoT) devices, are often too resource-limited to execute complex DNN models efficiently [3].

Edge DNN inferencing is often slow or infeasible due to limited resources on edge. Yet sending all data to the cloud on high-latency, bandwidth-limited links also incur high latency [4]. *Partitioning* [5] and *early exit* [6] address the mentioned challenges by attempting to bridge the gap between the computational demands of DNNs and available resources at the network's edges. On the one hand, DNN partitioning splits a DNN model into layers, distributing them across end devices, edge and cloud servers. On the other hand, early exit includes extra side branches at specific locations of the DNN model with auxiliary classifiers that enable some samples to leave the model earlier when meeting an entropy goal.

Previous work [5]–[10] falls short on (a) combining partitioning and early exit, (b) supporting multi-tier hierarchical settings, and (c) handling communication overheads such as (de)serialization. Additionally, many partitioning approaches assume a single partitioning point, neglect profiling intricacies, and focus only on minimizing latency. In contrast, early exit approaches overlook how to map these exits to multi-tier network topologies with constrained devices. Figure 1 illustrates how the mentioned challenges strongly impact the accuracy-latency tradeoff in a ResNet-110 model for jointly optimizing

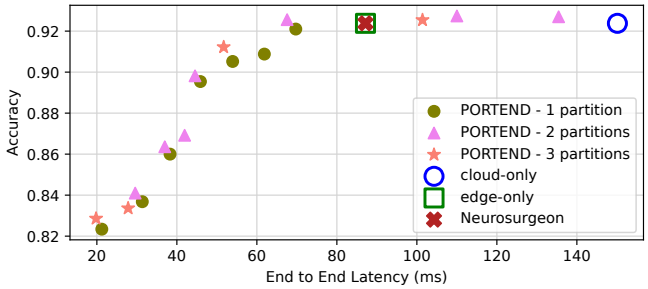


Fig. 1: The accuracy-latency tradeoff plot for ResNet-110.

DNN partitioning and early exits during the placement on a 3-tier edge computing setup (see Figure 3). Each point[1] shows the resulting average end-to-end latency (X-axis) and average accuracy (Y-axis). A pure partitioning approach, such as Neurosurgeon, creates a single partition and places it on the edge – as this is the optimal partitioning for this scenario. However, by jointly addressing partitioning and early exit and enabling the deployment of multi-partition configurations, we can further reduce latency without harming accuracy.

Our Contributions: We present PORTEND, a new performance model that combines partitioning and early exiting, enabling deployment in a multi-tier edge computing infrastructure, and allowing for simultaneous control of accuracy and latency. PORTEND holistically accounts for inference accuracy, computation time, (de)serialization overhead, transmission and propagation latencies, network topology, and device capabilities. PORTEND allows a customizable and flexible design where developers give hints to the optimization, such as the number and placement of partitions and early exists, which enables the exploration of a larger search space of DNN partitioning deployments. We evaluate PORTEND on a 3-tier infrastructure using AWS servers by emulating a topology with an end-user device, a resource-limited edge data center, and a powerful cloud server. Our extensive experiments with several DNNs and datasets demonstrate that PORTEND accurately estimates end-to-end inference performance and meets superior latency-accuracy tradeoffs than prior approaches. The results show that PORTEND's optimal configuration outperforms the state-of-the-art by up to 29% reduction in inference latency.

[1]Results show placement configuration samples on/near the Pareto frontier.

TABLE I: Summary of PORTEND's model parameters. Parameters listed at the top of the table are given as input, while those in the middle are measured during the offline profiling phase. The bottom lists model outputs for a partitioning P, early exit threshold T, and deployment setting D.

Notation	Description
M	Number of tiers in the topology
j	Tier index ($1 \leq j \leq M$)
PD_j	Propagation delay between tier j and $j+1$
BW_j	Link bandwidth between tier j and $j+1$
N	Number of blocks/branches in the backbone DNN
i	Block/branch index ($1 \leq i \leq N$)
O_i	Output size of block i
Rs	Result size (label and other information)
L	Number of partitions ($1 \leq L \leq M$)
k	Partition index ($1 \leq k \leq L$)
P_k	The index of partition k's last block
D_k	The deployment tier for partition k
A_i	Accuracy for branch i
ER_i^T	Exit rate of branch i, with threshold T
CT_k^j	Computation time of partition k on tier j
ST_O^j	Serialization time for data of size O on tier j
DT_O^j	Deserialization time for data of size O on tier j
$E_k(T,P)$	Estimated fraction of samples exiting at partition k
$S_k(T,P)$	Estimated stay rate after partition k
$Acc^{est}(T,P)$	Estimated inference accuracy
$Lat^{est}(T,P,D)$	Estimated average end-to-end latency

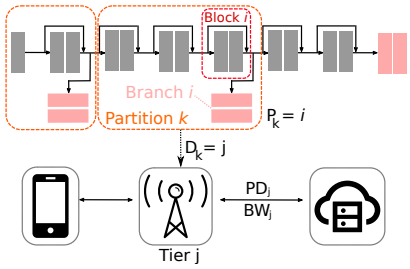


Fig. 2: A partitioned early exit deployment model.

II. PORTEND

PORTEND[2] offers a performance model for a flexible deployment on edge networks that simultaneously incorporates DNN partitioning and early exit. Based on input parameters such as the backbone DNN (i.e., DNN model), the dataset, the network topology, early exit threshold (see Section II-A), and/or the preferred latency and accuracy. PORTEND follows 3 steps: (1) offline profiling (Section II-B); (2) metrics estimation (Section II-C); and (3) optimization (Section II-D). Table I summarizes key notations.

A. System Model

PORTEND addresses the placement of DNN partitions with early exits in a geographically distributed M-tier network, as shown in Figure 2 (bottom). Each tier connects to the next tier via a link – non-neighboring tiers do not communicate directly. A link with index j is characterized by BW_j, the bandwidth measured in bits per second (bps), and by PD_j, the propagation delay defined as the time in milliseconds to transmit a packet from one end to the other.

PORTEND implements early exit by adding branches that act as auxiliary classifiers to a "backbone" DNN [6]. This backbone is a linear sequence of blocks, As depicted in Figure 2 (top), each block is a single layer or a sequence of layers (for simplicity, we assume exactly one connection between subsequent blocks). We denote the number of DNN blocks by N and index them by i, where O_i refers to the output size of a block i in bits. There is a potential exit point after each block, where adding an early exit branch is possible. There are thus N potential exit branches, each comprised of a copy of the final classifying layers of the backbone DNN. During inference, when a sample reaches an exit branch, we evaluate the normalized entropy of the output from the branch's classifier similar to BranchyNet [6]: $\eta(x) = -\sum_{i=1}^{|C|} \frac{x_i \log x_i}{\log |C|}$, where x is a probability vector from the output of the exit branch, and C is the set of possible labels. If the entropy is high, the classifier is not confident about the predictions, while low entropy means that the classifier is confident. We compare the output entropy to the provided *early exit threshold* T. If $\eta(x) < T$, the sample exits inference and we use x as the model's output. Otherwise, the inference process continues to the next layer.

A DNN *configuration* corresponds to a specific selection of exit points used for exit branches, while the *full configuration* is when all the possible exit branches are used. The maximum number of partitions is equal to the number of DNN blocks – the first partition always starts at block 1. PORTEND reduces the number of configurations by limiting one exit branch in each partition after the last block in the partition, as shown in Figure 2. PORTEND defines the last block because (a) exiting in an earlier block would not result in a significant reduction in memory and compute requirements as non-exiting samples must still go through all blocks in the partition and (b) later exit points are likely to be more accurate than early ones. More formally, L denotes the number of partitions in a configuration, and k indexes them. P_k represents the index of the partition's last block (or exit branch). More specifically, a partition k comprises blocks $P_{k-1}+1$ (the first block) to P_k (the last block). PORTEND allows partial DNN in such a way that $P_L < N$, but always $P_k > P_{k-1}$.

A *deployment setting* is a specific assignment of partitions to tiers. PORTEND assumes all partitions are assigned to some tier, and each tier has at most one partition assigned to it. An assignment D is a mapping from partitions to tiers: $D_k = j$ if partition k is assigned to tier j. PORTEND permits later partitions to be assigned to either higher or lower tiers. Hence, it allows $D_k > D_{k-1}$ and/or $D_k < D_{k-1}$.

B. Profiling Phase

PORTEND profiles the accuracy, exit rate, and computation latency of the full configuration model in the different tiers using a set of validation samples – it performs the profiling

[2]A preliminary performance model appeared in a workshop paper [11]. It did not account for various latencies (serialization, computation, and propagation), and it was not validated on an end-to-end system.

once for each backbone DNN and dataset on all servers and devices in the topology. First, PORTEND trains a full configuration model with all available exit branches by optimizing the sum of loss values of all branches, followed by back-propagating through the entire model. The profiling dataset is neither used for training the DNN nor part of the test set used for evaluation. The profiling for metrics happens as follows:

Computation Time (CT_k^j): the time in seconds required on tier j to generate the output of partition k using its inputs – the estimated computation latency of inference. PORTEND profiles each available device due to the variations in their capabilities. Previous work, such as [6], profiles the computation latency of each DNN layer/block separately and then sums them up to obtain the computation latency of a partition. This, however, neglects overheads between layers and other types of computation, such as entropy (see Sec. III-B). PORTEND overcomes these limitations by leveraging the partition-wise estimation, which profiles all[3] the possible partitions for employing them in the performance model.

Serialization and Deserialization Time (ST_O^j, DT_O^j): Sending data such as tensors between devices requires encoding it into a format that can be transmitted over the network and decoding it back into its original form at the destination tier. (De)serialization time depends on two factors: (1) the device's computation capability and (2) the size of the object – the intermediate data size between DNN layers. For example, serializing the output of block 1 on EfficientNet-B7 with ImageNet takes 26 milliseconds on an ARM-based AWS a1.medium instance. PORTEND, therefore, profiles serialization latency and deserialization latency on all available devices for all intermediate data sizes between blocks of the DNN model (i.e., where we might want to partition the model).

Exit Rate (ER_i^T): ER_i^T represents the exit rate of branch i with an early exit threshold of T, indicating the proportion of validation samples that branch i can exit at the given threshold. Later branches are generally more capable in classification, and thus, one expects them to exhibit higher exit rates. Since the exit rate is independent of the deployment device, it can be measured offline on any machine. To profile ER_i^T, PORTEND considers 10 early exit thresholds between 0 and 1 and stores the fraction of validation samples that would exit at branch i given each early exit threshold. PORTEND measures the exit of each branch separately from the other branches where it runs N times, and for each run, utilizes samples starting from the first layer (the DNN input) and sees whether they will exit at branch i, without testing any of the previous exit branches.

Accuracy (A_i): PORTEND uses all validation samples for each branch to correctly check how many are classified. Like the exit rate, PORTEND repeats it N times for each branch separately. Each time, PORTEND focuses on measuring the accuracy of one branch i, where samples start from the beginning of the model and are not allowed to exit before i – the profiled accuracy A_i of a branch does not depend on the early exit threshold.

[3]The profiling phase is fast (minutes for all models in Sec. III) as the number of (device, model, partitions) combinations to profile is moderate and profiling each only requires running inference a few times.

C. Estimation of Performance Metrics

PORTEND derives performance metrics for optimization by combining system model parameters, and full configuration profiling. This section describes methods for estimating the exit rate, accuracy, and end-to-end latency. However, PORTEND is not limited to them – PORTEND also enables developers to write their optimization goals and declare metrics like energy consumption, bandwidth utilization, monetary cost [12], among others.

1) Estimating Exit Rate and Accuracy: PORTEND estimates the exit rate per partition with $E_k(T, P)$. The function combines the target early exit threshold T with partitioning P (i.e., the last exit branch in each partition is used) to estimate the exit rate of samples at the partition k's exit branch. The profiled metric ER_i^T provides the exit rate of the branch i for a full configuration. However, PORTEND must account for samples that exit in previous partitions when the exit rate of a partition starts from a later layer. Therefore, PORTEND must subtract the exit rates of the last branch before partition k, $ER_{P_{k-1}}^T$, from the exit rate of the last branch of partition k, $ER_{P_k}^T$. In other words:

$$E_k(T,P) = \begin{cases} ER_{P_1}^T & k = 1 \\ ER_{P_k}^T - ER_{P_{k-1}}^T & 1 < k < L \\ 1 - \sum_{a=1}^{k-1} E_a(T,P) & k = L \end{cases}$$

For the last partition, $k = L$, the exit rate is 1 minus the sum of all previous exit rates since all samples exit from the last used branch. The stay rate of partition k, denoted by $S_k(T, P)$, is the fraction of samples that do not exit from its exit branch. These samples go to the next partition:

$$S_k(T,P) = 1 - \sum_{a=1}^{k} E_a(T,P)$$

PORTEND estimates the resulting accuracy of a particular partitioning by adding up the rates of samples correctly classified in each partition, weighted by the partition exit rate. In other words, the weighted mean of accuracies:

$$Acc^{est}(T,P) = \sum_{k=1}^{L} A_{P_k} E_k(T,P)$$

2) Estimating End-to-End Latency: PORTEND accounts for four sources of latency: the computation latency of executing DNN layers on tiers, the latency of serialization and deserialization of data, the propagation latency of the links, and the data transmission latency. The total estimated latency $Lat^{est}(T, P, D)$ of a choice of early exit threshold T, partitioning P, and deployment setting D is given by:

$$Lat^{est}(T,P,D) = \sum_{k=1}^{L} Lat_k^{comp} + Lat_k^{ser} + Lat_k^{prop} + Lat_k^{txn}$$

The *average propagation latency* $Lat_k^{prop}(T, P, D)$ of a partition k is the sum of samples that exited at partition k

($E_k(T,P)$) and the remaining samples ($S_k(T,P)$) going to the tier hosting the next partition. D_k maps the current partition k to a tier, and D_{k+1} gives the deployment tier of the next partition. Hence:

$$Lat_k^{prop}(T,P,D) = S_k \sum_{j=D_k}^{D_{k+1}} PD_j + E_k \sum_{j=D_K}^{Dest} PD_j$$

The *average transmission latency* $Lat_k^{txn}(T,P,D)$ refers to the time to transmit the entire data over a particular link. When the source and destination tiers have a direct link, the transmission latency is determined by dividing the data size by the link's bandwidth. However, in multi-hop scenarios with large data where, for example, the source is in tier 0, but data processing bypasses tier 1 in favor of tier 2, packet fragmentation results in a "pipelining" effect: tier 1 starts to transmit the first network packet received from tier 0 to tier 2 while receiving the second packet from tier 0. In this scenario, the single-hop model gives an error of tens and even hundreds of milliseconds, depending on the links' bandwidths and the data's size. PORTEND overcomes this limitation by modeling fragmentation [13]. First, PORTEND determines the effective bandwidth BW_{eff} between the source and destination tiers, which is the lowest bandwidth across the involved links: $BW_{\text{eff}} = \min_{i \le a \le j}(BW_a)$. Second, PORTEND computes the fragmentation latency, L_{frag}, of an MTU-sized package over the effective bandwidth: $L_{frag} = \frac{MTU}{BW_{\text{eff}}}$, and the number of such fragments, $F = ceil(\frac{D}{MTU})$.Thus, the latency of sending all fragments over all links from i to j is:

$$L_{i \to j}(D) = (F-1)L_{frag} + \sum_{a=i}^{j} \frac{MTU}{BW_a},$$

where $(F-1)L_{frag}$ is the total transmission time for all the fragments of the packet between hop i and j, and the rest is the transmission time for the last fragment of the packet. Based on that, the average transmission latency of a partition, considering exit and stay rates, is given by:

$$Lat_k^{txn}(T,P,D) = S_k L_{D_k \to D_{k+1}}(O_{P_k}) + E_k L_{D_k \to Dest}(Rs),$$

where the first component consists of the transmission latency of samples that do not exit and the second is the transmission latency of the result of size Rs of samples that exited – i.e., sent to the destination tier $Dest$.

The *average computation latency* $Lat_k^{comp}(T,P,D)$ uses all samples entering the partition k ($S_k(T,P) + E_k(T,P)$) to calculate the computation latency given at D_k based on the profiled latency $CT_k^{D_k}$. Thus:

$$Lat_k^{comp}(T,P,D) = (S_k + E_k)\, CT_k^{D_k}$$

The *average serialization/deserialization latency* $Lat_k^{ser}(T,P,D)$ depends on the incoming rate of partition k. First, when a tier receives samples, $S_k + E_k$, it needs to deserialize the data. The data size is $O_{P_{k-1}}$ because it is the output data coming from the previous tier, P_{k-1}. After, the tier does its inference. Then, the resulting data is serialized again to be sent to the next tier. The resulting data goes to the next tier for more inference, with the rate S_k. But if the inference is over and the sample is ready to exit, the result size that needs to be serialized is Rs. Hence:

$$\begin{aligned} Lat_k^{ser}(T,P,D) =& (S_k + E_k) DT_{O_{P_{k-1}}}^{D_k} + \\ & S_k \cdot ST_{O_{P_k}}^{D_k} + E_k \cdot ST_{Res}^{D_k} \end{aligned}$$

D. Optimizing and Extending PORTEND

PORTEND offers flexibility for defining optimization goals. This section gives an overview of how PORTEND declares an optimization problem (single or multi-objective optimization) based on the end-to-end latency and accuracy. P_{lat} minimizes latency while constraining accuracy to be at least ϵ_{acc}:

$$\begin{aligned} \underset{T,P,D}{\text{argmin}} \quad & Lat^{est}(T,P,D) \\ \text{s.t.} \quad & Acc^{est}(T,P) \ge \epsilon_{acc} \\ & |D| = L, \quad |P| = L \\ & D_k \in [M],\ P_k \in [N],\ P_k > P_{k-1} \quad 1 \le k \le L \end{aligned} \tag{P_{lat}}$$

P_{acc} maximizes accuracy while constraining latency:

$$\underset{T,P,D}{\text{argmax}}\ Acc^{est}(T,P)\ \text{ s.t. }\ Lat^{est}(T,P,D) \le \epsilon_{Lat}\ \ldots \tag{P_{acc}}$$

In both optimizations, P_{lat} and P_{acc}, the problem can be modeled as constrained mixed-integer nonlinear programs (MINLP) – a general class of optimization problems that are not easy to solve efficiently [14]. While MINLP problems are generally complex to solve efficiently, PORTEND search space is small enough to be amenable to exhaustive search on a laptop (as we show in Section III-A). Of course, the PORTEND model also permits developers to handle larger problems using other solvers such as dedicated MINLP solvers [14].

III. Evaluation

We evaluate PORTEND, looking at the precision of its model estimations, the end-to-end benefits of using it to find optimal configurations, and exploring new theoretical scenarios without conducting expensive experiments.

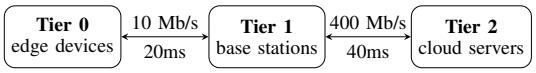
Our main two metrics are the average *end-to-end inference latency*, defined as the interval between the time an input sample is available on the edge device (tier 0) to the time the model output is available at the destination tier, and *accuracy*, the fraction of samples classified with the correct label.

A. Experimental Setup

We evaluate PORTEND on four different combinations of models and datasets: ResNet-20 and ResNet-110 [15] applied to CIFAR10 [16] for small image classification (32×32, 10 classes), EfficientNet [17] applied to ImageNet [18] for large image classification (224×224, 1000 classes), and VGGish [19] applied to AudioSet [20] for audio classification (96×64 log Mel-spectrograms [21], 632 classes). We focus on image and audio applications due to their popularity in edge computing. We also chose popular, up-to-date DNNs (ResNet, EfficientNet) rather than obsolete DNNs (AlexNet). We implement the optimization problem using MINLP in such a way that PORTEND finds the optimal solution using an

TABLE II: Models and datasets used in experiments, and the time to find the optimal 3-tier deployment using a laptop.

Model	Dataset	Blocks	Optimization Time
ResNet-20	CIFAR-10	10	21.2s
ResNet-110	CIFAR-10	55	30m
EfficientNet-B0	ImageNet	8	16.6s
VGGish	AudioSet	4	12.2s



Tier	EC2 Type	CPU	Cores	GPU
0 (edge device)	a1.medium*	Graviton (ARM)	1	–
1 (base station)	m4.large	Intel Xeon	2	–
2 (cloud server)	g4dn.xlarge	Intel Xeon	4	T4

*To avoid interference from the experimental harness (e.g., data loading, recording events), tier 0 was implemented on a 2-core a1.large with PyTorch constrained to 1 core, simulating an a1.medium.

Fig. 3: The topology used in our experiments.

exhaustive search. The optimizer is a single-threaded brute force search written in Python that enumerates all possible options and applies PORTEND to evaluate them. This solver was sufficiently fast for our goals (see Table II). Table II lists the models, the number of blocks in each, and the time it takes to evaluate PORTEND on all configurations (i.e., solve the optimization problem) on a 2021 MacBook M1 Pro.

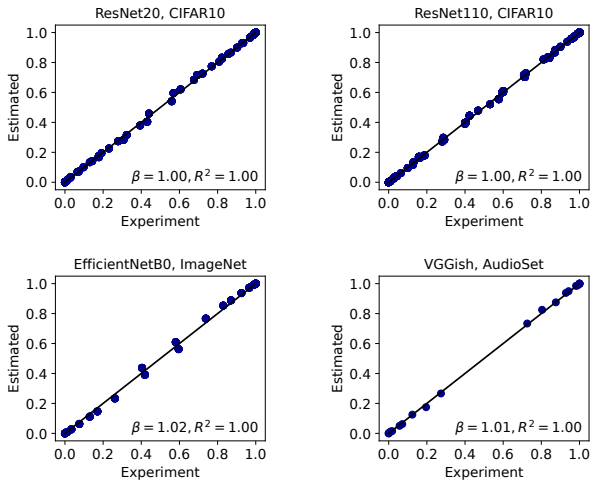
We trained the full configuration model with the training set of each dataset – 5k samples from the test set for the profiling phase and an additional disjoint set of 5k samples from the test set to evaluate the target metrics. For datasets with test sets larger than 10k samples, we select a stratified sampling of 10k points and split them into 5k for profiling and 5k for testing. We used PyTorch for training and inference.

All experiments use a 3-tier edge computing topology emulated on AWS EC2 instances, as depicted in Figure 3. We configured network bandwidth and latency using Linux Traffic Control and the routing with Linux IP route. The message exchange between tiers happens via ZeroMQ. Additionally, the evaluation employs a backpressure mechanism that limits the ingestion rate according to the system bottleneck. Note we do not use any AWS-specific features in our experiments.

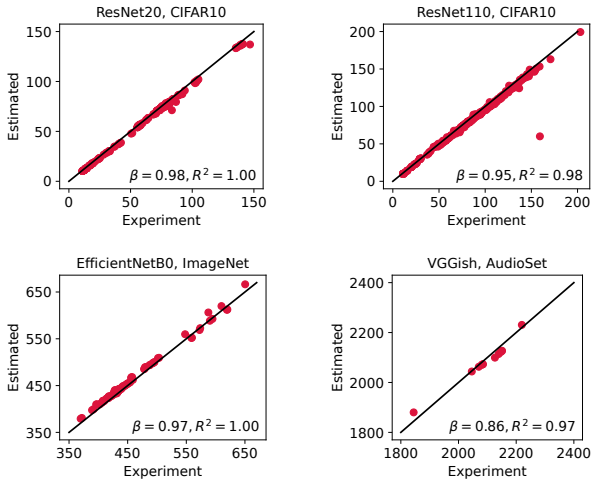
B. Validity of the Performance Model

This experiment aims to answer how well and accurately PORTEND does estimations by comparing the estimated exit rate, end-to-end latency, and accuracy from PORTEND with the results obtained from experiments on AWS. We evaluate each of the four models' performance on a test set by varying the configurations of early exit threshold T, partitioning P, and deployment setting D. We demonstrate in our graphs a mix of configurations from the Pareto frontier and outside it in such a way that we use 306 configurations for ResNet-20, 428 for ResNet-110, 92 for Efficient-Net, and 12 for VGGish.

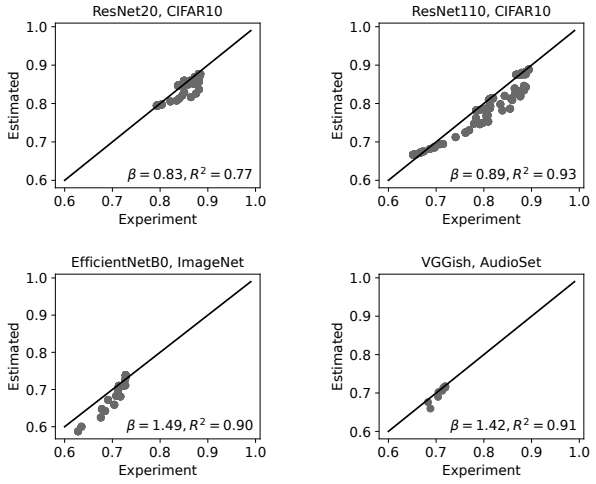
Exit Rate: The predicted exit rate of each branch, $E_K(T, P)$, is a key component of PORTEND, as it determines the esti-



(a) Estimated vs. measured exit rate of each branch.



(b) Estimated vs. measured end-to-end latency.



(c) Estimated vs. measured accuracy.

Fig. 4: Validating the accuracy of the performance model over different configurations. Each point shows the measured (X-axis) and estimated (Y-axis) performance of a single configuration (for exit rate, a single branch of a configuration).

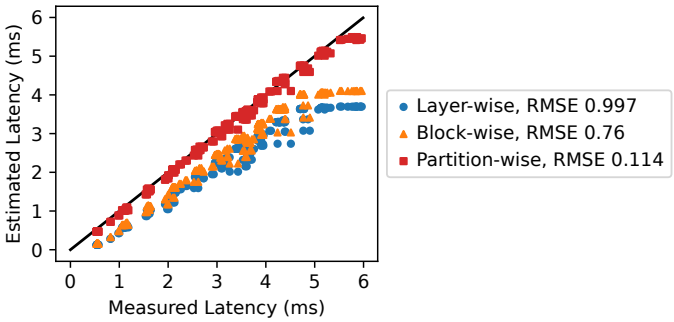


Fig. 5: Accuracy of estimating computation time for layer-wise, block-wise and partition-wise approaches.

mated accuracy and latency. Figure 4a compares the estimated exit rate $E_K(T, P)$ in the Y-axis with the measured exit rate in the X-axis. The plot shows one point for each branch k of each evaluated configuration. The closer the points are to the $Y = X$ line, the more accurate the estimation. We also use linear regression and show the regression coefficient (i.e., slope) β and the coefficient of determination R^2. R^2 near 1 means the points lie close to a straight line, while β close to 1 means that this line is close to $Y = X$. We observe that estimated exit rates match the actual exit rate as R^2 and β in all models are near to 1.

End-to-end Latency: Each point in Figure 4b shows the empirical and estimated end-to-end latency $Lat^{est}(T, P, D)$ for one configuration. Results demonstrate how accurate the estimations are for latency. The graph shows a single outlier for ResNet-110 due to an unusually large CPU inference time on tier 0 caused by external factors in the AWS environment that are beyond our control. We included this outlier when estimating β and R^2.

Computation Time: Figure 5 shows the estimated latency (Y-axis), the measured latency (X-axis), and the root mean square error (RMSE). Our proposed partition-wise outperforms layer-wise and block-wise as they systematically underestimate the computation latency of partitions.

Accuracy: Figure 4c compares the empirical and estimated accuracy $Acc^{est}(T, P)$. The estimations have few deviations, but the estimated points tend to underestimate accuracy rather than overestimate it. More importantly, the estimated points form a line, meaning that the measured accuracy is proportional to the estimated accuracy (i.e., high R^2). Results show that PORTEND is reliable in finding the optimal configuration and set accuracy constraints since (a) we can compare the estimated accuracy of different configurations and (b) the actual accuracy is equal to or higher than the estimation.

C. Finding Optimal Configurations

The primary goal of PORTEND is to find optimal configurations for early exits and partitionings, subject to application requirements. Hence, we target the optimization of P_{lat} in Section II-D. We set the minimum accuracy to be 2–4% points below the accuracy of the full model. As shown in Table II, the time for this NP-hard optimization is short, even running on a 2021 MacBook M1 Pro.

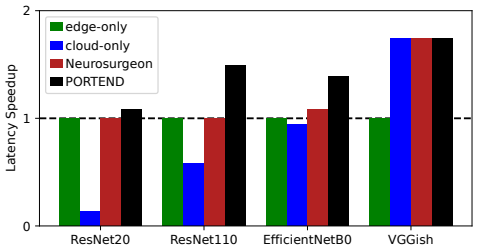


Fig. 6: Comparison of solutions' speedups (base edge-only).

TABLE III: Optimal configurations for P_{lat}.

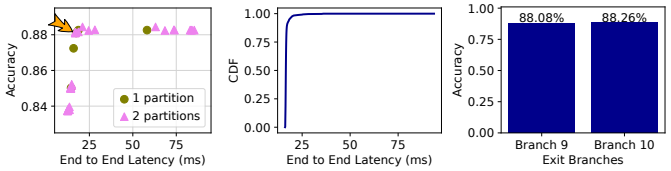
Model	Tiers [blocks]	Early exit threshold	Exit rates
ResNet-20	0 [1–9] , 1 [10]	0.7	99% , 1%
ResNet-110	0 [1–25] , 1 [26–40]	0.1	82% , 18%
EfficientNet-B0	0 [1–4] , 1 (5–8)	0.3	8% , 92%
VGGish	2 [all blocks]	–	100%

Figure 6 shows the speedup of PORTEND, cloud-only, and Neurosurgeon [5] compared to the end-to-end latency of edge-only deployment. Edge-only computes the DNN inference with a high computational overhead on an edge device. The cloud-only offloads the inference to a cloud server with a high communication overhead. Neurosurgeon splits computation between an edge device in tier 0 and the cloud in tier 2, ignoring early exits. PORTEND achieves a superior/equivalent speedup, demonstrating being a generic and crucial tool for identifying the optimal configuration on multi-tier edge computing. Note that the number of tiers in the optimal configuration highly depends on the required minimal accuracy. While the configurations in Table III use one or two tiers, for other accuracy constraints, the optimal configuration has three tiers (as shown in Figure 1).

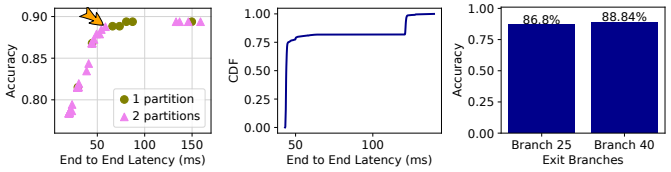
ResNet-20: Figure 7a (left-hand side) shows the Pareto frontier and the PORTEND's optimal configuration. Neurosurgeon identifies that ResNet-20 is comparatively lightweight, so fully computing it on tier 0 is faster than offloading it to the cloud over high-latency links. In contrast, PORTEND combines partitioning and early exit to gain additional speedup by placing blocks 1-9 on tier 0 and 10 on tier 1. Since 99% of samples meet the accuracy goal and exit after layer 9 as shown in Table III, PORTEND provides a 7% speedup.

ResNet-110: ResNet-110 is more computationally expensive. On the one hand, a pure partitioning approach, such as Neurosurgeon, will still determine that the optimal placement is on tier 0. On the other hand, PORTEND identifies a better deployment, which results in a significant speedup of 49% over Neurosurgeon's edge-only deployment while still maintaining the desired accuracy. Figure 7b) demonstrates that PORTEND utilizes early exit and partial DNN computation – the last partition ends after block 40 rather than 55 –, which permits PORTEND to improve its performance.

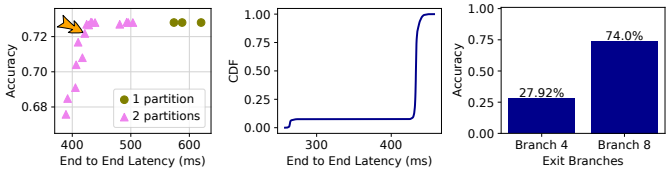
EfficientNet-B0: EfficientNet-B0 is computationally expensive. Hence, we may expect the cloud-only approach to be



(a) ResNet-20.



(b) ResNet-110.



(c) EfficientNet-B0.

Fig. 7: Pareto frontier (left), latency CDF of the optimal configuration (middle), and accuracy A_i of each selected branch (right) for different models.

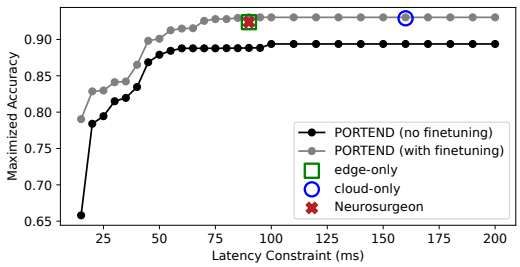
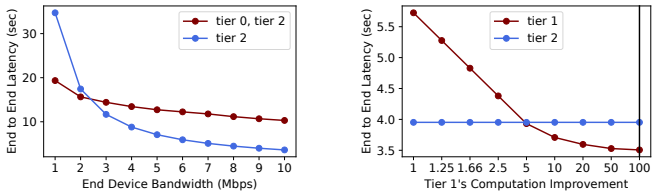


Fig. 8: Accuracy with and without fine-tuning when maximizing accuracy under a latency constraint on ResNet-110.



(a) Reducing bandwidth.

(b) Increasing compute power.

Fig. 9: Exploring hypothetical scenarios with PORTEND. The vertical line shows the relative compute power of tier 2.

beneficial. However, the larger input in this scenario incurs high transmission costs, making the cloud-only costly. In this experiment, Neurosurgeon achieves 8% speedup by splitting the model into two partitions, while PORTEND achieves a speedup of 39% using the same two partitions but benefiting from early exits. Figure 7c details the mentioned achievement with the latency CDF and branch accuracy for the selected optimal configuration. PORTEND outperforms Neurosurgeon because 8% of the samples exit in the first branch rather than going over the full model as in Neurosurgeon.

VGGish: VGGish inference is so expensive that the optimal deployment is cloud-only for all methods. In this case, PORTEND cannot utilize early exit or truncation, as no such configuration achieves the minimal desired accuracy: earlier branches are too inaccurate (e.g., $A_3 = 0.50$, well below the target accuracy of 0.71).

Alternative Optimization Problems: PORTEND is flexible and enables to handle other optimization problems. Hence, this experiment addresses the optimization of maximizing P_{acc} for the ResNet-110. Figure 8 demonstrates how PORTEND maximizes the accuracy (Y-axis) following a latency constraint (X-axis) – each point shows a configuration. Conversely, partitioning approaches such as Neurosurgeon are unable to trade-off accuracy for latency and are unable to meet the required latency constraints beyond a certain point – for ResNet-110, this point is the edge-only deployment.

Fine Tuning: PORTEND offers the option to do the fine-tuning for a certain model configuration to improve the inference accuracy further. PORTEND's estimation is based on a full configuration model, which includes all N possible exit branches. However, in practice, PORTEND selects only $L < N$ branches, depending on the chosen partitioning. In contrast, more exit branches can be more difficult to train to the same accuracy as a large number of branches (like the full configuration) tend to yield slightly lower accuracy than a model trained with fewer exit points. Thus, while the full configuration model may achieve the desired accuracy, we can often obtain additional accuracy with little effort by fine-tuning the model weights using only the selected exit branches. We fine-tuned the ResNet-110 configuration for 80 epochs to evaluate the benefit and compare the resulting accuracy. Figure 8 compares the accuracy achieved before (black) and after (gray) fine-tuning. As expected, fine-tuning improves model performance and recovers the accuracy lost when training the full-configuration model.

D. Exploring Hypothetical Scenarios

Having validated the accuracy of the PORTEND performance model, we can use it to explore theoretical scenarios without the time and cost of training models. PORTEND easily addresses questions about the impact of various factors on the performance of a new model, such as EfficientNetB7 [17] with ImageNet resized to 600×600, which would be expensive to train and evaluate for all scenarios.

Reduced Bandwidth: In a hypothetical scenario, we reduce the bandwidth between tiers 0 and 1 to reduce mobile communication costs. What would happen to performance? Figure 9a shows the effect of bandwidth variation on the end-to-end latency of the PORTEND's optimal configuration. We limit the available configurations to cloud-only deployments (blue) and those split between edge and cloud (red). As expected, latency increases as we reduce bandwidth. However, edge-

cloud deployments are less affected by this change. Thus, edge-cloud becomes the superior deployment when bandwidth drops below 2Mbps, despite the low computational power of the edge device. Of course, deploying a model on tier 0 in addition to tier 2 also incurs additional costs compared to deploying only on tier 2. PORTEND allows developers to explore the cost-effectiveness of such decisions.

Improving Compute: In a second hypothetical scenario, we consider the impact of tier 1's computational power on the choice of tiers, as there will be cost or security implications of sending the data to tier 2 for inference. How much more computational power would we need at tier 1 to avoid this? Figure 9b compares the end-to-end latency for tier 2 deployment (cloud-only) to a tier 1 deployment (without partitioning) as we multiply the computational power of tier 1 by different factors. The black vertical line shows the relative computational power of tier 2. By increasing the compute power of tier 1 five-fold, we match the performance of the tier 2 deployment. Since tier 1 is an m4.large AWS machine with 4 cores and no GPU, even a modest GPU will likely provide the required computational boost.

IV. Related Work

DNN partitioning has been proposed to accelerate inference and/or minimize the DNN's computation and transmission latency by leveraging computing resources at the edge. Previous work (Neurosurgeon [5] has focused on two-tier networks, such as one tier being edge and another tier being the cloud. Other work [7] has offered solutions for jointly optimizing partitioning and scheduling, dealing with communication and computation of multiple homogeneous DNNs placement. There has also been some effort [6] to enable DNNs to leave earlier with early exit approaches by skipping DNN layers when meeting a certain entropy threshold in strategic points of the DNN. Moreover, some works [8], [9] have focused on static partitioning with pre-defined early exit branches without exploiting entropy. More recently, solutions, such as AIMA [10], have been proposed to address DNN partitioning and early exiting. However, these solutions only provide 2-partitions, neglecting multi-tier edge networks, overheads incurred by properly transmitting data, and updated DNNs. The proposed PORTEND performance model is a comprehensive model that addresses these points.

V. Conclusions and Future Work

This work described PORTEND, a performance model for the joint optimization of DNN early exit and partitioning across a multi-tier edge network. PORTEND is a flexible and customizable performance model that enables developers to assess performance metrics with multiple configurations to achieve mono and multiple optimization goals. PORTEND leverages these features by providing an offline profiling method, a way to declare performance metrics and a way to define optimization goals. We evaluated PORTEND in an emulated edge computing platform on AWS. Results showed that PORTEND reduces the inference latency by 29% compared to the state-of-the-art Neurosurgeon. Additionally, PORTEND demonstrated how exploring hypothetical scenarios to meet environmental changes in computing and bandwidth is easy. In future work, we plan to integrate PORTEND with other techniques that trade accuracy and latency, such as model compression and quantization, and to explore environments where the optimal deployment adapts over time.

References

[1] D. Liaqat, S. Liaqat, J. L. Chen, T. Sedaghat, M. Gabel, F. Rudzicz, and E. de Lara, "Coughwatch: Real-World Cough Detection using Smartwatches," in *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2021, pp. 8333–8337.

[2] R. Hadidi, J. Cao, Y. Xie, B. Asgari, T. Krishna, and H. Kim, "Characterizing the Deployment of Deep Neural Networks on Commercial Edge Devices," in *Int. Symp. on Workload Characterization*, 2019.

[3] M. Merenda, C. Porcaro, and D. Iero, "Edge machine learning for AI-enabled IoT devices: A review," *Sensors*, vol. 20, no. 9, pp. 1–34, 2020.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE IoT Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[5] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge," in *Int. Conf. on Architectural Support for Programming Lang. and Operating Systems*, 2017.

[6] S. Teerapittayanon, B. McDanel, and H. Kung, "BranchyNet: Fast Inference via Early Exiting from Deep Neural Networks," in *Int. Conf. on Pattern Recognit.*, 2016.

[7] Y. Duan and J. Wu, "Joint Optimization of DNN Partition and Scheduling for Mobile Cloud Computing," in *Int. Conf. on Parallel Proc.*, 2021.

[8] R. G. Pacheco and R. S. Couto, "Inference Time Optimization Using BranchyNet Partitioning," in *IEEE Symposium on Computers and Communications (ISCC)*, 2020, pp. 1–6.

[9] L. Zeng, E. Li, Z. Zhou, and X. Chen, "Boomerang: On-Demand Cooperative Deep Neural Network Inference for Edge Intelligence on the Industrial Internet of Things," *IEEE Net.*, vol. 33, no. 5, 2019.

[10] Z. Huang, F. Dong, D. Shen, H. Wang, X. Guo, and S. Fu, "Enabling Latency-Sensitive DNN Inference via Joint Optimization of Model Surgery and Resource Allocation in Heterogeneous Edge," in *Int. Conf. on Parallel Processing (ICPP)*, 2023.

[11] M. Ebrahimi, A. d. S. Veith, M. Gabel, and E. de Lara, "Combining DNN partitioning and early exit," in *Proceedings of the 5th International Workshop on Edge Systems, Analytics and Networking*, ser. EdgeSys '22, 2022, p. 25–30.

[12] Z. Zhao, H. Cheng, and X. Xu, "Improved dqn-based computation offloading algorithm in mec environment," in *2022 IEEE 28th Int. Conf. on Parallel and Distributed Systems (ICPADS)*, 2023, pp. 25–32.

[13] J. McCann, S. E. Deering, J. Mogul, and B. Hinden, "Path MTU Discovery for IP version 6," RFC 8201, 2017.

[14] N. V. Sahinidis, "Mixed-integer Nonlinear Programming 2018," *Optimization and Engineering*, vol. 20, no. 2, pp. 301–306, 2019.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Conf. on Comp. Vis. & Pattern Recognit.*, 2016.

[16] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," University of Toronto, Tech. Rep., 2009.

[17] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," in *Int. Conf. on ML*, 2019.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-scale Hierarchical Image Database," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[19] S. Hershey, S. Chaudhuri, D. P. W. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, M. Slaney, R. J. Weiss, and K. Wilson, "CNN Architectures for Large-Scale Audio Classification," in *Int. Conf. on Acoustics, Speech & Signal Proc.*, 2017.

[20] J. F. Gemmeke, D. P. W. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio Set: An Ontology and Human-labeled Dataset for Audio Events," in *Int. Conf. on Acoustics, Speech, & Signal Processing*, 2017, pp. 776–780.

[21] G. Laput, K. Ahuja, M. Goel, and C. Harrison, "Ubicoustics: Plug-and-Play Acoustic Activity Recognition," in *ACM Symposium on User Interface Software and Technology*, 2018.