# Combining DNN Partitioning and Early Exit

Maryam Ebrahimi
University of Toronto
Toronto, Ontario, Canada
maryamebr@cs.toronto.edu

Alexandre da Silva Veith
University of Toronto
Toronto, Ontario, Canada
aveith@cs.toronto.edu

Moshe Gabel
University of Toronto
Toronto, Ontario, Canada
mgabel@cs.toronto.edu

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

## ABSTRACT

DNN inference is time-consuming and resource hungry. Partitioning and early exit are ways to run DNNs efficiently on the edge. Partitioning balances the computation load on multiple servers, and early exit offers to quit the inference process sooner and save time. Usually, these two are considered separate steps with limited flexibility. This work combines partitioning and early exit and proposes a performance model to estimate both inference latency and accuracy. We use this performance model to offer the best partitioned/early exit DNN based on deployment information and user preferences. Our experiments show that the flexibility in number and position of partitioning points and placement on available devices plays an important role in deciding the best output. In the future, we plan to turn this work into a "one-click" system to train and optimize models for edge computing.

## CCS CONCEPTS

• **Computing methodologies** → *Neural networks*; *Distributed computing methodologies.*

## KEYWORDS

Neural networks, partitioning, early exit, edge computing

**ACM Reference Format:**
Maryam Ebrahimi, Alexandre da Silva Veith, Moshe Gabel, and Eyal de Lara. 2022. Combining DNN Partitioning and Early Exit. In *5th International Workshop on Edge Systems, Analytics and Networking (EdgeSys '22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3517206.3526270

## 1 INTRODUCTION

Many mobile and edge applications such as pervasive health monitoring [9, 10, 14] and augmented reality [3, 15] require fast and accurate inference on data obtained at the edge device. Unfortunately, edge devices are often too resource-limited to execute state-of-the-art deep neural networks (DNN) models efficiently [12].

Two proposed approaches that address DNN inference on the edge networks are *partitioning* [5] and *early exit* [17, 18]. Partitioning offloads part of the inference computation by splitting the DNN's layers between the end-user devices, and more powerful edge and the cloud servers. Partitioning points are usually chosen to minimize end-to-end latency or maximize throughput. This approach effectively trades-off computation time for network latency.

Early exit allows some samples to execute just a portion of the DNN. This approach alters the architecture of the DNN by adding side branches that act like auxiliary classifiers. On each branch, the sample calculates a confidence measurement. If this value is above threshold, the sample takes the branch and exists in the DNN; otherwise, it continues to the next layer of the DNN. This approach effectively trades-off accuracy for inference latency.

Previous works have treated partitioning and early exit as separate steps. Partitioning approaches assume the model has already been trained and only consider how it will be deployed. Early exit approaches focus on making inference with available exits and do not consider the placement of these exits in the network topology.

**Contribution:** In this paper, we consider partitioning and early exit holistically, and propose the first performance model to jointly optimize them on multi-tier edge topologies.

Given a backbone DNN, edge network topology, device capabilities, and a dataset, the performance model estimates the expected inference accuracy and end-to-end latency including both network and DNN computation. Our model supports any number of partitions and exit points, non-consecutive placements, and positioning the consumer of inference (destination) at either the original edge device or in the cloud. This allows us to rapidly explore millions of configurations, and automatically propose the optimal partitioning and early exit configuration that minimizes latency while maximizing the accuracy.

We demonstrate the utility of the model on four difference cases, showing the benefits of joint optimizations. For example, we show a non-intuitive scenario predicted by the model where it is beneficial to place later layers of the DNN on the weak early tiers of the device hierarchy.

## 2 PERFORMANCE MODEL

We aim to offer a performance model that simultaneously captures DNN partitioning, early exit, and placement on hierarchical edge networks. Given the backbone DNN, the dataset, the network


*EdgeSys '22, April 5–8, 2022, RENNES, France*

ACM ISBN 978-1-4503-9253-2/22/04…$15.00
https://doi.org/10.1145/3517206.3526270

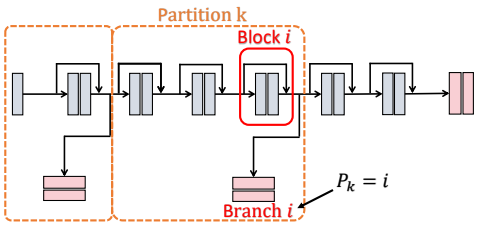



**Figure 1: The partitioned early exit model.**

topology, the capability of devices, and the preferred latency and accuracy, the model is designed to help find the optimal combination of partitioning, early exit, and placement. It must therefore support multiple partitioning points, both consecutive and non-consecutive placements on network servers, flexible destinations for results, and partial DNN computation.

## 2.1 Definitions and Notations

*Topology.* We consider an $M$-tier network comprised of end devices in the first tier, multiple edge or core servers, and a final cloud server tier; each tier is connected to the next one via a communication link. Tiers and links are indexed by $j$. We characterize a link $j$ by its *bandwidth* $BW_j$ measured in bits per second (bps) and by *propagation latency* $PL_j$ defined as the time it takes to transmit a bit from one end to the other. We characterize tier $j$'s devices with a computation ratio $R_j$, which denotes the time it takes to compute inference in this tier compared to the time it takes to compute it on a reference GPU.

*DNN.* The backbone DNN is a linear sequence of *blocks*, where each block is a single layer or a sequence of layers that form a single architectural element for architectures such as ResNet [4] and Inception [16]; we currently assume exactly one connection between subsequent blocks. The red box in Figure 1 is an example of how we assumed the blocks are. There are $N$ blocks in the backbone DNN and they are indexed by $i$. The blocks are characterized by the *output size* of the block $O_i$ in bits and the *block computation time* $CK_i$ which shows the time it takes to produce the output of block $i$ from the input of block $i$ (output of block $i-1$).

*Exits and Branches.* There is a potential *exit point* after each block, where we can add an early exit branch. We use the architecture of backbone DNN's final classifying layers as the auxiliary classifier in each branch. Thus, there are $N$ potential exit branches in the DNN model. A DNN *configuration* is a specific selection of exit points used for exit branches. A *full configuration* is when all the possible exit branches are used. Like blocks, branches are indexed by $i$. The branch $i$'s estimated accuracy, over all samples in the user-specified test set, is $acc_i$. Branch $i$ is also characterized by *branch computation time* $CB_i$ that is the time it takes to produce prediction vectors from the output of block $i$ (input of branch $i$).

The *exit rate* of the branch, denoted $\alpha_i(T)$, which is the fraction of samples that would exit at the branch $i$ in the full configuration DNN given threshold $T$. To determine whether a sample exits at a particular branch, we use the entropy of the output vector of the auxiliary classifier for that branch as a proxy for confidence. If entropy is above $T$, the sample moves to a deeper layer; if entropy is below, we stop inference and use the output of the auxiliary classifier [17].

**Table 1: Summary of notations used in Section 2.**

| Notation | Description |
|---|---|
| $M$ | Number of tiers in topology |
| $j$ | Tier index ($1 \le j \le M$) |
| $PL_j$ | Propagation latency between tier $j$ and $j+1$ |
| $BW_j$ | Link Bandwidth between tier $j$ and $j+1$ |
| $R_j$ | Computation ratio of devices in tier $j$ |
| $D$ | Destination tier |
| $N$ | Number of blocks in DNN |
| $i$ | Block/branch index ($1 \le i \le N$) |
| $O_i$ | Output size of block $i$ |
| $acc_i$ | Accuracy for branch $i$ considering all samples |
| $Ck_i$ | Computation time to produce block $i$'s output (on reference GPU) |
| $CB_i$ | Computation time to produce branch $i$'s output (on reference GPU) |
| $\alpha_i(T)$ | Fraction of samples that would exit at branch $i$ in full configuration DNN, given threshold $T$ |
| $L$ | Number of partitions ($1 \le L \le M$) |
| $k$ | Partition index ($1 \le k \le L$) |
| $P_k$ | The index of partition $k$'s last block |
| $A_k$ | The placement tier of partition $k$ |

*Partitioning and Placement.* We split each model to up to $M$ disjoint and contiguous sets of blocks, we call *partitions*. The first partition always starts with block 1. However, our performance model allows partial DNNs, in other words the last partition may exclude blocks at the end of the backbone DNN. We consider one exit branch in each partition, and it is the one after the last block in the partition, as shown in Figure 1. The number of partitions is $L$ and partitions are indexed by $k$. We denote the index of the partition's last block (or exit branch) by $P_k$. More specifically, a partition $k$ comprises blocks $P_{k-1}+1$ (the first block of the partition) to $P_k$ (the last block of the partition). Since we also allow partial DNN, it is possible that $P_L < N$. Also, always $P_k > P_{k-1}$.

*Partitioning* is the act of splitting the DNN into partitions by determining $P_k$ for all $L$ partitions. In other words, these $P_k$s are the DNN configuration that we talked about before. For Figure 1, $P = [2, 5]$ ($P_1 = 2$, $P_2 = 5$).

A *Placement* is a specific assignment of partitions to tiers. In the performance model, we assume that all partitions must be assigned to one tier, and all tiers have at most one partition. We define an assignment $A$ as a mapping from partitions to tiers: $A_k = j$ if partition $k$ is assigned to tier $j$. Note that since we are allowing both consecutive and non-consecutive placements, here we do not have constraints such as $A_k > A_{k-1}$ (consecutive), or $A_k < A_{k-1}$ (non-consecutive). Both cases are allowed.

## 2.2 Training and the Profiling Phase

To train a DNN with early exit, we optimize the sum of losses from all exit branches. However, doing so for every potential partitioning $P$ to estimate performance would be unfeasible.

Instead, the performance model estimates the resulting accuracy based on a quick offline profiling phase performed once for each backbone DNN and dataset combination.We first train a full configuration DNN, then profile $CK_i$, $CB_i$, $\alpha_i(T)$. We run the trained DNN on the dataset and set the block computation latency ($CK_i$) and branch computation latency ($CB_i$) to the 99th percentile over the training samples. To profile $\alpha_i(T)$, we consider 20 thresholds between 0 and 1 and store the fraction of samples that would exit at branch $i$ in the full configuration DNN, given each threshold.

We could use the accuracy of each branches $i$ of the full configuration DNN to set $acc_i$ for the performance model. However, as confirmed in Section 3, this risks substantially underestimating the accuracy of the resulting model since it balances the error of more exit branches than we will actually use in the final model. Instead, we use *separate training*, where we train an individual model for each exit branch $i$ and use its accuracy as $acc_i$. While this means training $N$ models, each can be trained more quickly than a joint model and to higher accuracy, giving us an upper bound for potential accuracy of the branch.

## 2.3 Estimating Performance

Given a triplet $(T, P, A)$ – a choice of assignment $A$, partition $P$, and threshold $T$ – the model estimates performance in accuracy and latency. For brevity, we omit $(T, P)$ and $(T, P, A)$ from the right-hand side of equations.

*2.3.1 Exit Rate.* The exit rate of each partition is the fraction of samples that would exit from partition $k$'s exit branch, which is the last exit branch of the partition. Therefore, to estimate the exit rate of a partition $k$ for a given threshold $T$, we add up the exit rates of all branches in that partition, from $P_{k-1}+1$, to $P_k$:

$$E_k(T,P) = \sum_{i=P_{k-1}+1}^{P_k} \alpha_i(T), \quad 1 < k < L$$

For the first partition ($k = 1$), the exit rate is $E_1(T,P) = \sum_{i=1}^{P_1} \alpha_i(T)$ since it starts with branch 1, and for the last partition ($k = L$) the exit rate is $E_L(T,P) = \sum_{i=Pk-1+1}^{N} \alpha_i(T)$, because even if the last branch of DNN is unused, still all samples will exit from the last used branch, so the summation is until last branch ($N$). As a result, the sum of the estimated exit rate for all the partitions (used branches) is always one.

*Stay rate* of partition $k$ is the fraction of samples that do not exit from its exit branch. These samples continue on to the next partition for additional computation:

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

*2.3.2 Estimated Accuracy.* To estimate the resulting accuracy of a particular partitioning, we add up the rates of samples correctly classified by each partition, weighted by the exit rate of that partition. In other words, the weighted mean of accuracies:

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

*2.3.3 Estimating Latency.* Our performance model considers three types of latencies: computation latency of executing layers on tiers, the propagation latency of the link between the tiers, and the latency caused by transmitting data over a link with limited bandwidth. The last two latencies must be tallied not only for intermediate data, but also for sending the input to the tier assigned to the first partition and the results to the consumer on either end of the network.

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

We next explain each component individually.

The *average propagation latency* $Lat_k^{prop}(T,P,A)$ of a partition $k$ comprises two parts. First, the fraction of samples exited at partition $k$ ($E_k(T,P)$) must go to the destination tier ($D$). Second, the rest of the samples that remain ($S_k(T,P)$) must move to the tier hosting the next partition. We use $A_k$ that maps the current partition $k$ to the tier it is assigned to, and $A_{k+1}$ that gives the tier assigned to the next partition. Assuming $A_k < A_{k+1}$, without loss of generality, the average propagation latency incurred by partition $k$ is:

$$Lat_k^{prop}(T,P,A) = S_k \sum_{j=A_k}^{A_{k+1}} PL_j + E_k \sum_{j=A_K}^{D} PL_j$$

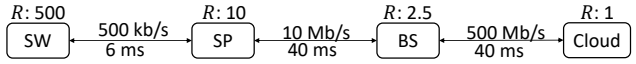
The *average transmission latency* $Lat_k^{trans}(T,P,A)$ is calculated similarly, but instead of $PL_j$ we use the transmission latency which is the size of the output data divided by the bandwidth of the link:

$$Lat_k^{trans}(T,P,A) = S_k \sum_{j=A_k}^{A_{k+1}} \frac{O_{P_k}}{BW_j} + E_k \sum_{j=A_K}^{D} \frac{O_{A_k}}{BW_j}$$

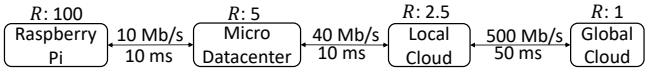
The *average computation latency* $Lat_k^{comp}(T,P,A)$ depends on which samples exit at which branch. This decision depends on the output of the branch. Thus, all samples entering a partition ($S_k(T,P) + E_k(T,P)$) incur the computation latency, regardless of whether they will exit after it or not. We introduced a computation ratio ($R_j$) for devices in each tier that shows how much these devices are slower than the reference GPU. Since $CB_i$'s and $CK_i$'s profiling are done on one reference GPU, we use the ratio $R_j$ for tier $j$ to have a better estimation of the computation latency for that tier.

$$Lat_k^{comp}(T,P,A) = (S_k + E_k)\, R_{A_k} \left( CB_{P_k} + \sum_{i=P_{k-1}+1}^{P_k} CK_i \right)$$

In the future, we plan to estimate aggregates such as the 99th percentile latency by tallying the latency of individual samples from the validation set as they run through the trained DNNs from the profiling phase. This only needs to be done once, since we can record the output of each branch.




**(a) Mobile Topology**




**(b) IoT Topology**

**Figure 2: Typologies used in the evaluation.**

## 2.4 Potential Applications

We briefly introduce two optimization problems that can be formed from our performance model. ($P_{lat}$) minimizes latency while constraining accuracy to be at least $\epsilon_{acc}$:

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

($P_{acc}$) maximizes accuracy while constraining latency:

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

The focus of this paper is on the performance model and its implications; we aim to explore optimization in future work.

# 3 EVALUATION

We first provide a preliminary evaluation of the validity of the performance model, focusing on its prediction of accuracy, exit rate, and overall latency. We then use the model to explore four different scenarios and discuss them in detail.

## 3.1 Experimental Setup

Our experiments use two networks, ResNet [4] and Inception V3 [16], and two datasets, CIFAR10 [6] and Cat and Dog [1] for image classification task. For each architecture and dataset combination, we profile branch accuracy ($acc_i$), exit rate per threshold ($\alpha_i(T)$), and block/branch computation time ($CK_i$ and $CB_i$) on a reference NVIDIA GeForce RTX 3080 GPU. Computational power $R$ is given as slowdown compared to the reference GPU, based on a review of DNN inference performance [3]. From the point of view of our model, all types of compute devices like smartphones, cameras, smart NICs, programmable switches, etc., are simply another form of "GPU". By using the computation ratio R, we estimate how much these devices contribute to computation latency.

We consider two different simulated network topologies from real-world scenarios. The mobile topology, shown in Figure 2(a), consists of a smartwatch ($R = 500$) connected to a smartphone ($R = 10$) using a 500 kb/s Bluetooth Low Energy link with 6ms latency The smartphone, in turn, is connected to an edge server ($R = 2.5$) at the base station using a 4G connection with 10 Mb/s bandwidth and 40 ms latency [2, 11, 19]. Finally, the edge server is connected to a cloud server ($R = 1$) over a 50 ms, 500 Mb/s terrestrial link. The second topology, depicted in Figure 2(b), is an industrial IoT setting that uses Raspberry Pi ($R = 100$) to collect information. It is connected via slow Wi-Fi (10 Mb/s, 10ms latency to a nearby micro datacenter ($R = 5$). The micro-datacenter talks to a local cloud server ($R = 2.5$) via a 40 Mb/s, 10 ms link, which in turn can connect to a more powerful but remote global cloud server ($R = 1$) over a faster 500 Mb/s link with 50 ms latency.

Because we simulate the network topology on a desktop-class machine, we focus on validating the accuracy and exit rate prediction of the performance model, which are not affected by the network. We plan to validate our network latency model in the full paper, as well as profile computation latencies on real devices rather than rely on a single blunt slowdown metric $R$. Also, in this work we mostly focus on image classification task. We will try other models and tasks, with different architectures and feature map sizes in the future.

## 3.2 Validity of the Performance Model

We investigate the validity of our performance model with ResNet20 and CIFAR10 on mobile topology. The performance model estimates the exit rate, accuracy, and latency of a given partitioning, assignment, and threshold. Ideally, estimated accuracy and exit rate would match those of a specialized *final model* that includes only the selected exit points (this is time consuming, as it requires training all the possible models from scratch). We trained final models for 28 randomly chosen partitionings and compared their predicted performance to the actual one for all assignments and thresholds.

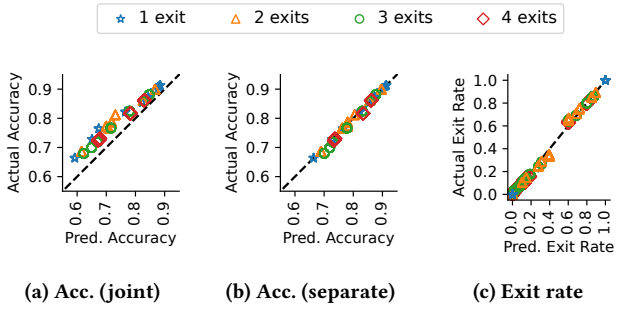
*3.2.1 Accuracy Validation.* Recall, the profiling can set $acc_i$ to either the accuracy of a full configuration model where all branches are jointly-trained or the accuracy of a separate model for each individual branch (Section 2.2).

Figures 3(a) and 3(b) compare the accuracy predicted by the performance model ($X$ axis) to the accuracy of the final DNN ($Y$ axis); the black line depicts $Y = X$. Using accuracy from joint training makes the performance model underestimate the actual accuracy achieved by the final model, since it includes fewer exit branches than the full configuration model. Conversely, using separate training accuracy for $acc_i$ leads to a much more accurate estimation. We therefore use separate training accuracy in our performance model throughout.

*3.2.2 Exit rate validation.* Exit rate plays an important role in the performance model, since it affects both accuracy and latency, and is dependant on the threshold. Figure 3(c) compares exit rates from the performance model to those of the final DNN, averaged over 20 thresholds $T$. As with accuracy, the model accurately estimates exit rates.

## 3.3 Using the Performance Model

We use the performance model to explore different combinations of application requirements, backbone, dataset, and topology. For each case, we estimate the resulting accuracy and latency of all possible partitioning ($P$), placement ($A$), and threshold ($T$) triplets,




(a) Acc. (joint) (b) Acc. (separate) (c) Exit rate

**Figure 3: Validating performance model estimates.**

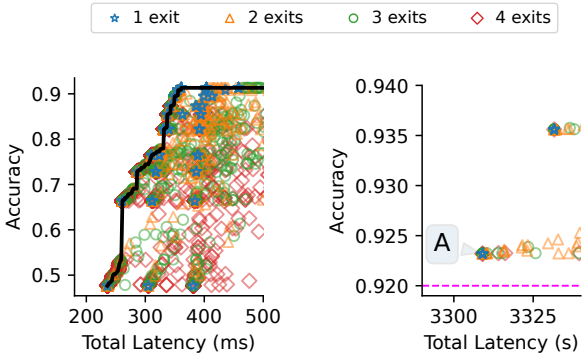



**Figure 4: Latency-accuracy trade-off plot and Pareto frontier (left). Zoomed view of case I (right) for accuracy above 92% (dashed line).**

resulting in an accuracy-latency tradeoff plot. Figure 4 (left) shows an example of such a plot, where every point is a possible triplet (We only show a sample of points). The black line is the Pareto Frontier which showing the optimal points in the trade-off. Given application latency and accuracy requirements, a user could choose a point on the frontier and the performance model would give the best partitioning, placement, and threshold to achieve it.

*3.3.1 Case I: Benefits of Trimming the DNN.* Case I demonstrates an application willing to forgo up to 3% accuracy for latency improvement. Since ResNet20 achieves a maximum of 95% accuracy on cat and dog dataset, so we ask the question: what is the optimal placement with minimal latency that achieves at least 92% accuracy on the mobile topology, which corresponds to solving $P_{Lat}$ with $\epsilon_{acc} = 92$ (Section 2.4). Figure 4 (right) shows a zoomed-in view of the area of interest (92% accuracy). In this case, the best choice (point A, 3309 ms latency and 92.32% accuracy) does not even require an early exit, merely removing the final layers of the model and using blocks 1-6 ($P = [6]$). Sometimes, simply trimming the DNN and using a partial one is the best answer, so performance models should include this possibility.

*3.3.2 Case II: Benefits of Partitioning and Early Exit.* The second case focuses on an application that requires the inference latency to be less than 300 ms on a mobile topology. The optimal answer

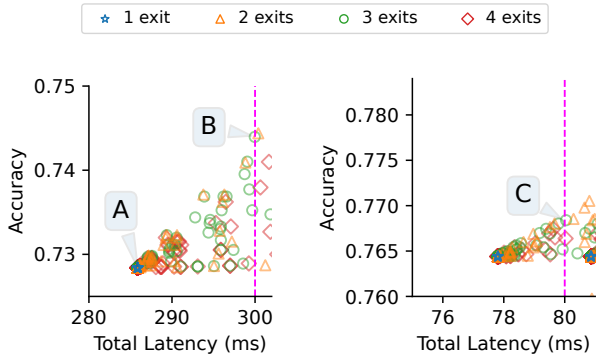



**Figure 5: Zoomed Latency-accuracy trade-off plots of case II (left) and case III (right); the dashed line represents the latency requirement for each case.**

in this application is achieved while we maximize the accuracy by solving $P_{acc}$ with a constraint on latency ($\epsilon_{Lat} < 300$). The zoomed Figure 5 (left) shows this scenario on ResNet20 model with CIFAR10 dataset. The best answer, point B, has 299.9 ms latency and 74.4% accuracy using 3 partitions ($P = [3, 9, 10]$). Point A is the best option with one exit point, which is far less accurate. In conclusion, considering more than one partitioning point with early exits can improve the latency-accuracy tradeoff.

*3.3.3 Case III: Benefits of Non-consecutive Placement.* Case III deals with a time-sensitive application on the IoT topology. Here, we constrain latency is less than 80 ms ($\epsilon_{Lat} < 80$). The result of solving $P_{acc}$ for this case with ResNet20 and CIFAR10 is shown in 5 (right). Point C shows the optimal placement $A = [3, 2]$, which surprisingly is non-consecutive, meaning later partitions are placed on early tiers. While intuitively we would expect later to be placed on later tiers (and indeed this is an implicit assumption in much existing work), this case shows that it is not always the best choice. Performance models should offer flexible placement.

*3.3.4 Case IV: Effects of Different Result Destination.* We next consider minimizing latency of Inception V3 on the IoT topology with CIFAR10, with an accuracy constraint $\epsilon_{acc} > 80\%$ (figure omitted for lack of space). We find that the optimal placement differs greatly depending on the destination of the results, even when partitioning is the same. When the result consumer is tier 1 (the edge devices), the optimal placement is $A = [2, 3]$; when it is tier 4 (the cloud), the optimal placement is $A = [2, 4]$. For both cases, partitioning $P$ is the same. A model that does not consider the destination of the result would yield a suboptimal configuration.

## 4 RELATED WORK

One of the first attempts on DNN partitioning to improve the inference latency happened in Neurosurgeon paper [5]. They used consecutive fixed placement and only one partitioning point, which is usually located at the beginning or at the end of the model based on their experiments.

BranchyNet [17] introduced the idea of early exit in inference also to save time and energy. In a follow-up paper, BranchyNet

authors decided to use their early exit model on a hierarchical computation system of end device, edge, and cloud [18]. However, the focus of [18] is not on improving latency but more on sensor fusion and different aggregations schemes and their effect on accuracy.

Li et al.[8] partitioned an AlexNet model with five pre-defined early exit branches. They chose one partitioning point and one exit point for the model while minimizing the computation and transmission latency.

Pachecom and Couto [13] considered having two partitions, where the first one is placed on edge servers and the second one on the cloud servers. This work, however, does not consider a flexible number of partitions. Moreover, the work does not explore the latency-accuracy trade-off.

ADDA [20], and SPINN [7] explored a limited number of partitions (only two) and a fixed network topology placement (first edge and second cloud). These papers do not consider the propagation latency nor the cost of sending the result to the destination; the results stay on the tier they are produced.

In summary, previous works used a limited number of partitions, network topologies, and placement strategies. Moreover, these previous works treat partitioning and early existing as separate steps. In contrast, our performance model jointly considers partitioning and early existing as it optimizes DNN deployment on multi-tier edge topologies.

## 5 DISCUSSION

We present the first performance model for joint DNN partitioning and early exit on hierarchical edge networks. We use it to investigate the accuracy/latency tradeoff in four representative scenarios and make the following observations: First, simply trimming the DNN is sometimes the optimal choice, even when allowing for early exit and partitioning; Second, having multiple partitions (and corresponding exit points) can indeed improve the accuracy/latency tradeoff compared to only one or two partitions; Third, placing later layers on weak early tiers can yield better performance than more intuitive placements; Finally, the destination of the output substantially affects the optimal placements.

We are working on enhancing the accuracy, exit rate, and latency estimation of the model, and plan to validate it on additional devices and DNNs. Using the performance model, we plan to develop an end-to-end system that receives deployment information, backbone, and application requirements and outputs a model with an optimized partitioning, early exits, and deployment plan.

## ACKNOWLEDGMENTS

This work was supported by Huawei Technologies Canada Co., Ltd and Natural Sciences and Engineering Research Council of Canada (NSERC) Grant # CRDPJ 543885-19.

## REFERENCES

[1] 2017. Kaggle Cats and Dogs Dataset. https://www.microsoft.com/en-us/download/details.aspx?id=54765
[2] Adnan Ghayas. 2020. Average 4G LTE Speed: How Fast Is 4G LTE? https://commsbrief.com/average-4g-lte-speed-how-fast-is-4g-lte/
[3] Ramyad Hadidi, Jiashen Cao, Yilun Xie, Bahar Asgari, Tushar Krishna, and Hyesoon Kim. 2019. Characterizing the Deployment of Deep Neural Networks on Commercial Edge Devices. In *IISWC*.
[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *CVPR*.
[5] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *ASPLOS '17*.
[6] Alex Krizhevsky. 2009. *Learning multiple layers of features from tiny images*. Technical Report.
[7] Stefanos Laskaridis, Stylianos I. Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D. Lane. 2020. SPINN: Synergistic Progressive Inference of Neural Networks over Device and Cloud. In *MobiCom '20*.
[8] En Li, Zhi Zhou, and Xu Chen. 2018. Edge Intelligence: On-Demand Deep Learning Model Co-Inference withDevice-Edge Synergy. *CoRR* (2018).
[9] Daniyal Liaqat, Mohamed Abdalla, Pegah Abed-Esfahani, Moshe Gabel, Tatiana Son, Robert Wu, Andrea Gershon, Frank Rudzicz, and Eyal De Lara. 2019. WearBreathing: Real World Respiratory Rate Monitoring Using Smartwatches. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 3, 2, Article 56 (June 2019), 22 pages.
[10] Daniyal Liaqat, Salaar Liaqat, Jun Lin Chen, Tina Sedaghat, Moshe Gabel, Frank Rudzicz, and Eyal de Lara. 2021. Coughwatch: Real-World Cough Detection using Smartwatches. In *ICASSP*.
[11] ISLA MCKETTA. 2019. The State of Mobile 5G in the United Kingdom. https://www.speedtest.net/insights/blog/5g-united-kingdom-2019/p
[12] Massimo Merenda, Carlo Porcaro, and Demetrio Iero. 2020. Edge machine learning for AI-enabled IoT devices: A review. *Sensors (Switzerland)* (2020).
[13] Roberto G. Pacheco and Rodrigo S. Couto. 2020. Inference Time Optimization Using BranchyNet Partitioning. In *ISCC*.
[14] Caleb Phillips, Daniyal Liaqat, Moshe Gabel, and Eyal de Lara. 2021. WristO2: Reliable peripheral oxygen saturation readings from wrist-worn pulse oximeters. In *WristSense*.
[15] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.
[16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *CVPR*.
[17] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. 2016. BranchyNet: Fast inference via early exiting from deep neural networks. In *ICPR*.
[18] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. 2017. Distributed Deep Neural Networks Over the Cloud, the Edge and End Devices. In *ICDCS*.
[19] B. Varghese, E. de Lara, A. Ding, C. Hong, F. Bonomi, S. Dustdar, P. Harvey, P. Hewkin, W. Shi, M. Thiele, and P. Willis. 2021. Revisiting the Arguments for Edge Computing Research. *IEEE Internet Computing* 25 (2021).
[20] Huitian Wang, Guangxing Cai, Zhaowu Huang, and Fang Dong. 2019. ADDA: Adaptive distributed DNN inference acceleration in edge computing environment. *ICPADS* (2019).