HiTDL: High-Throughput Deep Learning Inference at the Hybrid Mobile Edge

Jing Wu, Lin Wang, Qiangyu Pei, Xingqi Cui, Fangming Liu*, *Senior Member, IEEE*, Tingting Yang, *Member, IEEE*

Abstract—Deep neural networks (DNNs) have become a critical component for inference in modern mobile applications, but the efficient provisioning of DNNs is non-trivial. Existing mobile- and server-based approaches compromise either the inference accuracy or latency. Instead, a hybrid approach can reap the benefits of the two by splitting the DNN at an appropriate layer and running the two parts separately on the mobile and the server respectively. Nevertheless, the DNN throughput in the hybrid approach has not been carefully examined, which is particularly important for edge servers where limited compute resources are shared among multiple DNNs. This paper presents HiTDL, a runtime framework for managing multiple DNNs provisioned following the hybrid approach at the edge. HiTDL's mission is to improve edge resource efficiency by optimizing the combined throughput of all co-located DNNs, while still guaranteeing their SLAs. To this end, HiTDL first builds comprehensive performance models for DNN inference latency and throughout with respect to multiple factors including resource availability, DNN partition plan, and cross-DNN interference. HiTDL then uses these models to generate a set of candidate partition plans with SLA guarantees for each DNN. Finally, HiTDL makes global throughput-optimal resource allocation decisions by selecting partition plans from the candidate set for each DNN via solving a fairness-aware multiple-choice knapsack problem. Experimental results based on a prototype implementation show that HiTDL improves the overall throughput of the edge by $4.3\times$ compared with the state-of-the-art.

Index Terms—Deep learning inference, Edge computing, Resource allocation, Systems for machine learning.

1 Introduction

The rapid development of artificial intelligence has rendered deep learning (DL) into a promising solution for audio or video processing in modern mobile applications. Applications like Google Assistant or Apple AR typically employ pre-trained deep neural networks (DNNs) to perform inference tasks such as speech recognition [1], natural language processing [2], [3], and object recognition [4], [5], [6], [7]. Inference tasks take audio or image data as input and use DNNs to generate predictions. Thanks to its remarkable accuracy, DL has become the *de facto* approach for inference in mobile applications.

However, DNNs are hard to deploy in the mobile environment due to their intensive computation requirements. In general, there are three approaches for DNN deployment: *mobile-only*, *server-only*, and *hybrid*. The mobile-only approach relies on the mobile devices' local processing and energy power to deal with inference tasks. Due to the limitation in computation capacity and energy, mobile devices fail to support the computation-intensity state-of-the-art models, e.g., for speech recognition and natural language processing [8], [9]. Alternatively, there are pre-trained models offered by embedded devices-oriented frameworks, e.g., TensorFlow Lite[1], which optimize models by quantization, pruning, etc. [10], [11]. However, these optimizations are at the expense of accuracy [12]. The server-only approach leverages the more abundant server resources (with more powerful CPUs and sometimes also equipped with GPUs or other accelerators [13]) to run full-size DNNs with high accuracy by sending the raw input data to a cloud or edge server. However, the raw input data (such as images or audio/video clips) can be large in size, and sending it over the (wireless) network can introduce significant latency as well as performance variations [14], [15]. The hybrid approach aims to reap the benefits of the former two by partitioning the full-size DNN into two parts and run them on the mobile and the server respectively [8], [14], [16], [17], [18], [19]. The mobile and the server exchange intermediate data for the DNN layer at the partition point. Through adapting the partition point, the hybrid approach can ensure minimal inference latency under dynamic network conditions. With the rapid penetration of edge platforms, the hybrid approach has become a promising solution for mobile DNN inference [14], which we call the *hybrid mobile edge*.

Despite its high promise, provisioning DNNs at the hybrid mobile edge is challenging. An overview of the problem is shown in Figure 1. The DNN for a mobile application is partitioned between the mobile device and the edge,

- *This work was supported in part by the NSFC under Grant 61761136014 and 61520106005, in part by National Key Research & Development (R&D) Plan under grant 2017YFB1001703. Lin Wang was supported in part by DFG Collaborative Research Center 1053 MAKI B2. (Corresponding author: Fangming Liu)*
- *J. Wu, Q. Pei, X. Cui, and F. Liu are with the National Engineering Research Center for Big Data Technology and System, the Services Computing Technology and System Lab, Cluster and Grid Computing Lab in the School of Computer Science and Technology, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan 430074, China. E-mail: {wujinghust, peiqiangyu, xingqicui}@hust.edu.cn, fangminghk@gmail.com*
- *L. Wang is with VU Amsterdam, The Netherlands and TU Darmstadt, Germany. E-mail: lin.wang@vu.nl*
- *T. Yang is with Peng Cheng Laboratory, Shenzhen, China. E-mail: yangtt@pcl.ac.cn*

Manuscript received xxxx xx, 2021; revised xxxx xx, 2021.

1. https://www.tensorflow.org/lite

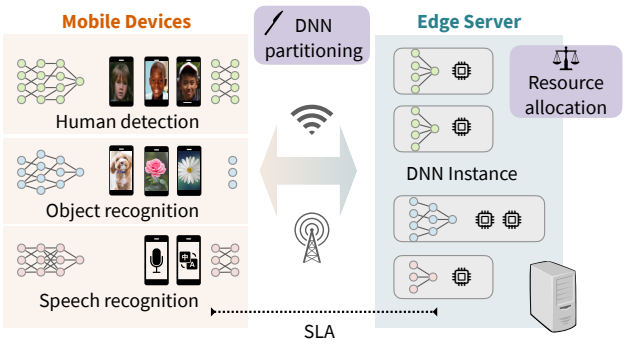


Fig. 1: Deep learning inference at the hybrid mobile edge.

where the first few DNN layers before the partition point run on the mobile device and rest layers run on the edge. The edge is shared among multiple mobile devices and each of the (partial) DNNs running on the edge is allocated a certain amount of resources for running its instance(s). On the one hand, DNN-powered mobile applications typically impose strict service-level agreements (SLAs) such as constrained end-to-end inference latency to ensure good user experience [8], [20], [21], [22], [23]. This requires to search for proper DNN-specific configurations (e.g., the DNN partition point and the server resource allocation) *under dynamic network conditions*. On the other hand, achieving high system throughput, defined as the number of served inference requests per unit time, is complicated due to the following intertwined factors: (1) Multiple DNNs share the limited edge resources which have to be allocated holistically for all co-located DNNs with fairness considered. (2) Applications running these DNNs have different priorities, e.g., mission-critical applications must be prioritized over entertainment applications when facing resource contention. (3) The co-location of DNNs on the same edge platform leads to inherently complex performance interference, which needs to be considered when guaranteeing SLAs.

Numerous management frameworks for deep learning inference exist today, including Clipper [24], GrandSLAm [22], InferLine [23], and ALERT [25]. However, these solutions mainly focus on the data center environment so far. A recent line of studies explore how to achieve high inference throughput via dynamic request batching and resource scaling (including device selection and horizontal scaling) for DNN inference serving pipelines [22], [23]. The hybrid mobile edge involves network dynamics which imposes unique challenges and requires specific treatment [8].

In this paper, we propose HiTDL (High Throughput Deep Learning)—a runtime framework for DNN provisioning at the hybrid mobile edge. HiTDL aims to achieve the optimal overall system throughput by making informed decisions on DNN partitioning and resource allocation, considering priority, fairness, and performance interference, while guaranteeing the SLAs of the DNNs. In essence, HiTDL strikes to reconcile the following conflicting relationships: (1) Pushing more DNN computation to the mobile would improve the overall system throughput since the DNN workload on the shared, resource-limited edge will be reduced. (2) Pushing more DNN computation to the edge would help with the SLA guarantee as the DNN runs much faster on the edge. Since the limited edge resources are shared among multiple DNNs, HiTDL needs to harmonize such reconciliations for all co-located DNNs.

To achieve the above goals, HiTDL first builds performance prediction models by profiling the throughput and latency of different DNNs, as well as their co-location performance interference, under varying resource availabilities and partition points. Based on these performance models, HiTDL then generates a bag of candidate partition plans that can meet the SLA for each DNN under the current network bandwidth condition. A partition plan consists of the DNN partition point and the amount of computing resources allocated to the (partial) DNN instance to run on the edge. Finally, HiTDL holistically decides the resource allocation for all DNNs to achieve the maximum overall system throughput, which is measured by a utility function defined as a weighted sum of the throughputs of all the DNNs running on the edge. The weight can be used to encode the priority of each DNN. More specifically, HiTDL optimizes the utility function by making joint decisions for the partition plan and the resources allocated for each DNN based on solving a variant of the fairness-aware multiple-choice knapsack problem (MCKP).

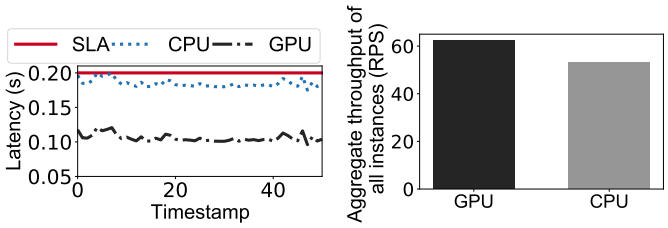
Overall, this paper makes the following contributions:

1) We design a new runtime framework to enable adaptive DNN provisioning at the hybrid mobile edge to achieve edge resource efficiency.
2) We build an accurate model for DNN performance prediction by performing comprehensive analysis on DNN throughput, latency, and co-location performance interference with respect to resource availability and DNN partition point.
3) We propose an efficient algorithm for joint DNN partitioning and edge resource allocation for hybrid DNN provisioning under limited edge resources, maximizing the system throughput while achieving SLA, priority, and fairness goals.
4) We implement HiTDL on a system prototype and carry out extensive performance evaluation. Our experimental results show that HiTDL can improve the overall edge throughput by 4.3× compared with the state-of-the-art solutions.

The rest of the paper is organized as follows: Section 2 presents the background and our motivation. Section 3 discusses our system design. Section 4 introduces our DNN performance prediction model for inference latency and throughput. Section 5 and Section 6 describe our problem formulation and algorithms for DNN partitioning and for collaborative resource allocation among multiple DNNs. Section 7 describes our implementation details and discusses our evaluation results. Section 8 represents related work. Finally, Section 9 concludes the paper.

2 Background and Motivation

This section introduces the background and discusses the motivation of our work.



(a) (b)

Fig. 2: (a) Inference latency under CPU and GPU: both can satisfy the SLA. (b) The aggregate inference throughput of the CPU-based instances is 85% of that of the GPU-based instance.

2.1 Deep Learning

DL has become an important component in many mobile applications. DL has been employed in intelligent personal assistant applications such as Google Now and Microsoft Cortana to provide intelligence by performing tasks like speech recognition [1], image and video processing [4], [5], [6], and natural language understanding [2], [3]. The proliferation of DL largely attributes to the significantly enhanced processing capability (e.g., more powerful CPUs and domain-specific accelerators like GPUs and TPUs) over the last year together with the sophistication of learning algorithms. Additionally, open-source frameworks (e.g., Tensorflow [26], Caffe [27], Pytorch [28], and Keras [29]) and publicly available well-processed datasets (e.g., LibriSpeech [30], ImageNet [31], and WordNet [32]) greatly simplify the development pipeline of DL algorithms as well as their deployment in production environments.

DL relies on DNNs which are typically structured in layers with different layers serving different purposes including convolutional, fully connected, and pooling. A DNN needs to be trained with labeled data before it can be used for inference. To attain high accuracy, DNNs usually contain a large number of layers [3] and are trained on large datasets. Both DNN training and inference are computation intensive. Efficient DNN training has been extensively studied [33], [34], [35], [36], [37], [38] and we focus on DNN inference in this paper. DNN-based mobile applications typically impose strict SLA requirements such as constrained end-to-end latency [8], [22], [39]. For example, a digital assistance service like Amazon Alexa typically require the end-to-end latency to be bounded within 200–300ms [23].

2.2 Inference Device Selection: CPU vs. GPU

For the majority of DNNs, GPUs can normally provide lower inference latency and higher throughput. However, the cost of GPUs is also higher in general. While GPUs are nowadays widely used for both DNN training and inference, it is generally preferable to use CPUs for cost efficiency, as long as CPUs can provide sufficient performance to meet the application requirements (e.g., meeting the SLA of the application). In addition, CPUs suffer low utilization in the edge environment, e.g., 74% VMs in Alibaba ENS [41] have less than 10% CPU utilization at average [42]. Such a high level of CPU under-utilization offers ample opportunity for cost optimization. Consequently, a central question to ask is how to select between CPUs and GPUs for the inference task of a specific application.

To verify this point we conduct a performance comparison between a CPU (Intel(R) Xeon(R) E5-2678 v3 2.50GHz) and a GPU (Nvidia Geforce RTX 2080 Ti) when running the Inception-v3 [40] model[2] —a popular model for image classification—with its SLA set to 200ms. The CPU contains 12 cores, where we use two cores to provision a model instance and run in total six instances. For the GPU case, we run one model instance with the GPU. Figure 2a shows the inference latency where the GPU-based model instance runs twice faster than the CPU-based model instances. However, both cases can meet the application SLA. Figure 2b shows that the aggregate throughput achieved by the CPU is 85% of that of the GPU. On the cost side, it is worth noting that the price of the CPU is around ten times lower than that of the GPU. Even with a cloud platform like Amazon EC2, the cost of a CPU-based instance is only half of that of a GPU-based instance when achieving the same inference throughput [41]. In essence, CPU achieves better cost efficiency given that both the CPU and GPU can satisfy the application SLA requirement. Therefore, in this work, we focus more on CPU-based DNN inference. Nevertheless, the proposed solution can be migrated to GPU-based scenarios as discussed in Section 6.4.

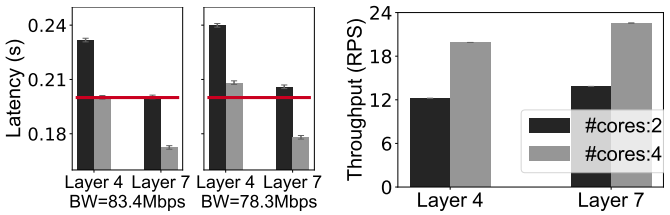
2.3 Inference Performance: Latency vs. Throughput

As discussed, the hybrid approach for DNN inference partitions the DNN into two parts and deploys these parts across the mobile and the edge respectively. This is inspired by the fact that intermediate data may be smaller than the raw input and thus, the network delay can be reduced [8], [16], [17]. However, the selection of the partition point, i.e., the DNN layer to split, is critical as partial processing, which happens on the mobile, is typically much slower than on the edge. Neurosurgeon shows that hybrid deployment of DNNs can achieve a speedup of $3.1\times$ on average and up to $40.7\times$ over the mobile-only and cloud-only approaches under varying network conditions [16].

There are multiple factors that can affect the overall inference latency, particularly the network condition, the DNN partition point, and the type and amount of allocated edge resources. While the former two have been explored by prior work [16], [17], [19], the question of how the amount of allocated edge resources affects the overall inference latency has not been carefully examined. Furthermore, they directly assume that there are sufficient resources at the edge, which are monopolized by a specific DNN. However, the resources are limited and shared/multiplexed among multiple DNNs. We take Inception-v3 [40] as an example again, and explore the inference latency with respect to the network bandwidth, the DNN partition point, and the amount of edge resources.

Figure 3a shows that under the bandwidth of 83.4 Mbps (i.e., at the 50% percentile of a real-world WiFi network trace), there are three feasible partition plans, corresponding to different numbers of CPU cores and different partition

2. The source code of Inception-v3 is given in https://github.com/tensorflow/models/blob/master/research/slim/nets/inception_v3.py



(a) (b)

Fig. 3: (a) Inference latency and (b) throughput under varying network bandwidths, when partitioned at different layers and with different numbers of CPU cores. The solid line stands for the SLA (i.e., 200ms), and the bandwidth 83.4 Mbps and 78.3 Mbps are at the 50% percentile and the 20% percentile of a real-world WiFi network trace, separately.

points. However, when the bandwidth decreases to 78.3 Mbps (i.e., at the 20% percentile), only the partition plan with the partition point of layer seven and the CPU core number of four is feasible. Hence, deciding the partition point and the edge resources requires exploring the complex relationship between DNN partitioning and resource allocation under dynamic network conditions. The DNN inference throughput, as another crucial performance metric to show if the allocated resources to DNNs are efficiently utilized, should also be explored. We evaluate the inference throughput of Inception-v3 at layer four and seven while allocated with two or four CPU cores, respectively[3]. Figure 3b shows that, without increasing the number of CPU cores, only adjusting the partition point from layer four to layer seven can improve the throughput significantly. Therefore, it is a challenge but also an opportunity to examine the compound impact of the network bandwidth, partition point, and allocated edge resources to improve the DNN inference performance.

At the hybrid mobile edge, DNN models require sufficient edge resources to guarantee SLAs. In general, the more edge resources a DNN receives, the faster the inference can run and the higher inference throughput the DNN can achieve. However, we find that the benefit of edge resources on reducing inference latency and increasing inference throughout declines marginally. Based on this finding, we claim that with a fixed amount of edge resources, it is more beneficial to have multiple small DNN instances than to have one large DNN instance given the small DNN instance can already satisfy the SLA. This is confirmed by the results shown in Figure 4a. As we can see, with in total 8 CPU cores, having four 2-core Inception-v3 instances (each partitioned at layer seven under the bandwidth of 83.4 Mbps) improves the inference throughput by over $1.6\times$ compared with having only one 8-core instance while both instance types ensure the SLA.

2.4 Edge Resource Sharing

The server resources at the edge are usually shared by multiple DNNs. Co-locating DNN instances on the same edge server improves the resource utilization, but it leads to performance interference which affects the inference latency. We measure the inference latency of Inception-v3 under different numbers of CPU cores while co-locating with other models[4] and normalize it to the latency when Inception-v3 monopolizes the server. Figure 4b shows clear performance interference. As a result, quantifying such interference is critical in guaranteeing the SLA of the DNN.

Overall, edge resource allocation with the goal of maximizing the overall throughput is a complex problem that requires taking into account all the factors we have discussed. To show this complexity, we perform experiments with three DNNs, namely Inception-v3, ResNet-50 [42], and MobileNet-v1 [43] which compete for the CPU cores on a single edge server. We propose two resource allocation strategies as Plan 1 and Plan 2, which are detailed in Table 1. Note that all the models can meet their SLAs (details in Section 7.1) under the two plans. As Figure 4c shows, adjusting the resource allocation can improve the overall throughput of the edge server by $1.7\times$.

TABLE 1: System Configuration Details in Two Resource Allocation Plans.

	Plan 1			Plan 2		
	Inc.	Res.	Mob.	Inc.	Res.	Mob.
Layer	0	2	7	7	9	8
#cores	2	4	1	4	5	1
#instance	1	2	2	1	1	3

3 System Design

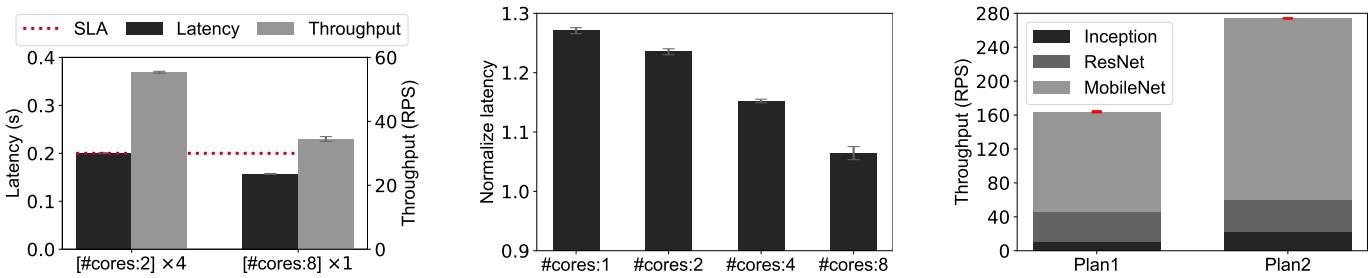
In this section, we present the architecture and key components of HiTDL – a runtime framework for adaptive DNN inference for the hybrid mobile edge. The goal of HiTDL is to maximize the overall throughput while guaranteeing application-specific SLAs, i.e., bounded end-to-end latency.

3.1 Overview

HiTDL consists of five major modules: mobile manager, model optimizer (MO), resource allocator (RA), model adapter (MA) and model zoo. Mobile manager collects information from all the connected mobile devices: computing capability, demanded DNN type, inference latency, and the estimated bandwidth to the edge server periodically. Using such information together with the detailed DNN architecture information from the model zoo, MO profiles the DNN inference performance and generates all feasible DNN partition plans that can guarantee the SLA of the application behind the DNN. RA takes all the feasible partition plans from all requested DNNs as input. It then picks specific partition plans for each DNN, and allocates resources according to each selected partition plan with the objective of maximizing the total system throughput. Finally, MA follows the model partition and resource allocation decisions to configure the mobile and the edge server and deploy the DNNs across the two ends.

3. Layer four and layer seven of Inception-v3 are MaxPool_3a_3x3 and MaxPool_5a_3x3 respectively, the shape of which are $(73, 73, 64)$ and $(35, 35, 192)$, respectively.

4. We increase the number of CPU cores for Inception-v3 and let another DNN instance take the rest cores.



(a) (b) (c)

Fig. 4: (a) Small DNN instances (with less CPU cores) are preferred over large ones; (b) Co-locating DNNs leads to performance interference; (c) Resource allocation makes a huge difference to the overall throughput of the edge server.

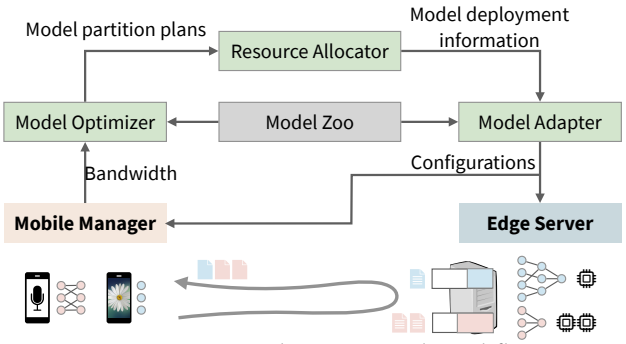


Fig. 5: HiTDL architecture and workflow.

3.2 Model Optimizer

MO is responsible for generating all the feasible partition plans for each DNN. A partition plan includes three components: the partition point (DNN layer), required edge server resources to guarantee SLA, and the achievable inference throughput. Guaranteeing the SLA is a challenging task due to at least the following two reasons: (1) A DNN usually has tens of layers and the partition point can be selected from any of the layers. (2) A variety of factors including network bandwidth, resource availability, and the cross-DNN interference can influence the decision making as we discussed in Section 2. Moreover, our goal is to achieve the maximum inference throughput while guaranteeing the SLA and the throughput is also conditioned by all these factors. Overall, the search space of the partition point is huge, and considering all these factors in resource allocation will result in exponential time complexity.

MO explores the partition plan space efficiently and generates a small set of feasible partition plans which can guarantee the application SLA. To this end, MO adopts a profiling-based method motivated by the fact that performance metrics like inference latency and throughput of DNN inference are predictable [16], [17], [22]. In particular, we build performance models for both DNN inference latency and throughput with respect to three factors: partition point, resource availability, and cross-DNN interference (Section 4). Based on these prediction models and insights, MO examines all partition points that are possible to achieve the SLA and finds out the minimum amount of resources to satisfy the SLA for each partition point (Section 5). These partition points and their corresponding resource allocation constitute the final feasible partition plans, which will be uploaded to RA for the global resource allocation on the edge server.

3.3 Resource Allocator

RA decides how to assign resources to all the co-located DNNs with the objective of achieving the highest overall system throughput. In particular, RA has to select the most suitable partition plans for each DNN from the set of all feasible partition plans (including the partition point and the demanded resources) generated by the MO as detailed above, and then determine the number of instances for each DNN. In this process, if the mobile-only execution is feasible, RA will select it as the final partition plan to improve the overall throughput. On the other hand, RA needs to prioritize the mission-critical DNNs when facing resource contention; meanwhile, it has to follow fairness constraints so that no DNN, especially of low priority, will be starved. We observe that this optimization problem can be transformed into a variant of the multiple-choice knapsack problem (MCKP) [44], which is well studied and the optimal solution can be obtained efficiently with branch-and-bound based methods [45], [46]. We will detail the problem formulation, transformation, as well as the solution in Section 6.

4 DNN Inference Performance Analysis

In this section, we provide an in-depth analysis of the performance of DNN inference with respect to inference latency and throughput. We explore the impact of three factors: resource availability, partition point, and cross-DNN performance interference, on the DNN inference performance. Based on our findings, we build prediction models for estimating the DNN inference latency and throughput.

4.1 Exploring Performance Behavior

Resource availability. We choose three popular DNNs: Inception-v3, ResNet-50, and MobileNet-v1, and measure the inference latency of these DNNs when varying the number of CPU cores from 1 to 8. Figure 7a depicts the result, where we can see that the inference latency decreases as the DNN gets more CPU cores. However, the latency improvement becomes marginal when sufficient CPU cores are supplemented. The inference latency will finally converge to a fixed value which is determined by the ratio of the non-parallelizable component in the DNN.

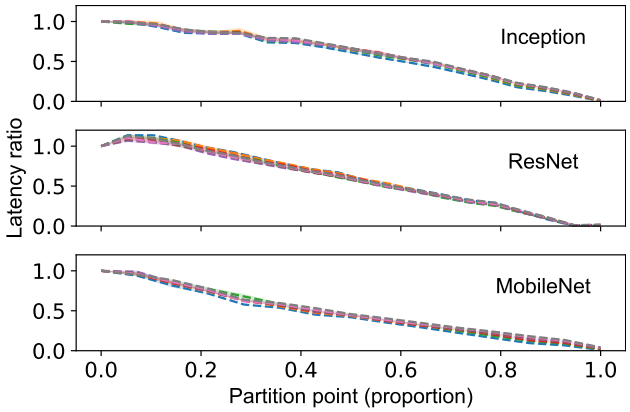


Fig. 6: Latency ratio of the partial DNN (righthand part) compared with that of the full DNN. The lines include results obtained under CPU cores from 1 through 8.

Partition point. Model partitioning is an effective way of reducing DNN inference latency [8], [16], [17]. The general practice is to partition the DNN into two parts and run them on the mobile and the server, respectively. Thus, it is important to understand the latency of the partial DNN that runs on the server under varying resources, which requires to build a prediction model for the DNN at every layer. This process is not scalable since it has to be done for every DNN considering that the DNN may contain a large number of layers. We argue that a single performance model can be used to predict the latency of a partial DNN by taking the ratio of the computing workload of the partial DNN to that of the full DNN. To verify this argument, we conduct experiments using again the three DNNs (i.e., Inception-v3, ResNet-50, and MobileNet-v1) and compare the ratio between the latency of the DNN righthand layers after the partition point (which will be run on the edge server) to that of the whole DNN under varying numbers of CPU cores. Figure 6 shows the result where we can see that the latency ratio is relatively stable under any resource conditions. As a result, we can simply build a performance prediction model for the full DNN and apply such a ratio at different layers to obtain the predicted latency of the partial DNNs resulted from the DNN partitioning at these layers.

Cross-DNN interference. As we discussed in Section 2, the co-location of DNN instances on the same edge server leads to considerable performance interference, which is detrimental for SLA guarantees. We first examine the impact of the amount of resources given to the DNN relative to that given to the co-located DNNs. In particular, we measure the inference latency of Inception-v3, ResNet-50, and MobileNet-v1 when their allocated CPU cores increase from one to eight, while the other cores are occupied by another DNN. We normalize the inference latency of the considered DNN to the inference latency when the DNN monopolizes the edge server with the same number of CPU cores. As Figure 7b shows, the increase in inference latency due to co-location interference decreases marginally when allocating relatively more resources to the considered DNN. This can be explained by the fact that when more CPU cores are used, the average load of each CPU core is reduced, thus reducing the possibility of resource contention with co-located DNNs.

TABLE 2: Parameter Definition of HiTDL.

Notation	Description
C	total number of available CPU cores
β	fairness
k	partition point
n	number of allocated CPU cores
$L(n)$	complete inference latency
$S(k)$	partial inference latency ratio
$I(n)$	interference ratio
$L(n,k)$	partial inference latency
$L^M(k), L^E(n,k)$	mobile/edge inference latency
T^U, T^D	upload/download latency
T^Q	queueing latency
$T(n,k)$	end-to-end inference latency
$H(n,k)$	inference throughout
$U(n,k)$	resource utility

Note that the amount of consumed memory is no more than 1GB [47], which the edge server can easily handle with its sufficient memory resources (128GB in our case).

We also explore the performance interference with respect to the number and the type of co-located DNN instances. As Figure 7c shows, the inference latency of Inception-v3, under different numbers of CPU cores, is quite stable when varying the number of co-located DNN instances which occupy the rest of cores. Meanwhile, we also vary the type of the co-located DNN instances, and the variation in the inference latency is negligible (less than 2ms).

4.2 Performance Modeling

Based on the above performance analysis, we model the DNN inference latency with respect to the three factors: resource availability, partition point, and cross-DNN interference. The notation is given in Table 2.

Resource availability. Inspired by Figure 7a, we use a second-order polynomial to model the full DNN inference latency with respect to the number of allocated CPU cores in the absence of cross-DNN interference. The inference latency L of a DNN when allocated n CPU cores is given by

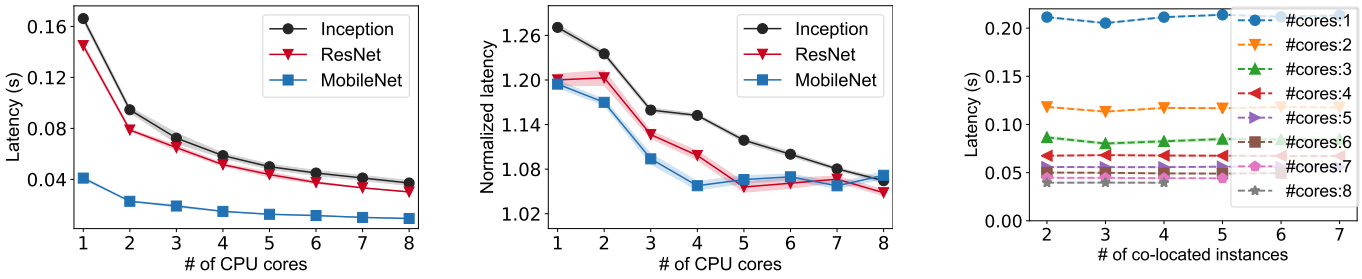
$$L(n) = c_0 + c_1/n + c_2/n^2, \tag{1}$$

where c_0, c_1 and c_2 are constants which can be obtained by performing a regression analysis on the profiled numbers for each DNN.

Partition point. As Figure 6 depicts, the latency ratio of a DNN is determined by its partition point irrespective of the number of allocated CPU cores. As a result, for each DNN we measure the latency ratio $S(k)$ at each partition layer k offline and derive the inference latency of the partial DNN when partitioned at layer k under n CPU cores as

$$L(n,k) = L(n) \cdot S(k). \tag{2}$$

Cross-DNN interference. From Figure 7b and Figure 7c we can observe that the number of CPU cores allocated to a DNN dominates the severity of performance interference,



(a) (b) (c)

Fig. 7: (a) DNN Inference latency with respect to the number of allocated CPU cores; (b) Normalized DNN inference latency under interference; (c) Inference latency under varying numbers of co-located DNN instances and allocated CPU cores.

rather than the number or the type of co-located DNN instances. Therefore, we use the interference ratio I to quantify the cross-DNN interference which is calculated as

$$I(n) = \hat{L}(n,k)/L(n,k), \tag{3}$$

where $\hat{L}(n,k)$ is the DNN inference latency when the DNN is assigned with n CPU cores and the co-located DNNs occupy the rest $C-n$ CPU cores, where C is the total number of available CPU cores on the edge server. We then employ a polynomial to model interference ratio I which can be obtained through regression as well. In a nutshell, the DNN inference latency under performance interference can be expressed by

$$L(n,k) = L(n) \cdot S(k) \cdot I(n). \tag{4}$$

Given the inference latency, the inference throughput of the DNN can be calculated as

$$H(n,k) = 1/L(n,k). \tag{5}$$

5 DNN Partition Plans Generation

A feasible partition plan for a DNN needs to meet the SLA, i.e., the end-to-end inference latency, when deployed at the hybrid mobile edge. The end-to-end latency is measured as the duration from the time an inference request is sent to the edge to the time the inference result is returned to the mobile. In the hybrid DNN deployment, a DNN with K layers is divided into two parts at the partition point k. The layers $[1,k]$ will run on the mobile, while the rest layers (i.e., layers $[k+1,K]$) will run on the edge server. The end-to-end latency is comprised of the execution time on both the mobile and the edge server, plus the network transmission delay as well as possible queuing delay in the pipeline.

We now formally analyze the end-to-end inference latency of a DNN. We denote by $L^M(k)$ the execution time of the first k layers of the DNN on the mobile and by $L^E(n,k)$ the execution time of the rest $K-k$ layers of the DNN on the edge server under the configuration $\langle n,k\rangle$ where n is the number of CPU cores. We use T^U and T^D to denote the upload and download transmission delay, respectively. The upload transmission delay can be computed as

$$T^U = d(k)/B^U, \tag{6}$$

where $d(k)$ represents the volume of the intermediate data when the DNN is partitioned at layer k. B^U is the upload bandwidth of the mobile measured at runtime. The download transmission delay is calculated as

$$T^D = O/B^D, \tag{7}$$

where O is the output size of the DNN and B^D is the download bandwidth of the mobile measured at runtime.

The edge server holds a queue for each DNN which buffers the incoming requests issued by the mobile devices before they are processed, as shown in Figure 5. We use T^Q to denote the queuing delay which can be calculated as

$$T^Q = (j-1)L^E(n,k), \tag{8}$$

where j represents that the current request is the j-th in the queue. This is because there are already $j-1$ requests waiting in the queue, and only when all of them have been processed, each of which corresponds to the execution time of $L^E(n,k)$, can the j-th request be served. Therefore, the overall time that request j spends at the edge server is $T^Q + L^E(n,k)$. To prevent the requests from queuing too long and thus violating the SLA, we limit the total queue length q as

$$q \leq R/L^E(n,k), \tag{9}$$

where R is the combined request arrival rate of the DNN and $1/L^E(n,k)$ represents the DNN inference throughput. Combining all the above analysis, the end-to-end DNN inference latency can be expressed by

$$T(n,k) = L^M(k) + T^U + T^Q + L^E(n,k) + T^D. \tag{10}$$

To meet SLA requirement, the end-to-end latency $T(n,k)$ under any feasible configuration $\langle n,k\rangle$ should satisfy

$$T(n,k) \leq SLA. \tag{11}$$

Here, we explore how to figure out all the feasible partition plans. To this end, we take two general steps in a greedy manner: 1) Filter — Depending on the current network bandwidth we filter out all the partition points where the sum of the mobile inference latency and the upload latency has already exceeded SLA (line 3-4 in Algorithm 1). 2) Evaluate — For any partition point reserved from step (1) we evaluate its demanded resources that can ensure SLA (line 5-8 in Algorithm 1). More specifically, we take the inspiration from the insights discussed in Section 2 that the inference throughput benefits more by creating more small instances rather than one big instance. We gradually increase the number of CPU cores until the SLA has been satisfied, and

then stop searching and check another partition point. The above greedy strategy ensures all feasible partition plans are allocated with the smallest possible number of CPU cores.

6 Resource Allocation

In this section, we discuss the global resource allocation problem in HiTDL. As we discussed, each feasible partition plan for the co-located DNNs has different resource demands and results in different inference throughputs. RA (a core component in HiTDL as shown in Figure 5) needs to take all the feasible partition plans of all DNNs into account to conduct resource allocation, with the goal of maximizing the overall throughput of the edge server. RA also needs to reconcile model priority and fairness. To address such a challenge, we first design a *utility function* to quantify the system throughput weighted by the DNN priority systematically. Then, we transform the global resource allocation problem, which consists of selecting specific partition plans and determining the number of instances for each DNN, into a variant of the classic multiple-choice knapsack problem (MCKP). Finally, we employ a branch-and-bound algorithm to solve the problem.

6.1 Problem Formulation

Utility function. DNNs are deployed as instances running on the edge server to offer the inference service to mobile users. Each instance occupies a certain number of CPU cores n and deploys a copy of a specific DNN model partitioned at k. We define the following utility $U(n,k)$ to evaluate the inference throughput of an instance as

$$U(n,k) = w \times H(n,k)/\min\{R, 1/T^U\}, \tag{12}$$

where $H(n,k)$ and $1/T^U$ are the expected DNN inference throughout and the network throughput, respectively. Parameter w is the relative priority of the DNN and R is the request rate (measured as requests per second or RPS) at which each mobile device issues requests. The denominator stands for the maximum number of requests that the instance has to process for each mobile device. There exists a large volume of research like tracking and result reusing [48], [49] to reduce the number of requests that the mobile offloads to the edge. Thus, R is generally smaller than $1/T^U$ in real-world scenarios. $H(n,k)/\min\{R, 1/T^U\}$ defines the number of mobile devices that the instance can serve simultaneously, representing the efficiency of the instance. The utility of a specific DNN is calculated by summing up the utility of all its instances. The overall utility of the edge server corresponds to the sum of the utilities of all its co-located DNNs. Note that the utility integrates both the throughput of each DNN and its relative priority. Therefore, the goal of RA, as maximizing the overall throughput of the edge server, can now be transformed into maximizing the utility of the edge server.

This objective of the resource allocation is to allocate the limited resources (i.e., CPU cores) of the edge server to DNNs to maximize the overall utility with SLA, priority, and fairness constraints. The problem can be formulated as

$$\begin{aligned}
\max \quad & \textstyle\sum_{i=1}^{M}\sum_{j=1}^{I_i} U_i(n_{i,j}, k_{i,j}) \cdot x_{i,j} && (13)\\
\text{subject to} \quad & \textstyle\sum_{i=1}^{M}\sum_{j=1}^{I_i} n_{i,j} \cdot x_{i,j} \le C, && (14)\\
& \textstyle\sum_{j=1}^{I_i} n_{i,j} \cdot x_{i,j} \le C\beta,\ \forall i, && (15)\\
& T_i(n_{i,j}, k_{i,j}) \le S_i,\ \forall j, && (16)\\
& x_{i,j} \in \mathbb{Z},\ \forall i,\ \forall j. && (17)
\end{aligned}$$

where M is the total number of co-located DNNs and C is the total number of CPU cores on the edge server, respectively. I_i is the number of feasible partition plans for DNN i, and $x_{i,j}$ stands for the number of instances for DNN i which implement partition plan j. $\beta \in (0,1]$ expresses the degree of fairness while allocating resources. For example, setting β to one means the least fairness and any DNN can monopolize all the CPU cores. Equation 14 limits that the total number of allocated resources can not exceed the total available resources on the edge server. Equation 15 ensures that the amount of resources allocated to each DNN follows the fairness requirement.

6.2 Problem Transformation and Solution

As formulated above, HiTDL needs to select specific partition plans for the DNNs while allocating them with appropriate resources such that the overall throughput can be maximized with SLA guarantee. This process is quite similar to the classic multi-choice knapsack problem (MCKP) regardless of certain differences. Particularly, MCKP is a knapsack problem, where the items, having specific weights together with values, are sorted into different classes. The goal of MCKP is to select one item from each class to put into the knapsack such that the total value of the selected items is maximized while the total weight does not exceed the knapsack's capacity. As for HiTDL, the feasible partition plans for each DNN model can be regarded as the sorted items in different classes, each class for a DNN.

However, there exist differences between HiTDL and MCKP. Specifically, HiTDL allows to select multiple feasible plans for each model and create multiple instances for each plan. This means that we need to release the constraint of MCKP by allowing to select and replicate specific items from each class, instead of one unique item. Besides, HiTDL needs to ensure fairness and priority while allocating resource, which can be transformed into the constraints as the maximal weights and selection order of the items from each class, respectively.

The above transformation enables HiTDL with the feasible algorithms for MCKP. In particular, the branch-and-bound (BB) algorithm, as an exact algorithm, has been widely used to solve MCKP [44], [45], [46]. BB divides the huge solution space into a series of subspaces to enumerate all possible solutions. Meanwhile, it utilizes pruning rules to evaluate the upper and the lower bound of each subspace and remove the regions that can not lead to a better solution so as to accelerate the exploration for the optimal solution.

6.3 Diverse Network Conditions

HiTDL may face the scenario where the users access the same DNN but experience different network bandwidth.

Algorithm 1: HiTDL

Input: C: total number of available CPU cores
F: β fairness, B: bandwidth
M: number of DNN models
$\mathbf{K} = \{K_i | i \in [1, M]\}$: number of model layers
$\mathbf{D} = \{d_i^j | i \in [1, M], j \in [1, K_i]\}$: volume of intermediate data
$\mathbf{S} = \{S_i | i \in [1, M]\}$: SLA requirements
Output: $\left\langle \mathbf{Z}', \mathbf{X} \right\rangle$: partition plans and number of instances

$\mathbf{Z} \leftarrow \emptyset$
/* Find feasible partition plans */
for $i \in [1, M]$ **do**
  for $k \in [0, K_i]$ **do**
    if $L_i^M(k) + d_i^k / B < S_i$ **then**
      for $n \in [1, \beta C]$ **do**
        if $T_i(n, k) \leq S_i$ **then**
          $Z[i]$.append$((k, n))$
          break
/* Conduct MCKP-based allocation */
$\mathbf{Z}', \mathbf{X} \leftarrow$ MCKP(C,β,$\mathbf{Z}$)
return $\left\langle \mathbf{Z}', \mathbf{X} \right\rangle$

To solve the issue, we can intuitively create separate DNN instances for each user, depending on their individual bandwidth. However, this may result in a large number of heterogeneous DNN instances, which can not be efficiently shared among users, thus deteriorating resource utility. As an alternative, we propose a threshold-based policy. Specifically, it adopts the lowest bandwidth of the users to allocate resource (i.e., DNN instances) when their difference in bandwidth is no more than a specific threshold; otherwise, separate instances are created accordingly. This threshold-based policy allows the users to share instances at the expense of over-allocation, especially for the users in good network conditions. HiTDL thus requires to carefully set its threshold to achieve a high utility.

6.4 Extension to GPU

While HiTDL focuses on the allocation of CPU cores among concurrent DNNs, it can be similarly transferred to GPU resource allocation. In the GPU case, we can leverage CUDA Multi-Process Services (i.e., MPS) to slice the GPU computing resource, i.e., Streaming Multiprocessors (SMs) of GPUs, into identical blocks [50], [51]. Then, a block can be treated as the basic computing unit like a CPU core to involve in profiling the inference performance of DNNs and conducting the multi-plan partition as well as the MCKP-based allocation.

7 Evaluation

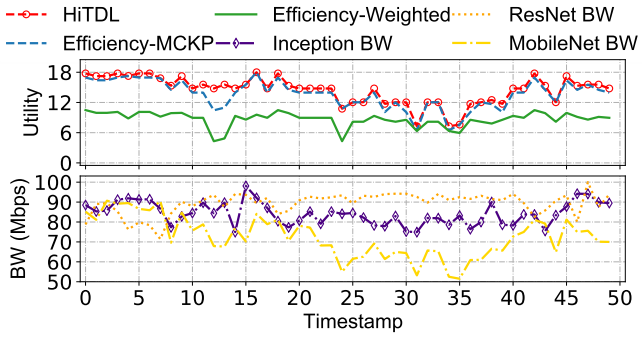
We have implemented a system prototype for HiTDL, based on which we have conducted extensive experiments to validate the performance of HiTDL under various conditions.

7.1 Implementation and Setup

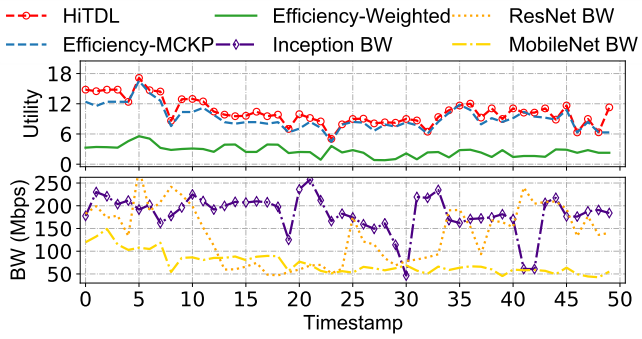
Our system prototype uses a server equipped with Intel(R) Xeon(R) E5-2678 v3 2.50GHz CPU with 12 physical CPU cores, running Ubuntu 16.04.4, as the edge server. To verify the scalability, we use another server equipped with a Intel(R) Xeon(R) Platinum 8269CY 2.50GHz CPU with 26 physical CPU cores. Since the system involves many mobile devices, we use Raspberry Pi 4 and Jetson Nano as the mobile device. To increase heterogeneity, we also use a separate server with less powerful CPUs, running Ubuntu 16.04.4, to emulate mobile devices. The DNN runtime is TensorFlow v1.14 in Python 3.7. For controlling the number of CPUs for each DNN, we configure the option `intra_op_parallelism_threads` in TensorFlow, which regulates the size of the thread pool used to accelerate the DNN operations. We implement each DNN instance as a separate subprocess while HiTDL runs in the main process, which controls the lifecycle (e.g., starting and termination) of DNN instances. The mobile device establishes a TCP connection to the edge server for exchanging the intermediate data. Besides, the mobile device depends on the partition decision from the edge server to initiate its partial DNN. In order to hide the network communication latency of this process, we let the mobile device execute the current partial DNN while listening for the new decision.

We implement five DNN models in our prototype, namely Inception-v3, ResNet-50, and MobileNet-v1, EfficientNet-B0 [52], and Conv-TasNet [53]. The first three models involve the comparison in overall performance in Section 7.2. Then, EfficientNet-B0 and Conv-TasNet are introduced additionally to verify the system's scalability in Section 7.6. The mobile-only latencies of these models (without interference) are 0.221s, 0.192s, 0.164s, 0.241ms, and 0.407ms, respectively. We set SLA as 90% of each model's mobile-only latency [8]. It is worth noting that due to the influence of the factors like available mobile resources and cross-application interference, the practical mobile inference latency may differ from the original measurement. To solve this issue, HiTDL makes Mobile Manager (a core module detailed in Section 3) monitor and report the performance deviation to the ones stored in Model Zoo (a core module) periodically to calibrate the stored model information.

HiTDL has hyper-parameters as fairness and priority. Particularly, the fairness defines the maximal ratio of CPU cores that each model can obtain. It should make the system approach the "absolutely" fair allocation as close as possible and meanwhile leave a reasonable space to improve the overall utility. On the other hand, the priority should promise the mission-critical inference tasks with sufficient CPU cores when facing resource contention. In this work, we set the priority based on the DNNs' top-1 accuracy [54], which will be extended to more dimensions, such as the popularity and the computation intensity of inference tasks in the future. Depending on the above principles, we set the fairness as 0.45, and the priority of Inception-v3, ResNet-50, and MobileNet-v1 as 0.4, 0.4, and 0.2 while evaluating the overall performance. When investigating the scalability, the fairness is set as 0.4 while the priority of Inception-v3, ResNet-50, and MobileNet-v1, EfficientNet-B0, and Conv-TasNet configured as 0.1, 0.4, 0.1, 0.3, and 0.1, respectively.



(a)



(b)

Fig. 8: Utility comparison between HiTDL and baselines under real network traces as WiFi (a) and 5G (b).

In addition, we use the Linux tc utility to simulate the real-world network traces, namely WiFi and 5G [55] shown in Figure 8a (bottom) and Figure 8b (bottom), during evaluating HiTDL's overall performance.

7.2 Overall Performance Comparison

We first compare HiTDL with six baselines in the aspects of the total utility and the resource allocation decisions under real-world network traces. The baselines are composed by combining different DNN partitioning methods with different resource allocation strategies. We consider three other DNN partitioning methods. The **efficiency-based** method partitions the DNN at the layer that achieves the highest efficiency (defined by the utility per unit resources). The **Neurosurgeon** [16] method figures out the layer that ensures the DNN with the minimal end-to-end inference latency, and the **input-based** method offloads the inference entirely to the edge server. HiTDL adopts a multi-plan partitioning method that figures out all the feasible partition plans. For resource allocation, we consider a **weighted** allocation strategy that allocates resources proportionally to the DNN's priority.

Utility. Figure 8a and Figure 8b show the utility of HiTDL, Efficiency-MCKP, and Efficiency-Weighted under real-world WiFi and 5G traces, respectively. Input-MCKP, Neurosurgeon-MCKP, Input-Weighted, and Neurosurgeon-Weighted obtain constant utility as 6.54, 4.55, 4.85, and 3.30 under WiFi traces. Their utility varies under 5G traces but does not exceed 5.13, 3.55, 3.85, and 2.11, respectively.

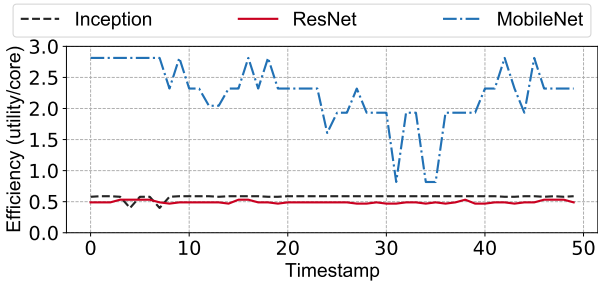


Fig. 9: Efficiency of each model under real-world WiFi traces (shown in Figure 8a (bottom)).

TABLE 3: Resource Allocation of Input-Weighted, Input-MCKP, Neurosurgeon-Weighted, Neurosurgeon-MCKP under Real Network Traces.

	Input			Neurosurgeon		
	Inc.	Res.	Mob.	Inc.	Res.	Mob.
Weighted	4	4	3	5	5	0
MCKP	4	2	5	5	0	5

We can see that HiTDL achieves the highest utility. This is because different from the other three partition methods, multi-plan partitioning dynamically figures out all the feasible partition plans, rather than one specific plan. All these feasible plans are evaluated by MCKP-based allocation comprehensively with the goal of maximizing the overall utility. Both the adaptivity of partition and the holism of allocation make HiTDL always outperform Efficiency-MCKP, and meanwhile achieve the utility improvement of around 2.2×, 3.17×, 1.69×, and 4.3× on average, when compared with Input-MCKP, Neurosurgeon-MCKP, Efficiency-Weighted, Input-Weighted, and Neurosurgeon-Weighted under WiFi, respectively. This improvement is even amplified under 5G due to the increase in bandwidth.

The utility achieved by HiTDL experiences more variations, especially when MobileNet-v1's bandwidth varies significantly. Through breaking down the overall utility, we find that HiTDL prefers to allocate more resources to MobileNet-v1 due to its high efficiency (measured by utility per core). This makes MobileNet-v1 dominate the overall utility. Furthermore, the efficiency is affected by the network bandwidth, and the decrease in bandwidth deteriorates the efficiency, thus reducing the overall utility (explained in Section 7.4) and vice versa. Particularly, Figure 9 illustrates the efficiency of each DNN under the real-world WiFi traces (shown in Figure 8a (bottom)). We can see that MobileNet-v1 experiences more fluctuations due to its severe variation in bandwidth, e.g., at timestamp 31 and timestamp 35.

Resource allocation. We now investigate the detailed resource allocation to each DNN when running under real network bandwidth traces. The baselines, namely Input-Weighted, Input-MCKP, Neurosurgeon-Weighted, and Neurosurgeon-MCKP, keep the same allocation to each model along the whole time slots, which are detailed in Table 3 where Inc., ResN., and Mob. stand for Inception-v3, ResNet-50, and MobileNet-v1, respectively. The others de-

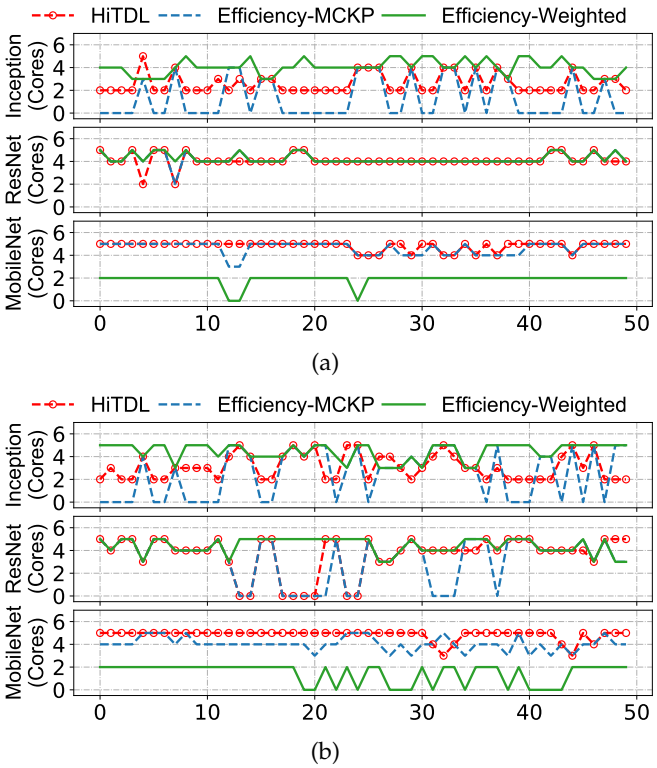


Fig. 10: Resource allocation under WiFi (a) and 5G (b).

pict in Figure 10a and Figure 10b. We can see that under both the WiFi and the 5G scenario, the weighted-based baselines allocate more resources to Inception-v3 and ResNet-50 due to their higher priority. In contrast, HiTDL, which depends on the MCKP-based allocation, allocates more resources to the DNNs of high efficiency like MobileNet-v1. Meanwhile, the partitioning method also has an impact on resource allocation. The baselines (i.e., solid green line and dashed blue line) that adopt the efficiency-based partitioning experience more variations than other partitioning methods. The reason is that efficiency-based baselines only consider one partition plan for each model. Although this plan own the highest efficiency, it calls for more CPU cores in general. When the system is unable to allocate sufficient CPU cores to create an instance for a model (e.g., Inception-v3) due to the constraint of fairness, the number of its allocated cores goes down to zero. In contrast, HiTDL adopts the multi-plan partitioning, which not only includes the plans that the efficiency-based partitioning selects but also has other candidates. When the system fails to support the plan of the highest efficiency, HiTDL can offer other feasible choices. This increases the possibility of getting resources and improves the overall utility.

Latency distribution. We now assess the latency distribution of HiTDL and the baselines under WiFi and 5G. We can see from Figure 11a and Figure 11b that HiTDL has larger inference latency than baselines due to the fact that HiTDL targets at maximizing the overall utility of the edge server instead of minimizing inference latency. Although HiTDL suffers longer latency, its SLA violation keeps at a reasonable low level, i.e., 0.08, 0.09, and 0.002 for Inception-v3, ResNet-50, and MobileNet-v1, respectively.

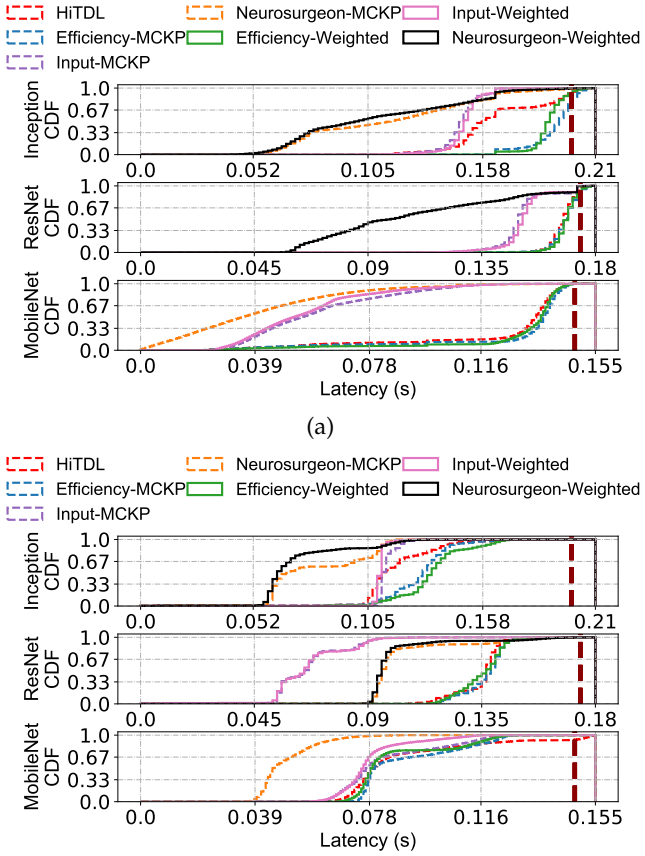


Fig. 11: Inference latency distribution under WiFi (a) and 5G (b). The dashed vertical lines represents the SLA of each DNN.

Runtime overhead and Efficiency. We evaluate the time cost and the memory consumption of HiTDL, relying on python module time[5] and perf[6], respectively. Specifically, the average time cost in evaluating three DNN models involved partition and allocation is 36ms. Meanwhile, the runtime memory consumption is no more than 2.38MB.

7.3 Performance model

This section investigates the prediction accuracy (measured by relative error) in inference latency of HiTDL and Neurosurgeon. For a fair comparison, we extend Neurosurgeon to consider the impact of available resources (i.e., CPU cores), by limiting, instead of monopolizing, the CPU cores allocated to the DNNs on the server. To understand how Neurosurgeon performs under varying resources, we build performance prediction models for Neurosurgeon as follows: We first analyze the types (e.g., Conv and Pooling) and the detailed settings (e.g., kernel size) of DNN layers, and then profile the per-layer execution time under a varying number of CPU cores, to generate the training dataset for the performance prediction model. To achieve high prediction accuracy, we train a specific prediction model for each

5. https://docs.python.org/3/library/time.html
6. https://man7.org/linux/man-pages/man1/perf.1.html

TABLE 4: Average Relative Errors of HiTDL and Neurosurgeon When Predicting DNN Inference Latency.

	Inc.	Res.	Mob.	Eff.	Conv.
Neurosurgeon	0.41	0.25	0.7	0.46	0.07
HiTDL	0.06	0.06	0.06	0.05	0.05

given number of CPU cores, with 80% of the corresponding dataset for training and 20% for testing.

Table 4 shows each DNN's average relative errors in partial latency under different numbers of CPU cores (ranging from 1 to 8). We can see that Neurosurgeon suffers higher relative errors by up to 11.6× when compared with HiTDL. The main reason is that the prediction model for Neurosurgeon works at a per-layer granularity, ignoring the fact that the DNN runtime frameworks typically apply optimizations on the execution of the DNN layers on the server [17]. This leads to that the predicted execution time is often higher than the actual execution time for the DNN layers, causing high prediction errors. In contrast, the prediction model of HiTDL works at the granularity of possible DNN segments (DNN layers that will be run together, either on the mobile device or on the edge server), taking into account the inter-layer optimizations by the DNN runtime frameworks and thus achieving higher prediction accuracy when compared with Neurosurgeon.

On the other hand, each performance model has prediction errors inevitably. Moreover, these errors will be amplified in modern DNNs (e.g., Inception-v3 and EfficientNet-B0), especially when adopting the per-layer prediction as done by Neurosurgeon. The reason is that the DNN layers have high similarity in both the type and parameters, e.g., 55.1% of the layers in Inception-v3 share the same structures. When the prediction error for a layer type that appears in the DNN frequently is high, this error will show up in all the layers of the same type, thus resulting in poor overall prediction accuracy. To mitigate this issue, one could utilize more advanced (but more complex) functions while considering more impacting factors to fit the prediction model. In the extreme case, one could even maintain a table containing the execution time for the high-frequency layer under varying resources, to improve overall prediction accuracy. However, these two optimizations either lead to additional manual efforts for decomposing the DNN, preparing training datasets, training prediction models, etc., or suffer from low generality and thus poor scalability. In contrast, HiTDL treats each layer as a part of the whole DNN (DNN segment) regardless of their specific layer type or parameters (detailed in Section 4), which simplifies the building of the prediction models effectively, without sacrificing prediction accuracy.

7.4 Model Partitioning

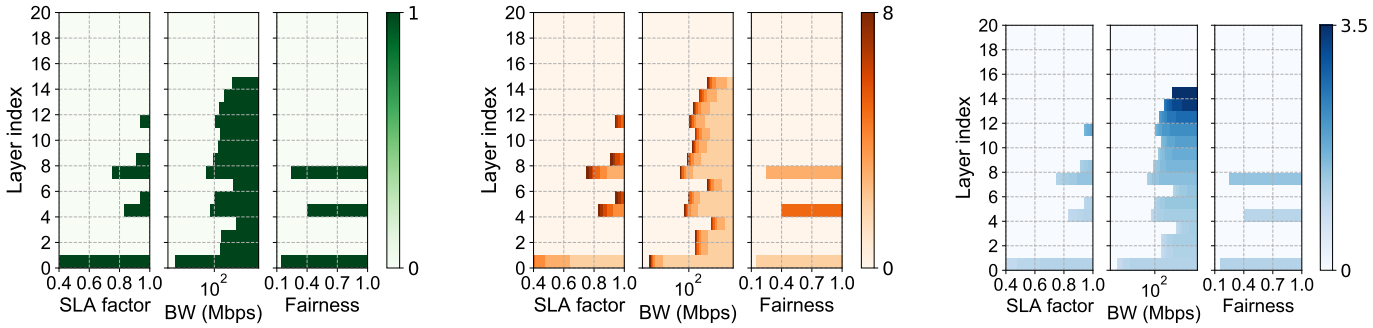
In this section, we dive into the multi-plan partitioning method used by HiTDL and analyze how network bandwidth, fairness, and SLA affect the partition performance in partition point, demanded CPU cores, and efficiency. To simplify exposition, the SLA for each model is abstracted as a ratio to its individual mobile-only latency and is marked as the SLA factor. We investigate the partition performance of Inception-v3, ResNet-50 and MobileNet-v3. We only present the results of Inception-v3, and the other two models show similar trends. Figure 12 depicts the results where the subfigures share the same axes, the y-axis is the layer index, the x-axis represents different factors, and the coordinate stands for different performance metrics. Moreover, only when the value of the coordinate is nonzero (e.g., dark green blocks in Figure 12a) is its corresponding layer (i.e., the y-axis of the coordinate) a feasible partition. Its demanded CPU cores and efficiency refer to the value that locates at the same coordinate in Figure 12b and Figure 12c, respectively.

Network. We evaluate the impact of the network bandwidth on DNN partitioning with respect to the partition layer, the number of CPU cores, and efficiency, respectively. Figure 12a (middle) shows that the number of feasible partition plans (i.e., the number of dark green blocks given each specific bandwidth) increases with the increase of the network bandwidth. This is because, given a partition layer, a lower bandwidth makes a larger network latency and leaves less time for the DNN execution on the edge server, which leads to increased demands in CPU cores. However, the impact of CPU cores on reducing the inference latency declines marginally. The edge's limited resource fails to support DNNs to achieve too short inference latency, making the partition infeasible. When the bandwidth gets higher, the constraints in inference latency are relaxed and the number of demanded CPU cores decreases, which makes the previous infeasible partitions, owing to the lack of resources, become feasible.

On the other hand, we can see from Figure 12c (right) that the efficiency benefits from the increase in network bandwidth due to more sufficient time budget left for the edge and vice versa. Meanwhile, the efficiency is higher when partitioned at a later layer given specific bandwidth and CPU cores. This is because partitioning at later layers corresponds to less computation on the edge server, and improves the inference throughput of the DNN. The increase in throughput given fixed CPU cores can effectively improve the efficiency.

SLA. To assess the impact of SLA, denoted as SLA factor, we vary it from 0.4 to 1. We can see from Figure 12a (left) that the number of feasible partition plans increases with the increase of the SLA. Similar to the increase of network, a higher SLA means looser constraints in edge inference latency as well as fewer demands in CPU cores. This allows the resource-limited edge server to support more partition plans. Note that feasible partitions for Inception-v3 take place at several specific layers (i.e., Layer 0, 4, and 7), which are generally of small volume in intermediate data to reap lower network latency.

Fairness. We investigate the impact of fairness on DNN partition performance by varying it from 0.15 to 1. As Figure 12a (right) shows, higher fairness values enable more feasible partition plans. This is because fairness defines the maximal number of CPU cores that one DNN can occupy. When it is set with a small value, the partition plans which require more cores than the upper bound become infeasible due to the lack of resources. Note that the partition point has no direct correlation with the number of CPU cores. Later partition points correspond to less computation on the edge server but lower allowable inference latency, which means



(a) Partition point (b) CPU cores (c) Efficiency

Fig. 12: Partition performance of Inception-v3 under varying SLA factors, network bandwidths, and fairness values. The y-axis is the layer index and the value at each coordinate stands for different performance metrics.

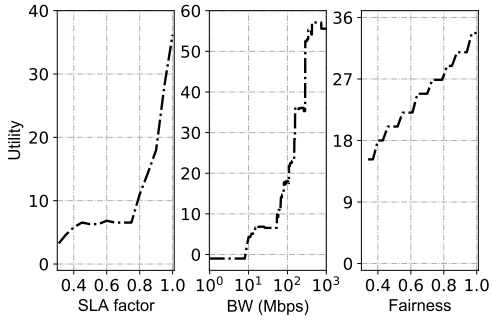


Fig. 13: System utility under varying SLA factors, network bandwidth conditions, and fairness values.

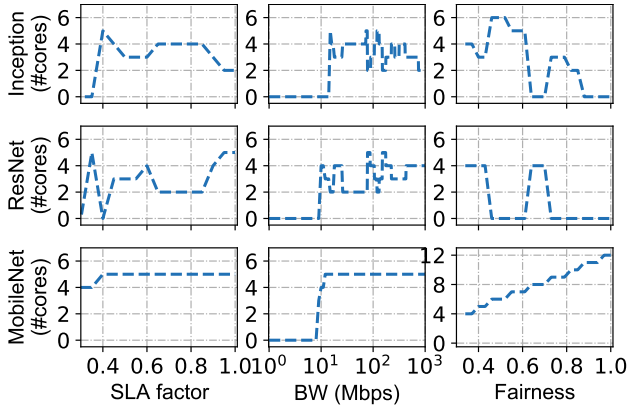


Fig. 14: Resource allocation under varying SLA factors, network bandwidth conditions, and fairness values.

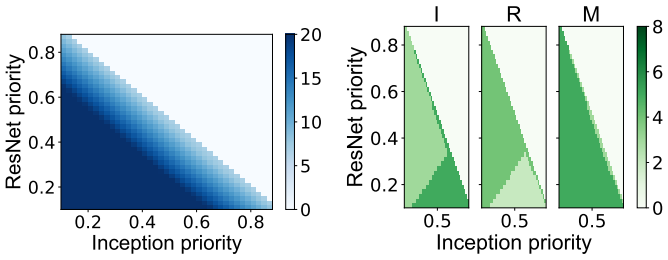
the edge server needs to finish a light task quickly. This tradeoff between computation and inference latency makes it unable to conclude that later partition layers correspond to less demanded cores and vice versa. The results shown in Figure 12b (right) support the argument that the demanded CPU cores of Inception-v3 at layer 4 is more than that of layer 7, and meanwhile, the input layer corresponds to the least cores among all feasible layers.

7.5 Resource Allocation

We now focus on the MCKP-based resource allocation in cooperation with multi-plan partitioning (explained in Section 7.4), and examine how SLA factor, fairness, network bandwidth, and DNN priority affect the allocation performance, namely the system utility and the detailed resource allocation to each DNN. During this process, we vary a specific factor while controlling the others to the default value as Section 7.1.

Utility. We can see from Figure 13 that the MCKP-based allocation is able to react to the relaxation of the constraints in SLA, network bandwidth, and fairness to continuously improve the overall utility.

DNN allocation. Excessively restricting SLAs makes the system fail to allocate any resources to some DNNs due to their lack of feasible partition plans (e.g., Inception-v3 and ResNet-50 under SLA factor of 0.3 in Figure 14 (left)). Fairness makes its impact by limiting the maximal number of CPU cores that can be allocated to a DNN. With the fairness increasing, both the number and the efficiency of



(a) Utility (b) #cores

Fig. 15: (a) System utility and (b) the number of allocated CPU cores to Inception-v3 (I), ResNet-50 (R) as well as MobileNet-v3 (M) under varying DNN priority. The x-axis and the y-axis for each subplot stand for the priority of Inception-v3 (P_I) and ResNet-50 (P_R), and the priority for MobileNet-v1 is calculated as $1 - P_I - P_R$.

feasible plans get increased. The MCKP-based allocation senses the improvement and takes actions to allocate more resources to the models of higher efficiency to improve the overall utility. Specifically, We can be seen from Figure 14 (right) that HiTDL keeps increasing the number of CPU cores allocated to MobileNet-v3, due to its higher efficiency.

The network bandwidth affects resource allocation by affecting the feasible partition plan. A higher network bandwidth brings more feasible partition plans, especially the

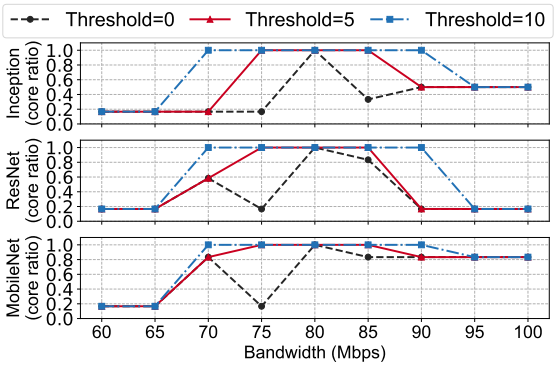


Fig. 16: Proportion of CPU cores (12 in total) allocated to the user when she faces varying bandwidths while accessing the same DNN with another user. Specifically, the latter has a fixed bandwidth as 80Mbps. Note that 1 means the two users share instances.

ones with high efficiency along with more demanded cores. The MCKP-based allocation reacts to the improvement in efficiency and adjusts resource allocations accordingly.

We now focus on how the DNN priority affects the resource allocation. As Figure 15b shows, the number of CPU cores allocated to a model increases with the increase of the model's priority as expected. Note that the number of CPU cores allocated to each DNN becomes zero when the sum of Inception-v3 priority and ResNet-50 priority exceeds 1, which renders an invalid configuration.

Diverse network conditions. We now focus on the scenario where the users access the same DNN but experience different network bandwidth. Specifically, we explore the detailed allocation for two users, one of whom experiences a fixed bandwidth as 80Mbps while the other faces varying bandwidths. Figure 16 illustrates the proportion of CPU cores (12 in total) allocated to the latter. We can see that the allocation is affected by the DNN type, network bandwidth, and threshold simultaneously. For example, HiTDL cuts down on its allocation to the user when the user suffers bad network conditions (e.g., at 60Mbps and 65Mbps), aiming at maximizing the overall utility. Meanwhile, the increase in bandwidth may also incur resource reduction (e.g., when the user accesses ResNet-50 at 95Mbps) because of the marginal impact of resources on improving utility (explained in Section 4).

7.6 Scalability.

We investigate the scalability of HiTDL when dealing with more DNNs while running on a server with a higher number of CPU cores. Here, we introduce two additional models as EfficientNet-B0 and Conv-TasNet, besides Inception-v3, ResNet-50, and MobileNet-v1. All of these involved DNNs co-run on a server equipped with 26 physical CPU cores. The detailed setup (i.e., fairness and priority) of the experiment is given in Section 7.1.

Figure 17 depicts the average utility of the baselines normalized by HiTDL. We can see that HiTDL outperforms the baselines up to 100× under both the WiFi and the 5G network traces. Meanwhile, due to the effect of multi-plan partition, HiTDL achieves the improvement by up to 8%

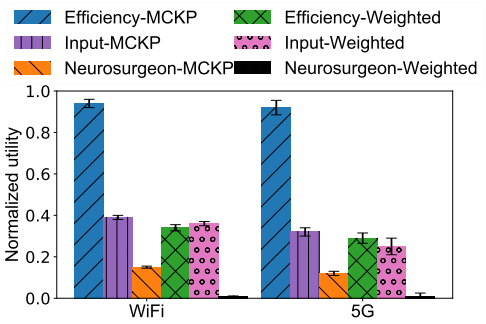


Fig. 17: Average utility (normalized by HiTDL) of baselines under different network conditions (i.e., WiFi and 5G).

when compared with Efficiency-MCKP. What's more, this improvement can be further amplified when configuring ResNet-50 with higher priority. For example, when setting ResNet-50 with the priority as 0.6 and the others with 0.1, together with the fairness as 0.35, the improvement goes by up to 25%.

8 Related Work

8.1 Edge-based DNN Inference

There have been many studies on improving the performance of mobile inference with an edge computing platform [21], [39], [56], [57], [58], [59], [60], [61]. Liu et al. propose an edge assisted object recognition system that jointly achieves high accuracy and low processing time by using dynamic RoI encoding, rendering pipeline decoupling, and fast object tracking [39]. Cachier is also an edge caching system for object recognition applications, which employs an adaptation engine to reduce the recognition delay by exploiting object spatiotemporal locality and adjusting the cache size dynamically [56]. Precog further extends the idea and not only uses edge servers but also leverages mobile devices to implement prefetching and caching on the device [57]. However, these are feature-based inference and do not employ DNNs. Guo et al. present FoggyCache, a computation reuse system that utilizes the computation results across devices at the edge by designing a two-level cache to reduce the redundant computation [58]. Guo et al. also take into consideration the fine-grained input similarity and achieve approximately deduplicate computation across applications [59]. Based on the partial-DNN sharing among applications, Jiang et al. propose Mainstream to achieve improments on aggregate application quality [60]. Wang et al. propose an adaptation-based strategy which explores applications' behavior to filter frames and adjust the number of concurrent instances within the edge so as to maximize the overall utility [21]. None of these works target the DNN resource sharing on edge servers following the hybrid DNN provisioning approach.

8.2 DNN Partitioning

Recently, DNN partitioning has been extensively studied [8], [10], [14], [16], [17], [18], [19], [62], [63]. Some works focus on the training of DNNs in a distributed environment. To optimize data parallel in NLP training, Kim et al. propose

Parallax, a data parallel training system for DNN leveraging the sparsity of model parameters [62]. Focusing on large-scale DNN models, Wang et al. presents Tofu, a system that partition the dataflow graph of large DNN models across GPU devices to speed up the training [18]. Others aims to optimize DNN inference on mobile devices. Neurosurgeon [16] is the first work to partition DNN, which profiles inference latency of each layer with respect to its categories like fully-connected, convolutional, or pooling. μLayer is an on-device inference system that achieves high inference efficiency by DNN layer partitioning and distribution between mobile CPU and mobile GPU [10]. Eshratifar et al. utilize an ILP-based method to obtain the optimal partition point for achieving the minimum latency or energy consumption [17]. Hu et al. propose DADS, a DNN partition scheme that can minimize the overall delay of a request or maximize the throughput [19]. PHsu et al. pay attention to these modern DNNs partitioning and container-based deployment for sliced DNN models to minimize the inference latency [14]. Recently, early-exits, as a new mechanism to decrease inference latency, has been introduced to DNN partitioning [64], [65]. SPINN further improves the inference latency via compressing the offloaded data and make early-exist decisions based on the complexity of the input [8]. In contrast, we study the problem of resource allocation for co-locating DNNs on the edge server with DNN partitioning and our goal is to achieve SLA guarantee and high throughput simultaneously.

9 Conclusions

HiTDL demonstrates how to provision multiple DNNs following the hybrid deployment approach over limited resources at the edge. HiTDL's goal is to achieve high aggregate throughput of all co-located DNNs while guaranteeing the SLAs of all the DNNs. HiTDL achieves its goal by building performance predication models for DNN inference throughput and latency with respect to a set of factors, generating a set of candidate partition plans for each DNN, and allocating resources by selecting specific feasible partition plans from the candidate set via solving a fairness-aware multiple-choice knapsack problem. Experimental results confirm the effectiveness of HiTDL, where HiTDL outperforms baseline solutions by $4.3\times$ in throughput.

References

[1] G. Cong, B. Kingsbury, C. Yang, and T. Liu, "Fast training of deep neural networks for speech recognition," in IEEE ICASSP, 2020, pp. 6884–6888.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in NIPS, 2017, pp. 5998–6008.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever et al., "Language models are unsupervised multitask learners," OpenAI blog, vol. 1, no. 8, p. 9, 2019.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in NIPS 2012, 2012, pp. 1106–1114.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in IEEE CVPR, 2016, pp. 770–778.

[6] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in AAAI, 2017, pp. 4278–4284.

[7] V. Nigade, L. Wang, and H. E. Bal, "Clownfish: Edge and cloud symbiosis for video stream analytics," in IEEE/ACM SEC, 2020, pp. 55–69. [Online]. Available: https://doi.org/10.1109/SEC50012.2020.00012

[8] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN: synergistic progressive inference of neural networks over device and cloud," in ACM MobiCom, 2020, pp. 37:1–37:15.

[9] A. Banitalebi-Dehkordi, N. Vedula, J. Pei, F. Xia, L. Wang, and Y. Zhang, "Auto-split: A general framework of collaborative edge-cloud AI," in ACM KDD, 2021, pp. 2543–2553.

[10] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, "μlayer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization," in ACM EuroSys, 2019, p. 45.

[11] W. Niu, P. Zhao, Z. Zhan, X. Lin, Y. Wang, and B. Ren, "Towards real-time DNN inference on mobile platforms with model pruning and compiler optimization," in IJCAI, 2020, pp. 5306–5308.

[12] https://www.tensorflow.org/lite/guide/hosted_models, 2020.

[13] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, "Cloud-dnn: An open framework for mapping DNN models to cloud fpgas," in ACM/SIGDA FPGA, 2019, pp. 73–82.

[14] K. Hsu, K. Bhardwaj, and A. Gavrilovska, "Couper: DNN model slicing for visual analytics containers at the edge," in ACM/IEEE SEC, 2019, pp. 179–194. [Online]. Available: https://doi.org/10.1145/3318216.3363309

[15] B. Zhang, X. Jin, S. Ratnasamy, J. Wawrzynek, and E. A. Lee, "Awstream: adaptive wide-area streaming analytics," in ACM SIGCOMM, 2018, pp. 236–252.

[16] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. N. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in ACM ASPLOS, 2017, pp. 615–629.

[17] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "Jointdnn: An efficient training and inference engine for intelligent mobile cloud computing services," CoRR, vol. abs/1801.08618, 2018. [Online]. Available: http://arxiv.org/abs/1801.08618

[18] M. Wang, C. Huang, and J. Li, "Supporting very large models using automatic dataflow graph partitioning," in ACM EuroSys, 2019, pp. 26:1–26:17.

[19] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in IEEE INFOCOM, 2019, pp. 1423–1431.

[20] J. Schwab, A. Hill, and Y. Jararweh, "Edge computing ecosystem support for 5g applications optimization," in ACM HotMobile, 2020, p. 103.

[21] J. Wang, Z. Feng, S. A. George, R. Iyengar, P. Pillai, and M. Satyanarayanan, "Towards scalable edge-native applications," in ACM/IEEE SEC, 2019, pp. 152–165.

[22] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, J. Mars, and L. Tang, "Grandslam: Guaranteeing slas for jobs in microservices execution frameworks," in ACM EuroSys, 2019, pp. 34:1–34:16.

[23] D. Crankshaw, G. Sela, X. Mo, C. Zumar, I. Stoica, J. Gonzalez, and A. Tumanov, "Inferline: latency-aware provisioning and scaling for prediction serving pipelines," in ACM SoCC, 2020, pp. 477–491.

[24] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in USENIX NSDI, 2017, pp. 613–627.

[25] C. Wan, M. H. Santriaji, E. Rogers, H. Hoffmann, M. Maire, and S. Lu, "ALERT: accurate learning for energy and timeliness," in USENIX ATC, 2020, pp. 353–369.

[26] https://www.tensorflow.org/, 2020.

[27] https://github.com/BVLC/caffe, 2020.

[28] https://github.com/pytorch/pytorch, 2020.

[29] https://github.com/keras-team/keras, 2020.

[30] http://www.openslr.org/12/, 2020.

[31] http://www.image-net.org/, 2020.

[32] https://wordnet.princeton.edu/, 2020.

[33] F. Xu, Y. Qin, L. Chen, Z. Zhou, and F. Liu, "λdnn: Achieving predictable distributed DNN training with serverless architectures," IEEE Transactions on Computers, vol. 71, no. 2, pp. 450–463, 2022.

[34] Q. Chen, Z. Zheng, C. Hu, D. Wang, and F. Liu, "Data-driven task allocation for multi-task transfer learning on the edge," in IEEE ICDCS, 2019, pp. 1040–1050.

[35] H. Zheng, F. Xu, L. Chen, Z. Zhou, and F. Liu, "Cynthia: Cost-efficient cloud resource provisioning for predictable distributed deep neural network training," in ACM ICPP, 2019, pp. 86:1–86:11.

[36] Q. Chen, Z. Zheng, C. Hu, D. Wang, and F. Liu, "On-edge multi-task transfer learning: Model and practice with data-driven task allocation," IEEE Transactions on Parallel and Distributed Systems, vol. 31, no. 6, pp. 1357–1371, 2020. [Online]. Available: https://doi.org/10.1109/TPDS.2019.2962435

[37] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. X. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in NeurIPS, 2019, pp. 103–112.

[38] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: generalized pipeline parallelism for DNN training," in ACM SOSP, 2019, pp. 1–15.

[39] L. Liu, H. Li, and M. Gruteser, "Edge assisted real-time object detection for mobile augmented reality," in ACM MobiCom, 2019, pp. 25:1–25:16.

[40] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," CoRR, vol. abs/1512.00567, 2015. [Online]. Available: http://arxiv.org/abs/1512.00567

[41] C. Zhang, M. Yu, W. Wang, and F. Yan, "Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving," in USENIX ATC, 2019, pp. 1049–1062. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/zhang-chengliang

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," CoRR, vol. abs/1603.05027, 2016. [Online]. Available: http://arxiv.org/abs/1603.05027

[43] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," CoRR, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[44] C. Cotta and J. M. Troya, "A hybrid genetic algorithm for the 0–1 multiple knapsack problem," in Artificial Neural Nets and Genetic Algorithms. Vienna: Springer Vienna, 1998, pp. 250–254.

[45] M. Visée, J. Teghem, M. Pirlot, and E. Ulungu, "Two-phases method and branch and bound procedures to solve the bi–objective knapsack problem," Journal of Global Optimization, vol. 12, no. 2, pp. 139–155, 1998.

[46] S. Martello and P. Toth, "A bound and bound algorithm for the zero-one multiple knapsack problem," Discrete Applied Mathematics, vol. 3, no. 4, pp. 275 – 288, 1981. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0166218X81900056

[47] P. Yu and M. Chowdhury, "Fine-grained gpu sharing primitives for deep learning applications," in MLSys, 2020, pp. 98–111.

[48] S. Naderiparizi, P. Zhang, M. Philipose, B. Priyantha, J. Liu, and D. Ganesan, "Glimpse: A programmable early-discard camera architecture for continuous mobile vision," in ACM MobiSys, 2017, pp. 292–305.

[49] L. Xie, X. Zhang, and Z. Guo, "CLS: A cross-user learning based system for improving qoe in 360-degree video adaptive streaming," in ACM MM, 2018, pp. 564–572.

[50] A. Dhakal, S. G. Kulkarni, and K. K. Ramakrishnan, "Gslice: Controlled spatial sharing of gpus for a scalable inference platform," in ACM SoCC, 2020, p. 492–506. [Online]. Available: https://doi.org/10.1145/3419111.3421284

[51] https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf, 2020.

[52] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," CoRR, vol. abs/1905.11946, 2019. [Online]. Available: http://arxiv.org/abs/1905.11946

[53] Y. Luo and N. Mesgarani, "Tasnet: Surpassing ideal time-frequency masking for speech separation," CoRR, vol. abs/1809.07454, 2018. [Online]. Available: http://arxiv.org/abs/1809.07454

[54] https://github.com/tensorflow/models/tree/master/research/slim, 2020.

[55] D. Raca, D. Leahy, C. J. Sreenan, and J. J. Quinlan, "Beyond throughput, the next generation: a 5g dataset with channel and context metrics," in ACM MMSys, 2020, pp. 303–308.

[56] U. Drolia, K. Guo, J. Tan, R. Gandhi, and P. Narasimhan, "Cachier: Edge-caching for recognition applications," in IEEE ICDCS, 2017, pp. 276–286.

[57] U. Drolia, K. Guo, and P. Narasimhan, "Precog: prefetching for image recognition applications at the edge," in ACM/IEEE SEC, 2017, pp. 17:1–17:13.

[58] P. Guo, B. Hu, R. Li, and W. Hu, "Foggycache: Cross-device approximate computation reuse," in ACM MobiCom, 2018, pp. 19–34.

[59] P. Guo and W. Hu, "Potluck: Cross-application approximate deduplication for computation-intensive mobile applications," in ACM ASPLOS, 2018, pp. 271–284.

[60] A. H. Jiang, D. L. Wong, C. Canel, L. Tang, I. Misra, M. Kaminsky, M. A. Kozuch, P. Pillai, D. G. Andersen, and G. R. Ganger, "Mainstream: Dynamic stem-sharing for multi-tenant video processing," in USENIX ATC, 2018, pp. 29–42.

[61] L. Zhou, M. H. Samavatian, A. Bacha, S. Majumdar, and R. Teodorescu, "Adaptive parallel execution of deep neural networks on heterogeneous edge devices," in ACM/IEEE SEC, 2019, pp. 195–208. [Online]. Available: https://doi.org/10.1145/3318216.3363312

[62] S. Kim, G. Yu, H. Park, S. Cho, E. Jeong, H. Ha, S. Lee, J. S. Jeong, and B. Chun, "Parallax: Sparsity-aware data parallel training of deep neural networks," in ACM EuroSys, 2019, pp. 43:1–43:15.

[63] W. Shi, Y. Hou, S. Zhou, Z. Niu, Y. Zhang, and L. Geng, "Improving device-edge cooperative inference of deep learning via 2-step pruning," in IEEE INFOCOM, 2019, pp. 1–6.

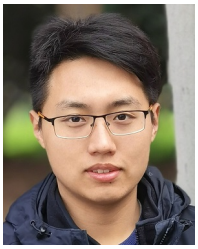
[64] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in IEEE ICPR, 2016, pp. 2464–2469.

[65] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in IEEE ICDCS, 2017, pp. 328–339.

Jing Wu received the M.S. degree in the School of Computer Science and Engineering, Northeastern University, Shenyang, China, in 2018. She is currently a Ph.D. student in the School of Computer Science and Technology, Huazhong University of Science and Technology, China. Her research interests include edge computing, augmented reality, and deep learning.

Lin Wang is an Assistant Professor at VU Amsterdam, The Netherlands and an Adjunct Professor at TU Darmstadt, Germany. He received his Ph.D. degree in Computer Science with distinction from the Institute of Computing Technology, Chinese Academy of Sciences. He has been a visiting researcher at IMDEA Networks Institute, Spain, from 2012-2014, a research associate at SnT Luxembourg from 2015-2016, and a group leader at TU Darmstadt, Germany from 2016-2018. His research interests broadly span distributed systems and networking with topics including edge AI inference, edge operating systems, and in-network computing. He received the Athene Young Investigator award from TU Darmstadt in 2018 and has been a PI of the Collaborative Research Center MAKI for future Internet in Germany.

Qiangyu Pei received his B.S. degree in physics from Huazhong University of Science and Technology, China, in 2019. He is currently a Ph.D. student in the School of Computer Science and Technology, Huazhong University of Science and Technology, China. His research interests include edge computing, green computing, and deep learning.



Xingqi Cui is currently an undergraduate student in the School of Computer Science and Technology, Huazhong University of Science and Technology, China. His research interests include edge computing and distributed computing systems.

Fangming Liu (S'08, M'11, SM'16) received the B.Eng. degree from the Tsinghua University, Beijing, and the Ph.D. degree from the Hong Kong University of Science and Technology, Hong Kong. He is currently a Full Professor with the Huazhong University of Science and Technology, Wuhan, China. His research interests include cloud computing and edge computing, datacenter and green computing, SDN/NFV/5G and applied ML/AI. He received the National Natural Science Fund (NSFC) for Excellent Young Scholars, and the National Program Special Support for Top-Notch Young Professionals. He is a recipient of the Best Paper Award of IEEE/ACM IWQoS 2019, ACM e-Energy 2018 and IEEE GLOBECOM 2011, the First Class Prize of Natural Science of Ministry of Education in China, as well as the Second Class Prize of National Natural Science Award in China.

Tingting Yang (M'13) received the B.Sc. and Ph.D. degrees from Dalian Maritime University, China, in 2004 and 2010, respectively. She is currently a Professor at Peng Cheng Laboratory. Her research interests are in the areas of Space-Air-Ground-Sea integrated networks, Network AI, and edge intelligence.