MDINFERENCE: Balancing Inference Accuracy and Latency for Mobile Applications

Samuel S. Ogden
Worcester Polytechnic Institute
ssogden@wpi.edu

Tian Guo
Worcester Polytechnic Institute
tian@wpi.edu

Abstract—**Deep Neural Networks are allowing mobile devices to incorporate a wide range of features into user applications. However, the computational complexity of these models makes it difficult to run them effectively on resource-constrained mobile devices. Prior work approached the problem of supporting deep learning in mobile applications by either decreasing model complexity or utilizing powerful cloud servers. These approaches each only focus on a single aspect of mobile inference and thus they often sacrifice overall performance.**

In this work we introduce a holistic approach to designing mobile deep inference frameworks. We first identify the key goals of *accuracy* and *latency* for mobile deep inference and the conditions that must be met to achieve them. We demonstrate our holistic approach through the design of a hypothetical framework called MDINFERENCE. This framework leverages two complementary techniques; a model selection algorithm that chooses from a set of cloud-based deep learning models to improve inference accuracy and an on-device request duplication mechanism to bound latency. Through empirically-driven simulations we show that MDINFERENCE improves aggregate accuracy over static approaches by over 40% without incurring SLA violations. Additionally, we show that with a target latency of 250ms, MDINFERENCE increased the aggregate accuracy in 99.74% cases on faster university networks and 96.84% cases on residential networks.

Keywords-**Mobile deep learning, performance**

I. INTRODUCTION

Deep learning on mobile devices is allowing for a wide range of new features such as virtual personal assistants [1], [2], visual text translation [3] and facial filters [4] to become commonplace in mobile applications. These diverse functionalities are made possible by recent advanced in machine learning models called *deep neural networks* (DNNs), which on some tasks can approach human-level accuracy [5].

However, DNNs achieve this high accuracy with high computational complexity [6] leading to high latency, especially when running on mobile devices [7]. This causes a necessary trade-off to be made between model accuracy and model execution latency. Modern frameworks such as TensorFlow allow for on-device execution, in-cloud execution, or some hybrid of these two, introducing a wide range of choices for this accuracy-latency trade-off.

Each of these three approaches each have strengths but introduce additional drawbacks. *On-device inference* allows for executing inferences entirely on the mobile device with easy to predict latency but the mobile developer has to choose between high execution latency or using lower accuracy models. *In-cloud inference* can execute high-accuracy models with low latency but the reliance on network communication means unpredictable, and potentially unacceptably long, overall response time [8]. *Hybrid inference* involves spreading execution between the mobile device and the cloud allowing for potential reductions in latency, but can result in worse latency and lower accuracy than purely on-device or in-cloud approaches.

In this paper we argue the need for mobile-oriented inference frameworks. We discuss the pros and cons of existing approaches and pinpoint the potential areas for improvement. We propose a holistic approach that considers mobile-specific factors when designing mobile inference frameworks. Finally, we demonstrate our approach through the design of a hypothetical framework called MDINFERENCE aiming to increase *aggregate accuracy*, defined as the average accuracy for all models used to service requests, while bounding latency for mobile inference requests. This is enabled by both utilizing a network-aware model selection algorithm to dynamically choose high-accuracy models that can execute within a target response time and duplicating requests to ensure a bounded latency response.

Instead of approaching the design of mobile inference frameworks as a static problem, where a single model is used and network time is disregarded, we consider a run-time approach to mobile inferences with a two-pronged design. **First,** by selecting the most accurate model for in-cloud inference based on the network delay we increase accuracy within an overall latency target. **Second,** by duplicating inference execution on-device using a low-latency model we can ensure that we can meet the latency target regardless of network connectivity and delay. In short, by dynamically selecting a model while running inference both in-cloud and on-device we improve accuracy while providing latency guarantees for mobile applications.

Our three main contributions are:
- We introduce a new mobile-oriented approach to designing deep inference frameworks that focuses on the specific goals and constraints of mobile devices, such
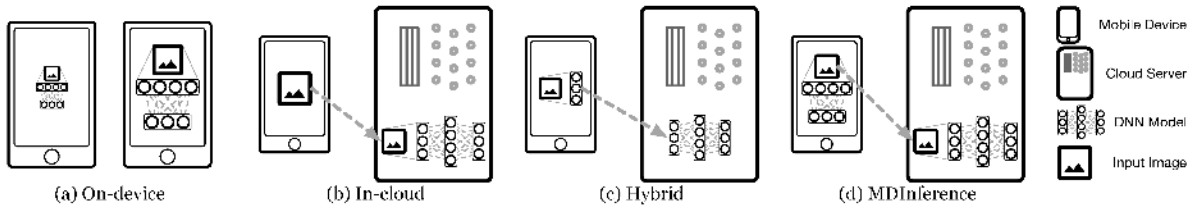
Fig. 1: Comparison of different mobile inference techniques. *(a) On-device inference allows models to run on-device in a resource constrained environment. Mobile-optimized models have lower latency but also lower accuracy. (b) Cloud-based inference allows for complex models but requires network transfer prior to inference execution, adding unpredictable network delay. (c) Hybrid inference shares execution between the mobile device and a cloud server, relying on both being available, to decrease latency. (d)* MDINFERENCE *uses runtime model selection and request duplication to select accurate cloud-based models and an on-device model to guarantee latency.*

as network condition variation. Making frameworks aware of these constraints will allow them to improve aggregate accuracy without sacrificing latency.

- We designed a hypothetical framework MDINFERENCE that demonstrates the ability of this mobile-oriented approach to improve the aggregate accuracy of inferences while meeting latency targets. Our evaluation shows MDINFERENCE achieved its target latency in 23% more cases than in-cloud approaches and increased aggregate accuracy over 39% compared to purely on-device approaches.

- We developed and integrated two algorithms to enable our MDINFERENCE to be mobile-aware. These algorithms opportunistically increase the aggregate accuracy of inferences and ensure that there are no SLA violations, improving user experience.

The remainder of this paper is structured as follows. In Section II we introduce a number of existing approaches mobile inference frameworks. The problem of mobile deep inference is formalized in Section III. Section IV discusses the key advantages of existing approaches and describes how we design a hypothetical framework for which we call MDINFERENCE. An evaluation of the techniques implemented in MDINFERENCE is presented in Section VI and a discussion of future directions is conducted in Section VII.

II. BACKGROUND AND MOTIVATION

Deep neural networks have become increasingly popular for embedding novel features into mobile applications. Two common forms of deep learning, convolutional neural networks (CNNs) and recurrent neural networks (RNNs) excel at image processing and speech-to-text, respectively. This allows for the addition of features such as Optical Character Recognition (OCR) [3] and virtual assistants [1], [2] to mobile applications.

State-of-the-art DNNs, with their accuracy-driven design, can contain tens of millions of parameters and hundreds of layers, and are therefore both computationally- and memory-intensive [6], [9]–[12]. To leverage these deep learning models, devices first need to preprocess the input data and load these models into memory. Once these models are
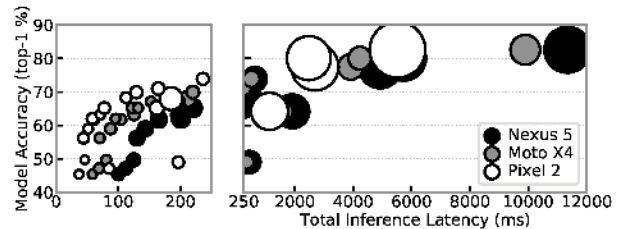

Fig. 2: DNN execution latency on a range of mobile devices. *We measure execution time for 21 pretrained models [7] via the TensorFlow Lite framework [14] running on three devices. The circle size corresponds to the standard deviation of the inference latency. We observe that high-accuracy models take multiple seconds to run on all devices and that newer devices, such as the Pixel 2, can support more models than older devices.*

loaded into memory, executing them requires large matrix multiplication operations with many millions, and often billions, of floating point operations [6], [10], [13]. Therefore, while these models can add rich functionality to mobile applications, actually leveraging them on mobile devices is difficult due to resource constraints [12].

Further, the number and extremity of otherwise common issues that mobile devices need to balance is extremely high. First, mobile devices experience a wide range of network conditions both in terms of connection quality and speed. They can be without a network connection for days or switch between high-speed WiFi and a cellular connection within the same minute. Second, they are inherently resource constrained as devices must be small and efficient enough to be carried by end-users throughout daily life. Finally, mobile applications are inevitably user-facing, compelling them to adhere to strict latency goals to improve user experience.

Three main approaches to enabling mobile deep learning are depicted in Figure 1 and discussed below.

A. On-device Inference

On-device inference refers to running DNNs on mobile devices, which is illustrated in Figure 1(a), and is supposed by frameworks such as Caffe2 [15] and TensorFlow Lite [14]. These frameworks often use models that have been

trained on powerful servers and exported to a format that is optimized for mobile devices. Mobile oriented optimizations to decrease the latency of on-device execution often aim to reduce the complexity of the models themselves [6], [13].

In Figure 2 we show the execution latency of 21 pretrained CNN models [7]. While many of the models that have been optimized for mobile devices completed execution in under 250ms, these models have lower accuracy results. Higher accuracy models often take much longer to run, even on devices with specialized hardware such as the Pixel 2 [16]. Further, we observe that the lower accuracy models show a distinct range of latencies, with latency increasing exponentially with accuracy, leading to the highest accuracy models having multi-second latency on all tested devices.

Further, even mobile-oriented models can still be orders of magnitude slower than running on dedicated servers with accelerators. The inference latency can be exacerbated when an application needs to load multiple models, such as chaining the execution of an OCR model and a text translation model [3], or requires higher accuracy.

In summary, even though on-device inference is a plausible choice for simple tasks and newer mobile devices, it is less suitable for complex tasks or older mobile devices.

B. In-cloud Inference

In-cloud inference, as illustrated in Figure 1(b), executes models on remote servers instead of on-device. Cloud-based servers, especially those with access to powerful accelerators such as GPUs, can execute models orders of magnitude faster than mobile devices. For example, execution of the *NasNet Large* model takes over 5 seconds on all of our mobile devices but takes only 113ms on a server with GPU (details in Table III). By leveraging this decrease in latency, in-cloud inference could decrease overall response time, even while using more accurate models. However, transferring the input data to the cloud-based servers can incur long and unpredictable network time [8], [17].

Model serving systems [18]–[21] allow mobile applications to leverage these cloud-based frameworks, often through REST APIs. However, many such systems require mobile developers to *manually* specify the exact model to use through the exposed API endpoints. These frameworks fail to consider the impact of dynamic mobile network conditions, which can take up a significant portion of end-to-end inference time [17], [22]. Moreover, such static development-time decisions can lead to using high-accuracy models whose high execution latency may be compounded by unexpectedly long network transfer time.

In summary, cloud-based inference has the potential to support many application scenarios, simple and complex, for heterogeneous mobile devices. However, current mobile-agnostic serving platforms fall short by not automatically adapting inference choices based on mobile constraints.

Symbol	Meaning
T_{sla}	Response time SLA
T_{budget}	Time allowed for model execution
T_{nw}	Estimated round-trip network time
M	A set of available models
$A(m)$	Accuracy of a model m
$\mu(m)$, $\sigma(m)$	Average and standard deviation of model execution time for model m

Tab. I: Symbols used throughout this paper.

C. Hybrid Inference

Hybrid inference spreads the execution of models across both the mobile device and a cloud-based server, as shown in Figure 1(c). By splitting the execution between two locations hybrid inference allows for decisions to be made at runtime to reduce overall response latency.

The division of model execution between the mobile device and the remote server is done by identifying partition points in models where intermediate data can be efficiently transferred from the mobile device to the remote server [23]. Executing the first layers of a model on-device and then the rest of the model on a remote server allows for transferring less data across the network. However, if the network is unavailable the entire model can be executed locally, but an unpredictable network can lead to an increase in latency.

To remove this reliance on the network, each segment of the model execution can calculate a confidence in its response [24], [25] where a high confidence will result in using the current response. If the confidence is too low on-device, the intermediate data can be sent for remote inference. This decreases the reliance on the network but potentially decreases accuracy.

In addition, since hybrid inference relies on continuing execution on the remote server this server has to host to same model as was used on the mobile device. In order to accommodate the possibility of no network connection this limits the models that can be used for hybrid execution.

In summary, hybrid inference allows for decreasing latency by partitioning the inference model and selecting where and whether each of the pieces should be executed. Network constraints may lead to longer latency and with a limited ability to improve accuracy by the models used.

III. PROBLEM STATEMENT

We target the problem of designing mobile deep inference frameworks for mobile applications. The core aspect of this problem is that the mobile device can have a variable, or nonexistent, network connection while request inferences. Additionally, an application developer has access to a set of models M that exhibit a range of different accuracies and latencies [7], [26] for the same task. Therefore, the problem is about how to enable high-accuracy inference results for mobile devices within a given target latency. Concretely, for a mobile device requesting an inference

within a target latency, T_{sla}, we want to select an inference model, $m \in M$ that maximizes accuracy and returns results within T_{sla}. Note, all symbols used throughout this paper can be referenced in Table I.

We consider two main metrics that measure the quality of solutions to this problem. *First* is *Service Level Agreement* (SLA) attainment, which is measured as the number of requests that return results within the specified response time target. The goal for a mobile-oriented framework is to return all results within a given SLA. *Second* is *aggregate accuracy*, which is the average accuracy of all models used by the framework. For example, if three inference requests are serviced by models with 40%, 60% and 60% accuracy, then the framework's aggregate accuracy is 53.3%. The goal of any framework is to maximize its aggregate accuracy.

System model and assumptions: We assume our mobile device is resource constrained and can only run a single on-device model. Further, we assume the mobile device *may* have access to an in-cloud server hosting a set of functionally-equivalent models, but transferring input data can take a variable network time T_{nw}. We call a set of distinct models *functionally-equivalent* if they all perform the same task, such as image classification. We further assume this network time can be calculated or estimated through a number of methods such as time synchronization, direct measurement, or network modeling [8].

Our hypothetical system is designed specifically for CNNs performing image classification tasks. We assume that any required preprocessing is completed on the mobile device and is not directly considered as part of the response time. We also assume that each request has an appropriate T_{sla}, representing the target request-response latency.

IV. Mobile Inference Framework Designs

Mobile-oriented inference frameworks have a number of unique goals and constraints that we discuss next. These represent a number of opportunities we discuss in Section IV-B.

A. Mobile-Aware Framework Design Goals and Challenges

As more mobile applications are leveraging DNNs it is becoming critical that inference frameworks be aware of the special demands of these applications. Existing approaches focus on optimizing for single goals, such as latency on mobile devices or inference server throughput, while ignoring mobile-specific needs. As an example, the *NasNet Mobile* model was designed to provide high-accuracy inference on mobile devices. On a Pixel 2 phone this model ran in 236ms but on other tested devices this model took up to 2.5X longer.

Goals for a mobile-specific inference framework: A mobile inference framework needs to *dynamically balance* two design goals: latency and accuracy. This need is driven by a dynamic mobile environments and network connections, and the inherent heterogeneity of devices.

	Goals		Factors (Awareness)		
	Accuracy	**Latency**	**Network**	**Resource**	**SLA**
On-Device	✗	✓	–	✓	✓
In-Cloud	✓	✗	✗	–	✓
Hybrid	✓/✗	✓/✗	✓	✓	–
MDINFERENCE	✓	✓	✓	✓	✓

Tab. II: Different mobile inference approaches and their goals and awareness. *The three approaches discussed each have different optimization goals. On-device inference relies on an awareness of available resources to optimize for inference latency. In-cloud inference has the goal to increase the throughput of inference servers for the most accurate models, showing an attention to SLA but ignoring the network. With hybrid approaches, the goals and awareness lie on a spectrum. Typically frameworks are aware of a subset of the various factors but no single approach is aware of all three.* MDINFERENCE *is aware of all three factors to achieve a reliable latency while increasing accuracy when possible.*

Latency is the time required to return an inference response to the mobile end-user. Keeping this metric low and consistent is important to mobile applications which are inherently user facing. Response times that are particularly long relative to the average will be obvious to users.

Accuracy is the ability a model to return the correct response on input data, which is often reported for image classifications models as the top-1 accuracy. This describes the model's average likelihood to correctly classify input images. In complex use cases accuracy is especially important.

Challenges for mobile-oriented frameworks: An ideal mobile inference framework should allow for both goals to be optimized by balancing them. To do this it would have to be aware of three major constraints, which we introduce below and have summarized in Table II.

First, mobile devices experience a wide range of network conditions that can lead to large variations in the latency of transferring input data for remote inference. Frameworks that performs remote inference should be aware of this variation and able to adapt its inference decisions to minimize the impact. *Second*, mobile devices are inherently resource constrained, making on-device inference difficult, which is exacerbated by device heterogeneity. A mobile-aware inference framework should reduce its reliance on on-device inference as these constraints are device-specific and may force each device to use a different low-accuracy model. *Finally,* mobile applications are user facing and thus are generally very sensitive to response time. Therefore any framework providing mobile devices with inference services should be able to provide results within a reasonable time, often defined by its latency SLA.

B. Inference Serving Opportunities

The existing approaches that mobile deep inference frameworks take introduce a number of potentially opportunities.

On-device inference aims to ensure that mobile users can always run inference but at a decreased accuracy. By decreasing the complexity of deep learning models it

is possible to run inference directly on the mobile device within a reasonable latency. This ensures that regardless of network connectivity mobile users can obtain inference results. One example of this is *MobileNets* [13] which by tuning the number of parameters within the model prior to training allows for a smooth trade-off curve between latency and accuracy based on the same model architecture.

The main drawback of on-device inference is that decreased latency is achieved by sacrificing inference accuracy. In the case of MobileNets, this can mean decreasing the top-1 accuracy by 29.6% (comparing the accuracy of the fastest and most accurate variations [7]). The problem of trading accuracy for latency is further compounded by the need to make such decisions prior to training. In particular, doing so at development time means an application either relies on a single model across all devices or needs to select the optimal model per device, which is challenging given the wide range of devices and models.

In-cloud inference allows for high-accuracy models to be run with low latency but neglects the needs of mobile applications. By leveraging hardware accelerators such as GPUs, cloud-based inference servers can greatly reduce the latency and improve the serving throughput even with complex high-accuracy models [18], [19], [27]. As an example, we observed that the time to execute the *NasNet Large* model (82.6% accuracy) in the cloud was faster than running inference requests with the *MobileNetV1_160 1.0* model (68.0% accuracy) on the fastest mobile device in our experiments. (For details see Figure 2 and Table III.) Cloud-based serving allows not only for high-accuracy inferences with low execution latency, but also opens up opportunities to serve inference requests with functionally-equivalent models that exhibit different latency-accuracy trade-offs.

The drawback of in-cloud inference frameworks is that they mobile-agnostic and are typically oriented towards service providers. This has two impacts. First, cloud-based servers aim to achieve a service level objective considering only on-server time and exclude the network latency of the input data [18], [28]. Due to this, poor mobile network connections can result in poor mobile performance [17]. Second, optimizations for throughput, such as batching, lead to an increase in the latency of individual requests [11], [18].

Hybrid inference spreads execution across multiple locations allowing for decreased latency but at the cost of relying on the availability of both locations. Spreading inference across multiple devices allows for a decrease in the amount of data transmitted across the network [23] or to exit early from execution when confidence in the intermediate result is above a threshold [25]. As a result, frameworks that support hybrid inference have the flexibility to selectively improve the inference performance by carefully spreading the model across different locations.

However, this requires both that intermediate data be transferred between locations and that the intermediate data can be used in both locations, leading to the same model be executed in both locations. In the case that network transfer of intermediate data is prohibitive the model must be executed entirely on-device. For complex models this leads to unacceptable latency, and simple models fail to benefit from the remote execution. Therefore, hybrid frameworks have similar limitations to on-device frameworks, in that the model used must be selected during development, and in-cloud frameworks with their sensitivity to the network.

V. MDInference Framework Design

The key insight of MDInference is that we can leverage a set of cloud-based functionally-equivalent models to improve accuracy. In addition, duplication of inference requests [29], [30] allows us to bound latency. For each inference MDInference submits an inference to a remote server that dynamically selects an accurate model, and at the same time executes a low model to ensure results will be available for uses within the SLA. This allows for increased accuracy and reliable latency.

MDInference combines the advantages of existing approaches in order to improve end-user performance. By dynamically selecting cloud-based models based on network information we can opportunistically use higher accuracy models and improve the aggregate accuracy. Additionally, MDInference and further improve the aggregate accuracy by using a more accurate on-device model, although this can impact the minimum achievable SLA. This combination of local and remote inferences allows MDInference to provide for reliable latency and improved aggregate accuracy.

MDInference consists of two components. First, a cloud-based server selects between a number of functionally-equivalent models for one that can complete within a specific SLA by estimating the time consumed for transferring input data. This algorithm is detailed in Section V-A. Second, a local inference is run on-device to ensure that results are available within the target SLA. The combination of these two components ensures that inference output is available within the SLA, possibly with improved aggregate accuracy from the cloud-based component. We discuss the implication of duplicating inference requests in Section V-B.

A. Model Selection Algorithm

MDInference's model selection algorithm is designed to manage a set of functionally-equivalent CNN models and pick the most accurate model that can return results within the specified SLA. It is designed to take advantage of the low variability of model execution latency to not only mitigate the impact of variations in the mobile network latency but opportunistically use them to improve accuracy. The key insight of our model selection algorithm is that the variations of transfer latency for an inference request can be compensated for with the appropriate choice of inference model. As functionally-equivalent models each

have different execution times and accuracies, by explicitly making inference latency and model accuracy trade-offs MDINFERENCE determines which CNN model to execute.

MDINFERENCE works by selecting the most accurate model that has a low enough execution time to return results to the end-user within the SLA. It accomplishes this by first calculating the request's time budget as the difference between SLA and the estimated network time. That is, $T_{budget} = T_{sla} - T_{nw}$ where T_{nw}, referred to as *network time*, denotes the time to transfer the inference request and to return the result. Consequently, T_{nw} can be estimated conservatively as $T_{nw} = 2 \times T_{input}$ where T_{input} is the time to transfer the data to the remote server. Estimating T_{nw} using T_{input} is straightforward as T_{input} can be measured by the server prior to inference execution. Further, such estimation is reasonable for application scenarios such as image recognition or image-to-text translations. These applications often need to send more data to the cloud (i.e., input data) which leads to $T_{input} \geq T_{output}$, the time to return results. For other application scenarios such as speech recognition where output data size is often larger, one could leverage past observations of T_{output} and estimate $T_{nw} = 2 \times T_{output}$ instead. Using this time budget we can then identify the set of models, M_E, that can complete execution within the request time budget T_{budget}.

The basic approach described above assumes that the execution times and accuracies of models previously measured stays the same. However, these two assumptions do not always hold, leading to a need to expand on the basic concept of model selection to *probabilistically* select models. Real-world serving systems [18], [19] often experience queuing delay or workload spikes [31] leading to transient increases in latency. Additionally, accuracy is affected over time by concept drift [32]. To handle these changes in latency and accuracy the model selection algorithm probabilistically selects models, thus exploring available models that might have been previously disregarded due to transient issues. We do this by selecting a model using a weighted probability based on the model's latency relative to T_{budget} and accuracy. We implement this probabilistic approach via a three stage algorithm described below.

Stage one: greedily picking the baseline model. In this stage, MDINFERENCE takes all the existing models and selects a base model m_j as follows.

$$\underset{j}{\text{maximize}} \quad \boldsymbol{A}(m) \tag{1}$$

$$\text{subject to} \quad \mu(m) + \sigma(m) < T_{budget}, \ m \in \boldsymbol{M} \tag{2}$$

To find the base model we first consider all models that have an expected inference time less than the time budget and use the most accurate of these models. This is to make it likely that the model will finish within the calculated budget. We use this model m_b as our base model. In the case that no models satisfy the time budget constraint the fastest model

available is chosen as the base model and execution begins.

Stage two: optimistically constructing the eligible model set. In order to account for unexpected performance variations, such as queueing delays or accuracy variations, the probabilistic model selection algorithm will expand around the base model to form an exploration set, M_E. This exploration set represents models that are similar to the base model in terms of execution time. Specifically, we construct the exploration set as

$$M_E = \{m \mid \mu(m) \in [\mu(m_b) - \sigma(m_b), \mu(m_b) + \sigma(m_b)]\} \tag{3}$$

which is the set of all models that have an average execution time within the typical execution time of the base model. It is important to note that M_E may include models that violate the latency variation constraints imposed on the base model. This is accounted for in stage three.

Stage three: opportunistically selecting the inference model. From the exploration set M_E we select a model m' to balance the risk of SLA violations and the exploration reward. Concretely, we calculate the utility for each model, $\boldsymbol{U}(m)$, based on its inference accuracy and its likelihood to violate response time SLA as:

$$\boldsymbol{U}(m) = \boldsymbol{A}(m) \frac{T_{budget} - \big(\mu(m) + \sigma(m)\big)}{|T_{budget} - \mu(m)|}. \tag{4}$$

MDINFERENCE than normalizes these utilities to calculate the selection probability as $\boldsymbol{Pr}(m) = \frac{\boldsymbol{U}(m)}{\sum\limits_{n \in \boldsymbol{M_E}} \boldsymbol{U}(n)}$ and picks m' based on its probability. This helps avoid choosing models with lower inference accuracy, wider inference time distribution, and outdated performance profile while still exploring the set of potentially eligible models.

B. Request Duplication

To ensure that all requests can be serviced within the SLA, MDINFERENCE duplicates requests to bound their tail response latency. As discussed in Section II-A, many mobile-oriented models can produce results on-device within a reasonable time limit, but with lower accuracy.

When an inference is initiated two requests are generated by the MDINFERENCE framework. The first is sent to a remote inference server that executes the model selection algorithm outlined previously. While this cloud request aims to return results within the SLA it is not guaranteed. Therefore, an inference request is duplicated and executed locally using the on-device model. In MDINFERENCE we chose the fastest available model to use on-device, supporting for SLAs as low as 50ms, although any model that satisfies the SLA goal can be used.

There are two potential outcomes to duplication. First, the SLA expires without the remote inference request having returned results, in which case MDINFERENCE uses the results of the on-device model. In our experiments this occurred in

Model Name	Top-1 Accuracy (%)	Inference Avg. (ms)	Inference Std. (ms)
SqueezeNet	49.0	4.91	0.06
MobileNetV1 0.25	49.7	3.21	0.08
MobileNetV1 0.5	63.2	4.21	0.06
DenseNet	64.2	25.49	0.14
MobileNetV1 0.75	68.3	4.67	0.07
MobileNetV1 1.0	71.0	5.43	0.11
NasNet Mobile	73.9	21.18	0.17
InceptionResNetV2	77.5	50.85	0.33
InceptionV3	77.9	31.11	0.19
InceptionV4	80.1	59.21	0.22
NasNet Large	82.6	112.61	0.36
NasNet Fictional*	50	112.61	0.36

Tab. III: Summaries of model statistics through empirical measurement. *Models are sorted based on their reported top-1 accuracy which is defined as the percentage of correctly labeled test images using only the most probable label. We measure the average inference time and standard deviation for each model running via TensorFlow on an AWS p2.xlarge GPU server. We used these models in simulations to study* MDINFERENCE*'s effectiveness in trading-off aggregate accuracy and latency. Note,* NasNet Fictional *is a copy of* NasNet *with lower accuracy used in Section VI-C.*

only 3.16% of cases, as we will see in Section VI-D. The second outcome has the remote inference response arrives before the SLA expires and the remote results are used.

VI. EVALUATION

Our evaluation goal is to quantify the effectiveness of MDINFERENCE in opportunistically improving aggregate accuracy for mobile devices. We do this by first demonstrating the effectiveness of our model selection algorithm at increasing the aggregate accuracy at a range of SLAs when compared to a number of alternative algorithms. Finally, we demonstrate the benefit of request duplication by analyzing its impact on SLA attainment and aggregate accuracy.

Key metrics. The first metric that we are measuring in many of our evaluations is aggregate accuracy, which we introduced in Section III. We additionally measure the SLA attainment, which is the percent of requests that return results to the user within a target SLA. With duplication this is no longer an issue thanks to leveraging low-latency on-device models. In these cases we measure the percentage of requests that rely on the on-device model and the aggregate accuracy improvements when leveraging in-cloud models.

Simulation methodology. In our simulations, we leverage a range of models, summarized in Table III [7], [9], [10], [13], [33]–[35], that expose different accuracy and inference time trade-offs. We empirically measured the inference time distributions of models using an EC2 *p2.xlarge* GPU-accelerated server over 1,000 inference executions. The accuracy of our functionally-equivalent pretrained model was reported against the ILSVRC 2012 dataset [7], [36], a widely used image classification test set. The mobile network we use as the basis for many of our simulations assumes that transferring an input image takes 100ms±50ms, based on real world measurements of our university network. For each simulation, we generate 10,000 inference requests with a predefined SLA target and recorded the model selected by MDINFERENCE (and baseline algorithms) and relevant
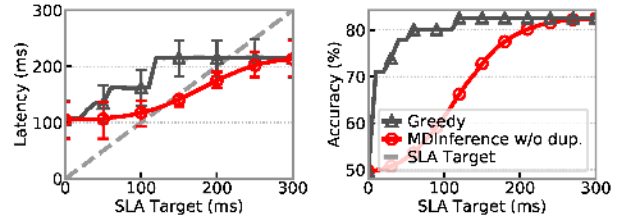
(a) *Comparison of average end-to-end latency and accuracy.* MD-INFERENCE*'s selection algorithm is able to track the SLA target when SLA* ≥*100ms while the greedy approach fails to do so.* MDINFERENCE *improved aggregate accuracy safely as the SLA target increases. Further, our model selection algorithm improves standard deviation of inference times as well keeping them below the SLA target.*
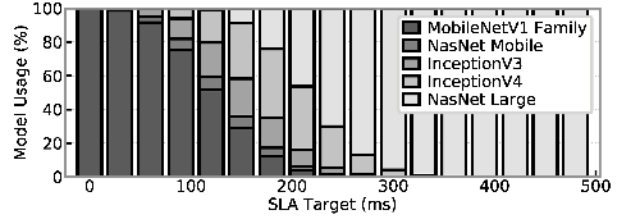
(b) *Models selected by* MDINFERENCE*. The use of a diverse set of models allows* MDINFERENCE *to minimize SLA violations while providing highly accurate inference results.*

Fig. 3: Comparison of MDINFERENCE to the *static greedy* **algorithm.** *For each SLA target, we simulated 10,000 inference requests and recorded the inference time incurred by both the greedy algorithm and* MDINFERENCE*.*

performance metrics. We repeated each simulation for a variety of SLA targets and network profiles combinations. For all tests except those in Section VI-D we evaluated the model selection capabilities of MDINFERENCE without duplication of requests.

A. Benefits over static greedy model selection

Figure 3(a) shows the average end-to-end inference time (left) and aggregate accuracy (right) achieved by our model selection algorithm and a *static greedy* algorithm, which picks the most accurate model that can complete within the given SLA. This figure shows that MDINFERENCE consistently achieved up to 42% lower inference latency, compared to *static greedy*. Moreover, MDINFERENCE can operate under a much more stringent SLA target (∼115ms) while *static greedy* continues to frequently incur SLA violations until SLA target is more than 200ms. The key reason is that MDINFERENCE was able to effectively trade off aggregate accuracy and response time by choosing from a diverse set of functionally-equivalent models. Consequently, MDINFERENCE had an aggregate accuracy of 68% (on par to using *MobileNetV1 0.75* which can take 2.9X more time running on mobile devices) under low SLA target (∼115ms), but was able to match the aggregate accuracy achieved by *static greedy* for higher SLA target. Note that *static greedy* achieved up to 12% higher accuracy by sacrificing inference latency.
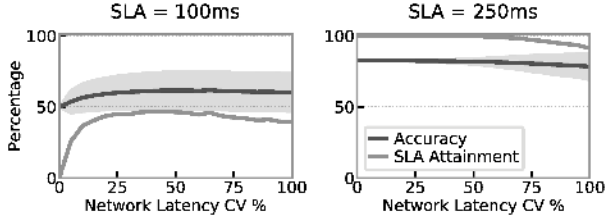
Fig. 4: **Aggregate accuracy of MDINFERENCE at different levels of CV with** T_{nw} = **100ms.** *The initial low level of SLA attainment is due to the initial network time of 100ms, leaving no time for inference execution. As the variability of the network increases MDINFERENCE can take advantage of the range of models available to it to quickly improve accuracy and SLA attainment. Similarly, at a higher SLA, MDINFERENCE can achieve high accuracy until the network variability exceeds the SLA.*

Figure 3(b) illustrates the CNN model usage patterns (i.e., percentage of model being used for executing the inference requests) under different SLA targets. At very low SLA target (less than 30ms), MDINFERENCE chooses the fastest model, *MobileNetV1 0.25*, as described in Section V-A. As the SLA target increases, MDINFERENCE aggressively explores more accurate, but slower models, commonly using our most accurate model, *NasNet Large*.

We make two observations. First, MDINFERENCE was effective in picking the more appropriate model to increase accuracy while staying safely within SLA target. For example, *InceptionResNetV2* was never selected by MD-INFERENCE because better alternatives such as *InceptionV3* and *InceptionV4* exist. Second, MDINFERENCE faithfully explored eligible models and was able to converge to the most accurate model when SLA target allows, as shown in Figure 3(a) at T_{sla} = 250ms.

In summary, MDINFERENCE outperformed *static greedy* with an end-to-end latency reduction of up to 43%, while matching its accuracy when the SLA budget is larger than 250ms. This is possible because MDINFERENCE adapted its model selection by considering both the SLA target and network transfer time, while *static greedy* naively selected the most accurate model.

B. Adaptiveness to dynamic mobile network conditions

One of the key goals of MDINFERENCE is to adapt to network variations in order to improve mobile user experience. To further examine how MDINFERENCE copes with these variations we simulated network profiles with increasing variability. Specifically, we fixed the average network latency to be 100ms, and varied the Coefficient of Variation (CV) from 0% to 100%, where CV is defined as the ratio between the standard deviation and average of the network time. CV ranged from 0% to 100%, to represent a perfectly stable network and a network which has a standard deviation equals its average, respectively. As a point of reference, our measured university WiFi network has a CV of 74%.
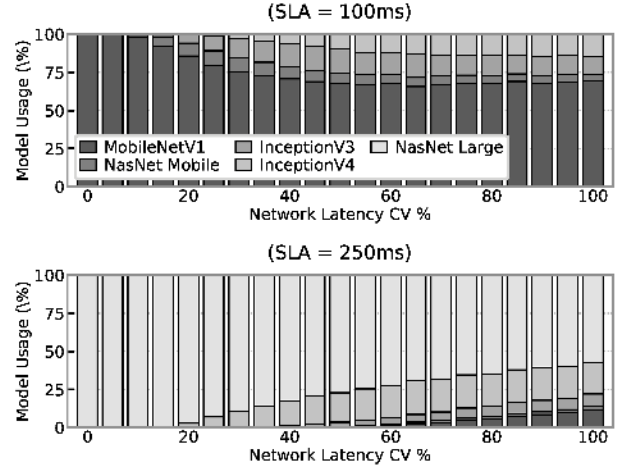

Fig. 5: **Model usage vs. network latency (CV) shown at two different SLA targets.** *When there is a reliable network (i.e. low CV) we see that single models are used for all requests since the high reliable network has leaves no room for other model choices. As the network becomes increasingly volatile a wider range of models is used to meet the SLA.*

Figure 4 shows the aggregate accuracy and SLA attainment achieved by MDINFERENCE. For low SLA target (100ms), when the network is relatively stable MD-INFERENCE had an SLA attainment of less than 50%. As the network condition became more variable, MDINFERENCE was able to increase the aggregate accuracy gradually while maintaining the SLA attainment. The low SLA attainment is due to on average half the requests needing the entire SLA just for network transfer. Conversely, the slight increase in accuracy is due to MDINFERENCE taking advantage of the network variation to use more accurate models.

It is important to note that when the network transfer took the entirely SLA, MDINFERENCE performed as expected by choosing *MobileNetV1 0.25*, the fastest available model. We note that, to satisfy such stringent SLA targets with high network latency variation, approaches such as provisioning inference servers at network edge.

When we consider an SLA target of 250ms we see a different outcome. This SLA is slightly more than the sum of the average network latency and the time to execute the most accurate model, *NasNet Large*. In this case we see that MDINFERENCE used high accuracy models, maintaining an accuracy of around 80% throughout the entire range of network variations.

Figure 5 shows the models chosen when varying CV of network time for SLAs of 100ms and 250ms. The SLA of 100ms is the RTT of the simulated network leaving no time left for inference. Similarly, when the SLA is 250ms then the target time is larger than the sum of the RTT of the network and the execution latency of *NasNet Large*, our most complex model, which MDINFERENCE leverages.

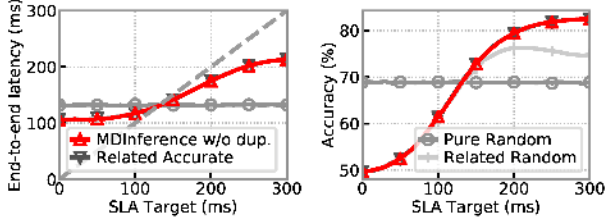We make the following two observations. First as the net-

Fig. 6: Decomposition of benefits of MDINFERENCE's three-stage algorithm. MDINFERENCE *achieves similar accuracy and SLA attainment compared to* related accurate, *indicating the effectiveness of our probabilistic approach. Both* pure random *and* related random *have poor aggregate accuracy due to their inability to distinguish models with different latency.*

	University		Residential	
	On-device Reliance	Aggregate Accuracy	On-device Reliance	Aggregate Accuracy
Static Latency	0.26%	41.40%	3.16%	41.40%
Static Accuracy	3.67%	81.09%	23.03%	73.11%
Random	0.42%	63.33%	5.06%	62.06%
MDINFERENCE	0.26%	82.39%	3.16%	80.43%

Tab. IV: Aggregate accuracy and on-device model reliance. MDINFERENCE *achieved the highest aggregate accuracy with lower on-device reliance, compared to other algorithms.*
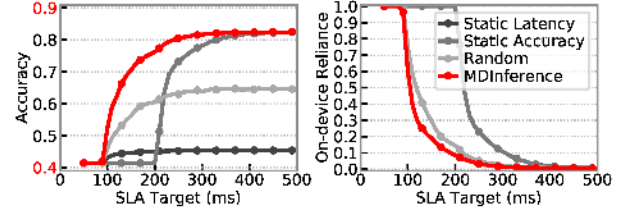

Fig. 7: Aggregate accuracy and on-device model reliance on residential network. MDINFERENCE *demonstrated improvements throughout all tested SLAs. At lower SLAs* MDINFERENCE *can quickly improve the aggregate accuracy. Meanwhile,* MDINFERENCE *also reduces the reliance on on-device models at low SLAs, much more quickly than other algorithms.*

work became more variable (i.e. high CV), MDINFERENCE matched the network variability by using a subset of faster models. As the network becomes more variable MDINFERENCE can exploit this to in some cases opportunistically use models with high inference accuracy. Second, the probability of exploring different eligible models is proportional to the SLA target and network variability. Faster models, such as those in the *MobileNetV1* family, are used as a basis for low SLA target while the most accurate model, *NasNet Large*, is used for higher SLA targets.

In summary, MDINFERENCE was effective in handling highly variable mobile network by exploring a diverse set of deep learning models that expose different inference latency and accuracy trade-offs. By taking advantage of network variation it could improve accuracy in many cases and even in cases with low target response times could often return responses within the SLA.

C. Decomposing the efficiency of MDINFERENCE

Next, we breakdown the performance benefits provided by MDINFERENCE by examining the stages of our probabilistic model selection algorithm (Section V-A). For each of the three stages we compare to an alternative algorithm. For stage one we compare to *random* model selection. For stage two we compare to *related random* that randomly selects a model from the exploration set M_E. For stage three we compare to *related accurate*, which selects the most accurate model from M_E to demonstrate that our probabilistic approach does not sacrifice accuracy. All algorithms were tested with a network latency with average $100ms$ and standard deviation of $50ms$.

Figure 6 shows the average inference latency and aggregate accuracy for all four model selection algorithms. All three algorithms, including MDINFERENCE, that choose from the exploration set M_E were able to meet reasonable SLA target while *pure random* had approximately the same latency across all SLAs, incurring a large number of SLA violations. This indicates that the construction of M_E, by stage one and stage two both enabled exploration and minimized risk of unnecessary SLA violations.

As the SLA target increases, *pure random* again achieved approximately the same aggregate accuracy across all SLAs. All three other algorithms were able to gradually increase the aggregate accuracy by using slower but more accurate models from Table III. However, once we have a large enough SLA target ($\sim$150ms), the exploration set M_E converges to two models: *NasNet Large* and *NasNet Fictional*. At this point, *related random* algorithm started to degrade aggregate accuracy since it cannot differentiate between these two models. Meanwhile, both *related accurate* and MDINFERENCE were able to steadily improve aggregate accuracy by avoiding *NasNet Fictional*.

It is important to note there is only a negligible difference in aggregate accuracy using MDINFERENCE when compared to *related accurate* algorithm. This small difference is due to *related accurate* always selecting the most accurate model from M_E while MDINFERENCE explores the eligible set. The probability of picking a less accurate model is low enough as to not overly impact the aggregate accuracy. The probabilistic behavior of MDINFERENCE is meant to allow for this exploration even while generally maintaining aggregate accuracy, as opposed to *related accurate*, which misses the opportunity to use models which may have improved accuracy or latency profiles.

In summary, MDINFERENCE's three-stage algorithm is effective in distinguishing and identifying the most appropriate model to use under dynamic conditions. All three stages contribute to and help MDINFERENCE cope with the variable network conditions and improve aggregate accuracy.
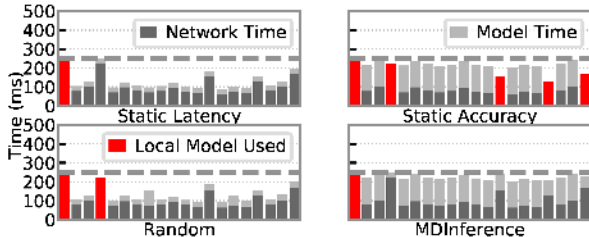
Fig. 8: Inference time for 20 randomly sampled requests over residential network. *In many cases* MDINFERENCE *chose a model that yields results within the time budget. Other approaches, such as Static Accuracy, returned high-accuracy results but rely more heavily on the on-device model due to network variation.*

D. Testing the on-device reliance Duplication

To test the on-device reliance of MDINFERENCE we used the network time from sample of 5000 requests on each of our university network and on a residential network. These requests consisted of a preprocessed image input that averaged 51.9KB with standard deviation of 53.6KB. The model chosen for the on-device was the fastest *MobileNetV1_128 0.25* model as it represents the single model most likely to complete within any SLA for all tested mobile devices. It was also excluded from the set of models available in the cloud to better demonstrate the ability of MDINFERENCE to improve over on-device inference.

For each of these measured requests we compared MD-INFERENCE to three other simulated model selection algorithms using the models detailed in Table III and an SLA target of 250ms. The three other algorithms we used were *static latency*, which picks the fastest model, *static accuracy*, which picks the most accurate model, and *random* which picks a random model.

Table IV compares the aggregate accuracy and on-device reliance for all four model selection algorithms. MD-INFERENCE achieved the highest aggregate accuracy for inference requests sent over both university and residential WiFi, improving over static accuracy by 7.32% on the variable residential network. Meanwhile, it improved aggregate accuracy compared to static accuracy by up to 19%.

The aggregate accuracy and on-device reliance is shown in Figure 7. We can observe that MDINFERENCE increases aggregate accuracy more quickly than the other algorithms tested and has a lower reliance on the on-device models, allowing it to maintain this higher aggregate accuracy.

Figure 8 shows the inference latency breakdown for 20 randomly sampled requests that were sent over the residential network. We observe that mosts requests were able to complete on the remote server but in some cases the on-device model must be used, in which we highlight the network latency in red. This shows the ability of MDINFERENCE to adapt its execution choice to match compensate for network variability, allowing it to decrease its on-device reliance and boosting is accuracy.

In summary, the duplication mechanism allows MD-INFERENCE to ensure that results are returned to the mobile user within the target response time. Combining this with the model selection algorithm, MDINFERENCE is able to increase the aggregate accuracy for inference requests in the vast majority of cases.

VII. DISCUSSION & FUTURE WORK

There are a number of potential avenues for future work in mobile-oriented deep inference frameworks. We discuss a number of important factors that should be considered, such as energy consumption and aggregate accuracy.

Energy Consumption. The duplication of inference requests solves the issues of SLA violations, allowing users to have reasonable performance. However, this requires energy consumption on the device for both network communication and inference. Therefore, identifying times when duplication is critical and avoiding unnecessary duplications could allow for reduced energy consumption.

On-device Model Selection. Currently, our proposed MDINFERENCE framework uses the same on-device model regardless of the mobile devices. There are a number of different approaches, discussed in Section II-A for improving on-device inference but generally rely on statically selected models. While some model optimizations can provide this ability to simplify models post-training [37], these provide only limited options.

Spanning Subsets of Models. Figure 3(b) demonstrated that many requests can be serviced by only a small subset of models. This potentially indicates that there exists a subset of models that could service nearly all requests, and thus form a *spanning subset* for all the models. This would be highly beneficial for decreasing the cost of inference serving, as only the models that fall into this subset would need to be available. Further, finding this subset for an arbitrary set of models without resorting to empirical measurement is another challenging problem to investigate.

VIII. CONCLUSION

In this work we introduced a holistic approach to designing mobile-oriented deep inference frameworks that focuses on identifying user needs and the constraints of mobile devices. We introduced the design of MDINFERENCE, a hypothetical framework utilizing this approach. MDINFERENCE can improve aggregate accuracy in over 96% of cases without introducing additional SLA violations. This improvement in accuracy was over 40% in some cases and was a 7.32% increase over statically serving the highest-latency model while duplicating inference execution locally. Our work shows the potential to better mobile inference serving by explicitly addressing mobile-oriented constraints.

REFERENCES

[1] T. Capes, P. Coles, A. Conkie, L. Golipour, A. Hadjitarkhani, Q. Hu, N. Huddleston, M. Hunt, J. Li, M. Neeracher *et al.*, "Siri on-device deep learning-guided unit selection text-to-speech system." in *INTERSPEECH*, 2017, pp. 4011–4015.

[2] V. Kepuska and G. Bohouta, "Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home)," in *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2018, pp. 99–103.

[3] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado, M. Hughes, and J. Dean, "Google's multilingual neural machine translation system: Enabling zero-shot translation," 2016.

[4] Z. Li, X. Wang, X. Lv, and T. Yang, "Sep-nets: Small and effective pattern networks," *CoRR*, vol. abs/1706.03912, 2017. [Online]. Available: http://arxiv.org/abs/1706.03912

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034.

[6] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[7] "Hosted models — tensorflow lite," Google Brain Team, [accessed Janurary 2020]. [Online]. Available: https://www.tensorflow.org/lite/guide/hosted_models

[8] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate record-and-replay for HTTP," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*. Santa Clara, CA: USENIX Association, Jul. 2015, pp. 417–429. [Online]. Available: https://www.usenix.org/conference/atc15/technical-session/presentation/netravali

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition."

[10] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[11] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark analysis of representative deep neural network architectures."

[12] T. Guo, "Cloud-based or on-device: An empirical study of mobile deep inference," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 184–190.

[13] 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."

[14] "Tensorflow lite," Google Inc., [accessed October 2019]. [Online]. Available: https://www.tensorflow.org/lite

[15] Facebook Research, "Caffe2," [accessed February 2019]. [Online]. Available: https://caffe2.ai

[16] "Pixel 2 - Wikipedia," https://en.wikipedia.org/wiki/Pixel_2, 2019.

[17] S. S. Ogden and T. Guo, "MODI: Mobile deep inference made efficient by edge computing," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. Boston, MA: USENIX Association, Jul. 2018. [Online]. Available: https://www.usenix.org/conference/hotedge18/presentation/ogden

[18] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 613–627.

[19] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, "Tensorflow-serving: Flexible, high-performance ml serving."

[20] P. Gao, L. Yu, Y. Wu, and J. Li, "Low latency rnn inference with cellular batching," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: ACM, 2018.

[21] M. LeMay, S. Li, and T. Guo, "Perseus: Characterizing performance and cost of multi-tenant serving for cnn models," in *2020 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2020.

[22] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, Oct. 2009.

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

[24] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2016, pp. 2464–2469.

[25] ——, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 328–339.

[26] "Caffe — Model Zoo," http://caffe.berkeleyvision.org/model_zoo.html.

[27] C. Zhang, M. Yu, W. Wang, and F. Yan, "Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving," in *2019 {USENIX} Annual Technical Conference ({USENIX}{ATC} 19)*, 2019.

[28] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, J. Mars, and L. Tang, "Grandslam: Guaranteeing slas for jobs in microservices execution frameworks," in *Proceedings of the Fourteenth EuroSys Conference 2019*. ACM, 2019, p. 34.

[29] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: distributed, low latency scheduling," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 69–84.

[30] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 10, pp. 1094–1104, 2001.

[31] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 241–252.

[32] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine learning*, vol. 23, no. 1, pp. 69–101, 1996.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[34] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size."

[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition."

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

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