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 (DNNs) 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 efficiently on resource-constrained mobile devices. Prior work has begun to approach the problem of supporting deep learning in mobile applications by either decreasing execution latency or utilizing powerful cloud servers. These approaches only focus on single aspects of mobile inference and thus they often sacrifice other aspects.**

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 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 40% without incurring SLA violations. Additionally, we show that with SLA = 250ms, MDInference can increase the aggregate accuracy in 99.74% of cases on faster university networks and 96.84% of cases on residential networks.

***Index Terms*—Mobile deep learning, Performance**

I. Introduction

Deep learning on mobile devices is allowing for a wide range of new features in mobile applications. Features as diverse as virtual personal assistants [1], [2], visual text translation [3] and facial filters [4] are now commonplace on mobile devices. These diverse functionalities are made possible by using deep neural networks (DNNs), which can even achieve better accuracy than humans on some tasks [5].

These models, however, are slow to run on mobile devices due to high computational complexity [6] leading to high latency for high accuracy models. To minimize this latency, existing deep inference frameworks allow for on-device, in-cloud, and hybrid inferences, each of which make different trade-offs between accuracy and latency. These approaches each have strengths but introduce additional drawbacks. On-device inference allows for executing inferences entirely on-device with predictable latency but has to make the choice between long execution or lower accuracy models. In-cloud inference can execute high-accuracy models with low latency but the reliance on network communication means overall response time can be unpredictable and unacceptably long [7]. Hybrid inference spreads execution between the mobile device and the cloud to potentially reduce response time, but poor network connections can curtail this advantage.

In this paper we argue the need for mobile-oriented inference frameworks. Additionally, we discuss the pros and cons of existing approaches and pinpoint the potential areas for improvement. We propose a holistic approach that considers the specific factors that impact mobile inference frameworks. Finally, we demonstrate our approach through designing a hypothetical framework called MDInference, which increases aggregate accuracy for mobile inference requests while bounding latency. This is enabled by utilizing a network-variation-aware model selection algorithm to select high-accuracy models while 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, we consider a run-time approach to mobile inferences with two main aspects. First, by selecting a model for in-cloud inference based on the network delay execution can match an overall latency target. Second, by duplicating this request on-device using a low-latency model we can ensure that we can meet the latency target regardless of network connectivity. By running inference both in-cloud and on-device we can 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 learning frameworks that focuses on the specific goals and constraints of mobile devices. By being aware of the impact of these constraints, such as unpredictable network latency variation, frameworks can improve aggregate accuracy without sacrificing latency.
- We design a hypothetical framework, MDInference, that demonstrates the ability of our mobile-oriented approach to improve the accuracy of inferences while meeting latency targets. Our evaluation shows MDInference can improve SLA attainment by up to 23% over in-cloud approaches and increase aggregate accuracy over 39% compared to purely on-device approaches, with no SLA violations.
- We develop and integrate two algorithms to enable our hypothetical framework to both increase accuracy of inferences and prevent SLA violations. These algorithms ensure that there are no SLA violations adversely impact-

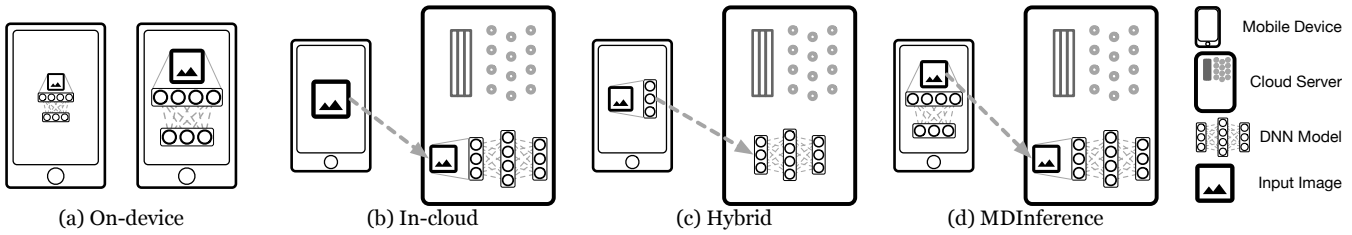


Fig. 1: Comparison of different mobile inference techniques. On-device inference (a) allows different sized models to run on-device with different latencies and accuracies. Generally, low latency models result in lower accuracy, which can be seen in Table III. Cloud-based (b) allows for more complex models but requires network transfer prior to inference execution, adding unpredictable network latency. Hybrid inference (c) spreads inference between the mobile device and a cloud server, relying on both being available, to decrease latency. MDInference (d) uses runtime model selection and duplication of requests to select cloud-based high accuracy models while running a back-up on a low-latency on-device model.

ing user experience and that inference requests have the highest possible accuracy.

The remainder of this paper is structured as followed. 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 these approaches and describes how we leverage similar tactics to design a hypothetical framework for mobile inferences which we call MDInference. An evaluation of the techniques implemented in MDInference is presented in Section V and a discussion of future directions is conducted in Section VI.

II. Background and Motivation

Deep neural networks (DNNs) 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 advanced features such as Optical Character Recognition (OCR) [3] and virtual assistants [1], [2].

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], [8]–[11]. To leverage these deep learning models devices first need to preprocess the input data and load these models into memory. Once these models are loaded into memory they require many millions, and often billions, of floating point operations [6], [9], [12] in order to perform large matrix multiplication operations. While these models can add rich functionality to mobile applications, due to the resources needed, actually leveraging them on mobile devices is difficult [11].

Further, the number and extremity of otherwise common issues that mobile devices need to balance is unique. First, they experience a wide range of network conditions both in terms of network 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 they must be small and efficient enough to be carried by end-users. Finally, mobile applications are inevitably user-facing, compelling them to target strict SLAs to improve user performance.

We next describe a number of approaches which are depicted in Figure 1 and described in Section II-A to II-C.

A. On-device Inference

On-device inference is when a DNN model is run directly on the mobile device, which is illustrated in Figure 1a. A number of frameworks, such as Caffe2 [13] and TensorFlow Lite [14], now support this. These frameworks use models that have been trained on powerful servers and exported to a format that is optimized for mobile devices. There has also been work towards reducing the complexity of models themselves to decrease their latency [6], [12].

In Figure 2 we show the execution latency of 21 pretrained CNN models [15]. While many of the mobile-optimized models complete execution in under 250ms, these are lower accuracy models. Higher accuracy models often take much longer to run, even on devices with specialized hardware such as the Pixel 2 [16]. We observe that the lower accuracy models do in fact show a distinct range of latencies, with increasingly large increases in latency for each step increase in accuracy, which is common in DNNs. However, on all of the test devices many of the high accuracy models have multi-second latency.

Even mobile oriented models can still be orders of magnitude slower than running on dedicated servers with accelerators. The inference latency is 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

Instead of running models on-device they can instead be run in the cloud, as illustrated in Figure 1b. Cloud-based servers, especially those with access to powerful accelerators, can execute models with orders of magnitude lower latency than mobile devices. For example, execution of the *NasNet Large*

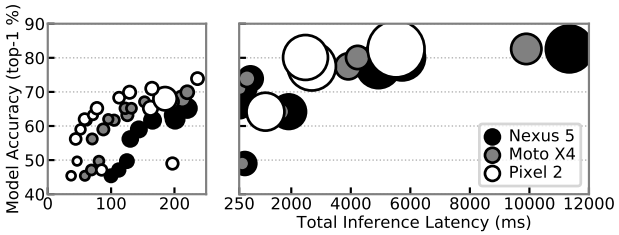


Fig. 2: Model execution latency on a range of mobile devices. Shown are the execution times for 21 pretrained models [15] running on three mobile devices via the TensorFlowLite framework. The size of the mark corresponds to the standard deviation of the inference latency. We observe both that high-accuracy models take multiple seconds to run on all devices and that newer devices, such as the Pixel 2, have access to many more models than older devices.

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

TABLE I: Symbols used throughout paper.

model takes over 5 seconds on all of our mobile devices but takes only 113ms on a server with GPU (shown in Table III). This decrease in latency means that leveraging these servers could allows for more complex models to run in less time than low accuracy models on-device. However, transferring the input data to the cloud-based servers can incur long and unpredictable network latency [7], [17].

Model serving systems [18]–[20] allow mobile applications to leverage these cloud-based systems, e.g. through a REST API. However, to utilize such systems as model backends, mobile developers need to *manually* specify the exact model to use through the exposed API endpoints. This manual model selection fails to consider the impact of dynamic mobile network conditions, which can take up a significant portion of end-to-end inference time [17], [21]. Such static development-time decisions can lead to mobile application developers either conservatively picking a low-latency but low-accuracy model that will meet the SLA or risk violating the SLA by choosing a high complexity model that will take longer to execute which may be compounded by unexpectedly long network transfer time.

In summary, cloud-based inference has the potential to support a plethora of application scenarios, simple or complex, and heterogeneous mobile devices, old or new. However, current mobile-agnostic serving platforms fall short by not automatically adapting inference accuracy to varying time requirements of mobile inference requests.

C. Hybrid Inference

Hybrid inference spread the execution of models across both the mobile device and a cloud-based server, as shown in Figure 1c. By splitting the execution between the two locations hybrid inference allows for decisions to be made at runtime regarding acceptable model execution time latency. This in turn allows for run-time inference latency improvements.

Hybrid approaches mainly differ in how they divide the model execution between the device and the remote server. Split inference approaches [22] break pretrained models into individual layers which can then be executed either on-device or in-cloud. Typically this is used to execute the first few layers of a model on-device before transferring intermediate data to the remote server for the completion of execution. The break point can be chosen so as to reduce the amount of data to be transferred and thus decrease the overall inference latency. However, this relies on the network connection being fast and reliable, or else may still lead to a large transfer latency.

To try to avoid this delay, early exit approaches [23], [24] design models in such a way as to allow for intermediate results to be reported along with the inference confidence. If this confidence is considered high enough then this intermediate response is returned; but if not then inference continues using the intermediate data. This intermediate data could potentially then be transferred to a remote server for further execution.

A shared limitation of split inference and early exit approaches is that execution can only leverage a single model across both locations. Therefore they need to either use low-accuracy models or risk long network latency.

In summary, hybrid inference allows for decreasing latency by breaking the inference model into parts and selecting where and whether each of the pieces should be executed. However, it is possible that this leads to longer latency and misses the opportunity to improve accuracy.

III. Problem Statement

We are targeting the problem of mobile deep inference where mobile applications request inferences from a framework. We assume that a mobile device which might be disconnected from the network while requesting inference services. Additionally, developer of applications has access to a set of models that present a range of different accuracies and latencies [15], [25] for the same task. Therefore, the problem resolves to how to get high accuracy inference results to the mobile device within a given SLA.

More specifically, for a mobile device requesting an inference within a target latency, T_{sla} we are interested in selecting a model from the set of available models, $m \in \boldsymbol{M}$ in order to maximize the accuracy of the inference and return results within the target latency. (Note, these symbols, and others used throughout the 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 describes 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 aggregate accuracy should be as high as possible.

System model and assumptions. These models all perform the same function (e.g. image classification) and hence we refer to them as *functionally-equivalent* models. 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 but can take a variable network time T_{nw} in order to use the inference server. Each cloud server hosts a set of functionally-equivalent models. We additionally assume that required preprocessing is completed on the mobile device, which is reasonable due to on-device frameworks. For simplicity we assume all inference requests are for image classification tasks but MDInference can be extended to other deep-learning tasks by providing additional inference APIs.

With regard to requests, we assume that network time, T_{nw}, can be calculated or estimated through time synchronization, direct measurement or network modeling [7]. We also assume that each request has an appropriate T_{sla}, representing the target request-response latency.

IV. Mobile Inference Framework Design

We first discuss important considerations, outlined in Table II, in making a mobile focused inference framework in Section IV-A. We then discuss how existing frameworks are aware of some of these considerations but fail to consider all of them in Section IV-B. Next, in Section IV-C we introduce a hypothetical framework, MDInference, that is designed around these mobile-specific considerations to balance aggregate accuracy and latency for mobile applications.

A. Key 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 mobile applications. Existing approaches focus on optimizing for particular design goals (e.g. latency on mobile devices or inference server throughput) while failing to consider an overall mobile-friendly approach. 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 our other devices this model took up to 2.5x longer to execute.

We argue that a mobile inference framework should *dynamically* balance two design goals: *latency* and *accuracy*. The need for dynamically balancing these two goals is driven by the inherently dynamic and heterogeneous mobile environment, which is particularly seen in the network latency.

Latency is the amount of time required to return results to the mobile end-user. This metric is essential for mobile applications as they are inherently user facing. Furthermore, for mobile end-users consistent performance is important. Response times that are particularly long relative to the average will be more obvious to users.

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

TABLE II: Different mobile inference approaches and their goals and awareness. The three common approaches discussed each have different optimization goals. On-device inference relies on an awareness of available resources and target SLA to to optimize for inference latency. In-cloud inference has the goal to increase the throughput of inference engines for the most accurate models, showing an attention to SLA but ignoring the network. With hybrid approaches, the goals and awareness varies with the approach. Typically they are aware of some of the various factors but no single approach is aware of all three. MDInference uses an awareness of all three factors to achieve a reliable latency while increasing accuracy whenever possible.

Accuracy is the probability of the framework to return the correct response on request input data. For image classification models this is often reported as the top-1 accuracy, which describes the model's ability to correctly classify input images across all requests. For example, a classification model that can correctly identify dog breeds would have high top-1 accuracy. Increasing this accuracy is important but in the existing mobile frameworks introduced in Section II-A, it is often considered secondary to latency.

An ideal mobile inference framework would allow for both considerations to be optimized without sacrificing one for the other. To do this it would have to be aware of three major constraints in order to optimize latency and accuracy, which we introduce below and have summarized in Table II.

First and foremost, mobile devices experience a wide range of network conditions that can lead to a huge variation in the latency of transferring input data for remote inference. Any system that performs remote inference should be aware of this latency variation and able to adapt its inference decisions to minimize its impact. *Second*, mobile devices are inherently resource constrained leading to making on-device inference difficult, especially for older devices. A mobile-aware inference system should reduce its reliance on on-device inference as this resource constraint is usually dealt with by using low-accuracy models. *Finally,* mobile applications are user facing and therefore are generally very sensitive to response time. Therefore any system providing mobile devices with inferences should be able to provide results within a reasonable time, often defined by an SLA.

B. Inference Serving Opportunities

Existing approaches to leveraging deep learning for mobile applications only focus on optimizing a single aspect of mobile inference, which can lead to a suboptimal experience for mobile end-users.

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. This ensures that regardless of the network connectivity mobile

users can get inference results within a reasonable latency. One example of this is MobileNets [12] which allows for tuning the number of parameters within the model; such parameter tuning allows for a smooth trade-off curve between latency and accuracy based on the same basic 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% [12], [15] comparing the fastest *MobileNet* model to the most accurate *MobileNet* model. The problem of trading off accuracy for latency is further exacerbated by the need to make such decisions at development time. In particular, doing so at development time means an application either relies on a single model across all devices or requires the training of many different models. Further choosing the one model with the best accuracy-latency trade-offs is challenging given the wide array of supported mobile devices.

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 throughput of serving inference requests even with complex models [18], [19], [26]. As an example, we observed that the time to execute the *NasNet Large* model (82.6% accuracy) in the cloud can be less than running inference requests with the *MobileNetV1 1.0 (160 pixel input)* model (68.0% accuracy) on the fastest mobile device in our experiments (see Figure 2 and Table III). This 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 are typically operated oriented towards service providers. This has two impacts. First, cloud-based servers aim to achieve a service level objective aimed at only on-server time rather than including the network latency of the input data, leading to poor mobile performance [18], [27]. Second, request batching, which is used to optimize for throughput, leads to an increase in the latency of individual inferences [10], [18].

Hybrid inference spreads execution across multiple locations, allowing for decreased latency, 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 [22] or to conclude execution when confidence in the intermediate result is above a threshold [24]. As a result, frameworks that support hybrid inference have the flexibility to selectively improve the inference performance by spreading the model across different locations.

However, this requires both that intermediate data be transferred between locations and that the same model be executed in both locations. In the case that network transfer of intermediate data is impossible or takes an unacceptably long time the model must be executed entirely on-device. If too complex a model has been selected this could lead to unacceptable latency, or if a simple model is chosen it reduces the advantages of remote execution. Therefore hybrid frameworks share the same limitations as on-device frameworks in that the model to be used must be selected during development. Compounding this, network variation can reduce or potentially remove the ability of hybrid inference to decrease latency.

C. MDInference Framework Design

The key insight of MDInference is that we can leverage a set of cloud-based functionally-equivalent models and duplicate inference requests to improve accuracy while bounding latency, allowing the user to have a reliable experience that is improved whenever possible. Duplication is widely used in computing clusters to reduce the impact of stragglers [28], [29]. For mobile deep inference, MDInference is designed to duplicate the inference request of an application and execute it both on-device and in-cloud. MDInference stores a model locally to ensure results will be available for users within the SLA, while submitting a remote request at the same time, potentially improve the accuracy of inferences.

MDInference combines the advantages of existing approaches in order to improve end-user performance. MDInference can additionally improve the aggregate accuracy of inferences by using a more accurate on-device model, although this can impact the minimum serviceable SLA. By leveraging a range of models in the cloud MDInference can opportunistically improve accuracy whenever possible. Running a combination of local and remote inferences allows MDInference to provide for reliable latency and potentially improve accuracy.

MDInference accomplishes this by using two components. *First*, a cloud-based server selects between a number of available functionally equivalent models for one that can complete within a specific time limit by estimating the time consumed transferring input data. This algorithm is detailed in Section IV-D. *Second*, a local inference is begun on-device to ensure that results are available within the target SLA. The combination of these components ensures that inference output is available within the SLA, with the possibility of the cloud-based inference providing improved accuracy. We discuss the model selection algorithm in the next section and further discuss the implication of duplicating inference requests in Section IV-E.

D. Model Selection Algorithm

MDInference's model selection algorithm is designed to manage a set of functionally-equivalent CNN models and to 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 mitigate the impact of large variations in the mobile network latency that is typically problematic for cloud-based inference. The key insight of the model selection algorithm is that the variations of transfer latency for an inference request can be compensated for with the correct choice of inference model. This works because different functionally-equivalent models have different execution

times and accuracies. Concretely, it determines which CNN model to execute the inference request by explicitly making inference latency and model accuracy trade-offs.

MDInference works by selecting the most accurate model suggest 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} denotes the time to transfer the inference request and to receive the result, referred to as *network time*. 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 simple 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 $T_{input} \geq T_{output}$. 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 time budget of the request T_{budget}.

The basic approach described above assumes that the execution times and accuracies of models previously measured are still accurate. 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. As examples, in real-world serving systems such as TensorFlow Serving [19] and Clipper [18] heavily utilized models may cause requests to experience some degree of queuing delay, impacting their overall execution time, while due to concept drift [30] the accuracy of models might vary over time. To handle these changes 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 its latency relative to T_{budget} and accuracy. We implement this probabilistic approach via the below three stage algorithm.

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 simply select the model that has the highest accuracy and is expected to have an execution time less than the time budget. This is to make it likely that the model will finish within the calculated budget. Out of the models that satisfy this constraint, the most accurate model is chosen as our base model, represented as m_b.

In the case that no models satisfy the time budget constraint the fastest model available is chosen as the base model.

Stage two: optimistically constructing the eligible model set. In order to account for unknown performance variations, such as queueing delays or workload spikes [31], as discussed previously, 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 generally similar to the base model in terms of execution time but were not selected as the base model. Specifically, we construct the exploration set as $M_E = \{m \mid \mu(m) \in [\mu(m_b) - \sigma(m_b), \mu(m_b) + \sigma(m_b)]\}$ 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.

$$\boldsymbol{U}(m) = \boldsymbol{A}(m) \frac{T_{budget} - \big(\mu(m) + \sigma(m)\big)}{|T_{budget} - \mu(m)|} \tag{3}$$

MDInference than normalizes these utilities to calculate the selection probability as $\boldsymbol{Pr}(m) = \frac{\boldsymbol{U}(m)}{\sum_{n \in \boldsymbol{M_E}} \boldsymbol{U}(n)}$ and picks the m' accordingly. This helps avoid choosing models with lower inference accuracy, wider inference time distribution, and outdated performance profile.

E. Request Duplication

To ensure that all requests can be serviced within the SLA, MDInference duplicates requests in order to bound the tail latency of requests. As discussed previously in Section II-A many efficiency-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 in Section IV-D. This request aims to return results within the SLA, but can not be guaranteed to due to the unreliable network connection. The second inference request is executed locally using the on-device model. This on-device model for MDInference is chosen to be the fastest available model, allowing for SLAs as low as 50ms, although any model that satisfies the SLA goal can be used.

There are two potential outcomes to this duplication. First, the SLA expires without the remote inference request having returned results. In this case MDInference uses the results of the on-device model. In our experiments this occurred in only 3.16% of cases. Second, the remote inference arrives before the SLA expires. In this case we use the results from the more accurate model, which are likely from the remote inference.

Although it is possible to combine results to boost performance via ensemble models [18] we leave this to future work.

V. Evaluation

Our evaluation goal is to quantify the effectiveness of our hypothetical system, MDInference, in opportunistically improving aggregate accuracy for mobile devices. We do this by first demonstrating the effectiveness of our model selection algorithm introduced in Section IV-D at increasing the aggregate accuracy under a range of SLAs when compared to a number of alternative approaches. We next demonstrate the benefit of request duplication in Section V-D by analyzing its impact on SLA adherence and accuracy.

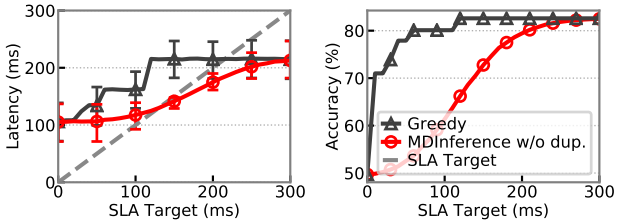
Key Metrics. The first metric that we are measuring in many of our evaluations is aggregate accuracy, which we introduced in Section III, and can be thought of as the average accuracy of all inferences. We additionally measure the SLA adherence which is the percent of requests that return results to 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 use the results of the on-device model and when aggregate accuracy was improved by leveraging in-cloud models. We examine the impact this has on the aggregate accuracy in Section V-D.

Simulation setup. In our simulations, we leverage a range of models, summarized in Table III [8], [9], [12], [15], [32]–[34], 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. Model accuracies are obtained from the source of pretrained models [15] unless otherwise noted. The mobile network we use as the basis for many of our simulations assumes that transferring an input image takes 100ms$\pm$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 record the model selected by MDInference (and baseline algorithms) and relevant performance metrics. We repeat each simulation for a variety of SLA target and network profiles combinations.

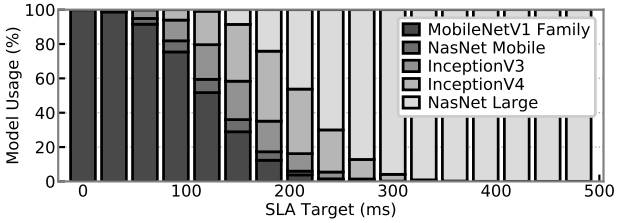
When comparing individual requests using different model selection algorithms we use 50 requests from our university WiFi and a residential WiFi network. For all tests except for those in Section V-D we evaluated the model selection capabilities of MDInference without duplication of requests.

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

TABLE 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 use these models in simulations to study MDInference's effectiveness in trading-off aggregate accuracy and time. Note, *NasNet Fictional* is a made-up model based on *NasNet Large* and is only used in Section V-C.



(a) Comparison of average end-to-end latency (shaded with one standard deviation) and accuracy. MDInference's selection algorithm is able to track the SLA target when SLA $\geq$100ms while the greedy approach fails to do so. MDInference is able to improve achieved model accuracy *safely* as the SLA target increases. Note that our model selection algorithm improves standard deviation of inference times as well, bounding them to the SLA target.



(b) Illustration of the models used by MDInference to enable smooth trade-off between latency and accuracy. 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.

A. Benefits over static greedy model selection

Figure 3a shows the average end-to-end inference time (left) and aggregate accuracy (right) achieved by our model selection algorithm and a static greedy algorithm that simply picks the most accurate model that can complete within the given latency. 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 ($\sim$115ms) while *static greedy* continues to frequently incur SLA violations until SLA target is more than 200ms. The key reason for this is that MDInference was able to effectively trade off aggregate accuracy and response time by choosing from a diverse set of models. Figure 3b illustrates the percentages of different CNN models selected by the model selection algorithm. Consequently, MDInference had an aggregate accuracy of 68% (on par to using *MobileNetV1 0.75* which can take 2.9x more time running on mobile devices) under low SLA target ($\sim$115ms), but was able to match the aggregate accuracy achieved by *static greedy*

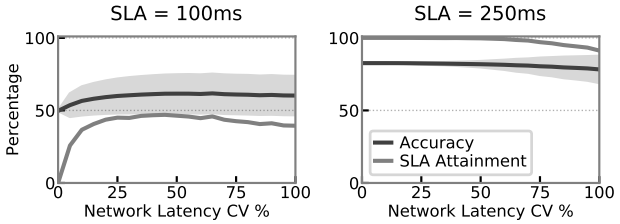


Fig. 4: Aggregate accuracy of MDInference at different levels of CV with T_{nw} = 100ms. The initial low level of SLA attainment is due to the fact that the network time is initially 100ms, leaving no time for inference execution. As the variability of the network increases MDInference can take advantage 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 becomes too much at which point it begins having to use lower latency models.

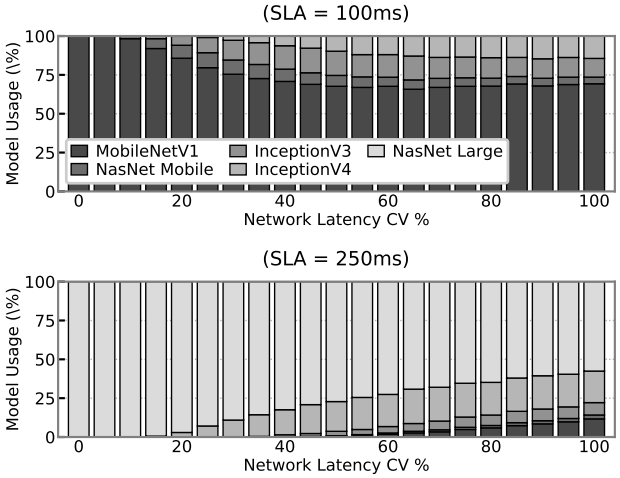


Fig. 5: Model usage vs. network latency (CV) shown at two different SLA targets. When there is a reliable network (i.e. low CV) single models dominate as all requests have the same inference time budget. As the network becomes increasingly volatile a wider range of models, including both higher and lower latency models, is used to meet the SLA.

for higher SLA target. Note that even though *static greedy* achieves up to 12% higher accuracy it does so by sacrificing inference latency.

Figure 3b illustrates the CNN model usage patterns (i.e., percentage of model being used for executing the inference requests) under differnet SLA targets. At very low SLA target (less than 30ms), MDInference aggressively chooses the fastest model, *MobileNetV1 0.25*, as described in Section IV-D. As the SLA target increases, MDInference explored more accurate but slower models than *MobileNetV1 0.25*.

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 MDInference 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 is sufficiently large, as shown in Figure 3a 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 always 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 user experience. To closely 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%. Here CV is defined as the ratio between standard deviation and average. For example, a CV of 0% indicates a very stable network condition while a CV of 100% means the network latency distribution is dispersed with its standard deviation equals to its average. As a point of reference, our measured Campus WiFi network has a CV of 74%.

Figure 4 shows the aggregate accuracy and SLA adherence achieved by MDInference. For low SLA target (100ms), when the network is relatively stable MDInference had an SLA attainment of less than 50%. As the network condition becomes more variable, MDInference was able to increase the aggregate accuracy gradually while maintaining the SLA attainment. This increase is because at first MDInference was only able to service 50% of requests because for the other half of the requests the network transfer consumes the entire SLA time. This, however, means that some requests take much less than the SLA allowing us to leverage more accurate models.

It is important to note that MDInference performed as expected by choosing the fastest possible model, i.e., *MobileNetV1 0.25*. We note that, to satisfy such stringent SLA targets with high network latency variation, alternative approaches such as provisioning inference servers near network edge or executing latency-optimized *MobileNetV1 0.25* on-device are generally preferable.

When given a reasonable SLA target, one which is slightly more than average network latency plus the time to execute the most accurate model *NasNet Large*, MDInference continued using high accuracy models regardless of network variation with high SLA attainment and maintains an accuracy around 80% (slightly less than *NasNet Large*).

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 this is enough to cover 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 network became more variable (i.e. high CV), MDInference

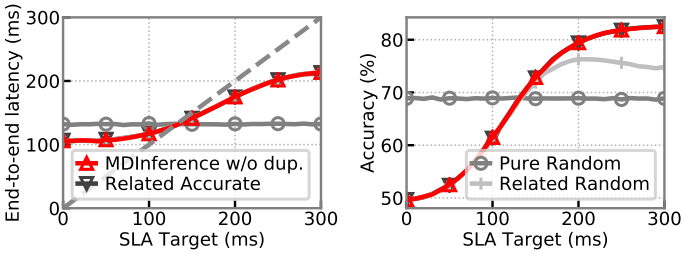


Fig. 6: Decomposition of benefits of MDInference's three-stage algorithm. MDInference achieves similar accuracy and SLA attainment compared to *related accurate*, indicating the effectiveness of our probabilistic approach. Both *pure random* and *related random* have poor aggregate accuracy due to their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

	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%

TABLE IV: ***Accuracy and on-device model reliance for different networks***

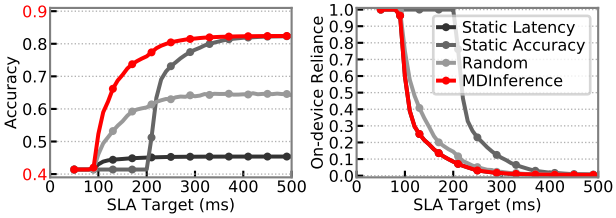


Fig. 7: Aggregate accuracy and on-device model reliance on residential network. The improvements of MDInference are seen not just at a single SLA but throughout all SLAs. At lower SLAs MDInference can quickly improve the aggregate accuracy. Meanwhile, MDInference also reduces the reliance on on-device models at low SLAs, much more quickly than other approaches.

matched the network variability by using a subset of faster models. This is because even as the network becomes less certain it affords MDInference with more opportunities to use models with higher 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. When tasked with a very low SLA, MDInference, even without request duplication, provided inferences in many cases. We analyze the request duplication that can further reduce SLA violations in Section V-D.

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 IV-D). For each of the three stages we compare to a different 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. These three algorithms were tested with a network with latency $100ms \pm 50ms$.

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. This indicates that the construction of M_E, by stage one and two, was effective and enabled good exploration opportunities to stay closely below SLA targets. The ability to adapt to increased SLA targets is important because it means we have the flexibility to use more accurate models.

Similarly, 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 (∼150ms), the exploration set M_E primarily consisted of two models: *NasNet Large* and *NasNet Fictional*. At this point, *related random* algorithm started to experience aggregate accuracy degredation since it did not 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 decrease in accuracy using MDInference when compared to *related accurate* algorithm. This is because that *related accurate* will always choose the most accurate model from M_E while MDInference has a low probability of picking *NasNet Fictional* so as to update its model performance profile. As mentioned before, models such as *NasNet Fictional* should not be completely ruled out from selection. The probabilistic behavior of MDInference is meant to allow for this exploration even while generally maintaining 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 inference conditions. All three stages contribute to and help MDInference cope with the variable network conditions and improve aggregate accuracy safely.

D. Effectiveness of Request Duplication

To test the reliance of MDInference on on-device inference we used a network 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

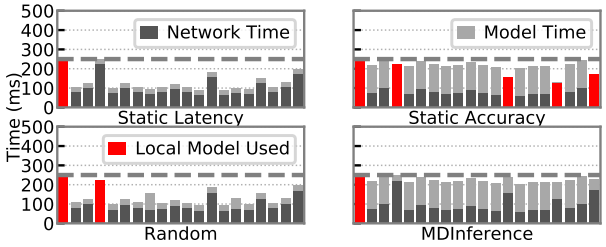


Fig. 8: Effectiveness of model selection approaches. The time needed for inference is shown for 20 randomly sampled requests from our residential network shows that in many cases MDInference can find a model that yields results within the requested budget. Other approaches, such as Static Accuracy, return high-accuracy results but rely more heavily on the on-device model due to network variation.

51.9KB±53.6KB in size. The model simulated for use on the mobile device was the fastest *MobileNetV1 0.25 (128 pixel input)* model, which was excluded from the in-cloud model selection. This *MobileNet* model was chosen as it represents the single model most likely to complete within any SLA for all tested mobile devices.

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

The aggregate accuracy and on-device reliance for these four approaches 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.

Table IV compares the aggregate accuracy and on-device reliance for all four model selection algorithms. MDInference 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 maintains the same miss rate as using our fastest model in all cases, improving over the static accuracy approach by over 19% in some cases.

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 be completed on the remote server but in some cases, which highlight the network latency in red, the on-device model must be used. This highlights how MDInference adapts its model selection to the networks variability allowing it to minimize the number of times it relies on the on-device model and thus boosting its aggregate accuracy.

VI. Discussion & Future Work

In this section we discuss further directions that this work can take centered around the idea of providing mobile-centric, network-aware inferences we have a number of possibilities for future work.

Energy Consumption. The duplication of inference requests solves the issues of SLA violations, allowing users to have consistent performance. However, this requires energy consumption on both the device and the cloud-based server, as well as requiring energy on-device for both network communication and inference. Therefore, identifying times when duplication is needed and only duplicate under these situations would allow for a decrease in overall 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 either statically selected models. While some model optimizations can provide this ability to simplify models post-training [35], these provide only limited options. Alternatively, models that are designed like MobilNets to provide a range of options, but tunable after training would be ideal.

Spanning subsets of models. Figure 3b demonstrated the number of models that are used to achieve a wide range of inference latencies. One striking factor in this is how the vast majority of the models chosen fall into a small subset of models. This potentially could indicate that there exists a subset of models that could service nearly all requests. 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.

VII. Conclusion

In this work we introduced a holistic approach to designing a mobile-oriented deep inference framework. By following this approach that focuses on identifying user needs and the constraints of mobile devices, we can better develop frameworks for providing inferences. We used this approach to introduce a hypothetical framework we call MDInference.

MDInference allowed us to 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 serving the highest latency model with duplication of inference requests. This shows the potential to improve user experience by focusing on how to address the constraints of mobile devices directly.

VIII. Acknowledgements

This work was supported by NSF Grants #1755659 and #1815619

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. Vigas, 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] 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

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

[9] 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.

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

[11] 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.

[12] 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."

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

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

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

[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. 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.

[22] 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.

[23] 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.

[24] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "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.

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

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[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] 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.

[33] 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."

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

[35] 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.