

# Characterizing the Deep Neural Networks Inference Performance of Mobile Applications

Samuel S. Ogden, Tian Guo

**Abstract**—Today’s mobile applications are increasingly leveraging deep neural networks to provide novel features, such as image and speech recognitions. To use a pre-trained deep neural network, mobile developers can either host it in a cloud server, referred to as *cloud-based inference*, or ship it with their mobile application, referred to as *on-device inference*. In this work, we investigate the inference performance of these two common approaches on both mobile devices and public clouds, using popular convolutional neural networks. Our measurement study suggests the need for both on-device and cloud-based inferences for supporting mobile applications. In particular, newer mobile devices is able to run mobile-optimized CNN models in reasonable time. However, for older mobile devices or to use more complex CNN models, mobile applications should opt in for cloud-based inference. We further demonstrate that variable network conditions can lead to poor cloud-based inference end-to-end time. To support efficient cloud-based inference, we propose a CNN model selection algorithm called CNNSelect that dynamically selects the most appropriate CNN model for each inference request, and adapts its selection to match different SLAs and execution time budgets that are caused by variable mobile environments. The key idea of CNNSelect is to make inference speed and accuracy trade-offs at runtime using a set of CNN models. We demonstrated that CNNSelect smoothly improves inference accuracy while maintaining SLA attainment in 88.5% more cases than a greedy baseline.

**Index Terms**—Mobile application, DNN inference service, DNN model management, performance optimization



## 1 INTRODUCTION

Resource intensive deep learning models are increasingly used in mobile applications [1, 2] to add features such as real-time language translation, image recognition and personal assistants [1]–[3]. To use deep learning models, mobile applications can either utilize cloud-based inference services or run directly on-device. However, achieving faster deep inference with high accuracy is often constrained by mobile computation, storage and network conditions.

In this work, we ask the question: what are the performance trade-offs of using on-device versus cloud-based inference for mobile applications. We conduct an empirical measurement study to quantify both the end-to-end inference time and resource consumption under different setups. In particular, we identify a number of key factors, including deep learning frameworks, mobile devices, and CNN model compression techniques, that impact on-device inference. Further, we study the cloud-based inference performance under different mobile network conditions and using different cloud servers. We identify that model startup latency can impose orders of magnitude time overhead and should be properly considered when managing cloud inference servers.

In sum, our observations reveal the relatively large, though shrinking, performance gaps between on-device and cloud-based inference. As such, it is still preferable for older

mobile devices that require more complex CNN models to resort to cloud-based inference. To support this efficient cloud-based inference under dynamic mobile network conditions, we propose CNNSelect to manage and select the most appropriate CNN model for each mobile inference request. The key insight for designing CNNSelect is that each model in a set of CNN models exposes different inference time and accuracy trade-offs that can be leveraged to mask the inherent mobile network variability. The dynamic model selection algorithm frees mobile developers from specifying CNN models before deployment and the probabilistic-nature makes it more robust towards unpredictable performance variations such as increased inference time due to sudden workload spikes.

We make the following contributions.

- **Performance characterization of mobile deep inference.** We conducted an extensive empirical performance analysis of using CNN models in mobile applications. We identified key performance factors, e.g., mobile hardware and network conditions, and quantified their impacts for both on-device and cloud-based inference.
- **Performance-aware CNN models selection algorithm.** We propose CNNSelect, a probabilistic-based algorithm that chooses the best CNN model to account for dynamic mobile environments when using cloud-based inference. We conducted an end-to-end evaluation and extensive empirical-driven simulations that demonstrate CNNSelect’s ability to smoothly trade-off between inference accuracy and time.

• Ogden and Guo are with the department of Computer Science, Worcester Polytechnic Institute, MA, 01609.  
Email: ssogden@wpi.edu, tian@wpi.edu

This work is supported in part by National Science Foundation grants #1755659 and #1815619 and Google Cloud Platform Research credits. Manuscript received September 9, 2019; revised month day, year.



mobile device	OS version	CPU	GPU	Memory (GB)	Storage (GB)	Battery (mAh)
Nexus 5	Android 6.0	2.26 GHz quad-core	129.8 GFLOPs Adreno 330	2	16	2300
LG G3	Android 6.0	2.5 GHz quad-core	129.8 GFLOPs Adreno 330	3	32	3000
Moto G5 Plus	Android 7.0	2.0Gz octa-core	130 GFLOPs Adreno 506	3	32	3000
Pixel 2	Android 9.0	4x 2.35GHz 4x 1.9GHz	567 GFLOPs Adreno 540	4	128	2700

TABLE 1: **Mobile devices used in on-device measurement.** We used four mobile devices that run their respectively most up-to-date OSes and have variable hardware resource capacities. Among which, Pixel 2 has a much faster mobile GPU.

tion fails to consider the impact of dynamic mobile network conditions, which can take up a significant portion of end-to-end inference time [26, 27]. Such static development-time efforts can lead to picking a faster but less accurate DNN model or risking SLA violations by choosing a more sophisticated DNN model.

Cloud-based inference has the potential to support a plethora of application scenarios, simple and complex, and heterogeneous mobile devices, old and new. However, current mobile-agnostic serving platforms fall short in automatically adapting inference accuracy to varying time requirements of mobile inference requests.

### 3 UNDERSTANDING ON-DEVICE INFERENCE

In this section, we perform an empirical measurement of on-device inference time and study the impact of key factors such as deep learning frameworks, mobile capacities, CNN models and model compression techniques. We then analyze the resource and energy implications of running on-device inference.

**Measurement methodology.** We implemented two Android-based mobile image classification applications and evaluated the on-device inference performance with four mobile devices outlined in Table 1. The first application compared two mobile inference execution frameworks, Caffe2 [13] and CNNDroid [28] in order to compare setup and execution time of CNNs on the Nexus5 device. The second application leveraged TensorFlow Mobile [29] in order to examine the impact of various model architectures and optimizations across all of our mobile devices.

For these tests we used a variety of CNN models. For the first application, we used AlexNet [4], NIN [8], and SqueezeNet [9]. These three models were chosen because they have similar top-5 accuracy on the ImageNet dataset [30] but differ vastly in terms of model size and complexity. The second application used four models that consisted of two accuracy-optimized models, i.e., InceptionV3 and InceptionV4 [31], and two time-optimized models, i.e., MobileNetV1 0.25 and MobileNetV1 1. [32]. These models were chosen for exposing a range of accuracy and complexity trade-offs. Using the second application we also did an in-depth study of the InceptionV3 model and optimizations thereof.

We furthermore used two different image datasets for generating inference requests. For the first application we used an image set (*images\_1*) that consists of 15 images. For

the second application we used a training set of 3314 to re-train our target models and another 1000 images (*images\_2*) to evaluate inference accuracy and time.

For each experiment we break the end-to-end inference time into three parts: loading input image, loading CNN model, and CNN model execution time. In the first application we output these time intervals to the Android log file which we accessed with `Logcat`, a command line tool, allowing us to additionally collect Android Runtime (ART) garbage collection information and application-level logs. We additionally measured the power and resource consumption with the Trepro profiler [33] to sample battery level, normalized CPU and GPU load every 100ms, following best practice techniques to minimize profile impact. Our second application recorded time intervals to an SQLite database.

#### 3.1 On-device Inference Performance Analysis

We begin by analyzing the performance differences by dissecting the on-device image recognition task with an in-depth study of time breakdown, and resource utilization. We focus on understanding the performance of on-device deep inference and the implications for potential performance improvement.

**Impact of Deep Learning Frameworks.** The choice of deep learning framework can impact the CNN model design and have a significant impact on the performance, due to different model sizes and complexities. We quantify such impacts through both the model loading and inference execution time. We plot the loading time in Figure 4(a) in *log* scale. For loading the same model (AlexNet and NIN), the ported Caffe library takes up to 4.12 seconds, about 22X faster than using CNNDroid. Furthermore, it only takes an average of 103.7 ms to load the smallest SqueezeNet model.<sup>1</sup> This loading happens whenever users first launch the mobile application, and potentially when a suspended background app is brought back. Our measurement of CNNDroid’s long loading time suggests that users need to wait for up to 88 seconds to be able to interact with the mobile app. Although long loading time might be amortized across a number of inference requests during one user interaction session, it still negatively impacts user experiences.

Next, we show the time taken to perform inference on the input image using five different configurations in Figure 4(b). For each configuration, we measure the computation time taken for all five images and collect a total of 75 data points. Each bar represents the average computation time across three versions of the same image and the standard deviation. CNNDroid-based AlexNet inference achieves the lowest average of 1541.67 ms, compared to the longest time of 13745.33 ms using ported Caffe NIN model. Even with the fastest device-based inference, it still takes three times more than CPU-based cloud inference [18]. In addition, we plot the end-to-end inference time in Figure 4(c). This total inference time includes the bitmap scaling time, the GC time, and the model computation time. CNNDroid-based approach takes an average of 1648.67 ms for performing object recognition on a single image, about

1. We did not measure the performance of SqueezeNet using CNNDroid due to the lack of support by CNNDroid.

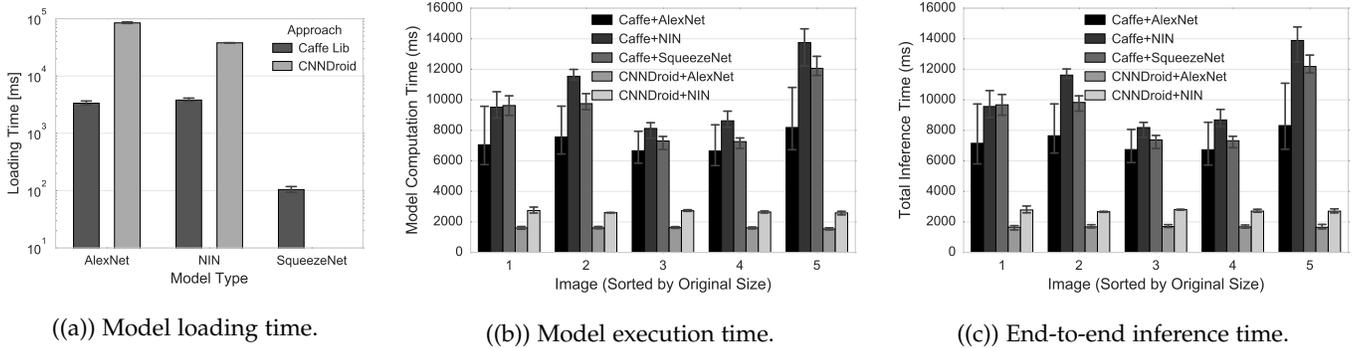


Fig. 4: **Caffe on-device inference time.** We compare two approaches for device-based inference using three CNN models. When using CNNDroid-based approach, trained models need to be converted to supported format.

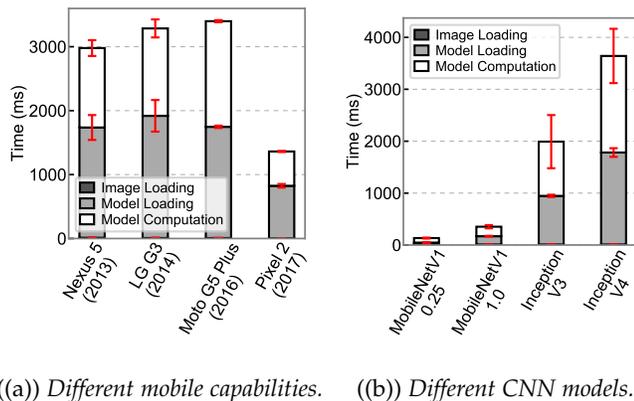


Fig. 5: **Comparisons of on-device inference using different mobile devices and CNN models.** The high-end Pixel 2 improves on-device inference speed by 2.3X compared to other devices. The mobile-specific CNN models, e.g., MobileNet family, only take an average of 150ms to run and are 27.2x faster than server-centric models, i.e., Inception. Note, the time to load the 330KB image is shown but negligible.

seven times faster than using ported Caffe models. Based on the response time rules [34, 35], it might lead to poor user experiences when using certain on-device inference.

**Impact of Heterogenous Mobile Capabilities.** Next, we measure the end-to-end image classification time for running InceptionV3 using TensorFlow Mobile framework on different mobile devices. In Figure 5(a), we plot the time breakdown of loading the image and the CNN model, as well as model computation time. Mobile devices vary in their ability to load and run the server-centric InceptionV3 model. As we can see, older or less capable devices (left three bars) take up to 3.5 seconds to return inference results to mobile users. However, the nearly 2 seconds of model load time could be amortized across multiple runs or reduced by using smaller CNN models. Although newer mobile hardware is able to deliver acceptable user performance even for relative complex CNN models, older mobile devices would benefit from having access to simpler CNN models or using cloud-based inferences.

**Impact of CNN Models.** CNN models differ from traditional deep learning models in that they use convolutional

layers as their input. This has two main effects. First, convolutional layers typically use shared weights for their convolutional layers, decreasing the amount of storage needed for the models themselves. Second, these convolutional layers increase accuracy for image classification tasks but greatly increase the memory usage of model execution due to the generation of intermediate data. Therefore we further study the performance differences when running different CNN models on a powerful mobile device. We choose four popular CNN models and run them using TensorFlow mobile framework on a Pixel2 mobile phone [36]. For each CNN model, we measured the inference time for running *images\_2* test set and calculated the average. In Figure 6(a), we show that Pixel2 is able to execute both MobileNetV1 models in less than 133ms (352ms) on average. However, Pixel2 takes 48.7X longer to load larger Inception models and is 5.6X-27.4X slower to perform the model computation using Inception models. We observed that the model loading time is more consistent compared to the model computation. This indicates that model computation might be more subject to the resource interference between foreground and background applications.

**Impact of CNN Model Compression.** Compactly storing deep learning models is key to our vision of supporting a wide selection of on-device models. However, compression techniques generally trade-off inference accuracy for compression effectiveness. In this section, we first quantify the storage savings of four post-training compression techniques and then compare each technique’s impact on inference performance.

Figure 6(a) compares the uncompressed and compressed model sizes. For each model, we plot the baseline uncompressed size (left bar) and the gzip version (right bar). As we can see, the 8-bit quantized model leads to the most storage saving of 75% regardless of gzip compression. In addition, the unquantized models (retrained and optimized) see only about 7% savings while the rounding quantized model sees a 72.6% storage reduction after being gzipped. Our observations suggest that both quantization and gzip compression can lead to significant storage savings, especially when combined.

Next, we compare the inference speed and accuracy of each model. Figure 6(b) shows the time taken by each model and its accuracy. It is important to note that majority of

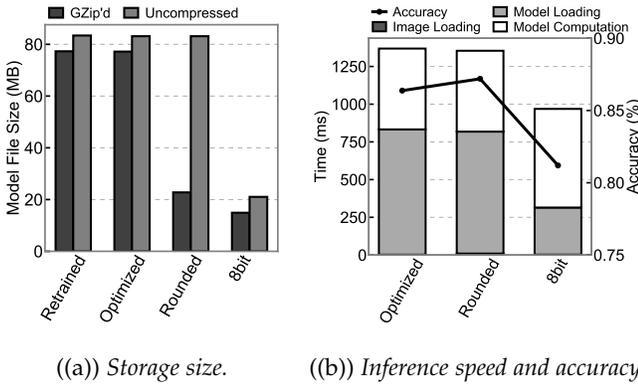


Fig. 6: **Comparisons of model compression techniques.** Model compressions can greatly affect the on-disk storage requirements as well as impacting the accuracy and latency of models. Overall, the 8-bit quantized model is superior in storage saving and inference speed but experiences 6% accuracy decrease. The retrained model is excluded due to unsupported operations on mobile devices.

TABLE 2: **Summary of GC activities when using CNNDroid-based inference.** ART uses the default CMS GC, and the GC time takes up to 9.89% during model loading, and up to 25% during user interactions. The average GC pause time can be up to 39.23 ms.

CNNDroid On-device Inference	Phase	Duration[ms]	Num. of GC	GC Time [ms]	GC Pause [ms]
AlexNet	Load	84537.33	4.33	513.55	10.42
NIN	Model	37975	16.67	3757.30	175.76
AlexNet	User	11800	4	536.55	4.60
NIN	Interaction	17166.67	7	4307.18	274.66

the inference time is in loading the model into memory and thus could be amortized over sequential mobile inferences. However, for one-off mobile inferences, the model loading time dominates the end-to-end inference time. The small 8-bit quantized model provides the fastest end-to-end response time with the lowest model loading time, but a slightly increased inference time. In our results, we do see a small accuracy increase for the rounding quantized model. But such observations are not common and would be detected through metadata tracking. In sum, model compression techniques have different impacts on model storage, inference speed and accuracy. We could leverage these observations to carefully select techniques that provide different tradeoffs.

**Impact of Limited Mobile Memory.** During loading CNNDroid-based models, we observe much more frequent, and long lasting garbage collecting activities performed by Android Runtime in our mobile device. When running our app using CNNDroid library, we have to request for a large heap of 512 MB memory.<sup>2</sup> Even with a large heap, the memory pressure of creating new objects has lead to a total of 8.33 (and 23.67) GC invocations when using CNNDroid-based AlexNet (and NIN) model, as shown in Table 2. Our evaluation suggests that by allocating more memory to deep learning powered mobile apps, or running such apps in

2. Running the app with the default 192 MB memory will lead to `OutOfMemoryError`.

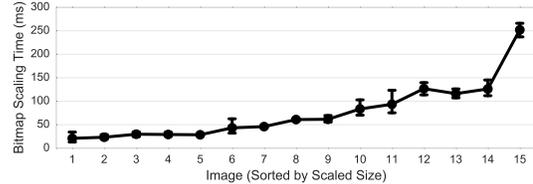


Fig. 7: **Bitmap downscaling time.** The time taken to downscale the image in the mobile device grow with the image size. Specially, larger images also experiences proportionally longer scaling time because of the limited memory resources assigned to the mobile application.

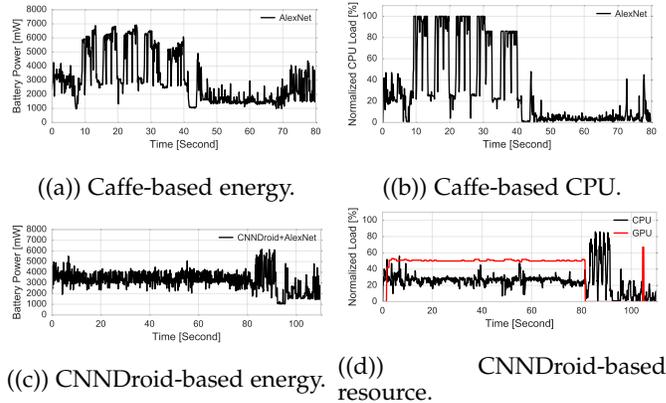


Fig. 8: **Energy consumption and resource utilization of on-device object recognition task.**

more powerful mobile devices can mitigate the impact of garbage collection.

**Impact of Image Size.** Because the CNN models in this test only require images of dimension 224 by 224 pixels to perform inference tasks, we can scale input image to the required dimension before sending. Figure 7 shows the time taken to scale images with different sizes. Each data point represents the average scaling time across five different runs. The time taken to resize image grows as its size increases. It is only beneficial to downscale an image of size  $x_1$  to  $x_2$  if  $T_d(x_1, x_2) + T_n(x_2) \leq T_n(x_1)$ , where  $T_d(x, y)$  represents the time to downscale an image from size  $x$  to  $y$  and  $T_n(x)$  denotes the time to upload an image of size  $x$  to a cloud server. For example, based on our measurement, it takes an average of 36.83 ms to upload an image of 172 KB to our cloud server. Also, from Figure 7, we know that it takes up to 38 ms to resize an image less than 226 KB. By combining these two observations, it is easy to conclude that directly uploading image one to five is more time efficient. We can expect to make informed decisions about whether resizing an image of size  $x$  before uploading is beneficial or not given enough time measurements of resizing and uploading steps. Our analysis shows that on-device inference’s performance bottlenecks mainly exhibit in loading model and computing probability steps.

### 3.2 On-device Inference Resource and Energy Analysis

In Figure 8, we analyze both the energy consumption and resource utilization when running our app in different con-

figurations. We compare the time-series plots of running AlexNet model using Caffe Android library and CNNDroid framework. The plots correspond to experiment runs that perform inference tasks on image set one.

For Caffe Android library based approach, we observe an initial energy consumption (and CPU utilization) that increases corresponding to loading AlexNet CNN model into the memory, a continuation of energy spike during model computation, and the last phase that corresponds to displaying images and the most probable label texts, in Figure 8(a) and Figure 8(b). The other two CNN models, NIN and SqueezeNet, exhibit very similar usage pattern<sup>3</sup>. Specifically, in the case of NIN, the initial model loading causes the energy consumption to increase from baseline 1081.24 mW to up to 5000 mW; when performing the model computation, both the energy consumption and CPU utilization spikes to more than 7000 mW and 66.2%. Note in the case of SqueezeNet, we only observe a very small window of both energy and CPU spikes at the very beginning of measurement. This is because SqueezeNet can be loaded in 109 ms, compared to more than 3000 ms to load either AlexNet or NIN.

In contrast, we observe two key usage differences in CNNDroid approach, as shown in Figure 8(c) and Figure 8(d). First, CNNDroid-based AlexNet exhibits a longer period of more stable and lower energy consumption compared to its counterpart in Caffe-based approach. This is mainly because CNNDroid explicitly expresses some of the data-parallel workload using `RenderScript` and is able to offload these workload to more energy-efficient mobile GPU [37] (indicated by the high GPU utilization during model loading). Second, the total model computation time is significantly shortened from 40 seconds to around five seconds. In all, by shifting some of computation tasks during model loading, CNNDroid-based approach successfully reduces the user perceived response time. However, the CNNDroid approach consumes 85.2 mWh energy, over 42% more than Caffe-based approach. Note 91% of CNNDroid energy is consumed during model loading phase, and therefore can be amortized by performing inference tasks in batch. In other words, the CNNDroid-based approach is more energy-efficient in performing inference tasks compared to the Caffe-based approach when CNN models are preloaded into the mobile memory.

## 4 UNDERSTANDING CLOUD-BASED INFERENCE

Next, we measured the cloud-based inference performance under different key factors such as cloud server capacity, CNN models, and mobile network. We then analyze the resource and energy implications of running cloud-based inference. We used four types of cloud-based servers (summarized in Table 3) with different capacities. These cloud servers range from burstable servers to GPU-accelerated servers. We run TensorFlow framework behind a flask-based web server and executed inference requests using different images and CNN combinations.

3. Interested readers can refer to our prior work [18] for additional results of Caffe-based NIN and SqueezeNet models, and CNNDroid-based NIN.

cloud server	vCPU	GPU	RAM (GB)	Storage	Network (Gbps)
t2.medium	2	N/A	4	N/A	N/A
c5.large	2	N/A	4	EBS only	Up to 10
p2.xlarge	4	1	61	N/A	High
g2.2xlarge	8 Intel Xeon E5-2670	1 Nvidia GRID K520	15	60GB instance store	N/A

TABLE 3: Cloud servers used in cloud-based measurement.

inference mode	image recognition time breakdown (ms)			
	model loading	image resizing	image uploading	probability computing
g2.2xlarge+CPU	N/A	76.2	36.8	238.6
g2.2xlarge+GPU	N/A	76.2	36.8	18.6
Nexus5+Caffe	2422.1	80.0	N/A	8910.6
Nexus5+CNNDroid	61256.2	70.4	N/A	2131.7

TABLE 4: Impact of hardware acceleration. We measured the image recognition time with and without using GPU. We also included the on-device performance from Figure 4 as baselines.

### 4.1 Cloud-based Inference Performance Analysis

In this section, we present the measurement results of image recognition time and mobile resource utilization for both cloud-based and on-device inference.

**Impact of hardware acceleration.** Table 4 summarizes the average end-to-end performance and resource consumption of executing object recognition using both cloud-based and on-device inference modes. We use CPU-only Caffe framework and GPU accelerated Caffe framework for toggling the CPU and GPU mode in our `g2.2xlarge` server. For each inference mode, we repeat the recognition tasks using all fifteen images and three CNN models. We measure the time to execute each step and calculate the average. Similar to on-device inference the task of image recognition is further broken down into four steps: loading CNN models into memory, downscaling image input to desired dimension, uploading input data to the cloud server, and executing the inference. In contrast to on-device deep inference, in the cloud-based scenario the time to load models is negligible because models already reside inside the memory and can be used to execute the inference task immediately. Similarly, on-device mode does not incur any time for uploading image bitmaps.

Recall, the inference time is the sum of rescaling, uploading bitmap and inference execution over one bitmap, and the recognition time is the sum of amortized model loading time over a batch of images and inference time. The average cloud-based inference time is 351.59 ms/131.59 ms when using CPU-only/GPU of a well-provisioned cloud instance hosted in a nearby data center. As shown, because inference tasks are typically data-parallel and therefore can be accelerated by up to 10x when using GPU. However, we should note that such results represent a lower bound performance of real-world setting. In a real-world deployment scenario, image recognition time can last much longer due to reasons such as overloaded cloud servers and variable mobile network conditions. The total inference time when running on-device is almost 9 seconds when using Caffe-based model, and 2.2 seconds when using CNNDroid model.

**Impact of model startup latency.** Next, we studied the inference execution time difference of *hot start* versus *cold start* model. Here, hot start inference time refers to the time

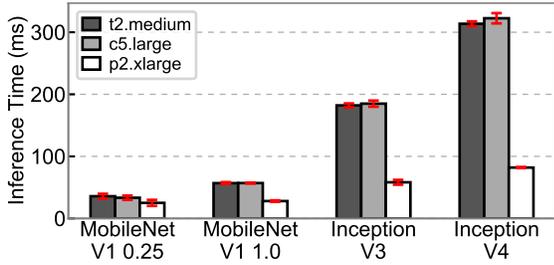


Fig. 9: Comparison of inference latency of cloud-based inference with four CNN models. The InceptionV4 that runs on the `p2.xlarge` GPU-accelerated server is over 2.5X faster than the `MobileNet 0.25` on the MotoX. This highlights the advantage of cloud-based inference in providing high-accuracy and low-latency results and demonstrates the potential of cloud-based inference for enabling inference latency and accuracy trade-offs.

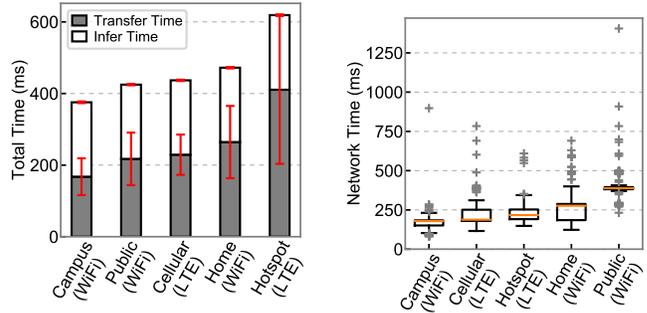
CNN model	accuracy (%)		inference time (ms)	
	top1	top5	hot start	cold start
SqueezeNet	49.0	72.9	28.61 ± 1.13	173.38 ± 25.73
MobileNetV1 0.25	49.7	74.1	25.73 ± 1.22	272.81 ± 45.00
MobileNetV1 0.5	63.2	84.9	26.34 ± 1.19	302.77 ± 45.50
DenseNet	64.2	85.6	49.55 ± 3.21	1149.04 ± 108.00
MobileNetV1 0.75	68.3	88.1	28.02 ± 1.14	351.92 ± 47.38
MobileNetV1 1.0	71.8	90.6	28.15 ± 1.22	421.23 ± 47.14
NasNet Mobile	73.9	91.5	55.31 ± 4.09	2817.25 ± 123.73
InceptionResNetV2	77.5	94.0	76.30 ± 5.74	2844.29 ± 106.49
InceptionV3	77.9	93.8	55.75 ± 1.20	1950.71 ± 101.21
InceptionV4	80.1	95.1	82.78 ± 0.89	3162.24 ± 133.99
NasNet Large	82.6	96.1	112.61 ± 6.09	7054.52 ± 238.36

TABLE 5: Summaries of CNN model statistics through empirical measurement. We measured the average inference time (with standard deviation) with/out startup latency for each model running on an EC2 `p2.xlarge` GPU server.

taken if the CNN model has been loaded into the memory, i.e., the model has been used to service inference requests previously. In the case of cold start, we reloaded the CNN model every time before executing the inference request and measured the cold start inference time starting from loading the CNN model until the inference response is generated.

Table 5 summarizes the CNN model accuracies and inference execution time averaged across 1000 inference requests. First, we observe a distinct correlation between accuracy and inference time. Additionally, the cold start time is generally much larger than the hot start time, and increases at a much faster rate. Another important observation is that cold start time is harder to predict than hot start time. For example, in the case of DensNet, we observed about 23X increase compared to 12X increase for a similarly sized MobileNet model. The increased time can be attributed to complex CNN structure which requires longer setup time and large CNN model size that leads to longer time to load into GPU memory. Therefore it is critical to keep important and often used CNN models in the memory.

**Impact of CNN models.** We next study how different CNN models and cloud servers lead to different inference execution time. We evaluated four different types of CNN models with reasonable inference accuracy (more detailed in Table 5). For each CNN model, we used an image of 110KB and repeated the measurement for 1000 times. We measured



((a)) Cloud-based inference response time. ((b)) Mobile network latency.

Fig. 10: Comparisons of cloud-based inference under different mobile network conditions. Under poor network connectivity, such as when using cellular hotspot, the transfer time almost doubled when compared to university WiFi. In addition, we demonstrate that network conditions play a critical role in cloud-based inference.

the inference execution time, which is defined as the time to generate output labels based on input images. In Figure 9, we plotted the average inference execution time for running these four CNN models in three cloud servers with different capacities. We make a few observations. First, regardless of the cloud servers in use—whether the cloud server is a burstable `t2.medium` that might be subject to performance fluctuations [38] or a powerful GPU server `p2.xlarge`, the simplest of the four CNN models `MobileNetV1 0.25` takes less than 50 milliseconds with up to 30% difference between `p2.xlarge` and `t2.medium`. Second, as the CNN models become more computation-intensive, both CPU servers start to take more than 313.7 milliseconds to finish in the case of `InceptionV4`. Meanwhile, the GPU server `p2.xlarge` was able to deliver low execution time in less than 82 milliseconds, which is still over 2.5X faster than running the simplest CNN model `MobileNetV1 0.25` on the MotoX mobile phone. Our measurements suggest that CNN models with different computation complexities, in addition to cloud servers, take different amount of execution time. These results also demonstrate the potential of using cloud-based inference for making inference latency and accuracy trade-offs dynamically.

**Impact of mobile network conditions.** We evaluate the end-to-end classification time when using models hosted in edge servers with the optimized model pre-loaded. Figure 10 shows the average classification time using an edge server in a variety of network conditions. The majority of the classification time is network transfer time with up to 66.7% in the cellular hotspot case. Overall, the edge-server based classification ranges from 375ms to 600ms in a well-provisioned `t2.medium` cloud server. When running the same classification task the Pixel 2 takes 536ms with a preloaded model and is on par with edge based inference. Mobile-based inference only delivers acceptable performance for newer and high-end mobile devices while edge-based inference is a viable option even under poor network condition. CNNSelect can leverage this observation to dynamically select inference locations.

inference mode	mobile resource consumption			
	CPU (%)	GPU (%)	Mem (GB)	battery (mW)
g2.2xlarge+CPU	6.2	0.9	1.28	1561.6
g2.2xlarge+GPU	6.4	0.4	1.31	1560.2
Nexus5+Caffe	35.0	80.0	1.64	3249.0
Nexus5+CNNdroid	22.2	70.4	1.75	2962.6

TABLE 6: **Mobile resource consumption.** We measured the mobile resource consumption when using cloud-based inference. We also included the on-device performance from Figure 4 as baselines.

## 4.2 Cloud-based Inference Resource and Energy Analysis

Table 6 shows the resource consumption of mobile device when running the object recognition mobile application. As a baseline, we measure the performance when the device is idle and the screen is turned on. The CPU utilization and power consumption is 3.17% and 1081.24 mW respectively. Cloud-based mode consumes roughly the same amount of CPU and 44.4% more power consumption comparing to the baseline. However, on-device mode not only incurs significantly higher CPU utilization (and in the case of CNNdroid, GPU utilization as well), but also require two times more power consumption when compared to the baseline. In all, we can calculate the energy consumption of different inference modes by multiplying the average inference (recognition) time by the average power consumption. Cloud-based inference requires as low as 0.057 mWh energy when using faster GPU computation, and on-device based inference consumes up to 8.11 mWh.

In sum, cloud-based inference exhibits substantial benefits in terms of inference response time and mobile energy savings over on-device inference, in this case by two orders of magnitude. This is due to more powerful processing power, shorter durations of inference and efficient use of network interfaces.

## 5 MANAGING AND SELECTING CNN MODELS

In this section, we describe our *multi-models* approach, CNNSelect, to mitigating the impact of the mobile network variations on cloud-based inference performance. CNNSelect manages a set of CNN models that exhibit different execution time and accuracy trade-offs, and selects the bests CNN model for a given mobile inference request. The key insight is that the variations of transferring the input data for an inference request can be masked with CNN models that take differing amounts of time to execute. When considering which CNN model to use, we assume that all CNN models are already loaded into the memory and that CNN model performance profiles are measured and managed by individual inference servers. We further assume that image preprocessing is handled in an intelligent way by the mobile device. In Table 7, we list all the symbols used in this paper.

For a mobile inference request, we assume that developers provide the target response time that indicates how long the end-to-end inference should take. This target time can alternatively be expressed as Service Level Agreement between mobile developers and cloud-based inference service

providers. Without loss of generality, we refer to this target response time as  $T_{sla}$ .

Next, CNNSelect estimates the remaining time  $T_{budget}$  that an inference request should finish executing.  $T_{budget}$  is calculated by taking the difference between  $T_{sla}$  and the network transfer time  $T_{nw}$ .  $T_{nw}$  can be estimated conservatively with  $2 * T_{input}$  where  $T_{input}$  denotes the time taken to send input data from the mobile device to the cloud-based inference server. In our application scenarios, we could expect  $T_{input} \geq T_{output}$  given inference requests, e.g., images, are often larger than inference responses, e.g., text labels. To summarize,  $T_{budget}$  can be calculated as:  $T_{budget} = T_{sla} - 2 * T_{input}$ .

To take into account of the CNN performance profiles getting outdated—leading to less accurate estimation of execution time, we define a threshold  $T_{threshold}$  which indicates how uncertain we are about the model performance profiles. The larger the value of  $T_{threshold}$ , the more outdated the inference performance profiles. In order to effectively explore all potentially high-accuracy models without violating SLA, we then expand the notation of time budget  $T_{budget}$  to a range  $T_R = [T_L, T_U]$ , where  $T_U = T_{budget}$  and  $T_L = T_U - T_{threshold}$  (see Figure 11). Intuitively,  $T_U$  represents the maximum amount of time that CNNSelect can use for generating an inference response without risking SLA violations. We refer to  $T_L$  as the *hard time limit*. On the other hand,  $T_U$  is referred to as the *soft time limit* and provides CNNSelect the flexibility to explore a subset of high-accuracy models  $M_E$  that exhibit different execution time  $\{T(m) | \forall m \in M_E\}$ .

Currently,  $T_{threshold}$  is configured by CNNSelect’s users, e.g., mobile application developer, as any values in the range of  $[0, T_D]$ , where  $T_D$  represents the expected on-device inference time. We choose to bound  $T_{threshold}$  this way to restrict the exploration set  $M_E$  and mitigate the undesirable behavior of starting on-device inference prematurely when cloud-based inference can finish without violating the SLA. However, CNNSelect could also dynamically adjust  $T_{threshold}$  based on its confidences of model performance profiles and will be explored as part of future work.

### 5.1 Opportunistic Model Selection

Next, we describe in detail how CNNSelect utilizes both the model performance profiles and the time budget range  $T_R$  to first pick a base model, then construct a set of eligible models  $M_E$  that is worth exploring, and last probabilistically select the model for executing the inference request. Our three-staged algorithm is designed to gradually improve our estimation of model performance profiles without incurring additional profiling overhead. In addition, if we are under time pressure to select models, our algorithm could be stopped any time after the first stage and will still select a quality model for performing the inference. In Figure 11 we provide an example walkthrough of CNNSelect.

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

Symbol	Meaning
$T_{sla}, T_{start}, T_{budget}$	Response time SLA, start time, and remaining name of a mobile inference request.
$T_{threshold}$	Confidence threshold of inference performance.
$T_{input}, T_{output}, T_{nw}$	Network time to send inference request and response, and both.
$T_D$	Expected on-device inference time.
$T_U, T_L$	The soft and hard time budgets.
$T_R, T_E$	The time budget and exploration ranges.
$K$	The total number of models.
$M_E$	The exploration set of models.
$\mu(m), \sigma(m)$	Average and standard deviation of historical inference time of model $m$ .
$Pr(m)$	Probability of model $m$ for performing inference.
$A(m), T(m), U(m)$	Accuracy, inference time and utility of CNN model $m$ .

TABLE 7: **Symbols summary.** Shaded symbols are specific to inference requests from different mobile devices, while others are related to CNN models.

$$\begin{aligned} & \underset{j}{\text{maximize}} && \mathbf{A}(m_j) && (1) \\ & \text{subject to} && \mu(m_j) + \sigma(m_j) < T_U, \quad j = 1 \dots K. && (2) \\ & && \mu(m_j) - \sigma(m_j) < T_L, \quad j = 1 \dots K. && (3) \end{aligned}$$

The high level idea is to select the most accurate model Equation (1) that are likely to finish execution within specified SLA target Equation (2) without triggering on-device inference Equation (3). Given that cloud-based inference execution might experience performance fluctuations that lead to a wider inference execution distribution [39, 40], we take into account of the standard deviation of model inference time and only select models that satisfy both the soft time limit  $T_L$  and the hard time limit  $T_U$ . By doing so, the selected models are of high accuracy and are very likely to finish execution within specified SLA. In the example walkthrough in Figure 11, CNNSelect will select model  $m_3$  as the base model. In scenarios where there is no base CNN models that satisfy the constraints, e.g., variability of mobile networks, CNNSelect chooses the CNN model with the lowest average inference time  $\mu(m_j)$  in order to provide a best-effort at SLA attainment.

**Stage two: optimistically constructing the eligible model set.** To account for cloud-based inference variations, either due to workload spikes [41] or insufficient/outdated CNN model performance profiles, CNNSelect leverages the basic idea of exploiting and exploration [42]. Given the base CNN model  $m^*$ , we then explore other potential CNN models that should also be inside the exploration set  $M_E$ . To do so, we leverage  $m^*$  performance profile and construct the exploration range  $T_E$ .

Specifically, we expand the hard time limit  $T_L$  with  $\mu(m^*)$  and  $\sigma(m^*)$  to construct the exploration range  $T_E$  with an acceptable distance  $(|T_L - \mu(m^*)|) + \sigma(m^*)$  as below.

$$T_E = \begin{cases} [\mu(m') + \sigma(m'), 2T_L - \mu(m') + \sigma(m')], & \text{if } T_L > \mu(m') \\ [2T_L - \mu(m') + \sigma(m'), \mu(m') + \sigma(m')], & \text{otherwise.} \end{cases}$$

Given this, we construct the exploration CNN model set  $M_E = \{m \mid \mu(m) \in T_E \text{ and } \mu(m) + \sigma(m) < T_U\}$ . As such, all CNN models in  $M_E$  satisfy our target performance while providing CNNSelect the opportunity to exploit the trade-offs between inference accuracy and time. In Figure 11, only model  $m_2$  and  $m_3$  are marked as the members of  $M_E$ .

**Stage three: opportunistically selecting the CNN model.** CNNSelect selects the CNN model  $m'$  that balances the risk

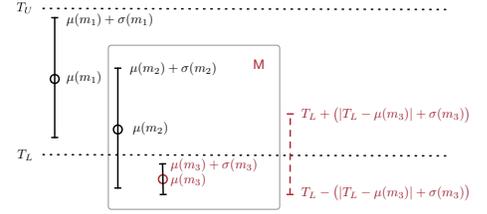


Fig. 11: **Walkthrough of CNNSelect.** Here, CNNSelect has access to CNN model performance profiles and inference accuracy:  $\mathbf{A}(m_3) > \mathbf{A}(m_1) > \mathbf{A}(m_2)$ .

of SLA violations and the exploration reward. Concretely, we calculate the utility for each CNN model  $U(m)$  based on its inference accuracy and the likelihood to violate response time SLA.

$$U(m) = \mathbf{A}(m) \frac{T_U - (\mu(m) + \sigma(m))}{|T_L - \mu(m)|} \quad (4)$$

CNNSelect then calculates the selection probability  $Pr(m) = \frac{1}{\sum_{n \in M_E} U(n)} U(m)$  and picks the  $m'$  accordingly.

This helps avoid choosing CNN models with lower inference accuracy, wider inference time distribution, and outdated performance profile.

## 5.2 Experimental Evaluation of CNNSelect

We quantify the effectiveness of CNNSelect, in dynamically selecting the most appropriate CNN model to avoid missing target response time while achieving good inference accuracy. We use a mix of experiments and simulations with real-world CNN models. We run our experiments using an Virginia-based Amazon EC2 p2.xlarge GPU server that manages two retrained CNN models, `MobileNetV1 0.25` and `InceptionV3`. After warming up the inference server, we have our Massachusetts-based image recognition Android application on MotoX (late 2017) that send inference requests of preprocessed images (average 330KB) over campus WiFi. For each SLA target, our mobile application sends 1000 inference requests and measure both the inference accuracy and end-to-end inference time. For our simulations, we leverage a number of CNN models, summarized in Table 5 [4, 22, 31, 43]–[46], that expose different accuracy and inference time trade-offs. Our simulations are seeded with empirical measurements of CNN model execution time and mobile network conditions. For each simulation, we generate 10,000 inference requests with a predefined SLA target and record the model selected by CNNSelect (and baseline algorithms) and relevant performance metrics. We repeat each simulation for different SLA target and network profiles combination.

### 5.2.1 Prototype evaluation

In Figure 12, we plot the percentage of inference requests that exceed the SLA (left y-axis) and the percentage of inference requests that are correctly classified (right y-axis) for different SLA targets. We further annotate the figure with two important timelines: on-device inference time with `MobileNetV1 0.25` and on-device inference time with

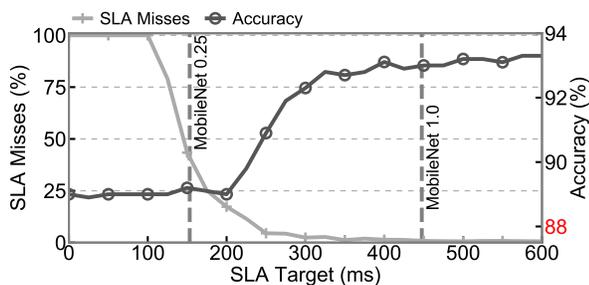
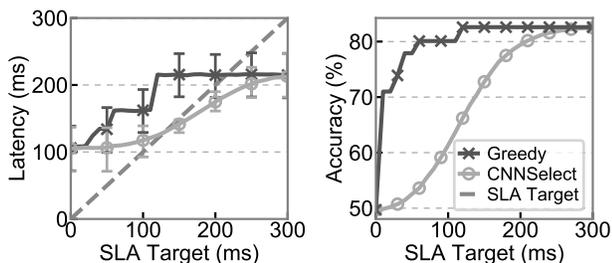
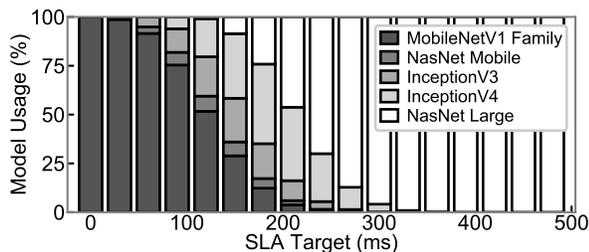


Fig. 12: **End-to-end performance of CNNSelect.** CNNSelect automatically transitions between two CNN models as the target SLA increases. This increased budget allows CNNSelect to use the higher latency but more CNN model.



(a) Comparison of average end-to-end latency (error bars show 25th and 75th percentiles) and accuracy.



(b) *CNN models selected by CNNSelect.* As the SLA increases the time budget for inference increases and CNNSelect chooses more accurate models.

Fig. 13: **Comparison of CNNSelect to the greedy algorithm.** CNNSelect tracks the time left to the SLA target allowing it to meet the SLA consistently when network time allows it to at  $\geq 100$ ms, while the greedy approaches fails to do so. As such, CNNSelect improves effective accuracy *safely* as the SLA target increases. Meanwhile, by selecting a model probabilistically CNNSelect absorbs the network variation *better* by using a diverse set of CNN models.

MobileNetV1 1.0 (from left to right) to better illustrate CNNSelect performance. The mobile device is connected to our campus WiFi which has an average network time of 63ms over the course of the test.

As we can see, CNNSelect is able to gradually reduce the percentage of SLA misses as the SLA target increases. In particular, we start to observe reduction in the number of SLA violations and improved inference accuracy once the SLA target is larger than 115 ms. This is due to CNNSelect recognizing that the time budget is extremely small and utilizing a low-latency model, MobileNetV1

0.25. As the SLA increases further, the overall inference accuracy begins to improve but still exhibits some variation. The improved accuracy is due to CNNSelect identifying the increased time budget and beginning to use the more accurate InceptionV3 model while the continuing variation in accuracy is due to CNNSelect accounts for network variability and occasionally chooses MobileNetV1 0.25.

**Result:** CNNSelect is able to adapt its model selection with the goal to minimize SLA violations while improving inference accuracy, even when SLA target is set to be as low as executing a mobile-optimized model on-device.

### 5.2.2 Benefits over a greedy model selection

To examine CNNSelect’s ability to handle the trade-offs between inference response time and accuracy, we compare CNNSelect to a *greedy* algorithm that always chooses the most accurate CNN model for a given SLA. In Figure 13(a), we plot the average end-to-end inference time (left) and inference accuracy achieved by these two algorithms. This figure shows that CNNSelect consistently achieves up to 42% lower inference latency, compared to *greedy*. Moreover, CNNSelect can operate under a much more stringent SLA target ( $\sim 115$ ms) while *static greedy* continues to incur SLA violations until SLA target is more than 200ms. The key reason is because CNNSelect is able to effectively trade-off accuracy and inference time by choosing from a diverse set of models (see Figure 13(b)). Consequently, CNNSelect has an 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 ( $\sim 115$ ms), but is able to match accuracy achieved by *greedy* when SLA target is higher. Note that even though *static greedy* achieves up to 12% higher accuracy, it does so by sacrificing inference latency.

In Figure 13(b), we further analyze CNNSelect performance by looking at its model usage patterns under different SLA targets. At very low SLA target ( $< 30$ ms), CNNSelect aggressively chooses the fastest model MobileNetV1 0.25 since none of the managed models satisfy Equation (2) and (3). As the SLA target increases, CNNSelect explores more accurate but slower models than MobileNetV1 0.25. There are two key observations: (1) CNNSelect is effective in picking the more appropriate model to increase accuracy while staying safely within SLA target. For example, InceptionResNetV2 is never selected by CNNSelect because better alternatives InceptionV3 for lower SLA target and InceptionV4 for higher SLA target exist. (2) CNNSelect faithfully explores eligible models and is able to “converge” to the most accurate model when SLA target is sufficiently large.

**Result:** CNNSelect outperforms greedy with up to 43% end-to-end latency reduction, while is able to keep up with accuracy with SLA budget is larger than 250ms. The key reason is because CNNSelect is able to adapt its model selection by considering both the SLA target and network transfer time, while greedy naively selects the most accurate model.

## 6 RELATED WORK

To keep up with the increasing popularity of using deep learning within mobile applications [47], there has been a

wide range of work on providing efficient mobile deep inference. These efforts range from optimizing mobile-specific models to improving the performance of inference serving systems.

**On-device execution.** Efforts for enabling executing deep learning models directly on mobile devices fall in two broad categories: mobile-specific model optimizations and redesigning mobile deep learning frameworks.

Concretely, researchers have investigated various ways to make DNN models efficient [48]. First, post-training optimizations such as quantization uses simpler representations of weights and bins weights to improve compressibility [49, 50] allowing for reduced load time. Second, techniques such as pruning [50], removing model weights with low contributions, reducing the number of computations needed for inference as well as model sizes. Third, redesign of networks can also lead to improved inference time. An early example was the mobile-specific SqueezeNet [44] and this trend has continued with MobileNet [22] which was designed as a compact alternative to the complex InceptionV3 model [31].

To enable running models across different hardware architectures [51], researchers have redesigned deep learning frameworks [52]–[54] with the goal of providing optimized runtimes. For instance TensorFlowLite [19] and Caffe2 [13] both leverage mobile-specific optimizations that allows deep learning models to execute smoothly on mobile hardware. Recently, researchers investigated system-level optimizations for supporting multiple on-device mobile applications [55]–[57]. Our work can leverage these proposed optimizations to further improve mobile deep inference performance by judiciously selecting models at runtime.

**Remote execution.** Cloud-based solutions have demonstrated their effectiveness in handling traditional workloads [58]. Code offloading [59, 60] has been widely used to augment the performance of mobile applications with constrained hardware resources. Due to the reliance on network connectivity, code offloading is often done at runtime [59]. Determining the optimal partition of computation graphs can be solved optimally [59] with approaches such as Integer Linear Programming (ILP). However, these optimal solutions fall short because they assume access to prior performance information, such as execution time and energy [59, 60] and often incur long decision time. CNNSelect leverages the key idea of runtime computation offloading by selecting from different deep learning models, for both inference speed and accuracy gain.

Recently, various model serving platforms [23, 24, 61] provides web-based services that mobile applications can leverage. Further, researchers have started to understanding the performance and cost trade-offs when running inference services in the cloud [62]–[64]. These platforms are often designed with the key focus of managing model lifecycle from training to deployment, providing low-latency and high-throughput serving systems, and cost-effective cloud resource managements. These projects are beneficial to CNNSelect as they provide infrastructure supports for hosting a range of models. Moreover, CNNSelect complements these works with system and algorithm designs that gear towards mobile applications.

## 7 CONCLUSION

In this paper, we conducted comprehensive empirical measurements that geared towards understanding the performance implications of running deep learning models on mobile devices and in the cloud. We identified a number of key performance factors, such as mobile networks and CNN models, and demonstrated the need of cloud-based inference, especially for complex CNN models and older mobile devices. Towards mitigating the impact of the mobile network variations on cloud-based inference performance, we proposed CNNSelect that manages a set of CNN models and uses probabilistic-based to adapt its model selection to the heterogeneous mobile requirements. Our evaluations show that CNNSelect is able to transparently switch between CNN models as SLA relaxes, and that CNNSelect improves SLA attainment by 88.5% while achieving comparable accuracy compared to greedy algorithms.

## REFERENCES

- [1] A. van den Oord *et al.*, “Deep content-based music recommendation,” in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges *et al.*, Eds. Curran Associates, Inc., 2013.
- [2] Y. Wu *et al.*, “Google’s neural machine translation system: Bridging the gap between human and machine translation,” vol. arXiv:1609.08144, 2016.
- [3] “Deep Learning for Siri’s Voice: On-device Deep Mixture Density Networks for Hybrid Unit Selection Synthesis,” <https://machinelearning.apple.com/2017/08/06/siri-voices.html>, 2017.
- [4] A. Krizhevsky *et al.*, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira *et al.*, Eds. Curran Associates, Inc., 2012.
- [5] B. Zoph *et al.*, “Learning transferable architectures for scalable image recognition,” vol. arXiv:1707.07012, 2017.
- [6] A. Graves *et al.*, “Speech recognition with deep recurrent neural networks,” in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May.
- [7] I. Goodfellow *et al.*, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [8] S. Y. M. Lin, Q. Chen, “Network in network,” *International Conference on Learning Representations, 2014 (arXiv:1312.4400v3)*.
- [9] F. N. Iandola *et al.*, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size,” arXiv:1602.07360.
- [10] Y. LeCun *et al.*, “Gradient-based learning applied to document recognition,” in *Intelligent Signal Processing*. IEEE Press, 2001.
- [11] S. CS231n, “Convolutional neural networks for visual recognition.” <http://cs231n.github.io/convolutional-networks/>, 2017.
- [12] Y. Jia *et al.*, “Caffe: Convolutional architecture for fast feature embedding,” in *Proceedings of the 22Nd ACM International Conference on Multimedia*, ser. MM ’14. New York, NY, USA: ACM, [Online]. Available: <http://doi.acm.org/10.1145/2647868.2654889>
- [13] “Delivering real-time AI in the palm of your hand,” <https://code.facebook.com/posts/196146247499076/delivering-real-time-ai-in-the-palm-of-your-hand/>, 2016.
- [14] R. Collobert *et al.*, “Torch: a modular machine learning software library,” Idiap, Tech. Rep., 2002.
- [15] M. Abadi *et al.*, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015, software available from tensorflow.org.
- [16] S. Bahrampour *et al.*, “Comparative study of deep learning software frameworks,” arXiv:1511.06435.
- [17] B. Caffe, “Model zoo,” <https://github.com/BVLC/caffe/wiki/Model-Zoo>, Accessed on 2017.
- [18] T. Guo, “Cloud-based or on-device: An empirical study of mobile deep inference,” in *Proceedings of 2018 IEEE International Conference on Cloud Engineering*, ser. IC2E ’18, 2018.
- [19] “Introduction to TensorFlow Lite,” <https://www.tensorflow.org/mobile/tflite/>, 2018.
- [20] “How Google Translate squeezes deep learning onto a phone,” <https://research.googleblog.com/2015/07/how-google-translate-squeezes-deep.html>, 2015.

- [21] "Zero-Shot Translation with Googles Multilingual Neural Machine Translation System," <https://research.googleblog.com/2016/11/zero-shot-translation-with-googles.html>, 2016.
- [22] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," vol. arXiv:1704.04861.
- [23] D. Crankshaw *et al.*, "Clipper: A low-latency online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017.
- [24] C. Olston *et al.*, "Tensorflow-serving: Flexible, high-performance ML serving," 2017.
- [25] P. Gao *et al.*, "Low latency rnn inference with cellular batching," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. New York, NY, USA: ACM, 2018.
- [26] S. S. Ogden *et al.*, "MODI: Mobile deep inference made efficient by edge computing," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. Boston, MA: USENIX Association, 2018.
- [27] M. Satyanarayanan *et al.*, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, Oct. 2009.
- [28] S. S. Latifi Oskouei *et al.*, "Cnndroid: Gpu-accelerated execution of trained deep convolutional neural networks on android," in *Proceedings of the 2016 ACM on Multimedia Conference*, ser. MM '16.
- [29] "TensorFlow Mobile Overview," <https://www.tensorflow.org/mobile/>, 2018.
- [30] BVLC Caffe, "Models accuracy on imagenet 2012 val," <https://github.com/BVLC/caffe/wiki/Models-accuracy-on-ImageNet-2012-val>.
- [31] C. Szegedy *et al.*, "Rethinking the inception architecture for computer vision," vol. arXiv:1512.00567, 2015.
- [32] M. Z. Andrew G. Howard, "Mobilenets: Open-source models for efficient on-device vision," <https://research.googleblog.com/2017/06/mobilenets-open-source-models-for.html>, June 14, 2017.
- [33] Q. T. Inc., "Trepn power profiler," <https://developer.qualcomm.com/software/trepn-power-profiler>, Accessed on 2017.
- [34] J. NielSen., "Website response times," <https://www.nngroup.com/articles/website-response-times/>, June 21, 2010.
- [35] appdynamics, "16 metrics to ensure mobile app success," Accessed on 2017.
- [36] "Pixel Visual Core: image processing and machine learning on Pixel 2," <https://www.blog.google/products/pixel/pixel-visual-core-image-processing-and-machine-learning-pixel-2/>, 2017.
- [37] nvidia, "Gpus are driving energy efficiency across the computing industry, from phones to super computers." <http://www.nvidia.com/object/gcr-energy-efficiency.html>, Accessed on 2017.
- [38] C. Wang *et al.*, "Using burstable instances in the public cloud: Why, when and how?" *Proc. ACM Meas. Anal. Comput. Syst*.
- [39] P. Sharma *et al.*, "Containers and virtual machines at scale: A comparative study," in *Proceedings of the 17th International Middleware Conference*, ser. Middleware '16. New York, NY, USA: ACM, 2016.
- [40] F. Yan *et al.*, "Efficient deep neural network serving: Fast and furious," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, March 2018.
- [41] P. Bodik *et al.*, "Characterizing, modeling, and generating workload spikes for stateful services," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM.
- [42] M. Yogeswaran *et al.*, "Reinforcement learning: exploration-exploitation dilemma in multi-agent foraging task," *OPSEARCH*, vol. 49, no. 3, Sep 2012.
- [43] "List of hosted models," <https://www.tensorflow.org/lite/models>.
- [44] F. N. Iandola *et al.*, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," vol. arXiv:1602.07360, 2016.
- [45] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," vol. arXiv:1409.1556, 2014.
- [46] K. He *et al.*, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
- [47] M. Xu *et al.*, "A first look at deep learning apps on smartphones," in *The World Wide Web Conference*, ser. WWW '19, 2019, pp. 2125–2136.
- [48] I. of Electrical *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, 2017-12.
- [49] "NVIDIA TensorRT," <https://developer.nvidia.com/tensorrt>.
- [50] S. Han *et al.*, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," vol. arXiv:1510.00149.
- [51] Q. Cao *et al.*, "Mobirnn: Efficient recurrent neural network execution on mobile gpu," in *Proceedings of the 1st International Workshop on Deep Learning for Mobile Systems and Applications*, ser. EMDL '17.
- [52] N. D. Lane *et al.*, "Deepex: A software accelerator for low-power deep learning inference on mobile devices," in *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*.
- [53] L. N. Huynh *et al.*, "Deepsense: A gpu-based deep convolutional neural network framework on commodity mobile devices," in *Proceedings of the 2016 Workshop on Wearable Systems and Applications*, ser. WearSys '16.
- [54] S. Han *et al.*, "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '16, 2016.
- [55] B. Fang *et al.*, "Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '18, 2018, pp. 115–127.
- [56] M. Xu *et al.*, "Deepcache: Principled cache for mobile deep vision," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '18.
- [57] Q. Cao *et al.*, "Deqa: On-device question answering," in *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '19. New York, NY, USA: ACM, 2019, pp. 27–40. [Online]. Available: <http://doi.acm.org/10.1145/3307334.3326071>
- [58] T. Guo *et al.*, "Latency-aware virtual desktops optimization in distributed clouds," *Multimedia Systems*.
- [59] E. Cuervo *et al.*, "Maui: Making smartphones last longer with code offload," in *ACM MobiSys 2010*. Association for Computing Machinery, Inc., June 2010.
- [60] X. Gu *et al.*, "Adaptive offloading inference for delivering applications in pervasive computing environments," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications*, ser. PERCOM '03. Washington, DC, USA: IEEE Computer Society, 2003.
- [61] T. Chen *et al.*, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," vol. arXiv:1512.01274, 2015.
- [62] A. Samanta *et al.*, "No DNN left behind: Improving inference in the cloud with multi-tenancy," vol. arXiv:1901.06887, 2019.
- [63] C. Zhang *et al.*, "Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pp. 1049–1062.
- [64] V. Ishakian *et al.*, "Serving deep learning models in a serverless platform," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, April 2018, pp. 257–262.



**Samuel S. Ogden** is a Ph.D. student at Worcester Polytechnic Institute. He received his M.S. in Computer Science from the University of Vermont in 2013 and his B.S. in Pure Mathematics and Electrical Engineering in 2010. His research interests include cloud and mobile computing and deep learning systems. Contact him at [ssogden@wpi.edu](mailto:ssogden@wpi.edu).



**Tian Guo** is an Assistant Professor in the Computer Science Department at Worcester Polytechnic Institute. She received her Ph.D. and M.S. in Computer Science from the University of Massachusetts Amherst in 2013 and 2016, respectively, and her B.E. in Software Engineering from Nanjing University in 2010. Her research interests include distributed systems, cloud computing and mobile computing. Contact her at [tian@wpi.edu](mailto:tian@wpi.edu).