Mobile-Aware Deep Inference

Fine- and Coarse-Grained Approaches

Samuel S. Ogden

CSU: Monterey Bay

March 17, 2022

Big Idea

Mobile-oriented deep-learning inference needs to dynamically adapt to its environment and workload

How do we approach this?

1. Consider the end-to-end workflow of deep learning inference (ICPE'21, HotEdge'18)
2. React to environmental changes (IC2E'20)
3. Allocate resources for large, variable workload (ACSOS'21)
4. Improve execution at a low level (PERFORMANCE'20, DIDL'20)

Peer-reviewed Publications

1. Samuel S. Ogden, Guin R. Gilman, Robert J. Walls, Tian Guo, (2021), "Many Models at the Edge: Scaling Deep Inference via Model-Level Caching" (10 pages), *2^{nd} IEEE International Conference on Autonomic Computing and Self-Organizing Systems* (**ACSOS'21**) (Acceptance Rate 23%)
2. Samuel S. Ogden, Xiangnan Kong, Tian Guo, (2021), "PieSlicer: Dynamically Improving Response Time for Cloud-based CNN Inference" (8 pages), 12^{th} *ACM/SPEC International Conference on Performance Engineering* (**ICPE'21**) (Acceptance Rate 29%)
3. Guin R. Gilman, Samuel S. Ogden, Tian Guo, Robert J. Walls, (2020), "Demystifying the Placement Policies of the NVIDIA GPU Thread Block Scheduler for Concurrent Kernels" (7 pages), *38^{th} International Symposium on Computer Performance, Modeling, Measurements and Evaluation* (**PERFORMANCE'20**) (Acceptance Rate 23.5%)
4. Samuel S. Ogden, Tian Guo, (2020), "MDInference: Balancing Inference Accuracy and Latency for Mobile Applications" (11 pages), *IEEE International Conference on Cloud Engineering* (Invited) (**IC2E'20**) (Acceptance rate 51%)
5. Guin R. Gilman, Samuel S. Ogden, Robert J. Walls, Tian Guo, (2019), "Challenges and Opportunities of DNN Model Execution Caching", (5 pages) *MiddleWare DIDL Workshop* (**DIDL'19**)
6. Tian Guo, Robert J. Walls, Samuel S. Ogden, (2019), "EdgeServe: Efficient Deep Learning Model Caching at the Edge" (3 pages), *4^{th} ACM/IEEE Symposium on Edge Computing* (**SEC'19**)
7. Samuel S. Ogden, Tian Guo, (2018), "MODI: Mobile Deep Inference Made Efficient by Edge Computing" (7 pages), *USENIX Annual Technical Conference HotEdge Workshop 2018* (**HotEdge'18**)

Outline

Background

Background

What is deep learning?

Deep-learning are large and complex artificial neural networks used to interpret inputs

- A common example is CNNs, which are often used for image analysis

Two main phases to deep learning models

- **Training**: We use large amounts of data (often TBs of data) to train models
  - Training a single model can emit as much $C0_2$ as six cars
- **Inference**: We take novel input data and use the model to make a prediction

Background

Where is it used?

Many mobile applications use deep learning

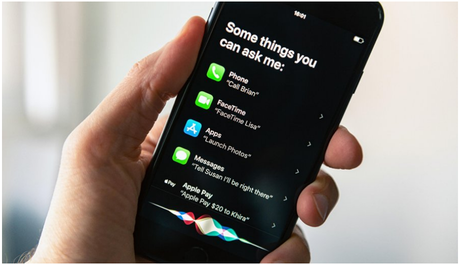
- **Snapchat uses deep learning for face-aware filters**

Background

Where is it used?

Many mobile applications use deep learning

- Snapchat uses deep learning for face-aware filters
- **Siri and Alexa perform speech-to-text and question answering**

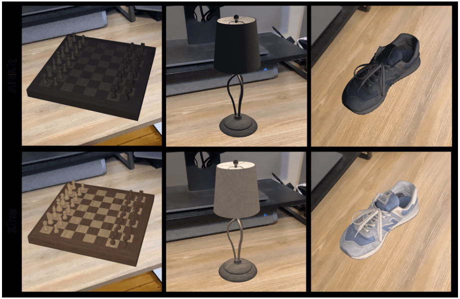


Background

Where is it used?

Many mobile applications use deep learning

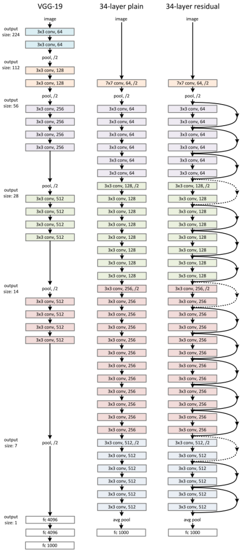
- Snapchat uses deep learning for face-aware filters
- Siri and Alexa perform speech-to-text and question answering
- **Augment reality uses this for realistic shadowing**



Background

Challenges

Deep Learning models are big and complicated

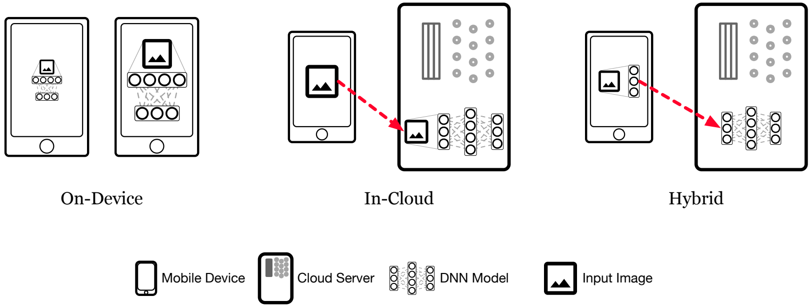


Background
Challenges

Mobile devices prioritize battery over computational power

Background

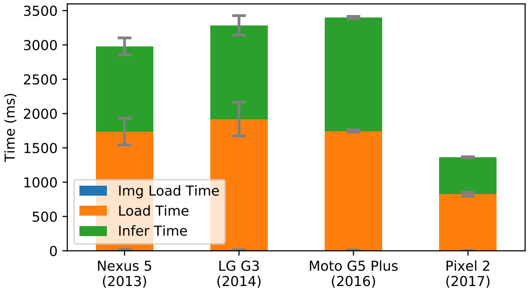
Mobile deep inference options

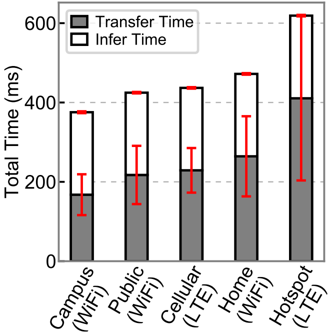


Three main ways to enable deep learning inference on mobile devices

Background

Latency comparison

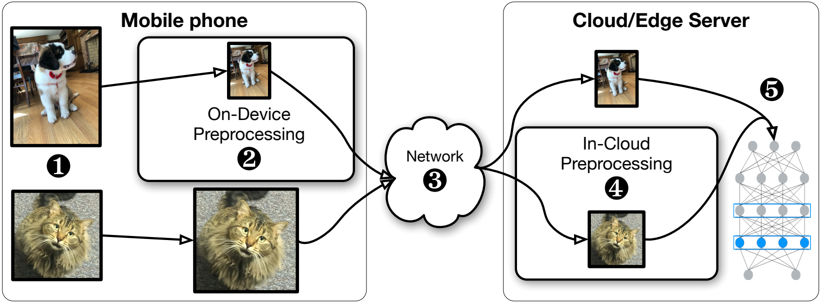




Executing on-devicedevice can be much slower than executing remotely

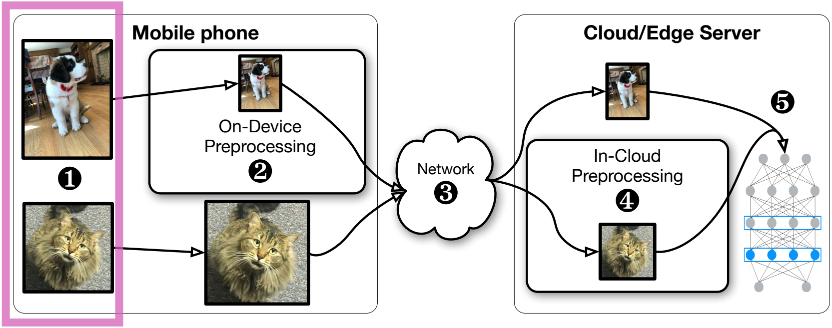
Background

Mobile Inference Request Workflow



Background

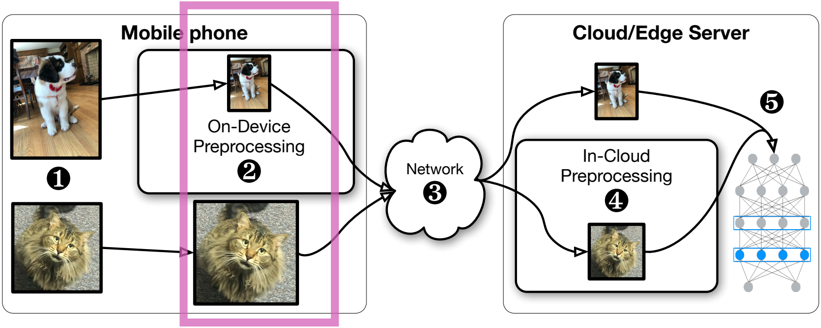
Mobile Inference Request Workflow



1. **Input Capture**

Background

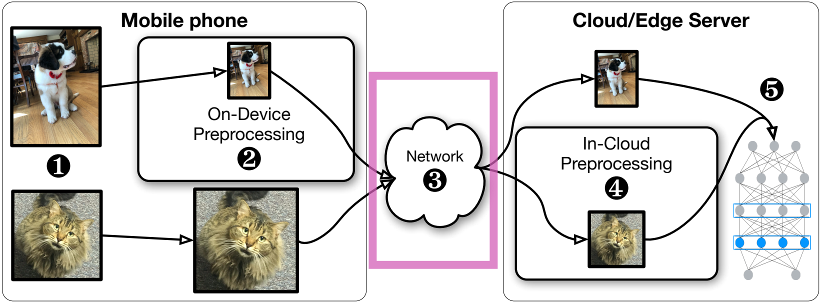
Mobile Inference Request Workflow



1. Input Capture
2. **On-device preprocessing**

Background

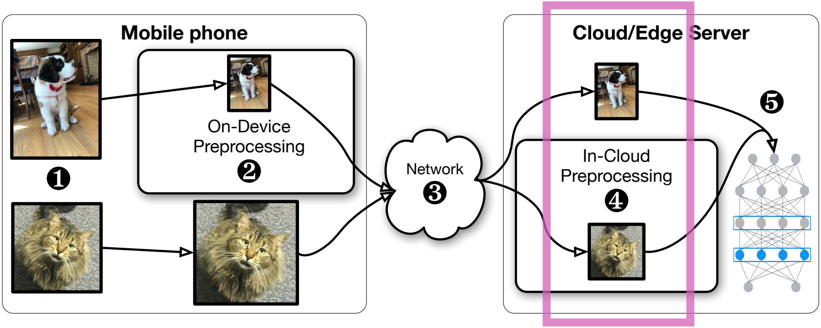
Mobile Inference Request Workflow



1. Input Capture
2. On-device preprocessing
3. **Network Transfer**

Background

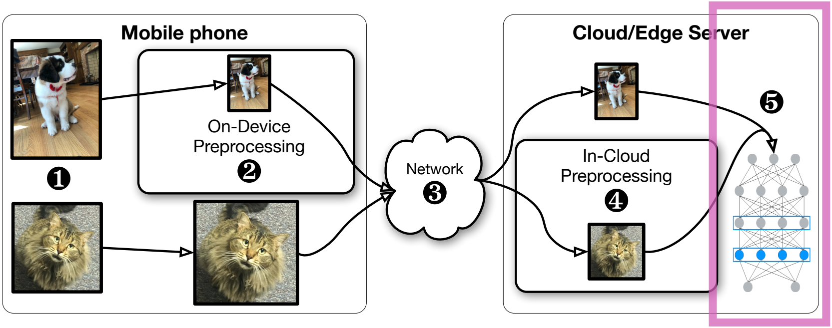
Mobile Inference Request Workflow



1. Input Capture
2. On-device preprocessing
3. Network Transfer
4. **In-cloud preprocessing**

Background

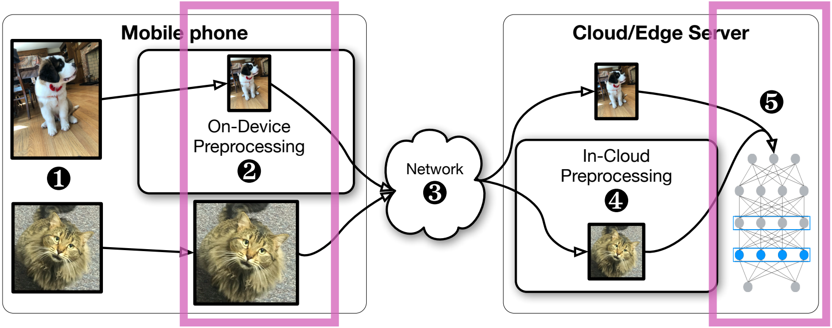
Mobile Inference Request Workflow



1. Input Capture
2. On-device preprocessing
3. Network Transfer
4. In-cloud preprocessing
5. **Execution**

Background

Mobile Inference Request Workflow – Decisions points



1. Input Capture
2. **On-device preprocessing**
3. Network Transfer
4. In-cloud preprocessing
5. **Execution**

Big Idea

Mobile-oriented deep-learning inference needs to dynamically adapt to its environment and workload

On-device Preprocessing Decisions

On-device Preprocessing

PieSlicer: ICPE'21

PieSlicer: Dynamically Improving Response Time for Cloud-based CNN Inference

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

Xiangnan Kong
xkong@wpi.edu
Worcester Polytechnic Institute

Tian Guo
tian@wpi.edu
Worcester Polytechnic Institute

ABSTRACT

Executing deep-learning inference on cloud servers enables the usage of high complexity models for mobile devices with limited resources. However, *pre-execution time*—the time it takes to prepare and transfer data to the cloud—is variable and can take orders of magnitude longer to complete than inference execution itself. This pre-execution time can be reduced by dynamically deciding the order of two essential steps, *preprocessing* and *data transfer*, to better take advantage of on-device resources and network conditions. In this work we present PieSlicer, a system for making dynamic preprocessing decisions to improve cloud inference performance using linear regression models. PieSlicer then leverages these models to select the appropriate preprocessing location. We show that for image classification applications PieSlicer reduces median and 99^{th} percentile pre-execution time by up to 50.2ms and 217.2ms respectively when compared to static preprocessing methods.

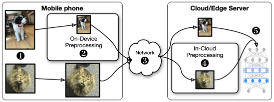
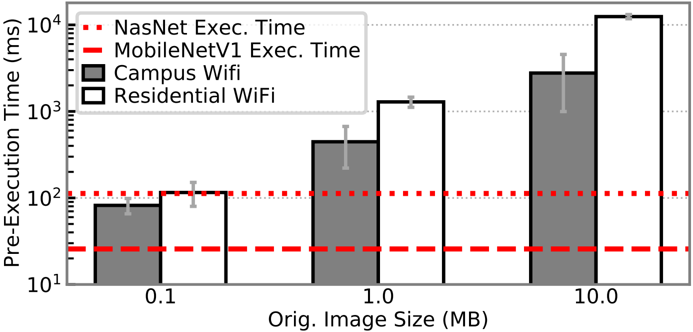


Figure 1: *Cloud-based Deep Inference Workflow.* *In general, there are five steps:* input capture ❶, on-device pre-processing ❷, network transfer ❸, in-cloud preprocessing ❹, *and* deep learning model execution ❺. *Steps* ❷-❹ *comprise pre-execution and present opportunities to make dynamic decisions to reduce latency.*

In this work, we characterize pre-execution time and investigate ways to reduce it. Our first goal is to identify and understand factors

On-device Preprocessing

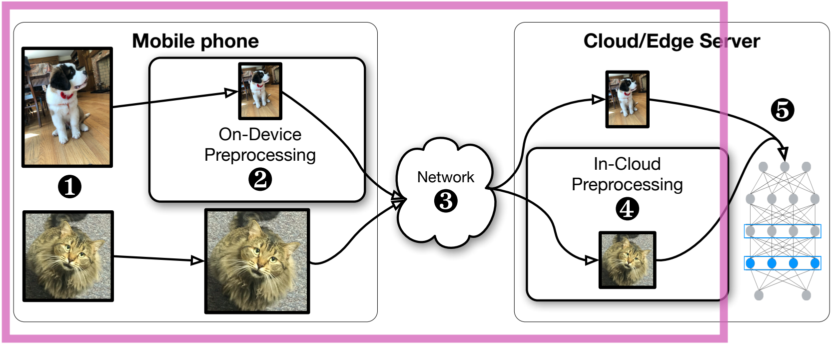
Pre-execution Latency by Size



As file size increases pre-execution latency surpasses execution latency

On-device Preprocessing

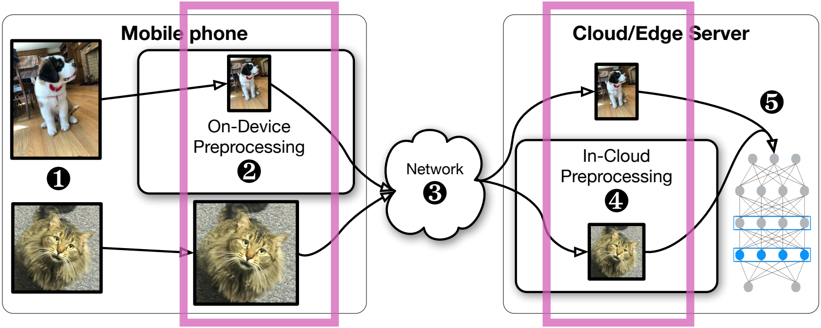
Mobile Inference Request Workflow



Pre-execution time is all the time prior to execution.

On-device Preprocessing

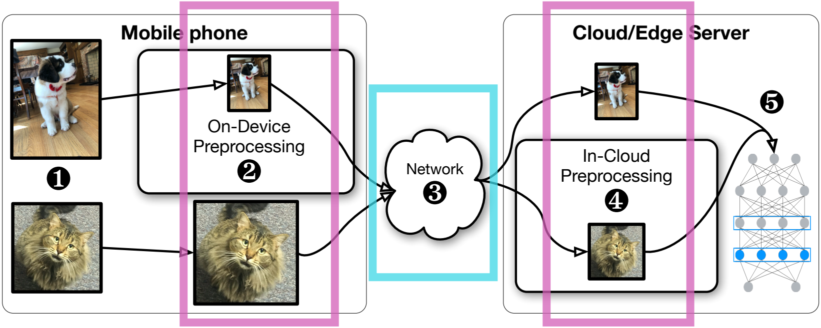
Mobile Inference Request Workflow



We can select preprocessing location

On-device Preprocessing

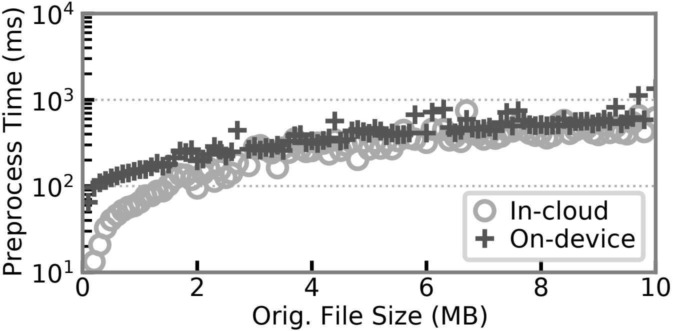
Mobile Inference Request Workflow



We can select preprocessing location
and change how much data we sent across the network

On-device Preprocessing

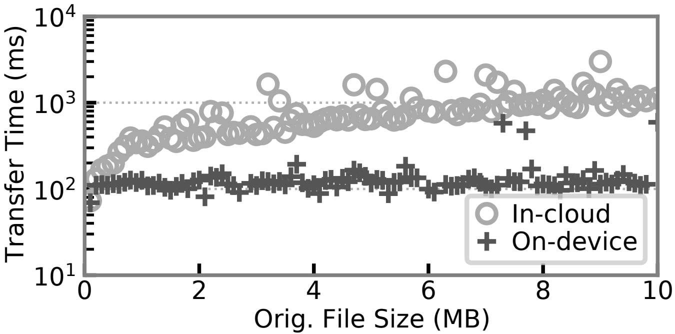
Preprocessing Latency Comparison



Preprocessing in-cloud is faster than on-device at all measured sizes

On-device Preprocessing

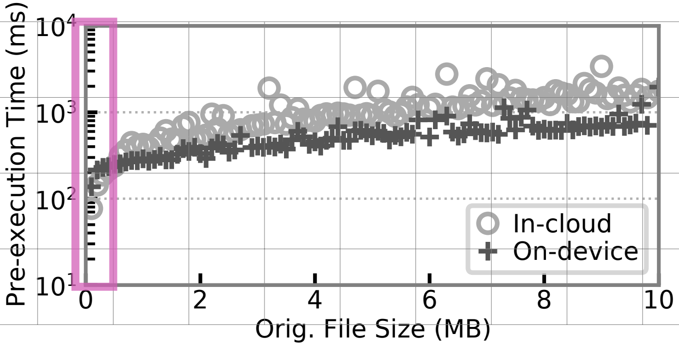
Network Latency Comparison



Transferring smaller, already preprocessed images is almost always better

On-device Preprocessing

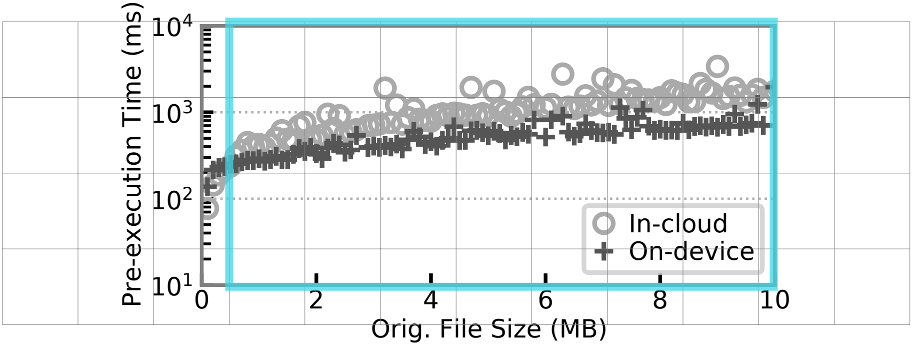
Pre-execution Latency Comparison



There's a trade-off between preprocessing on-device and in-cloud to be made

On-device Preprocessing

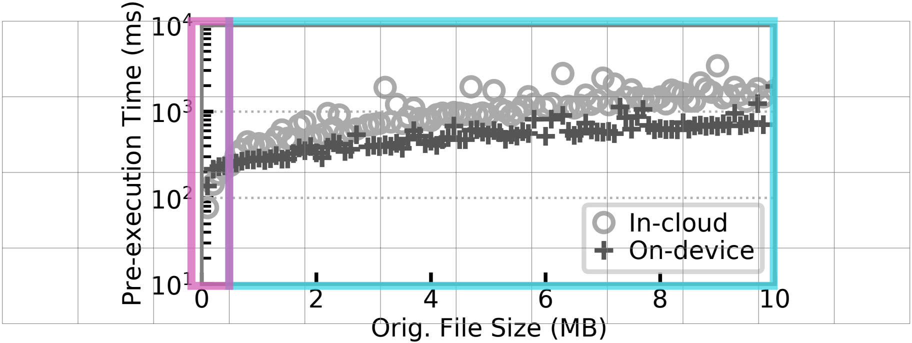
Pre-execution Latency Comparison



There's a trade-off between preprocessing on-device and in-cloud to be made

On-device Preprocessing

Pre-execution Latency Comparison



There's a trade-off between preprocessing on-device and in-cloud to be made

On-device Preprocessing

Core idea

If we change the preprocessing location we can change our overall latency to reduce latency

On-device Preprocessing

Core choice

Slow on-device resizing & small transfer

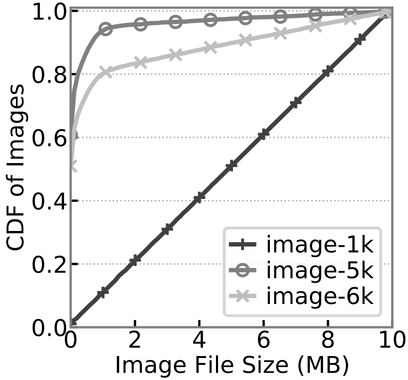
vs.

Big transfer & fast in-cloud resizing

On-device Preprocessing

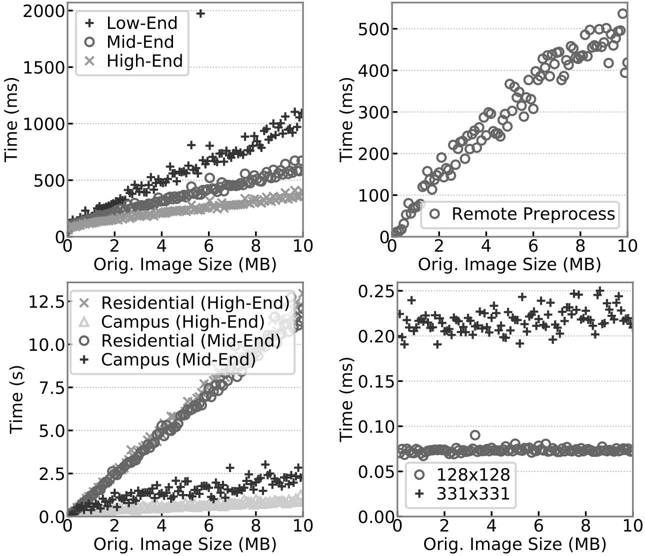
Measurements Overview

- Three different phones: Nexus 5, MotoX4, Pixel2
- Two networks: University, Residential
- Two datasets: *image-1k*, *image-5k*
- Measured data:
    - Time (*Target*)
    - Input Size in MB
    - Transfer Size in MB
    - Resolution in Megapixels
    - Input height
    - Input width



On-device Preprocessing

Measurements with *image-1k*



On-device Preprocessing

Model Types

Models to try

- Linear
- K-Nearest Neighbors (KNN)
- Random Forest (RF)
- Lasso
- Support Vector Regression (SVR)

Model Combination Factors

- Phones (3)
- Networks (2)
- Dataset (2)
- *Total variations: 36*

On-device Preprocessing

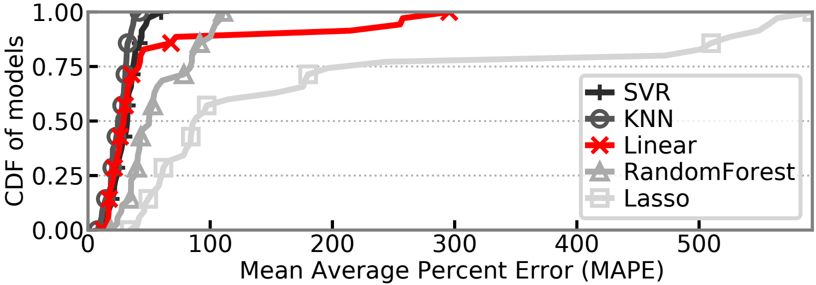
Modeling goals

Goals

- Accurate
- Fast to use
- Fast to train

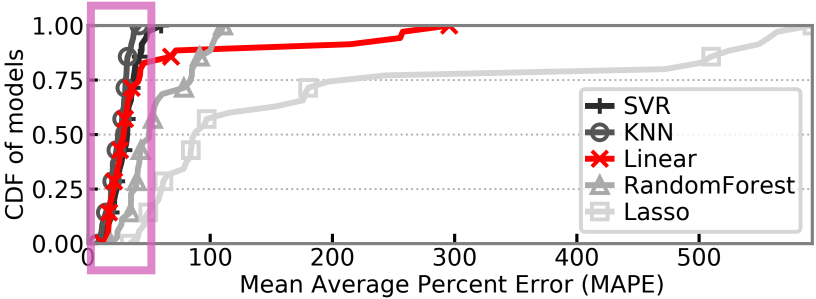
On-device Preprocessing

Modeling performance



On-device Preprocessing

Modeling performance



On-device Preprocessing

Modeling Options

	KNN	SVR	Linear
Accuracy			
Time to Use			
Time to Train			

On-device Preprocessing

Modeling Options

	KNN	SVR	Linear
Accuracy	Best	Very Good	Good
Time to Use			
Time to Train			

On-device Preprocessing

Modeling Options

	KNN	**SVR**	**Linear**
Accuracy	Best	Very Good	Good
Time to Use	Slow	Fast	Fast
Time to Train			

On-device Preprocessing
Modeling Options

	KNN	**SVR**	**Linear**
Accuracy	Best	Very Good	Good
Time to Use	Slow	Fast	Fast
Time to Train	N/A	Slow	Fast

On-device Preprocessing

Modeling Options: Linear

	KNN	SVR	Linear
Accuracy	Best	Very Good	Good
Time to Use	Slow	Fast	Fast
Time to Train	N/A	Slow	Fast

On-device Preprocessing
Experimental Summary

Baselines

- *Static local*
- *Static remote*
- *Static minimum* (empirical optimal)

3 Mobile Devices

- Nexus 5
- MotoX4
- Pixel 2

2 WiFi Networks

- Residential (slow)
- University (fast)

1 Datasets

- *image-1k*

On-device Preprocessing

Decision Accuracy

	Residential	University
Low-End	0.987	0.980
Mid-End	0.988	0.987
High-End	0.990	0.983

On-device Preprocessing

Latency Comparisons Summary

		Residential			University		
Device	**Algorithm**	50^{th}	95^{th}	99^{th}	50^{th}	95^{th}	99^{th}
Low-End	*Optimal*	713.2ms	1231.0ms	1876.6ms	707.2ms	1215.7ms	1984.5ms
	In-Cloud	922.6%	1094.7%	1524.9%	274.2%	288.8%	316.9%
	On-Device	100.1%	100.0%	100.0%	100.5%	101.0%	100.0%
	PieSlicer	95.0%	100.3%	113.8%	93.4%	94.5%	94.1%
Mid-End	*Optimal*	582.4ms	875.6ms	1316.1ms	502.4ms	749.7ms	1090.2ms
	In-Cloud	1082.3%	1353.0%	1003.1%	275.4%	599.5%	502.6%
	On-Device	100.1%	100.0%	103.1%	100.3%	100.0%	100.0%
	PieSlicer	97.3%	96.7%	83.5%	97.6%	96.6%	94.1%
High-End	*Optimal*	448.7ms	690.0ms	979.8ms	384.2ms	666.7ms	951.7ms
	In-Cloud	1457.6%	1818.5%	1454.4%	234.9%	238.8%	223.9%
	On-Device	100.1%	100.0%	100.0%	100.2%	102.1%	100.0%
	PieSlicer	98.9%	96.3%	104.7%	98.1%	98.7%	105.7%

On-device Preprocessing

Other Benefits: Bandwidth reduction

	All	**Residential**	**University**
All	4.47%	1.88%	4.10%
Low-End	5.44%	1.91%	4.93%
Mid-End	4.74%	1.86%	4.79%
High-End	7.26%	1.86%	7.33%

On-device Preprocessing Decisions

What did we see?

1. By looking at the overall workflow we can find better potential optimizations
2. Using simple but accurate models is often sufficient for our cases
   - Simple models let us be really quite accurate!
3. We can save a significant amount of time and latency!

On-device Preprocessing Decisions

Remaining questions

1. How can we make use of this extra time? (next section)
2. Are there cases when this doesn't work as well? (current work direction)

In-cloud Execution Adjustment

In-cloud Execution Adjustment

MDInference: IC2E'20

2020 IEEE International Conference on Cloud Engineering (IC2E)

MDInference: Balancing Inference Accuracy and Latency for Mobile Applications

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

Tian Guo
Worcester Polytechnic Institute
tian@wpi.edu

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

In this work we introduce a holistic approach to designing mobile deep inference frameworks. We first identify the key goals of *accuracy* and *latency* for mobile deep inference and the conditions that must be met to achieve them. We demonstrate our holistic approach through the design of a hypothetical framework called MDInference. This framework leverages

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

In this paper we argue the need for mobile-oriented infer-

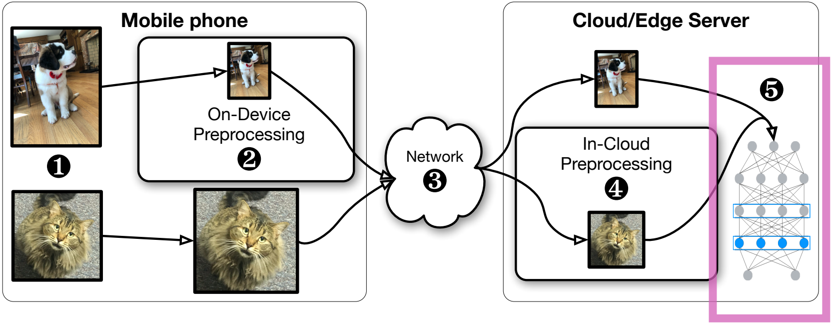
In-cloud Execution Adjustment

Core idea

Adjust execution based on the request

In-cloud Execution Adjustment

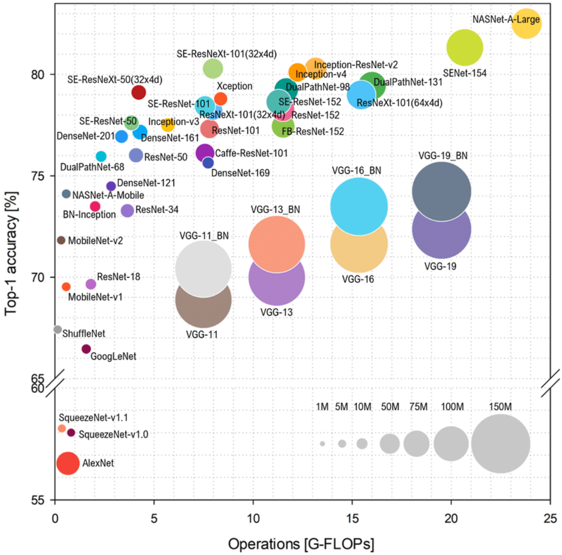
Mobile Inference Request Workflow



Focus on better utilizing cloud execution time

In-cloud Execution Adjustment

Trade-offs between accuracy and latency



Constraints

- Requests are submitted over an unpredictable network
- Needs to enforce a Service Level Agreement (SLA) of maximum response latency latency
  - Measured from user pressing "go!" to the response being back at the device

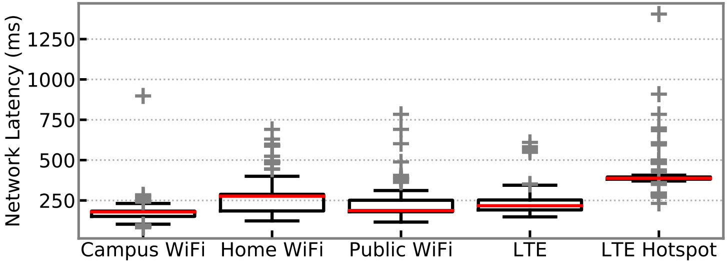
In-cloud Execution Adjustment

Key insight

Chosing different models allows us to adjust execution latency to compensate for the network

In-cloud Execution Adjustment

Network variation



Network variation can be quite large for networks

In-cloud Execution Adjustment

Proposed Solution

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

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

- $\boldsymbol{A}(m)$: accuracy of model m
- T_{budget} : time budget, calculated as $T_{SLA} - 2 \times T_{network}$
- $\mu(m)$: average of execution latency for model m
- $\sigma(m)$: standard deviation of execution latency for model m

In-cloud Execution Adjustment

Proposed Solution

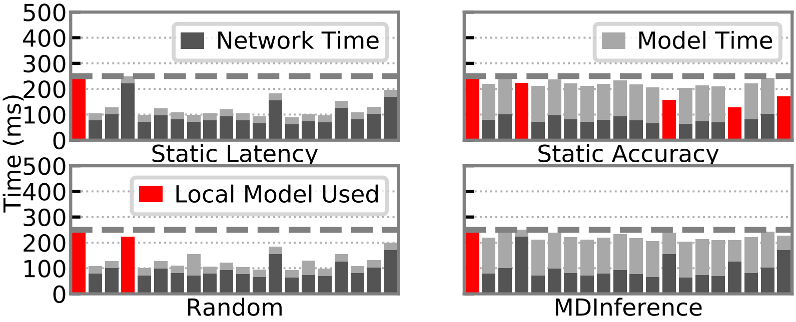
$$\underset{j}{\text{maximize}} \quad \boldsymbol{A}(m) \tag{1}$$

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

For each request, calculate a time budget and pick the most accurate model that will execute within that budget

In-cloud Execution Adjustment

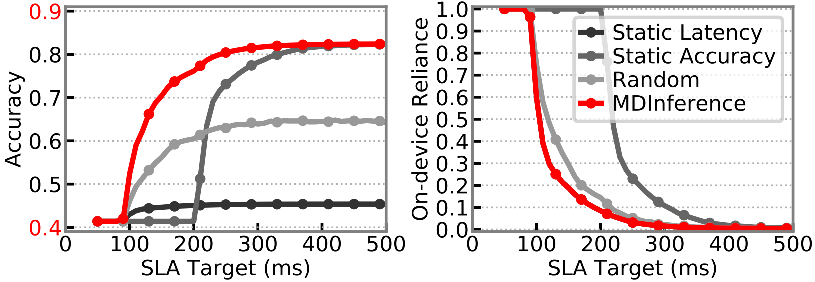
What does this look like?



Given a reasonable SLA, we can match an SLA closely while using more complex models

In-cloud Execution Adjustment

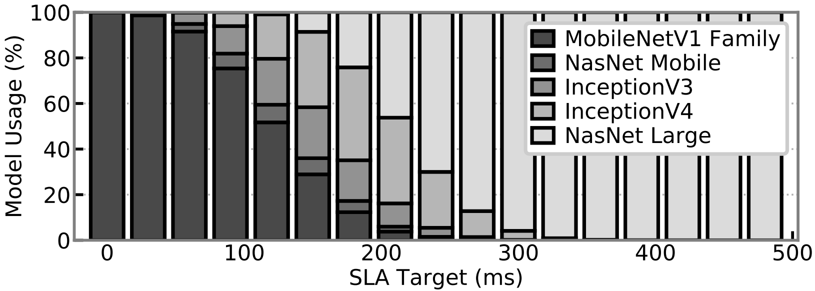
What is a reasonable SLA?



MDInference quickly stops using on-device backup and improves accuracy

In-cloud Execution Adjustment

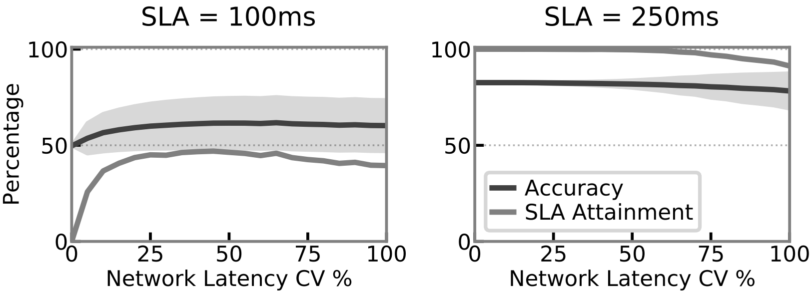
How do we increase accuracy?



As the SLA increases the time budget allows MDInference to use more complex models to improve average accuracy

In-cloud Execution Adjustment

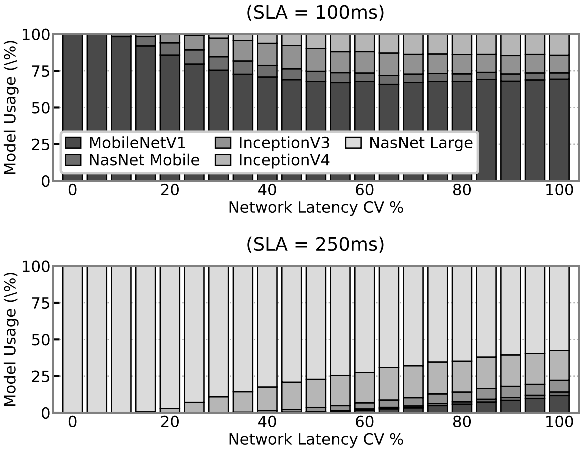
What is the impact of noise?



As noise (e.g. Coefficient of Variation) increases, MDInference takes advantage to increase accuracy, and maintains SLA attainment

In-cloud Execution Adjustment

How do we leverage noise?



As the noise increases we can opportunistically use more accurate models, or can compensate with fast models

In-cloud Execution Adjustment

What did we see?

What we saw

- Many tasks have a range of available models with different latency-accuracy trade-offs
  - There's a lot of active work on model optimizations, like quantization
- By selecting an appropriate model we can maintain SLAs and yet use higher accuracy models
- By always keeping a minimal backup model running on-device we can use this in the rare cases that we can't respond in time

In-cloud Execution Adjustment

What did we see?

What we could do better

- Having all of these models loaded is a large use of resources (addressed briefly next)
- What if we knew that an inference would fail to complete? (current work)

Resource Management

In-cloud Execution Adjustment

CremeBrulee: ACSOS'21

Many Models at the Edge: Scaling Deep Inference via Model-Level Caching

Samuel S. Ogden , Guin R. Gilman , Robert J. Walls , and Tian Guo

Computer Science Department, Worcester Polytechnic Institute
{ssogden,grgilman,rjwalls,tian}@wpi.edu

Abstract

Deep learning (DL) models are rapidly expanding in popularity in large part due to rapid innovations in model accuracy, as well as companies' enthusiasm in integrating deep learning into the existing application logic. This trend will inevitably lead to a deployment scenario, akin to the content delivery network for web objects, where many deep learning models—each with different popularity—run on a shared edge with limited resources. In this paper, we set out to answer the key question of *how to manage many deep learning models at the edge effectively*. Via an empirical study based on profiling more than twenty deep learning models and extrapolating from an open-source Microsoft Azure workload trace, we pinpoint a promising avenue of leveraging cheaper CPUs, rather than commonly promoted accelerators, for edge-based deep inference serving.

Based on our empirical insights, we formulate the DL model

managing static, and more recently dynamic, content in CDNs, the complexity of deep learning models, and the requirements of using them make model serving complex.

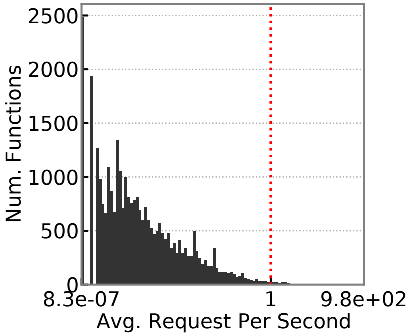
Deep learning models are large in size, over 4GB in some cases [42], with complex execution graphs that need to be constructed upon model load. As such, naive memory management may encounter difficulties handling these models, experiencing unexpected latency variations, and not fully exploiting the characteristics of models. The scale of the workload can further compound the memory management complexity. As deep learning models proliferate, they are being used in myriad applications that were traditionally served by central servers or, more recently, run in serverless platforms. Extrapolating from a serverless trace [35], we expect deep learning models will see not only a huge number of requests but also a wide range of popularity, with some models being requested many orders of magnitude more often than others.

Deep Learning models must be managed like objects in a cache to improve resource utilization

Resource Management

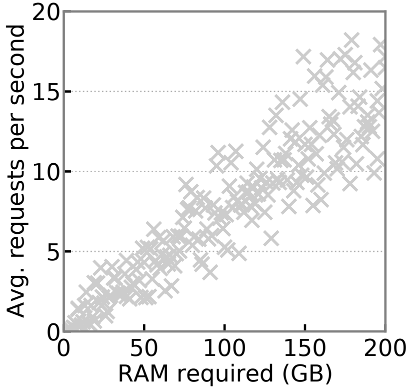
Workload extrapolation

Extrapolating from existing workloads, most deep learning models will be *rarely* used



Resource Management

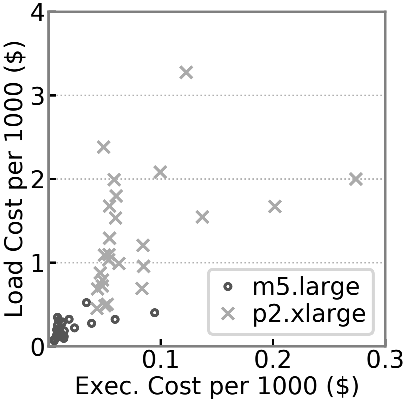
Workload extrapolation

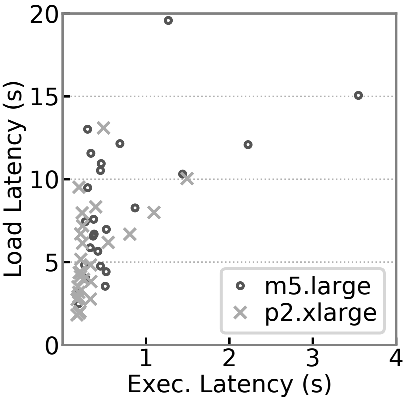


Keeping models in memory is more resource intensive than executing models

Resource Management

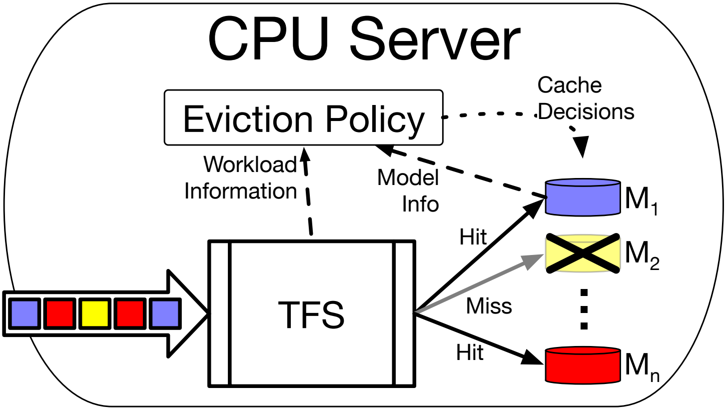
Contribution #1: CPUs are cost efficient





Resource Management

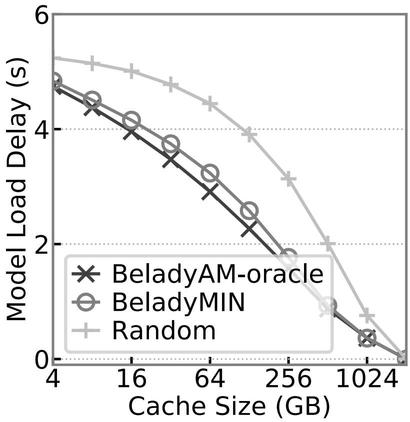
Contribution #2: Caching of deep learning models



Resource Management

Contribution #1: CPUs are cost efficient

By considering characteristics of deep learning models we can decrease added cost due to caching misses



Ongoing Work: On-device execution decisions

On-device execution

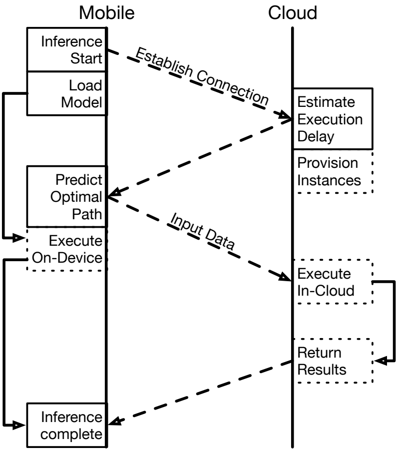
Core Idea

On-device execution can offload work from caching server

Resource Management

Contribution #1: CPUs are cost efficient

Communication in parallel with execution on-device allows for more efficient resource utilization both on-device and in-cloud



Conclusions

Conclusions

- Approaching deep learning serving from a mobile-oriented approach can greatly reduce latency and variability for mobile devices
- Close analysis of workflows can help identify large time savings
- Making inference serving aware of end-to-end behavior allows us to opportunistically improve serving quality
- Deep learning workloads need to be approached in new ways to help improve resource utilization

Future Directions

- Not all work needs to be done by servers, so move some work off-device
    - Improved processing power and network performance will continue to shift the balance of on-device and in-cloud performance
- Improved awareness of inter-model interactions
    - Interconnected workloads introduce dependencies and resource contention