Daydream: Accurately Estimating the Efficacy of Optimizations for DNN Training

Hongyu Zhu[1,2], Amar Phanishayee[3], Gennady Pekhimenko[1,2]

1 UNIVERSITY OF TORONTO

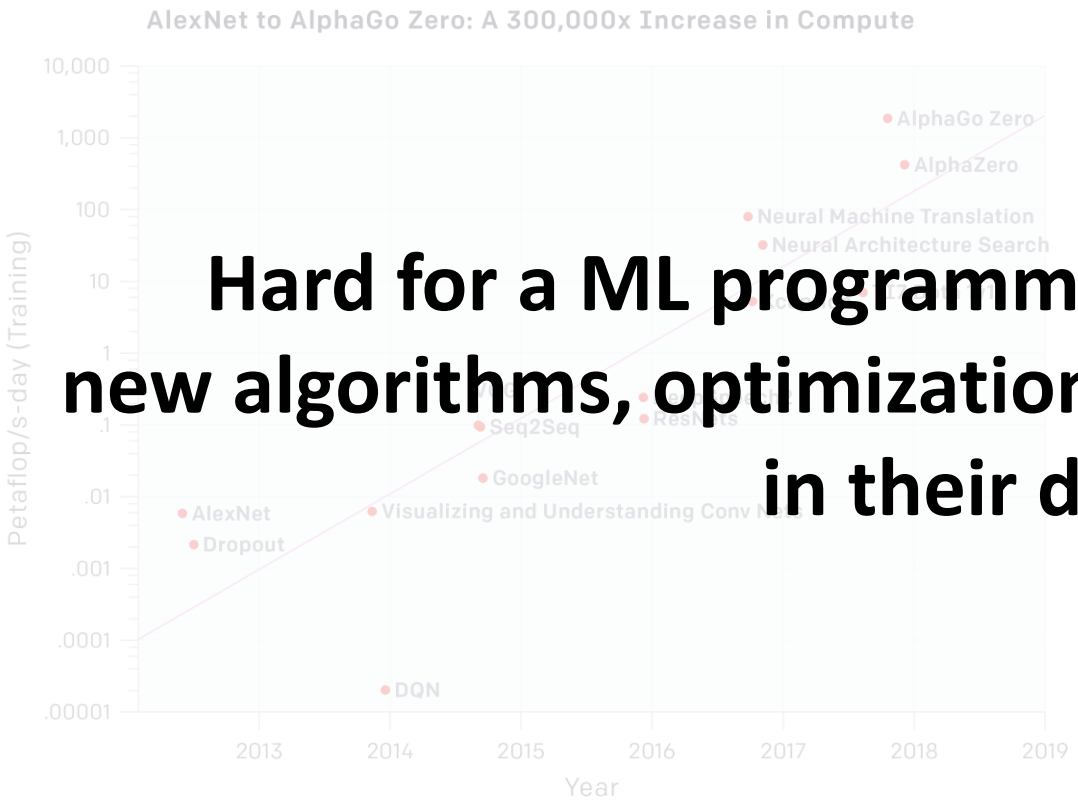
2 VECTOR INSTITUTE

3 Microsoft Research

Executive Summary

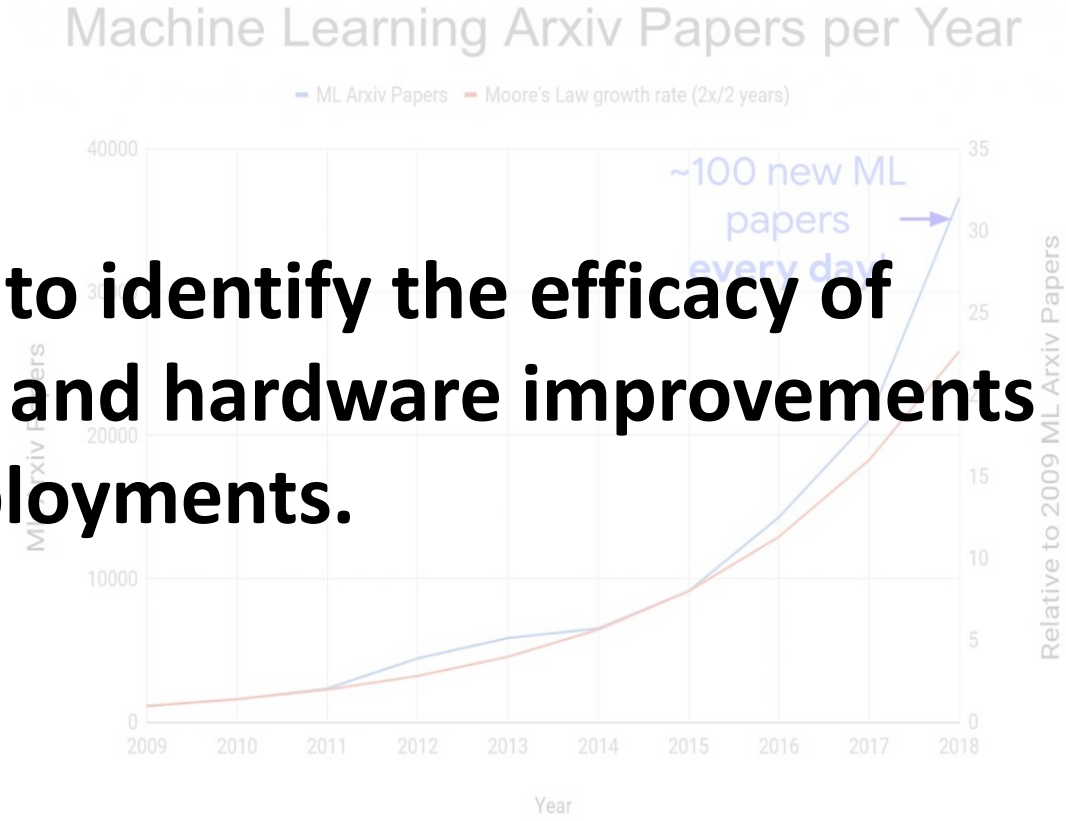
- Motivation: Benefits of many DNN optimizations are not easy to exploit because
  - Efficacy varies for different HW/SW deployments
  - It is onerous to implement optimizations
- Goal: Need to quickly find the effective optimizations for a given deployment
  - No need to FULLY implement the optimizations
- Our proposal: a system called **Daydream,** that can estimate runtime improvement of various DNN optimizations, using **dependency graph analysis**:
  - Tracking dependencies at the **abstraction of GPU kernels** (graph size is large)
  - **Correlating** low-level traces with layer organization of DNN models
  - Ability to model a **diverse** set of optimizations
- Evaluation: Low estimation error (8% average) on 5 optimizations, 5 DNN models
  - Accurately estimating distributed training runtime based on single-GPU profile

Advances in ML Full Stack Research



DNN compute requirements are growing exponentially

https://openai.com/blog/ai-and-compute/

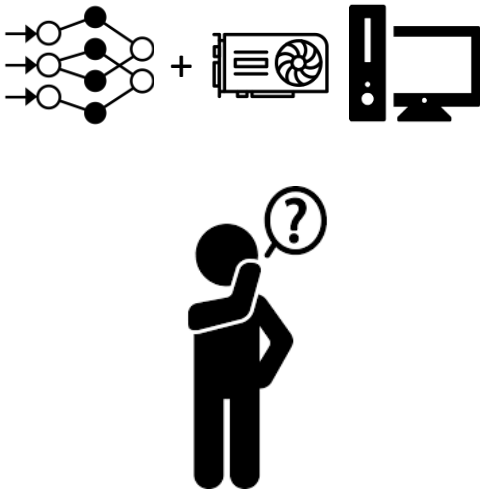


Rapid advances in algorithms, systems optimizations & hardware architectures

https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8259424&tag=1

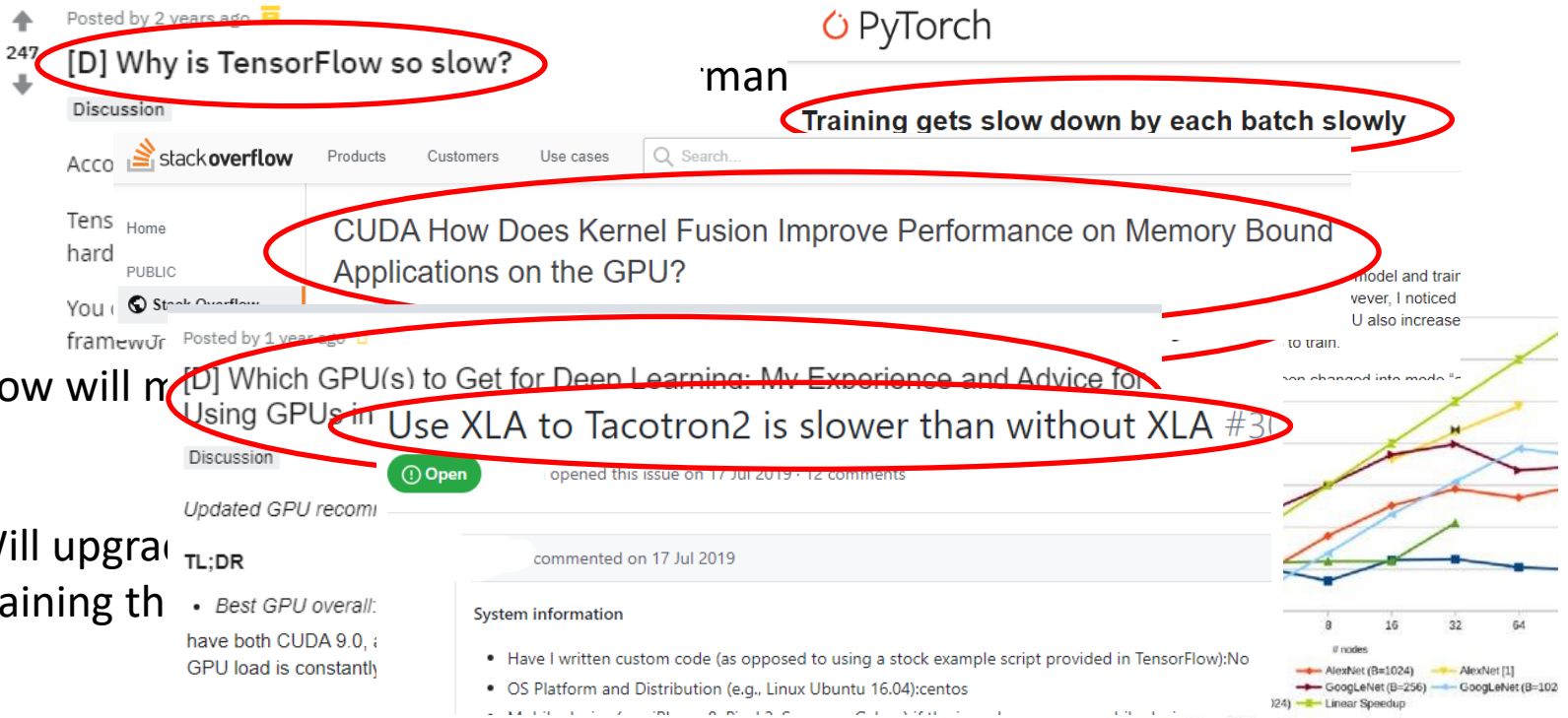
Hard for a ML programmer to identify the efficacy of new algorithms, optimizations, and hardware improvements in their deployments.

What-if Questions

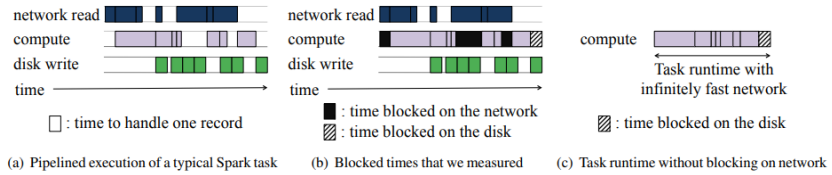


ML Programmer

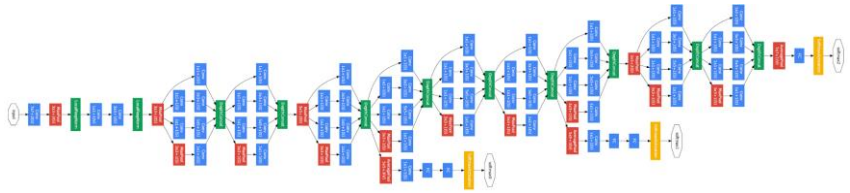
Why is my DNN training workload running slow? What is the bottleneck?



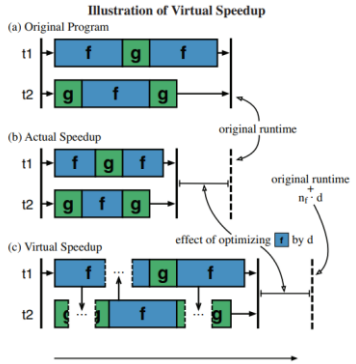
Why Dependency Analysis



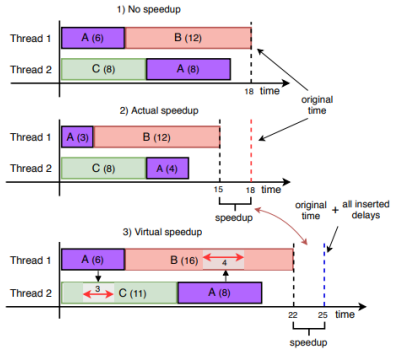
Making Sense of Performance in Data Analytics Frameworks (Ousterhout et al., NSDI 15)



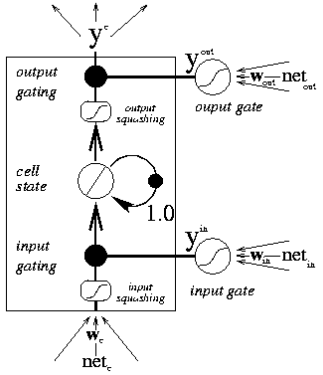
Inception (2014)



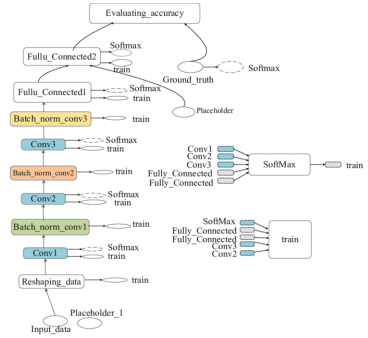
COZ: Finding Code that Counts with Causal Profiling (Curtsinger et al., SOSP 15)



What-If Analysis of Page Load Time in Web Browsers Using Causal Profiling (Pourghassemi et al., SIGMETRICS 19)



LSTM (2014)



TensorFlow's computational graph (2016)

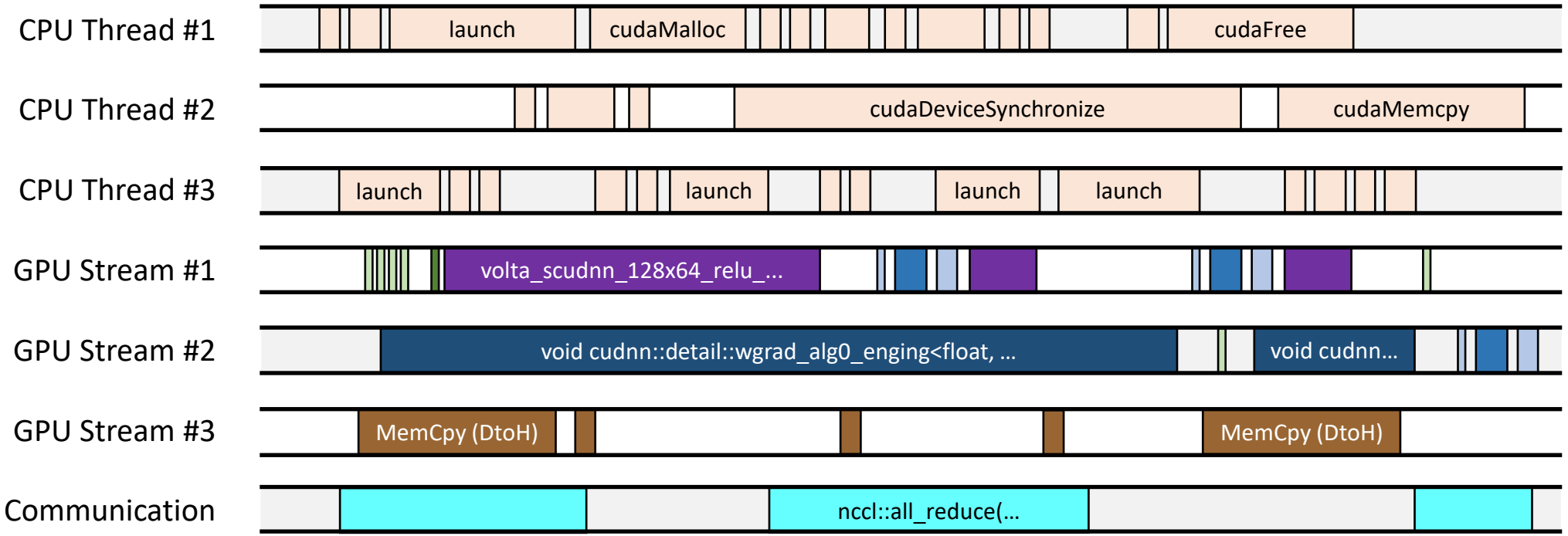
Answering what-if questions in non-ML contexts

DNN Computational Graph

Similarities between the graph structures, unique challenges and opportunities for the ML context

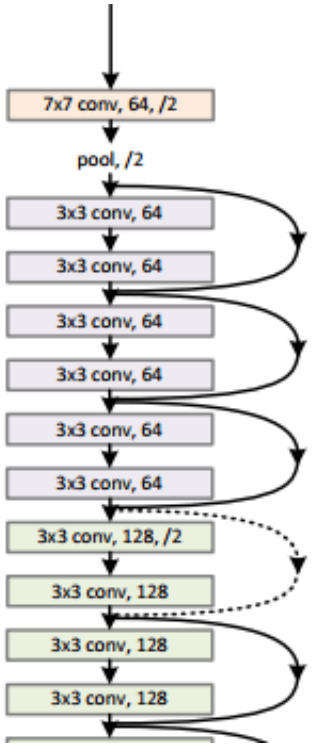
Challenges for Dependency Graph Analysis in the ML context

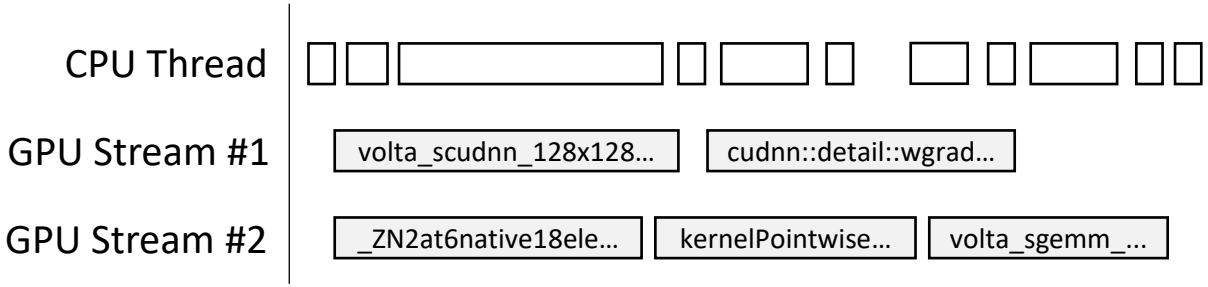
Challenge #1: Thousands of tasks, and dependency needs to be tracked across CPU threads, GPU streams, and interconnects.



Challenges for Dependency Graph Analysis in the ML context

Challenge #2: Modeling DNN optimizations requiring correlation between kernel and layer abstractions.

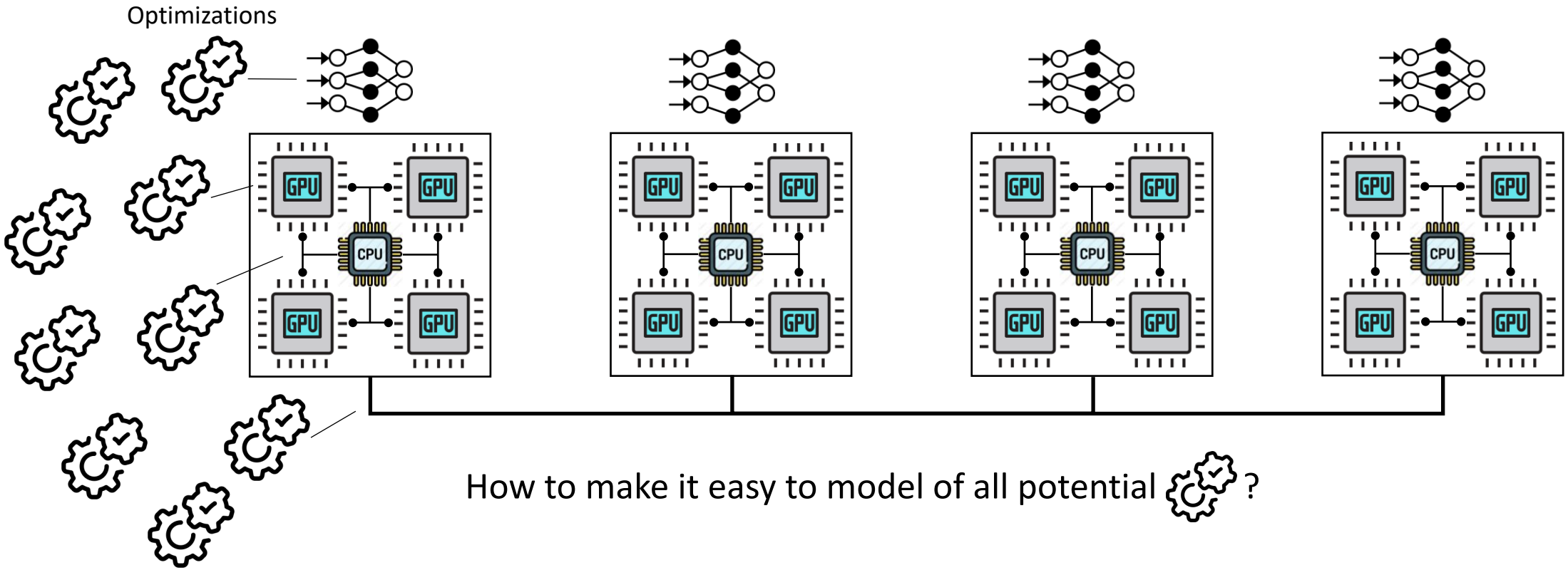




What if I improve CONV layers?
Which kernels belong to these layers?

Challenges for Dependency Graph Analysis in the ML context

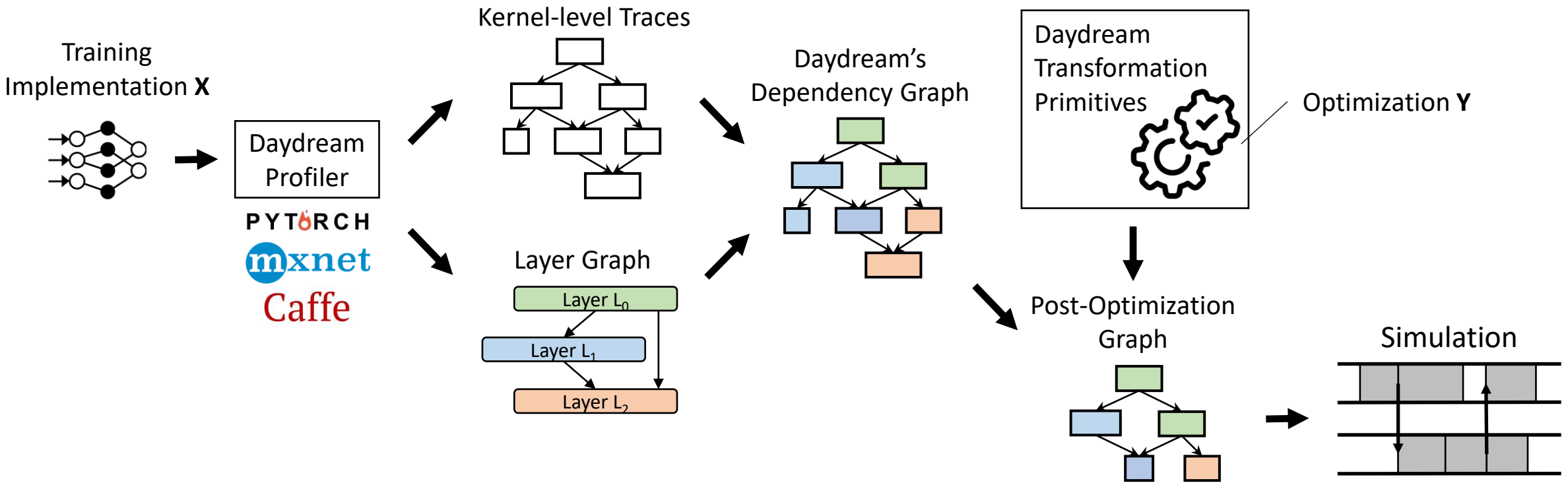
Challenge #3: Ability to easily model diverse DNN optimizations.



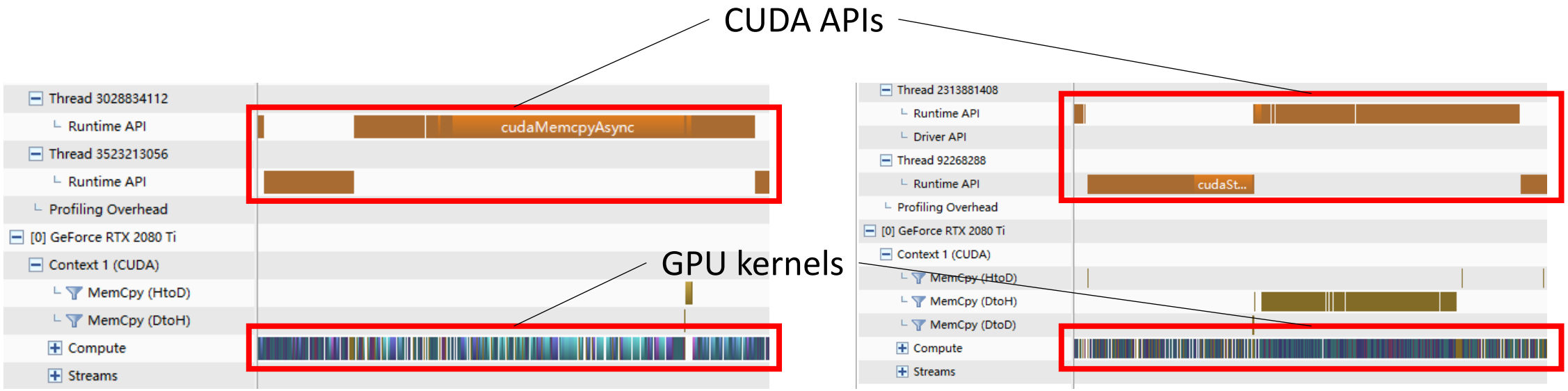
Daydream Overview

Input: an DNN training implementation **X**, an optimization **Y**

Output: the estimation of runtime when applying **Y** to **X**



Challenge 1: Tracking Dependencies



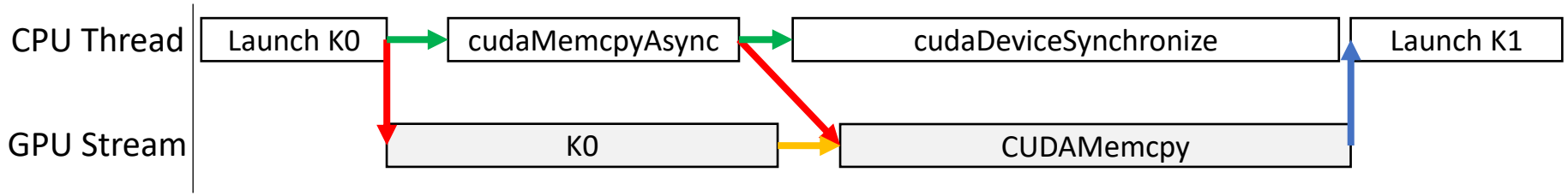
NVProf profile of one ResNet50 iteration

NVProf profile of one $BERT_{LARGE}$ iteration

Observation: GPU kernels are highly serialized for most DNN training workloads

Daydream's Graph Construction

We identify the **six** types of dependencies:



(1) → Sequential CPU-CPU: two consecutive CPU calls on the same CPU thread

(2) → Sequential GPU-GPU: two consecutive GPU kernels on the same stream

(3) → CPU-GPU launching: A CPU call launching a GPU kernel/CUDA memory copies

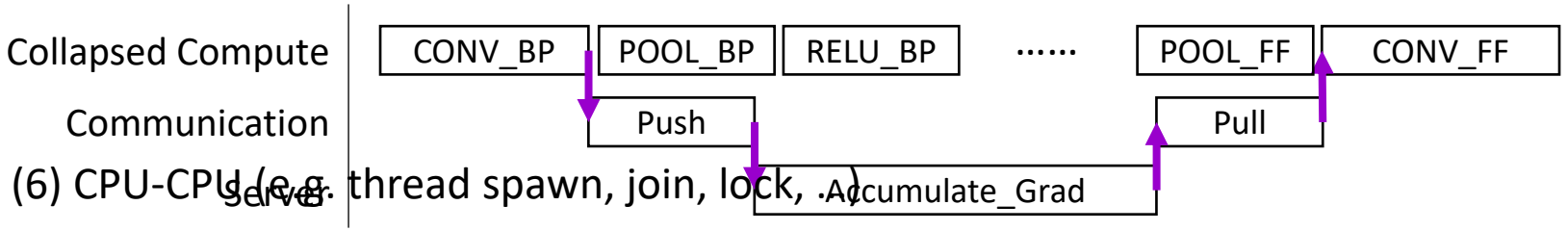
(4) → GPU-CPU sync: A CPU synchronization call waiting for GPU kernel to finish

Daydream's Graph Construction (cont.)

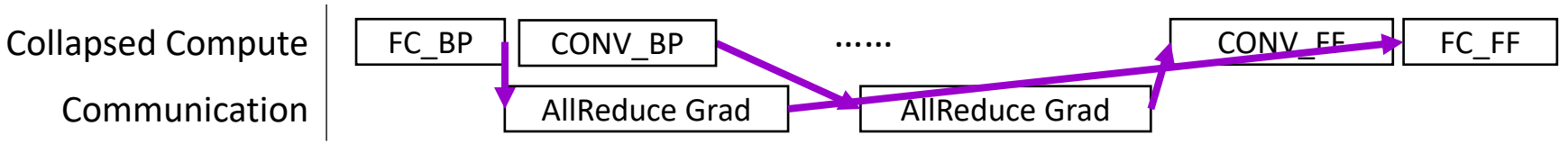
(5) → CPU-Communication

(6) CPU-CPU (e.g. thread spawn, join, lock, ...)

Parameter Server Architecture:



MPI-like Architecture:

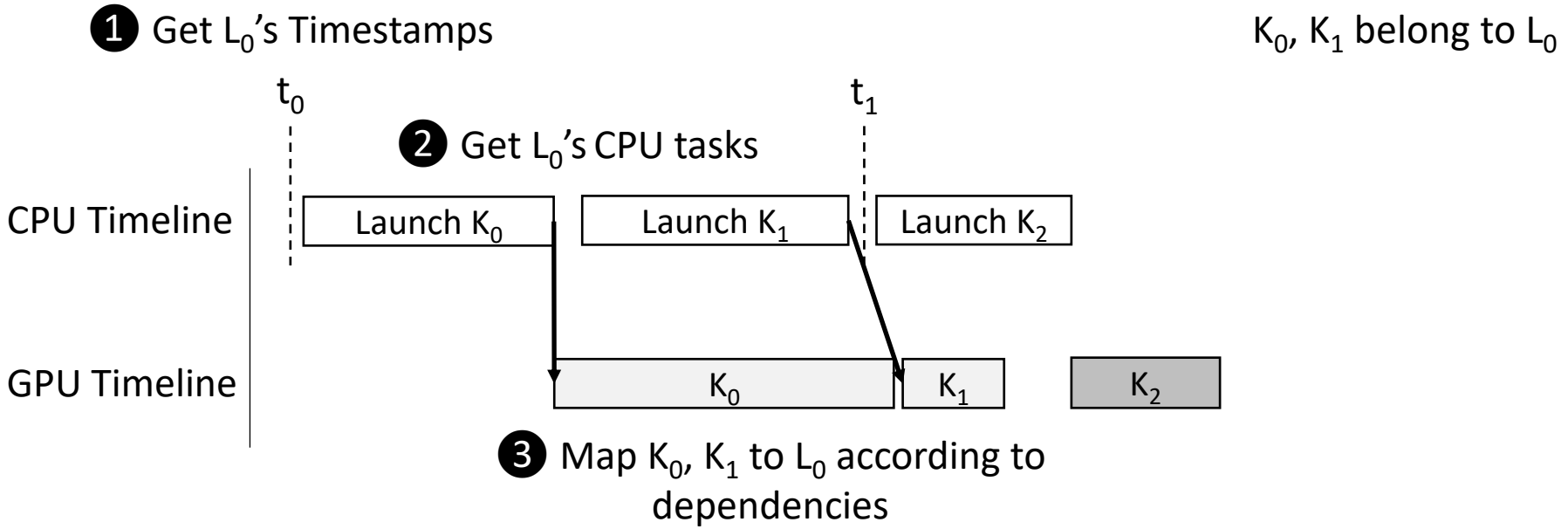


Challenge 2: Trace-Layer Correlation

- Optimizations requiring correlation between low-level traces and DNN layers:
  - E.g., Fusing CONV and RELU layers
  - Low-level traces have NO domain knowledge

- Naïve approach: adding synchronization

Daydream's Kernel-Layer Mapping



Little overhead (only need to instrument frameworks for per-layer timestamps)

No alternation to the dependency graph (synchronization-free)

Challenge 3: Optimization Diversity

Optimization Goals	Strategy	Technique Examples
Improving Hardware Utilization in Single-Worker Environment	Increasing Mini-batch Size by Reducing Memory Footprints	***vDNN*** (MICRO16), ***Gist*** (ISCA18), Echo (ISCA20)
	Reducing Precision	***Automatic Mixed Precision*** (arxiv17)
	Kernel/Layer Fusion	***FusedAdam***, ***MetaFlow*** (MLSys19), TASO (SOSP19)
	Improving Kernel Implementation	***Restructuring Batchnorm*** (MLSys19), TVM (OSDI18), Tensor Comprehensions (arxiv18)
Lowering Communication Overhead in Distributed Training	Reducing Communication Workloads	***Deep Gradient Compression*** (ICLR18), QSGD (NeurIPS17), AdaComm (MLSys19), Parallax (EuroSys19), TernGrad (NeurIPS17)
	Improving Communication Efficiency/Overlap	***Wait-free Backprop*** (ATC17), ***P3*** (MLSys19), ***BlueConnect*** (MLSys19), TicTac (MLSys19), BytePS (SOSP19), Blink (MLSys19)

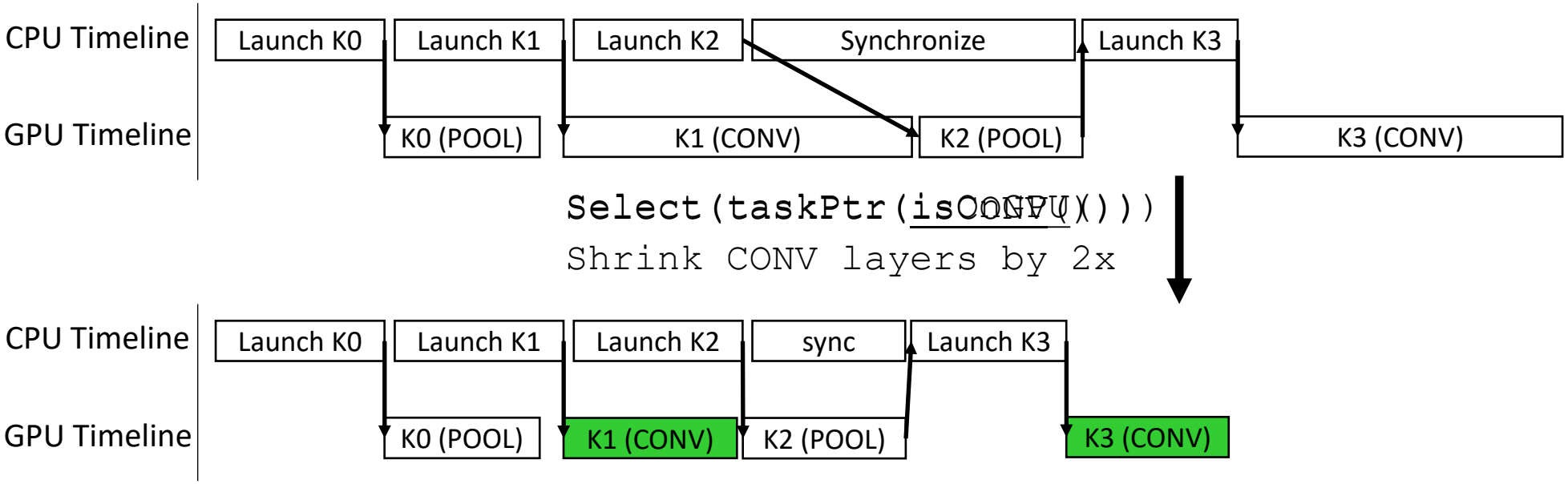
We evaluate "***some optimizations***", and show that we can conveniently model "***others***" using Daydream

Daydream's Transformation Primitives

Most DNN optimizations can be described as a combination of the following primitives:

(1) Select(expr): return tasks of interests for further process

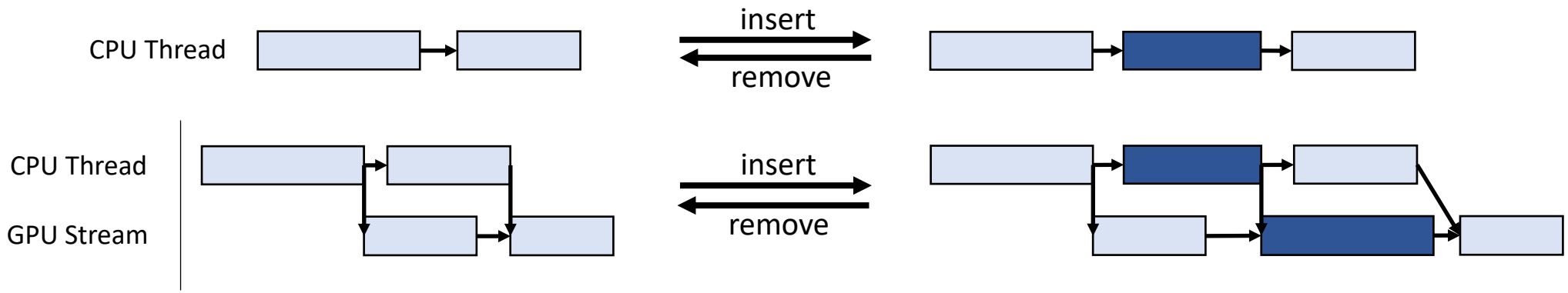
(2) Shrinking/Scaling the task duration



Daydream's Transformation Primitives (cont.)

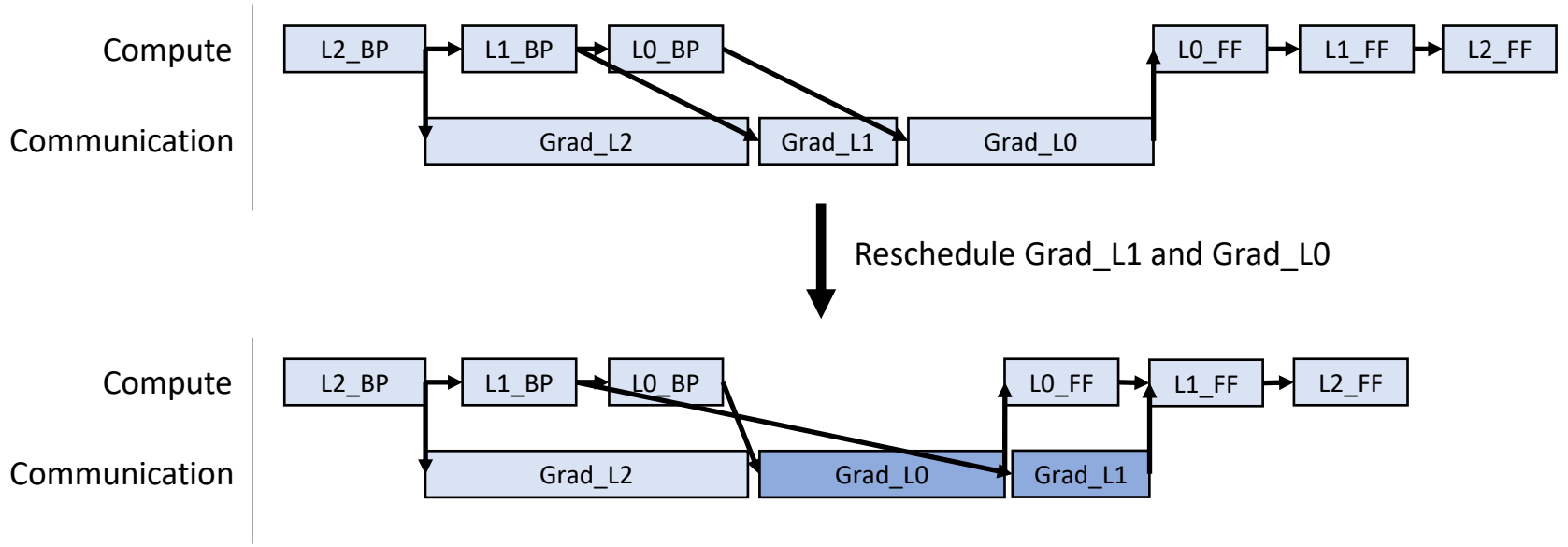
(3) Insert(s, task, t): Insert a task between s and t

(4) Remove(task): Remove a task from the graph



Daydream's Transformation Primitives (cont.)

(5) Schedule(Q: a queue of tasks that are ready to execute): --> task
Decide which task to execute when multiple tasks are ready



Example – Automatic Mixed Precision

Using Daydream to estimate the efficacy of AMP (Micikevicius et al., arxiv 2017)

Low-level traces

Per-layer timestamps

```
def estimate_AMP(cupti_file, timestamps_file):
    graph = Graph(cupti_file)
    graph.mapping(timestamps_file)

    GPUNodes = [node for node in graph.nodes() if node.kind == "KERNEL"]
    for node in GPUNodes:
        if "wgrad" in node.name or "sgemm" in node.name:
            node.dur /= 3
        else:
            node.dur /= 2

    return graph.simulate()
```

Constructing kernel-level dependency graph

Map low-level traces to DNN layers using per-layer timestamps

Select all GPU tasks from the graph

If we expect this task to use TensorCore

Otherwise, use half-precision cores

Simulate the timeline, return the elapsed execution time

10 optimization examples, each around 20 lines of code (refer to our paper)

Methodology

Woakloads:

Application	Model	Dataset
Image Classification	VGG-19	Imagenet
	DenseNet-121	
	ResNet-50	
Machine Translation	GNMT (Seq2Seq)	WMT
Language Modeling	BERT	SQuAD

Setup:

RTX 2080 Ti

Quadro P4000

NVIDIA CUDA v10.0

cuDNN v7.4.2

NCCL (GPU0, GPU1, GPU2, GPU3) v2.4.2

PYTORCH v1.0

mxnet v1.1

Caffe v1.0

Optimizations:

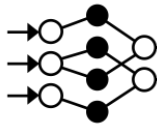
Improving hardware utilization:

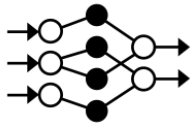
Automatic Mixed Precision (AMP), FusedAdam, Reconstructing Batchnorm

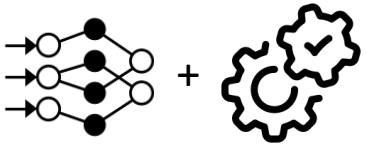
Distributed training:

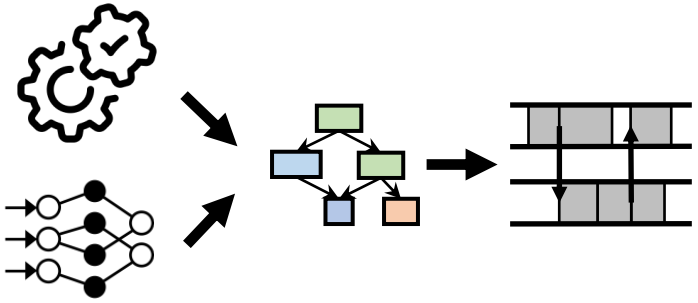
Data-parallel distributed training, Priority-based parameter propagation (P3)

Methodology (cont.)

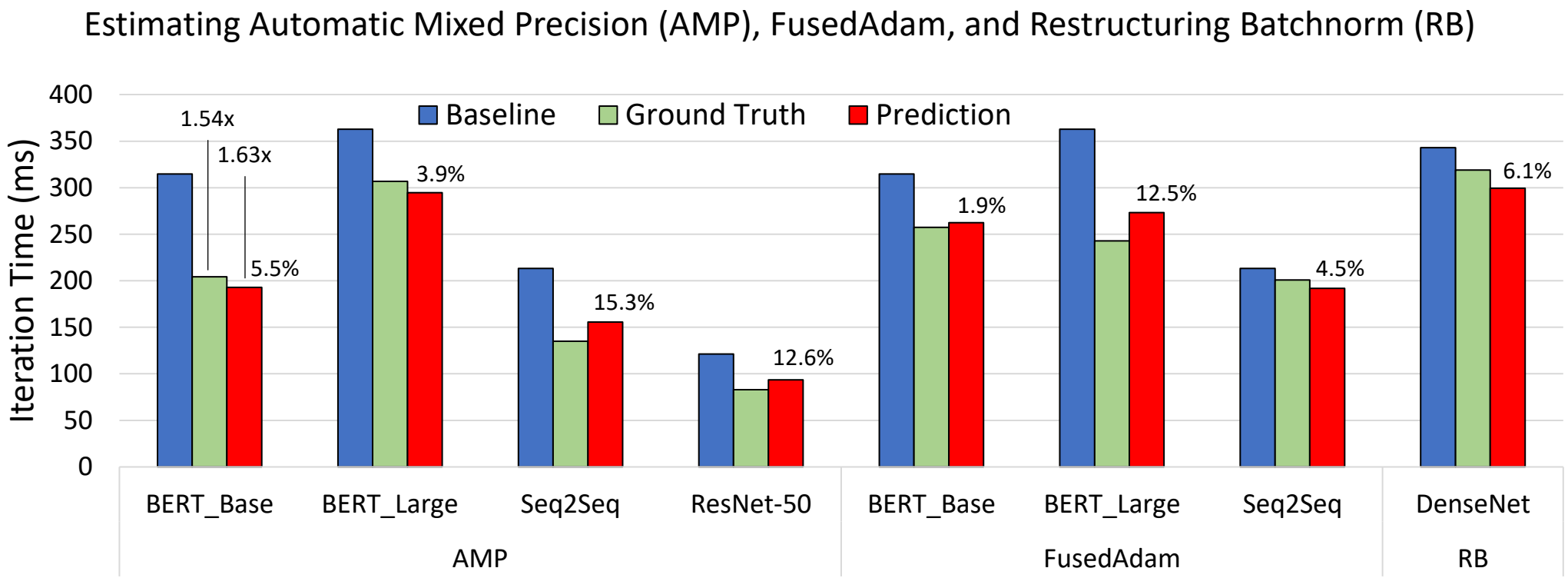
Given a  and a  , we evaluate:

Baseline: 

Ground Truth: 

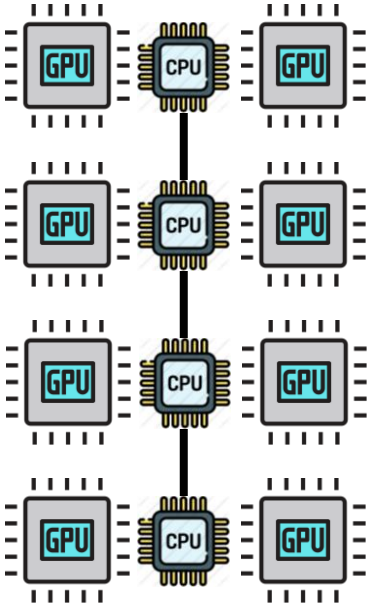
Prediction: 

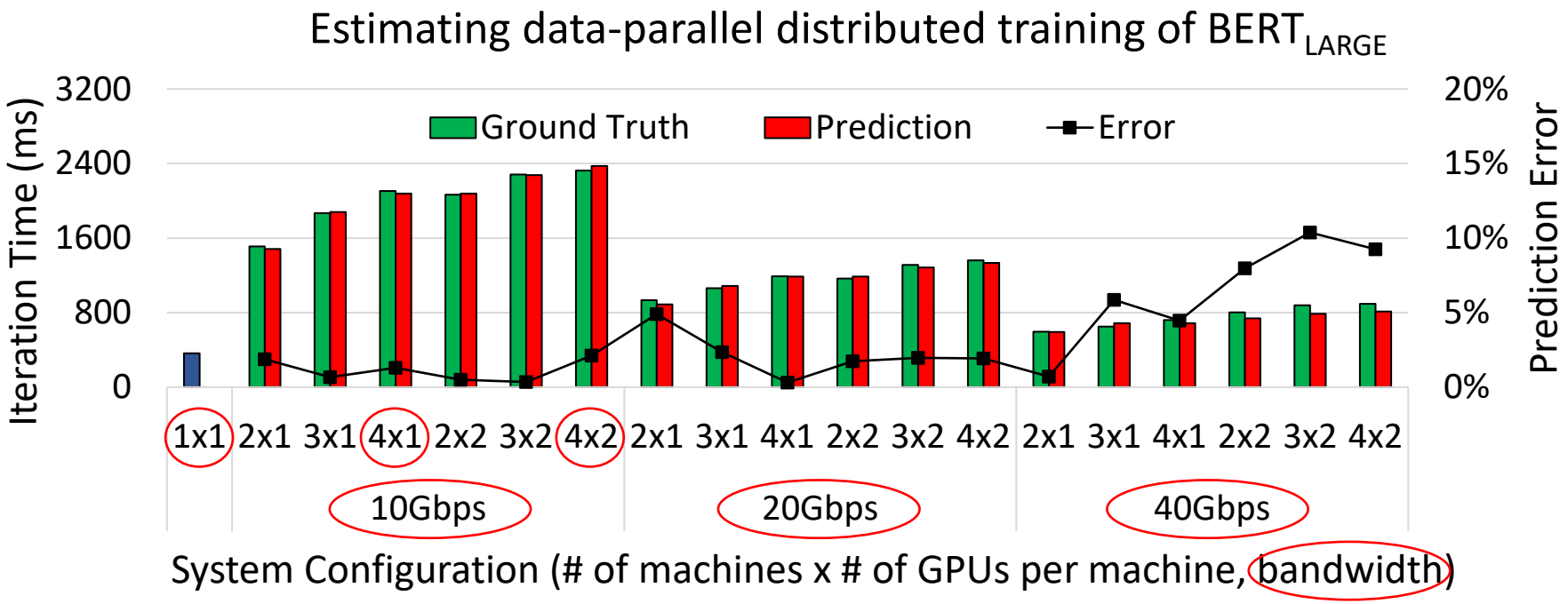
Runtime Estimation Accuracy



Daydream achieves 8% estimation error on average (15% maximum)

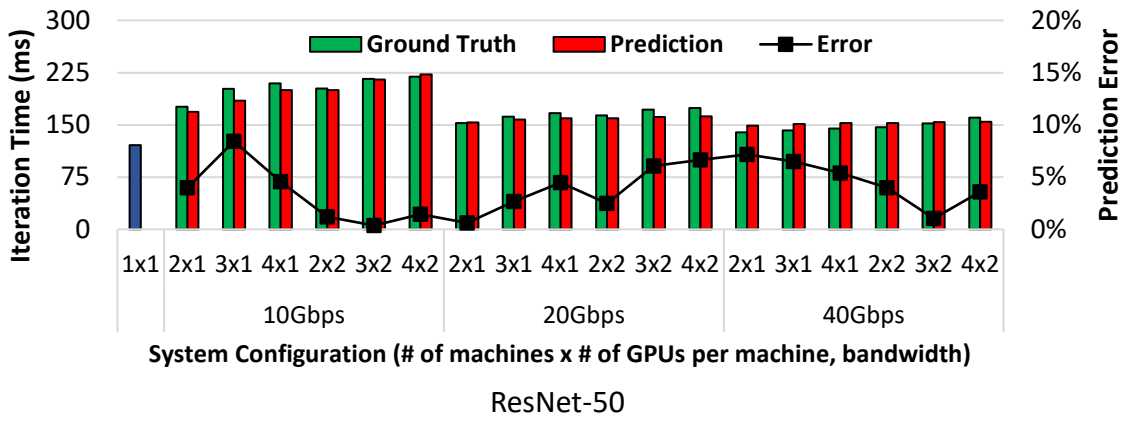
Estimating Distributed Training



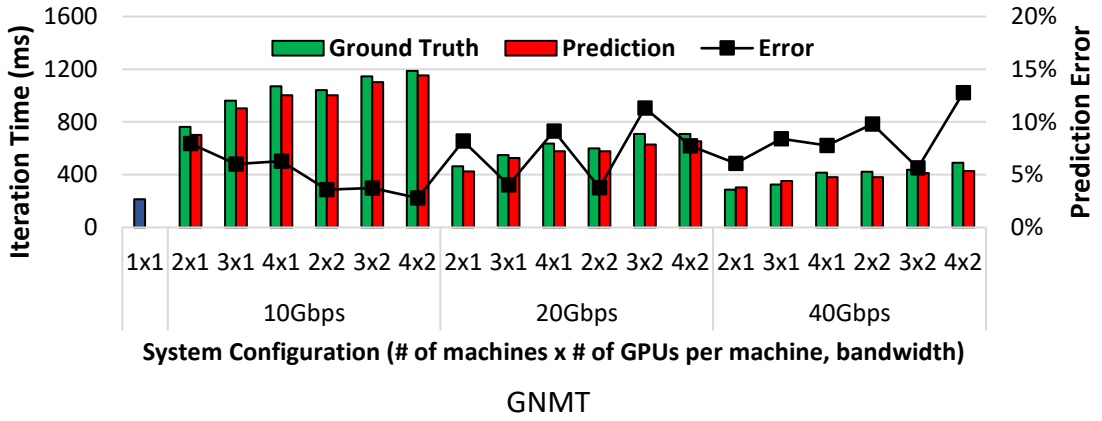


Daydream can accurately estimate the distributed performance for various system configurations

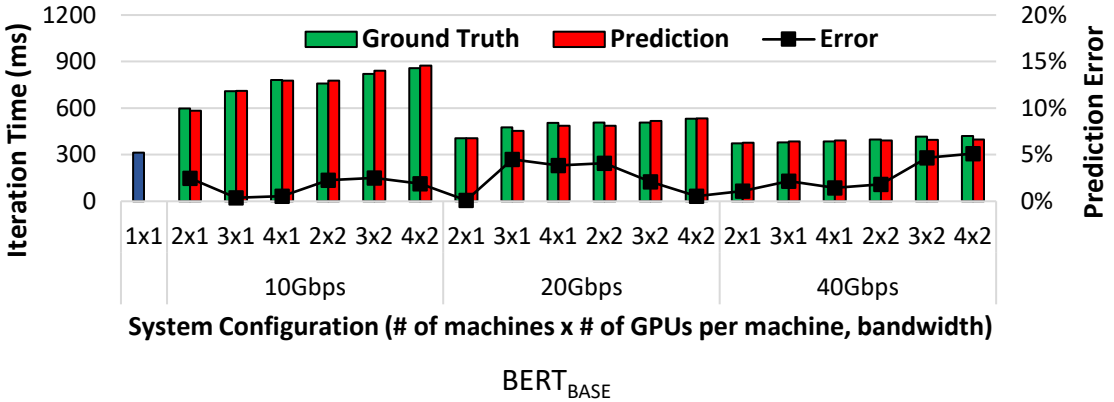
Estimating Distributed Training



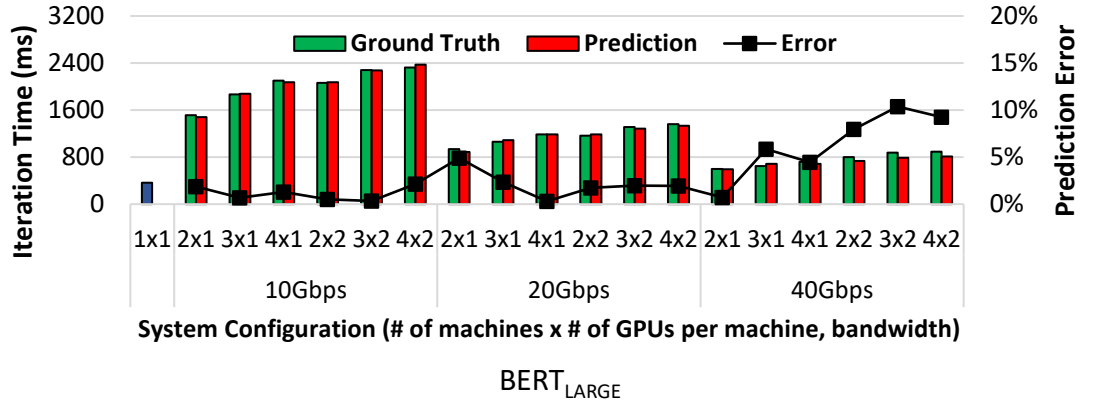
ResNet-50



GNMT



$BERT_{BASE}$



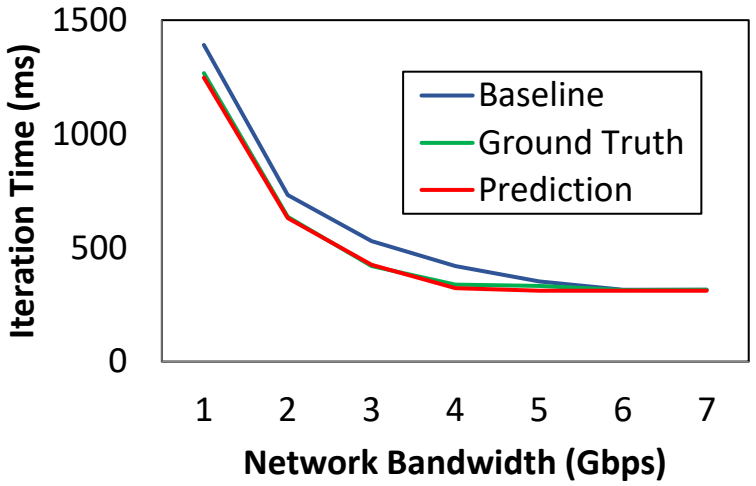
$BERT_{LARGE}$

Daydream can accurately estimate the distributed performance for a variety of DNN models

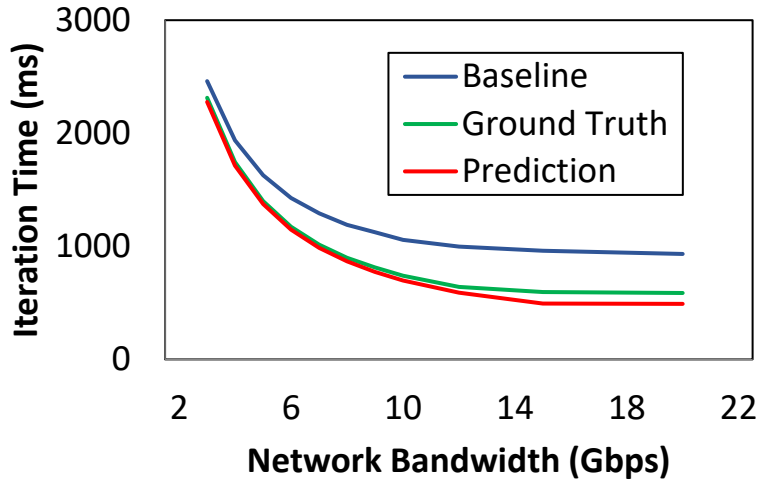
Estimating Efficacy of P3

Prediction accuracy for Priority-Based Parameter Propagation (P3)

(we use 4 machines and 1 P400 GPU on each machine)



Runtime Prediction for ResNet-50



Runtime Prediction for VGG-19

Using Daydream, we can successfully estimate whether P3 would provide significant or subtle improvement

Conclusion

Benefits of DNN optimizations are not easy to exploit:

- Efficacy various across different hw/sw deployments
- Often onerous to implement and debug

Basic Idea: **Dependency graph analysis**

Our Solution: The **Daydream** system allowing users to quickly estimate the performance of various DNN optimizations:

- Tracking dependencies at the **kernel-level granularity**
- **Sync-free** trace-to-layer mapping
- **Simple graph transformation primitives**

Key Results: Estimation error of 8% on average (15% maximum)
Modeling a wide range of optimizations (only 20 lines of code each)

Daydream: Accurately Estimating the Efficacy of Optimizations for DNN Training

Hongyu Zhu[1,2], Amar Phanishayee[3], Gennady Pekhimenko[1,2]

Thank you!

1 UNIVERSITY OF TORONTO

2 VECTOR INSTITUTE

3 Microsoft Research

serailhydra@cs.toronto.edu