

Theo Radig
 Laser-Plasma-Institute
 Technologies and Software
 Potsdam, Germany
 theo.radig@lpi.de

Holger Karl
Hans-Plattner-Institute
Internet Technologies and Softwareization
Potsdam, Germany
holger.karl@hpi.de

[illegible]

1. INTRODUCTION

Deep Neural Networks (DNNs) comprise a series of interdependent calculations at the layers of the neural network. These layers are conventionally associated to one or more arithmetic operations, for example, a matrix multiplication as an activation function. The operations are executed on dedicated hardware and are typically implemented in specialized libraries like cuDNN [1] or deep-learning frameworks like PyTorch [2].

DNN applications, such as training a model or serving inference requests, require high throughput and low latency. To ensure quality of service (QoS), cloud-facing services such as ML inference are often deployed on dedicated machines [3]. But the load on these machines caused by inference services is bursty, with long periods of inactivity between requests [4], resulting in low average compute utilization [Fig. 1a]. Furthermore, popular DNN models like ResNet101 [5] often do not utilize all streams

multiprocessors (SbMs) of a graphics processing unit (GPU) (Figure 1b), even with larger batch size.

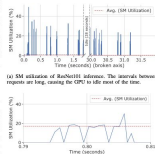


Fig. 1: Measured SM utilization of ResNet101 inference on an NVIDIA A100 [12] GPU. We show the first 30 seconds of load based on the inter-arrival times (IATs) of inference traces published and analyzed by Microsoft Azure in [4]. Most of the SMs idle, even with load.

all applications replicate the same contents. This enables us to assign different contents to different applications at runtime.

sity of the application, the applications, and the number of jobs. We use shared memory across the different

ment, J. Tervilais, and H. Choudry. "Distributed architectures for performance in AI environments." In *Proceedings of the 22nd Annual Conference on Artificial Intelligence*, I. Fagan, "Deep reasoning for image recognition," M. Bajcsy, "TOOT: Toward common sense," and "Introduction to the 22nd Annual Conference on Artificial Intelligence," IJCAI-88, August 1988, pp. 308-313.

10. M. Hagiwara, and P. Tassi. "An analysis of the structure of the knowledge base in the *Learning and Systems*, see Tervilais H. Association for Computing Machinery, April 1988.

11. J. and M. Schulz. "Hierarchical reasoning in a reinforcement learning approach." *Science on Cluster Computing* (ICLUSTER-88), 1988.

12. M. Naghibadshahi, H. Fessing, and H. Choudry. "A comparison of concurrent kernels in a distributed system." *Journal of Systems Management*, vol. 31, no. 4, pp. 766-773, 1986.

13. J. H. Morris. "On-line, intranet-type, systems applications." In *Proceedings of the 22nd Annual Conference on Artificial Intelligence on Computer Systems*, see Tervilais H. Association for Computing Machinery, April 1988.

14. H. Cheri. "Microsecond-scale processing in a distributed system." In *Proceedings of the 22nd Annual Conference on Artificial Intelligence*, I. Fagan, "Deep reasoning for image recognition," M. Bajcsy, "TOOT: Toward common sense," and "Introduction to the 22nd Annual Conference on Artificial Intelligence," IJCAI-88, August 1988, pp. 308-313.

[illegible]

Wang, B., Lu, Y., Li, X., Cho, and H. L. are-level GPU resource isolation for multi-tenancy. *Proc. ACM*, 3(4):2024, 2024.

100%

Onco

g 202

Inte

International Symposium on Cluster, Cloud and Internet Computing 2025 | Presentation

Theo Radig and Holger Karl
Technologies and Softwarization
Hasso-Plattner-Institute

Challenges & Goals

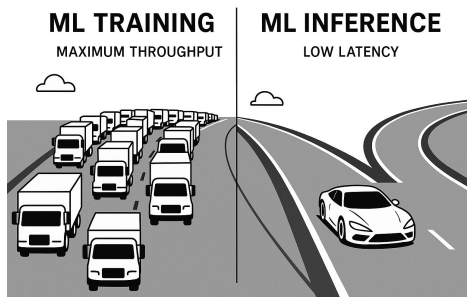
System
Design

Summary

Requirements
& Contributions

Evaluation

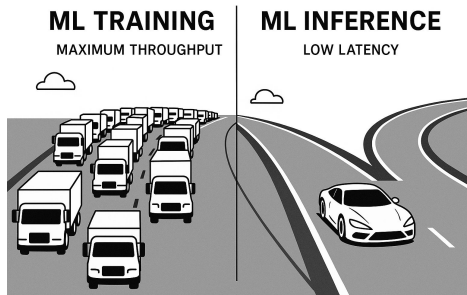
Challenges



Created by ChatGPT.

- ❑ ML workloads are **heterogenous**
 - Batch training
 - Low-latency inference

Challenges



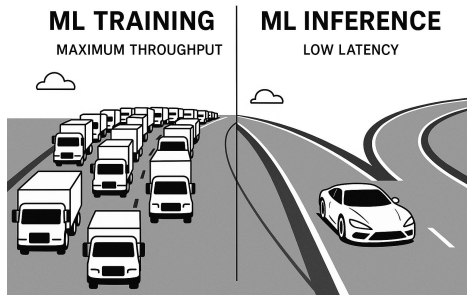
Created by ChatGPT.



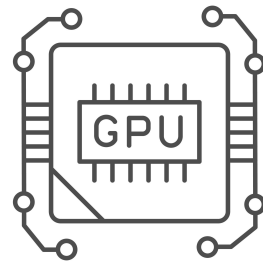
- ❑ ML workloads are **heterogenous**
 - Batch training
 - Low-latency inference

- ❑ Inference services **guarantee latency**

Challenges



Created by ChatGPT.



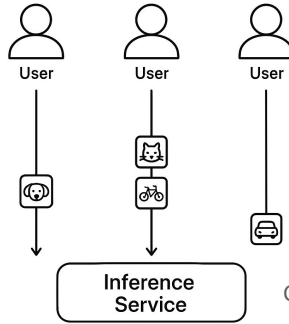
- ❑ ML workloads are **heterogenous**
 - Batch training
 - Low-latency inference

- ❑ Inference services **guarantee latency**

- ❑ **Exclusive** GPU usage
- ❑ Spatial sharing **over-provisions**

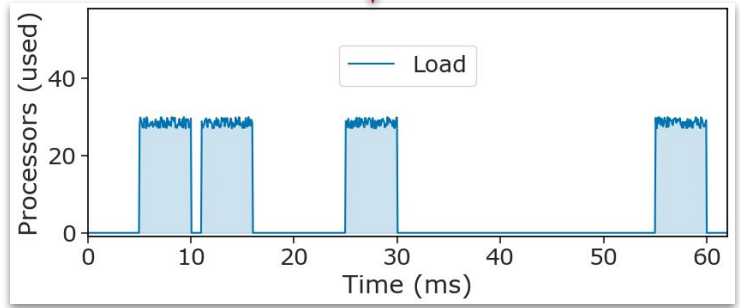
Example

Requests arrive irregularly



Created by ChatGPT.

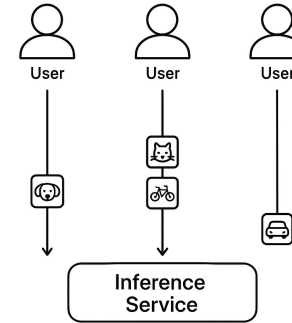
Requests



Dynamic Workload

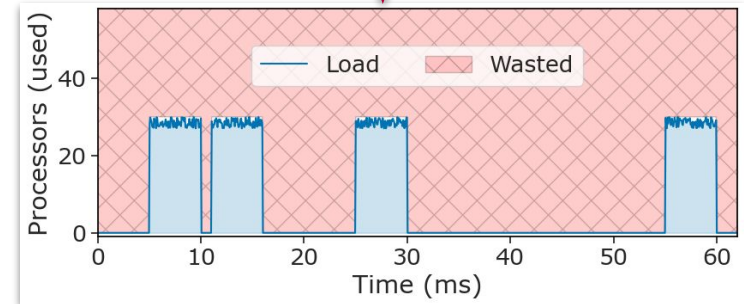
Example

- ❑ Requests arrive irregularly
- ❑ Compute resources idle



Created by ChatGPT.

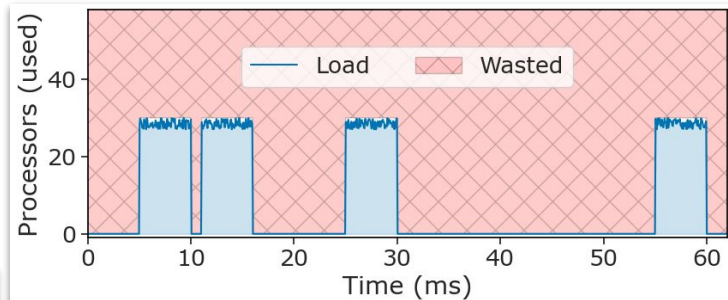
Requests



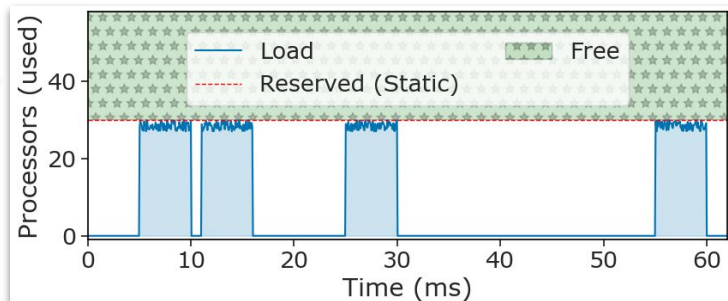
Dynamic Workload

Example

- ❑ Requests arrive irregularly
- ❑ Compute resources idle
- ❑ Spatial sharing (MIG¹ / MPS²)



Dynamic Workload

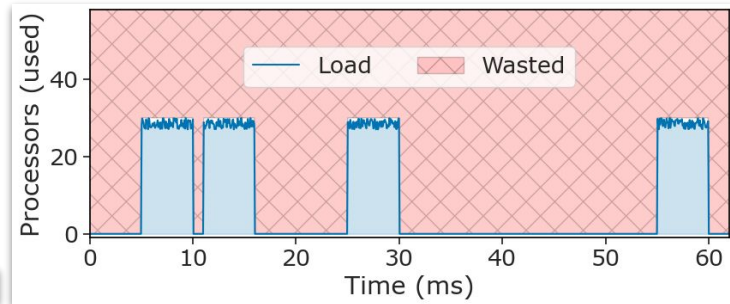


Static Partitioning

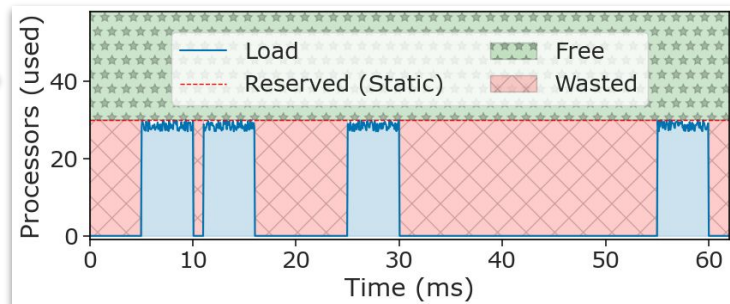
¹ NVIDIA Multi-Process Service ² NVIDIA Multi-Instance GPU

Example

- ❑ Requests arrive irregularly
- ❑ Compute resources idle
- ❑ Spatial sharing (MIG¹ / MPS²)
- ❑ Inference service still over-provisioned
- ❑ Wasted resources



Dynamic Workload



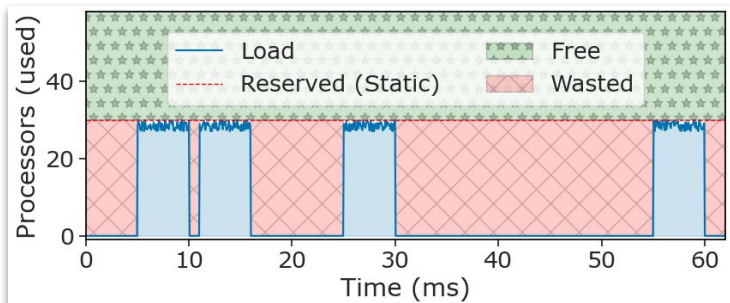
Wasted Resources

¹ NVIDIA Multi-Process Service ² NVIDIA Multi-Instance GPU

Constraints & Goals

Constraints (MPS / MIG)

- ❑ Static partitions ensure low latency
- ❑ Cannot be quickly reconfigured
- ❑ Static partitions over-provision
 - Cost-ineffective

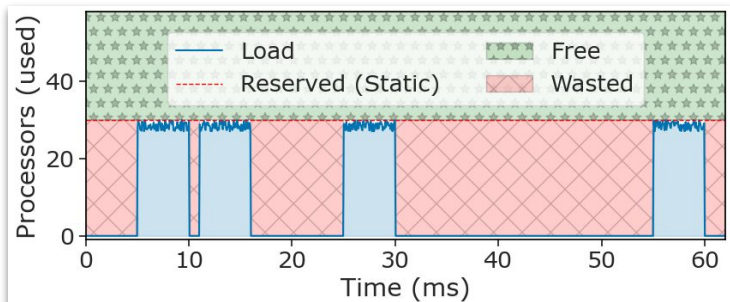


Wasted Resources

Constraints & Goals

Constraints (MPS / MIG)

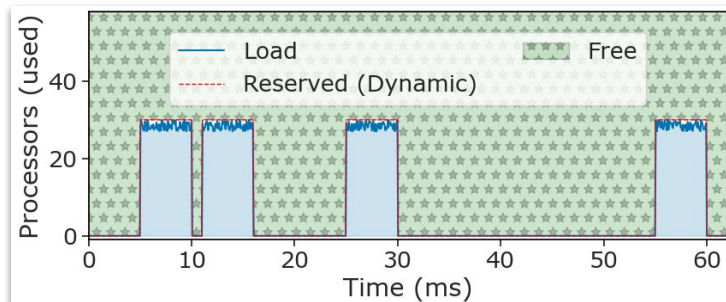
- ❑ Static partitions ensure low latency
- ❑ Cannot be quickly reconfigured
- ❑ Static partitions over-provision
 - Cost-ineffective



Wasted Resources

Goals

- ❑ Dynamically allocate / free resources
- ❑ Co-locate inference and training
 - Low inference latency
 - High training throughput



Dynamic Partitioning

Challenges
& Goals

System
Design

Summary



Requirements & Contributions

Evaluation

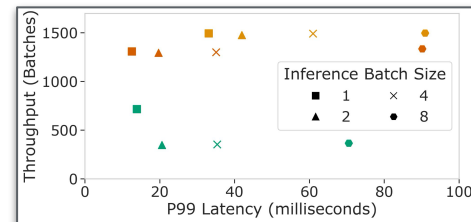
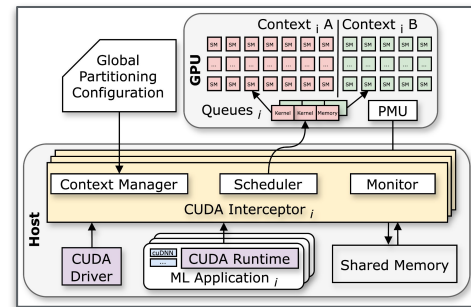
Requirements

- ❑ **Applicable**
 - Zero modifications to ML frameworks
- ❑ **Online**
 - No offline profiling phase
- ❑ **Isolation**
 - Co-located applications operate independently
- ❑ **Low overhead**
 - Profiling must not increase latency
 - Moderate memory footprint



Contributions

- ❑ **Explore NVIDIA Green Contexts**
 - For dynamic SM¹ partitioning
- ❑ **Draco**
 - Automatically assign resources
 - Fine-granular partitioning
 - Live sampling the PMU²
- ❑ **Evaluate**
 - Compare with state-of-the-art MPS
 - Model different workloads



¹ Streaming-Multiprocessor ² Performance Monitoring Unit

System Design

Challenges
& Goals

Summary



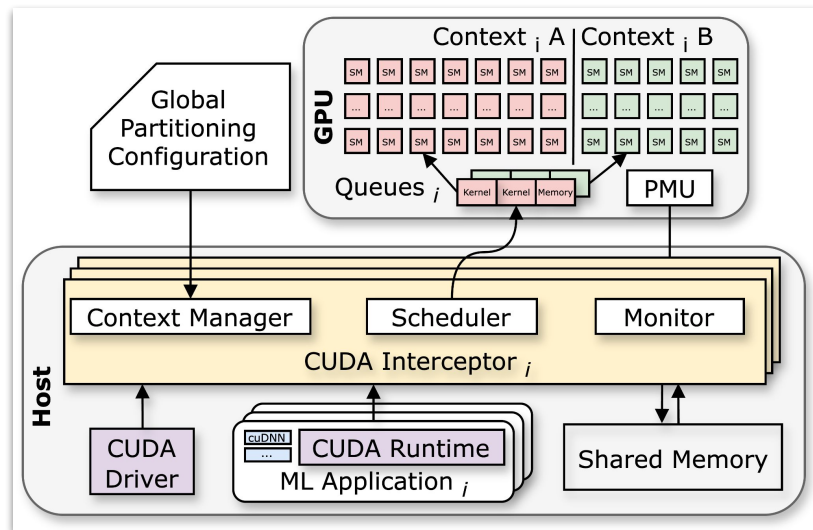
Requirements
& Contributions

Evaluation

Architecture

Highlights

- ❑ Effortlessly integrate with `ld_preload`
- ❑ Intercept CUDA runtime / driver
- ❑ Separate processes (CPU) / contexts (GPU)
- ❑ Low overhead profiling at runtime



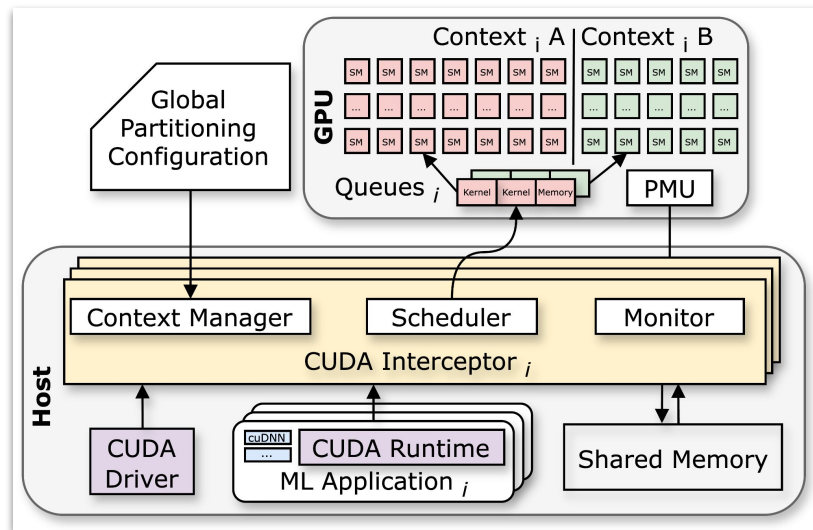
Architecture

Highlights

- ❑ Effortlessly integrate with `ld_preload`
- ❑ Intercept CUDA runtime / driver
- ❑ Separate processes (CPU) / contexts (GPU)
- ❑ Low overhead profiling at runtime

Core Components

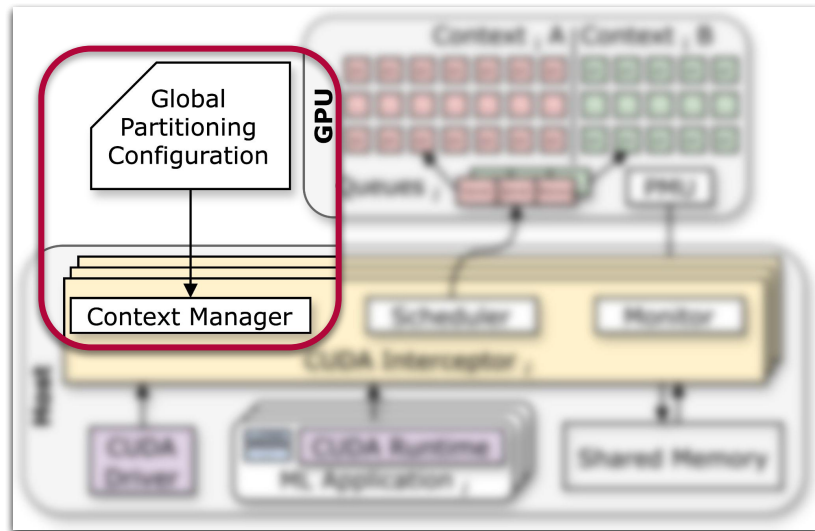
- ❑ Context Manager
- ❑ Monitor
- ❑ Kernel Scheduler



Context Manager

Traditional Context

- ❑ Comparable to OS process
- ❑ Isolates resources
- ❑ Automatically created
- ❑ Large memory footprint (**429MB**)



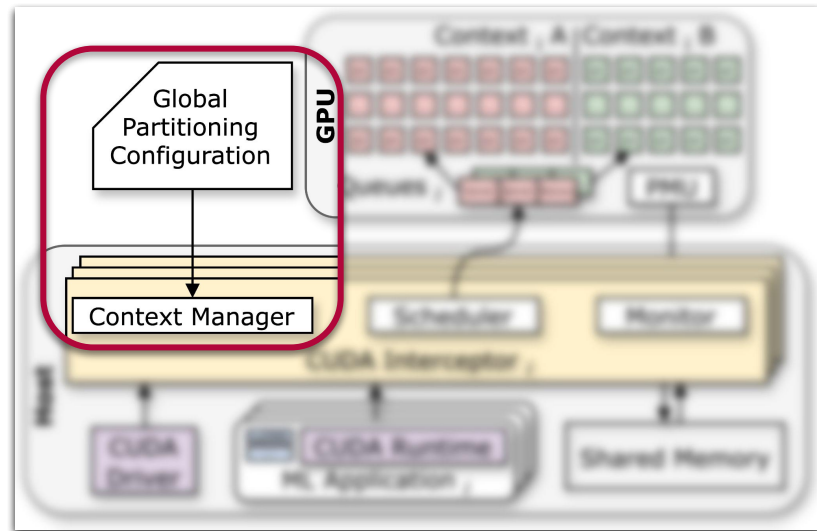
Context Manager

Traditional Context

- ❑ Comparable to OS process
- ❑ Isolates resources
- ❑ Automatically created
- ❑ Large memory footprint (**429MB**)

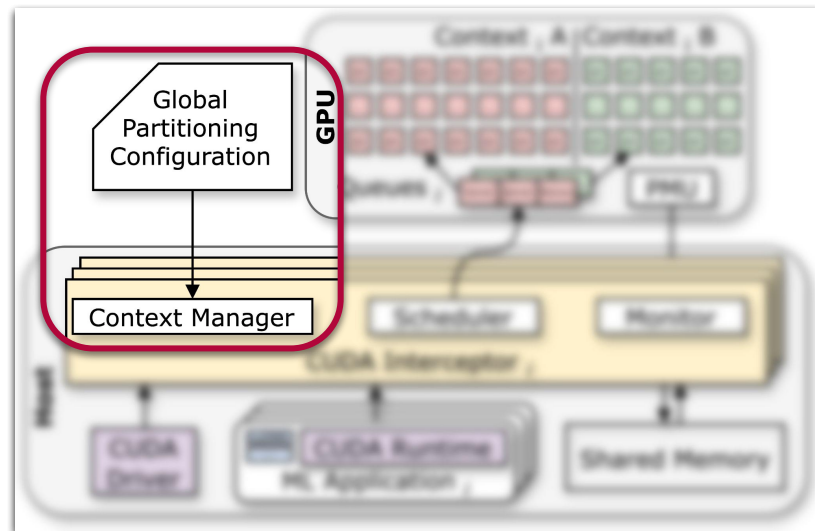
Green Context

- ❑ Manually created
- ❑ Low memory footprint (**4MB**)



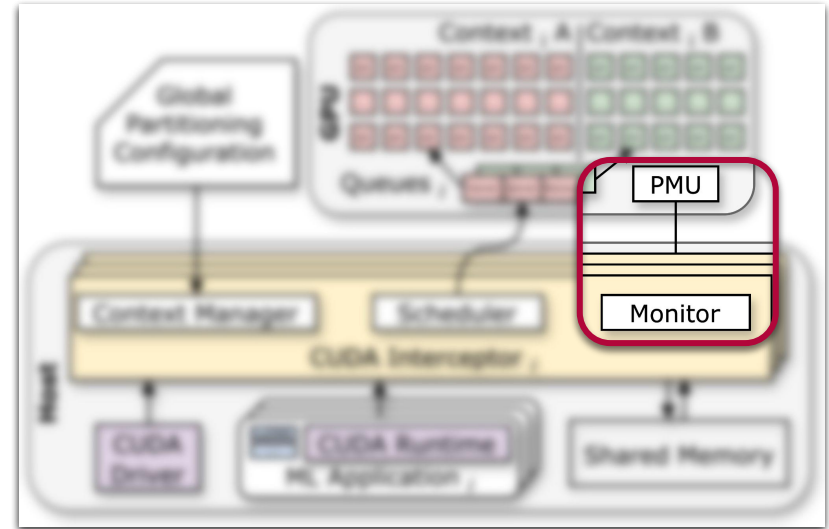
Context Manager

- ❑ Self-manage a pool of contexts
- ❑ Applications replicate this pool
- ❑ Fine-granular partitioning granularity



Monitor

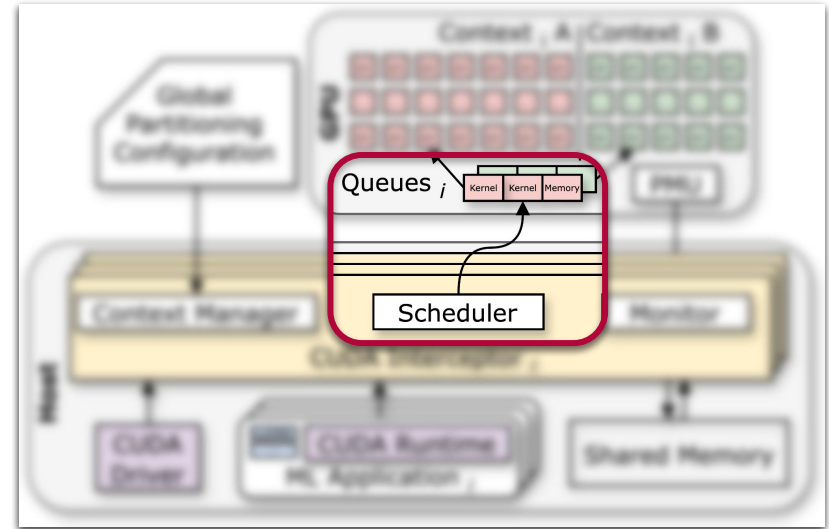
- ❑ Hardware-counter sampling every 1ms
- ❑ Percentage of active SMs
- ❑ Estimate SM requirement for inference
- ❑ Detect inference idle times



Kernel Scheduler

Kernel

- ❑ Function that executes on a GPU
- ❑ Data-parallel operation
- ❑ Invoked through CUDA APIs



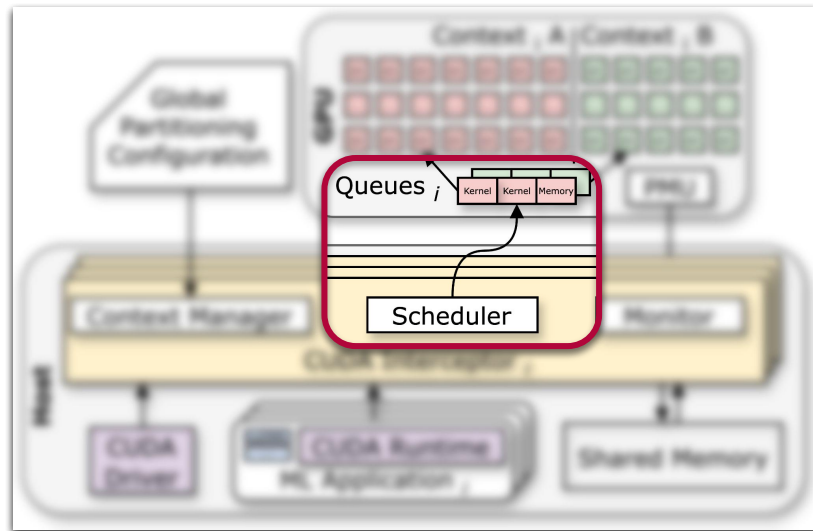
Kernel Scheduler

Kernel

- ❑ Function that executes on a GPU
- ❑ Data-parallel operation
- ❑ Invoked through CUDA APIs

Scheduler

- ❑ For every kernel
 - Delay submission
 - Initiate context switch



Challenges
& Goals

System
Design

Summary



Requirements
& Contributions

Evaluation

Testbed & Workloads

Testbed

- ❑ NVIDIA A30 GPU (**56** SMs)
- ❑ Intel Xeon Platinum 8468V (**96** Cores)
- ❑ Latest CUDA version



Testbed & Workloads

Testbed

- ❑ NVIDIA A30 GPU (**56** SMs)
- ❑ Intel Xeon Platinum 8468V (**96** Cores)
- ❑ Latest CUDA version

Workloads

Workload	Model	Operation	Batch Size	MPS Thread Limit (%)
A				
B				
C				



Testbed & Workloads

Testbed

- ❑ NVIDIA A30 GPU (**56** SMs)
- ❑ Intel Xeon Platinum 8468V (**96** Cores)
- ❑ Latest CUDA version

Workloads

Workload	Model	Operation	Batch Size	MPS Thread Limit (%)
A	Bert-base-uncased	Training	16	
B	Bert-base-uncased	Training	16	
C	Bert-base-uncased	Training	16	



Testbed & Workloads

Testbed

- ❑ NVIDIA A30 GPU (**56** SMs)
- ❑ Intel Xeon Platinum 8468V (**96** Cores)
- ❑ Latest CUDA version

Workloads

Workload	Model	Operation	Batch Size	MPS Thread Limit (%)
A	ResNet101	Inference	1, 2, 4, 8	16
	Bert-base-uncased	Training	16	
B	Distil-bert	Inference	1, 2, 4, 8	16
	Bert-base-uncased	Training	16	
C	Bert-base-uncased	Inference	1, 2, 4, 8	16
	Bert-base-uncased	Training	16	



Testbed & Workloads

Testbed

- ❑ NVIDIA A30 GPU (**56** SMs)
- ❑ Intel Xeon Platinum 8468V (**96** Cores)
- ❑ Latest CUDA version

Workloads

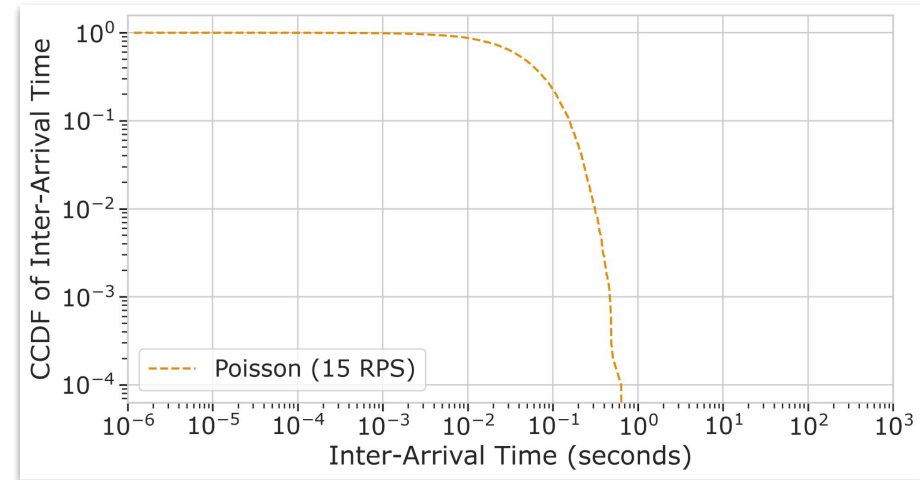
Workload	Model	Operation	Batch Size	MPS Thread Limit (%)
A	Bert-base-uncased	Training	1, 2, 4, 8, 16	60, 80, 80, 90, 40, 20, 20, 10
B	Distil-bert Bert-base-uncased	Inference Training	1, 2, 4, 8 16	60, 80, 80, 90 40, 20, 20, 10
C	Bert-base-uncased	Training	1, 2, 4, 8, 16	60, 80, 80, 90, 40, 20, 20, 10



Inter-Arrival Times

Poisson

- ❑ Exponentially distributed
- ❑ No long gaps
- ❑ Low variability and burstiness

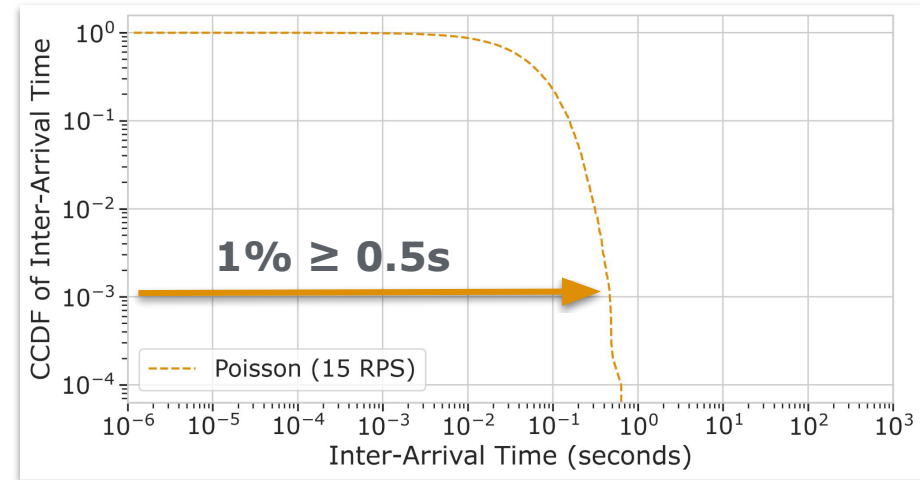


Inter-arrival Times

Inter-Arrival Times

Poisson

- ❑ Exponentially distributed
- ❑ No long gaps
- ❑ Low variability and burstiness



Inter-arrival Times

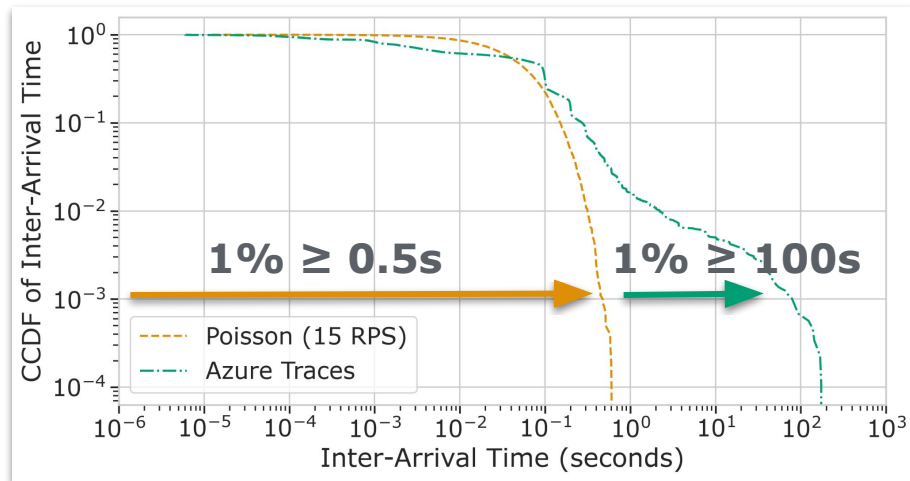
Inter-Arrival Times

Poisson

- ❑ Exponentially distributed
- ❑ No long gaps
- ❑ Low variability and burstiness

Real-world Traces

- ❑ Same median inter-arrival times
- ❑ Rare long gaps ($\geq 100s$)

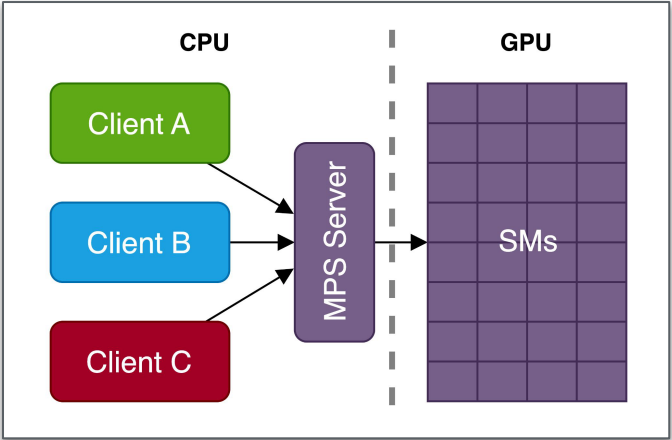


Inter-arrival Times

Systems under Test

Transparent	Dynamic	Autonomous

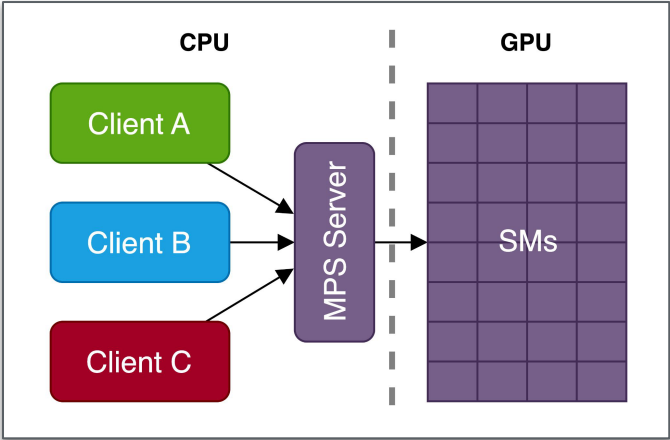
Systems under Test



Transparent Dynamic Autonomous

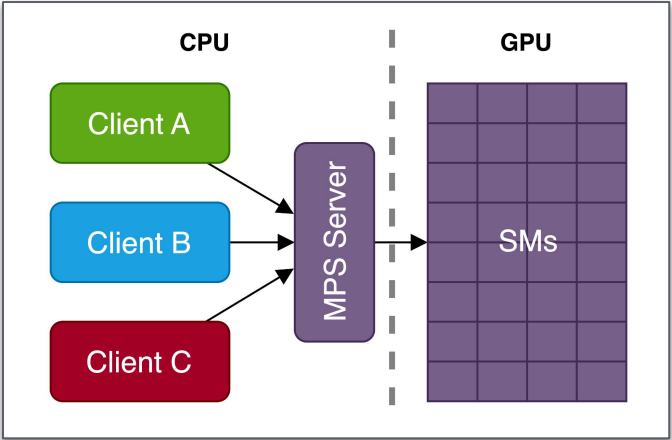
MPS **U**nlimited

Systems under Test



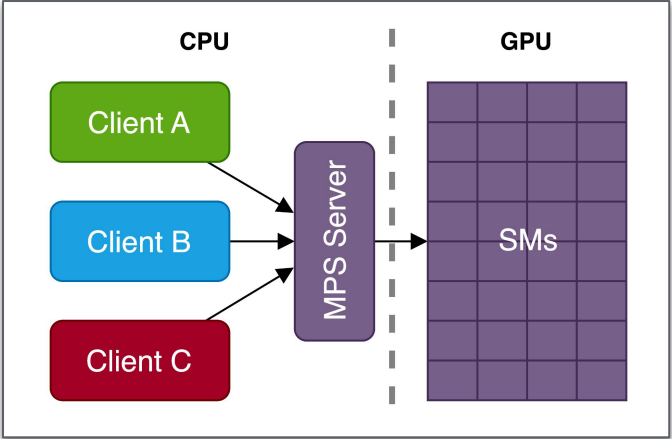
	Transparent	Dynamic	Autonomous
MPS U nlimited	×		

Systems under Test



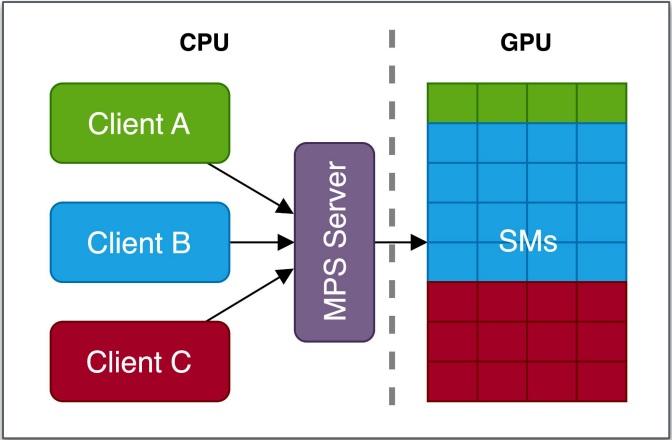
	Transparent	Dynamic	Autonomous
MPS U nlimited	×	✓	

Systems under Test



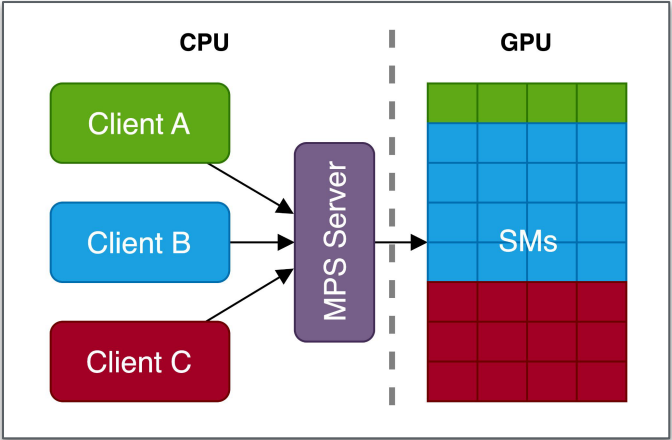
	Transparent	Dynamic	Autonomous
MPS U nlimited	✗	✓	✓

Systems under Test



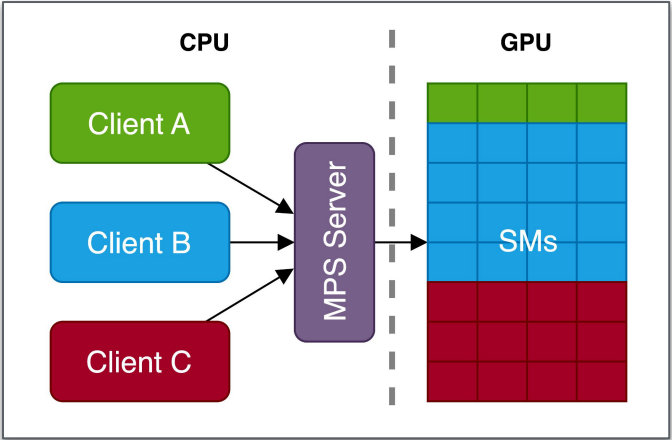
	Transparent	Dynamic	Autonomous
MPS U nlimited	✗	✓	✓
MPS L imited			

Systems under Test



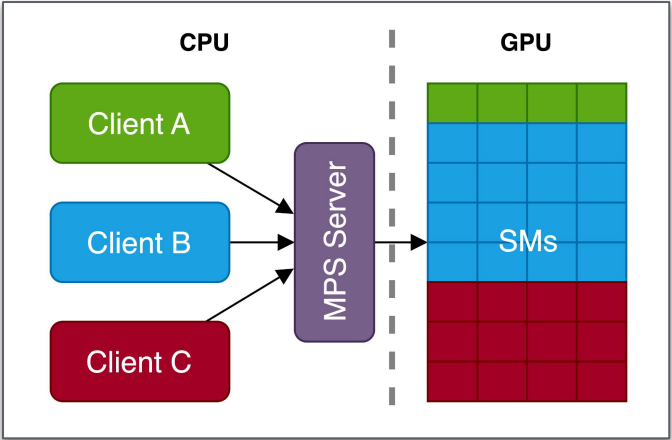
	Transparent	Dynamic	Autonomous
MPS U nlimited	×	✓	✓
MPS L imited	×		

Systems under Test



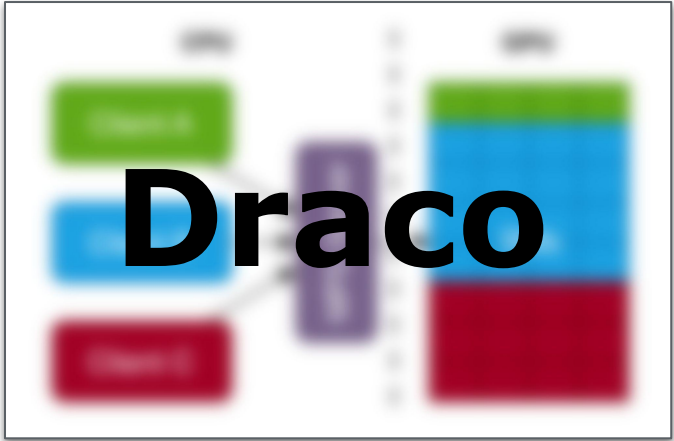
	Transparent	Dynamic	Autonomous
MPS U nlimited	×	✓	✓
MPS L imited	×	×	

Systems under Test



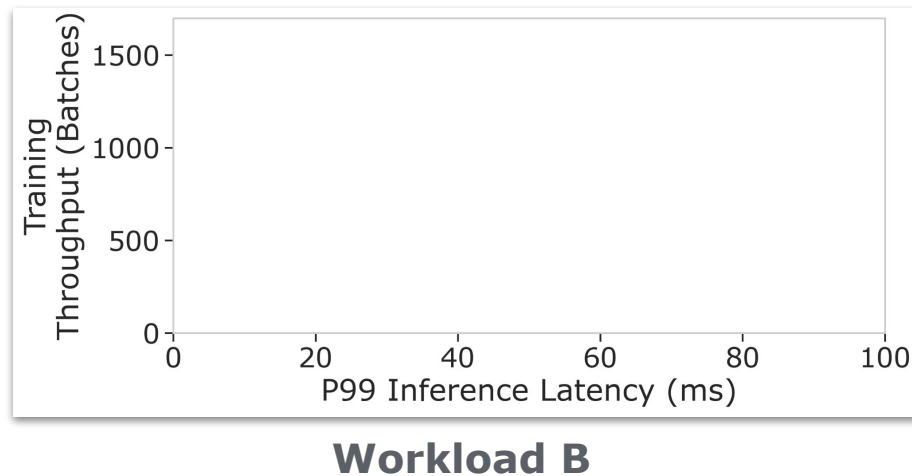
	Transparent	Dynamic	Autonomous
MPS U nlimited	X	✓	✓
MPS L imited	X	X	X

Systems under Test

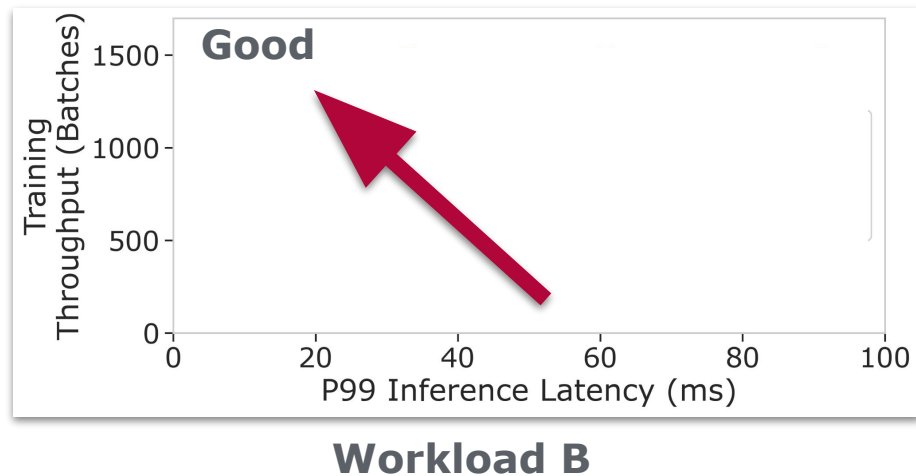


	Transparent	Dynamic	Autonomous
MPS U nlimited	✗	✓	✓
MPS L imited	✗	✗	✗
Draco	✓	✓	✓

Results



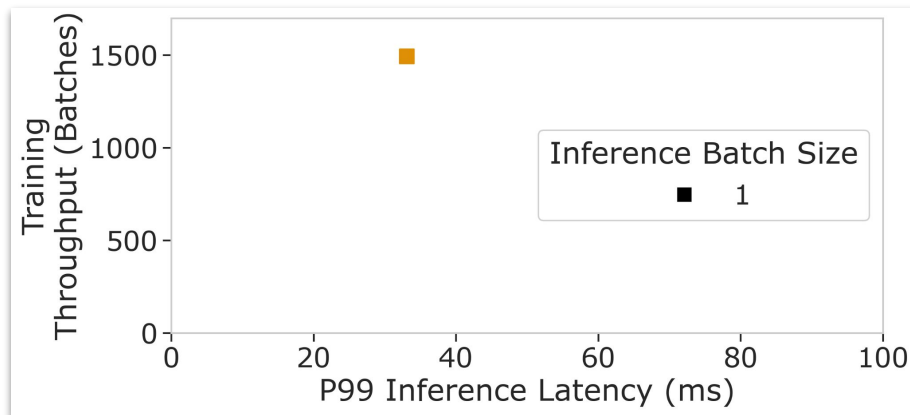
Results



Results



■ MPS (U)

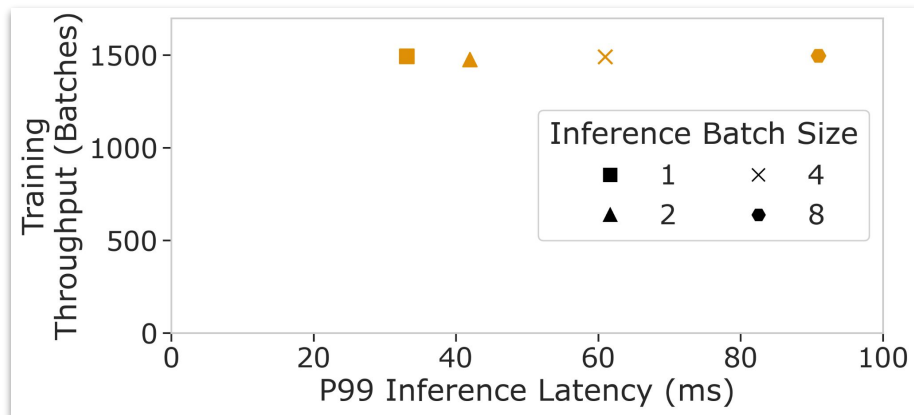


Workload B

Results



■ MPS (U)

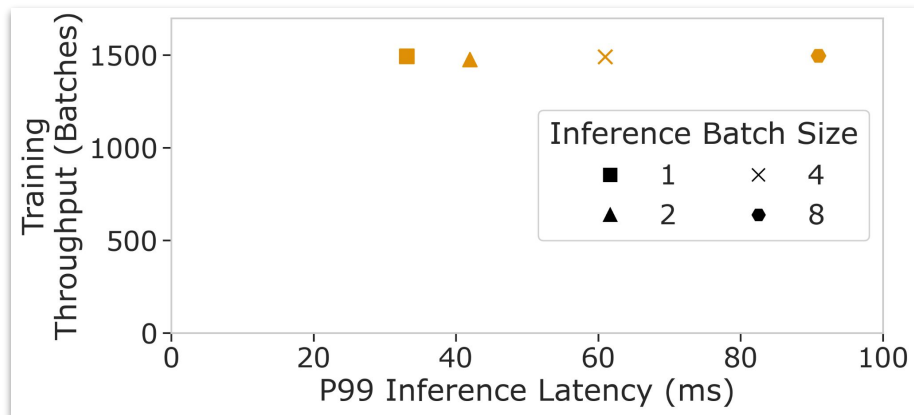


Workload B

Results

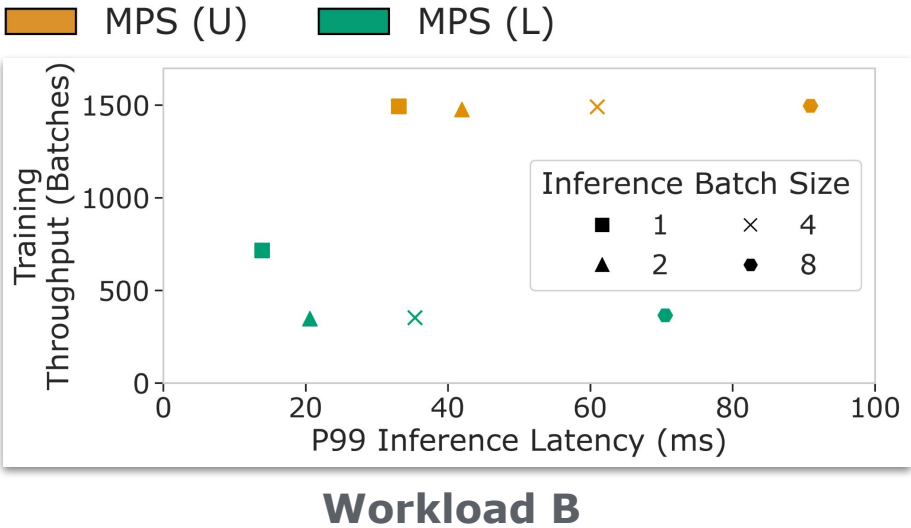
- ❏ MPS (U) favours throughput over latency

■ MPS (U)



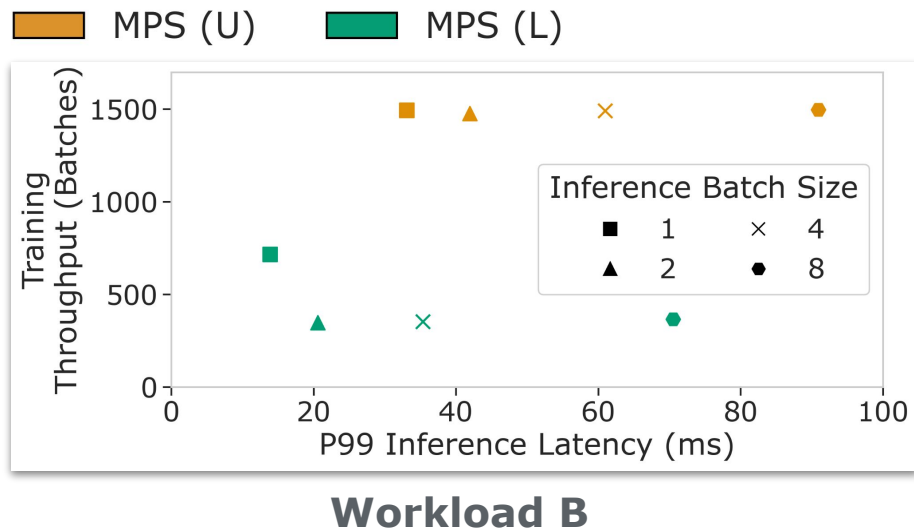
Results

❏ MPS (U) favours throughput over latency



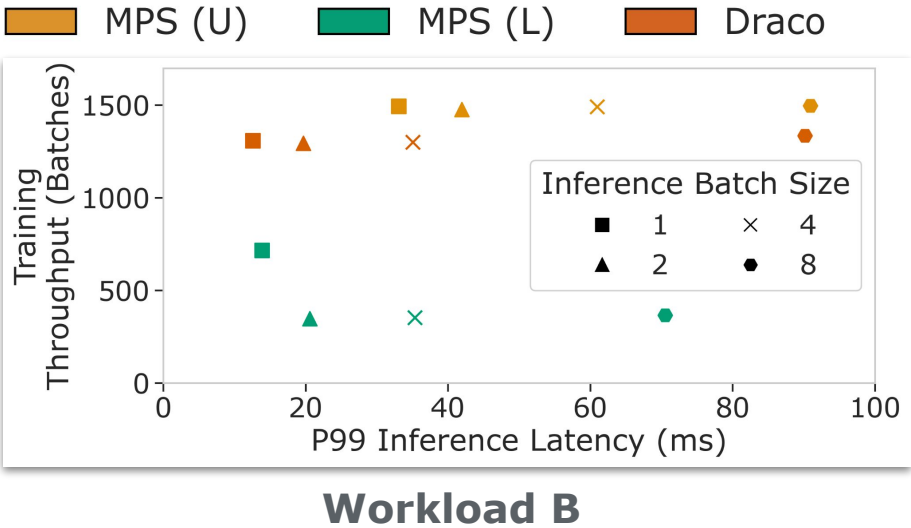
Results

- ❑ MPS (U) favours throughput over latency
- ❑ MPS (L) favours latency over throughput



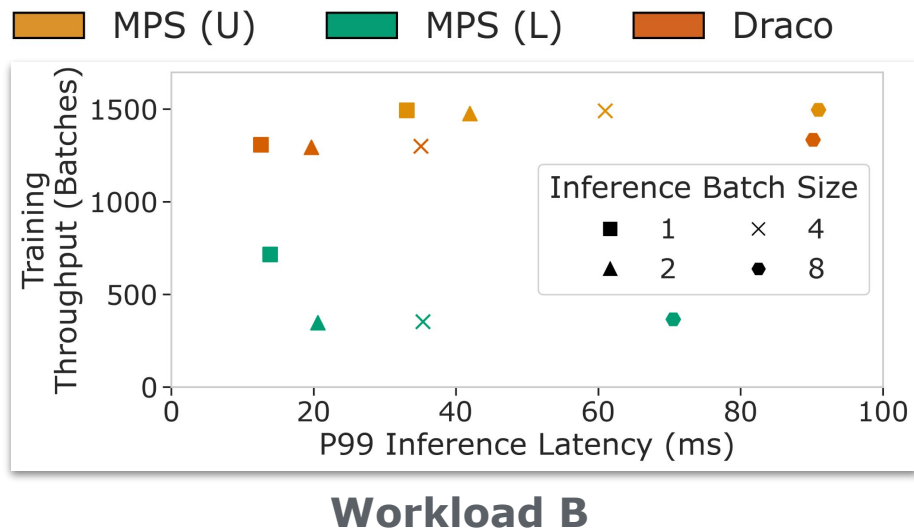
Results

- ❑ MPS (U) favours throughput over latency
- ❑ MPS (L) favours latency over throughput



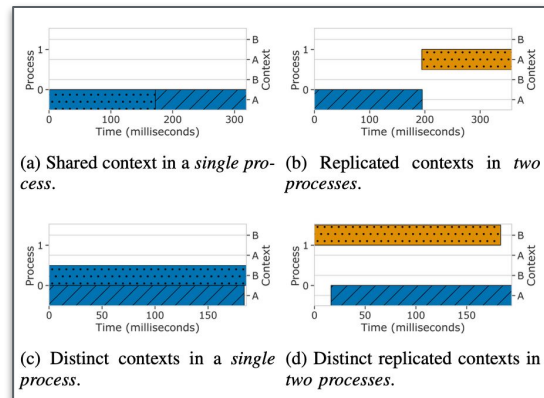
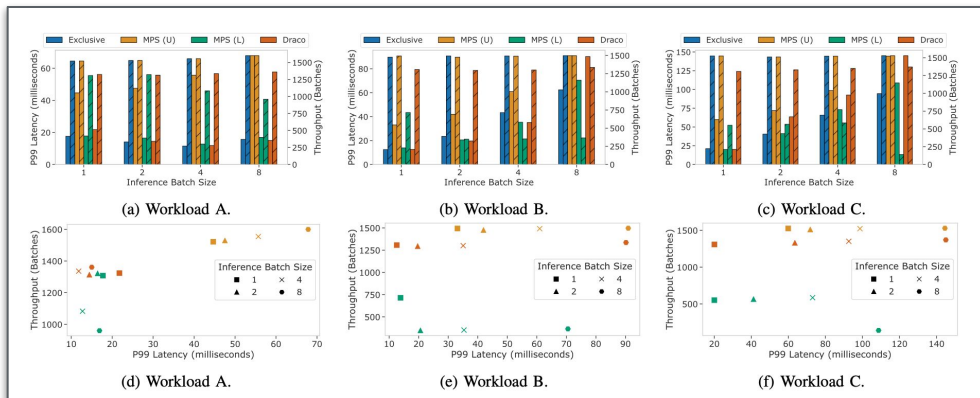
Results

- ❑ MPS (U) favours throughput over latency
- ❑ MPS (L) favours latency over throughput
- ❑ Draco inference latency matches MPS (L) **and** achieves high training throughput



Results

Workloads A, C and more experiments in the paper



Challenges
& Goals

System
Design

Summary

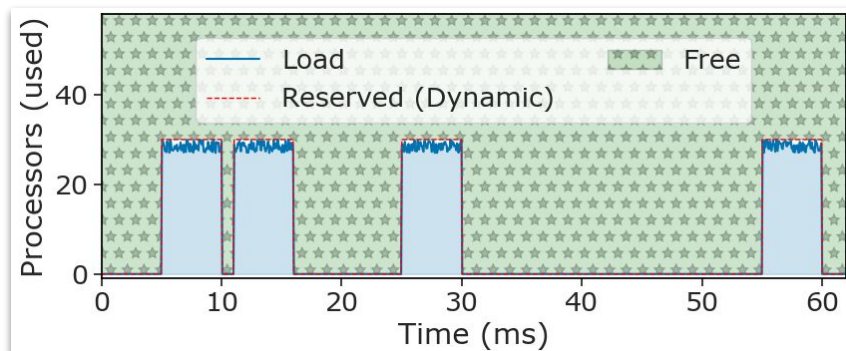


Requirements
& Contributions

Evaluation

Summary

- ❑ Draco often outperforms MPS
- ❑ Low inference latency
- ❑ High training throughput
- ❑ Draco is **applicable** and **autonomous**
- ❑ Not limited to ML workloads
- ❑ Open source project
- ❑ Schedule multiple inference clients



Dynamic Partitioning



Thanks for your attention! Questions?



Theo.Radig@hpi.de

SPONSORED BY THE



Federal Ministry
of Education
and Research

