Predicting workload colocations under GPU spatial sharing

Bing-Shiun Han

RPE committee

Professor Dongyoon Lee (Chair)

Professor Anshul Gandhi (Co-advisor)

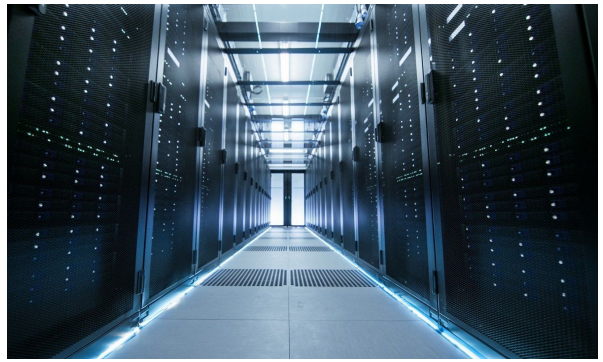
Professor Zhenhua Liu (Co-advisor)

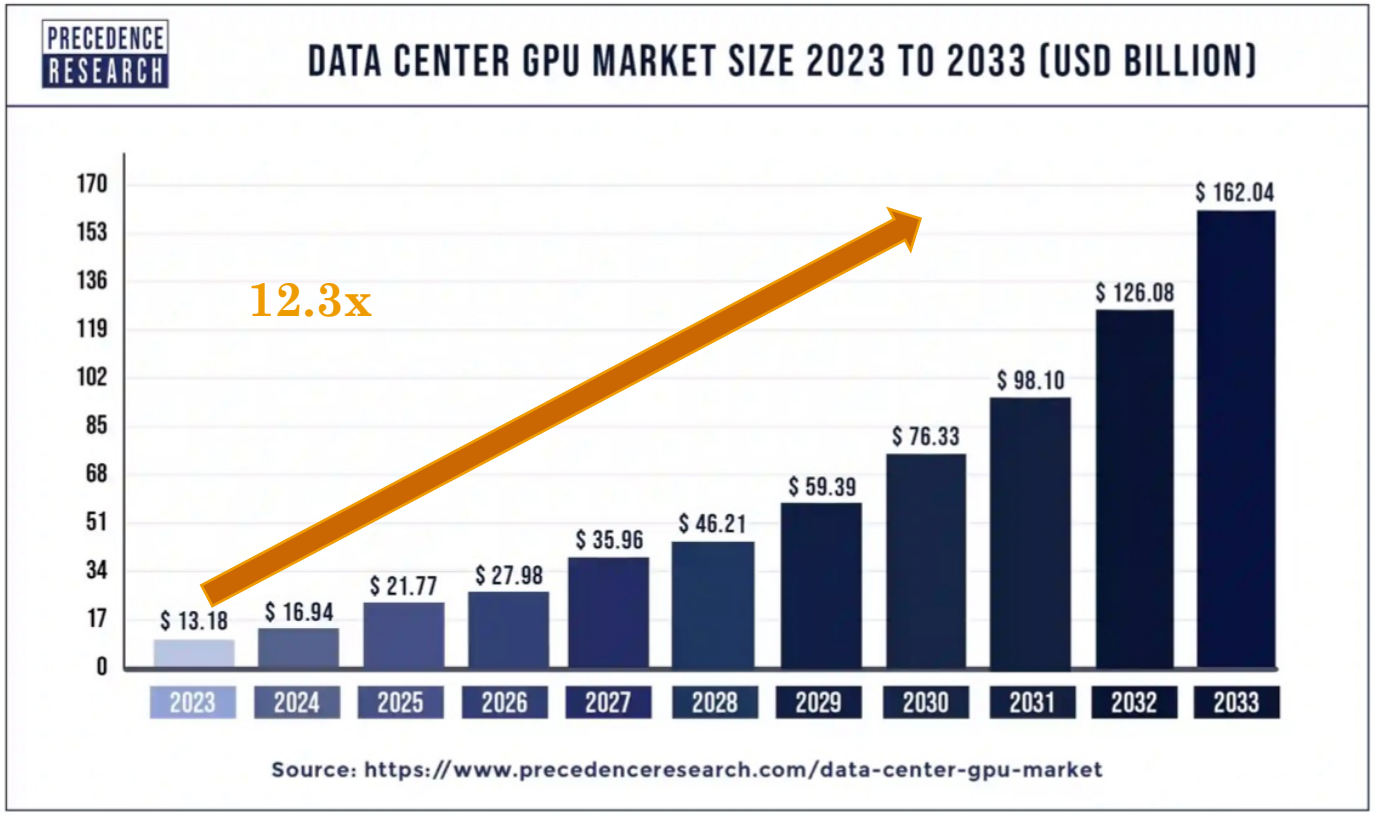
GPU as a Service is Booming

LLM

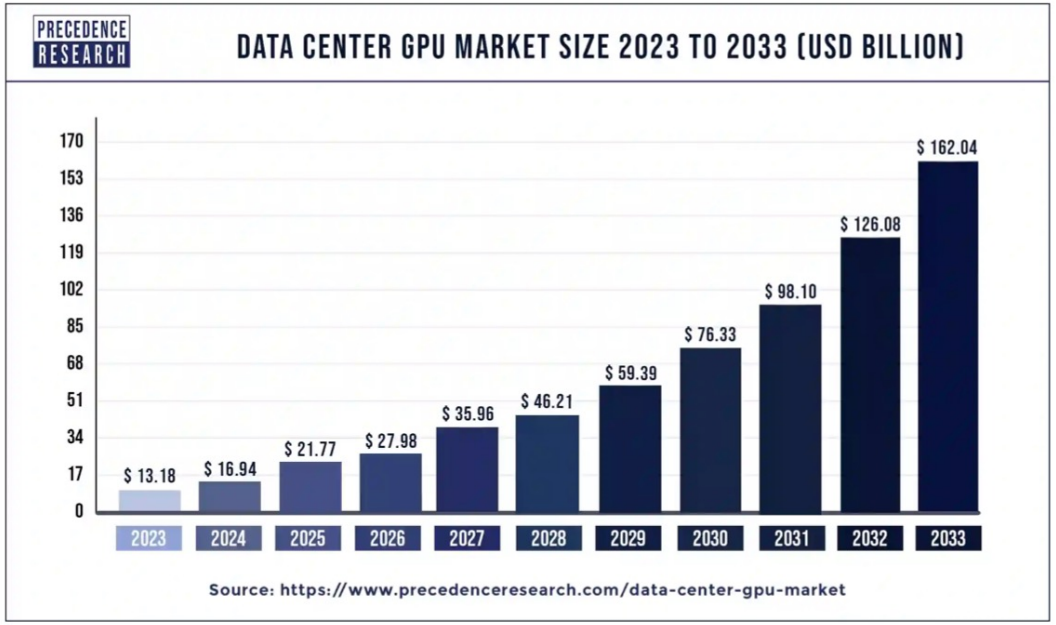
AI

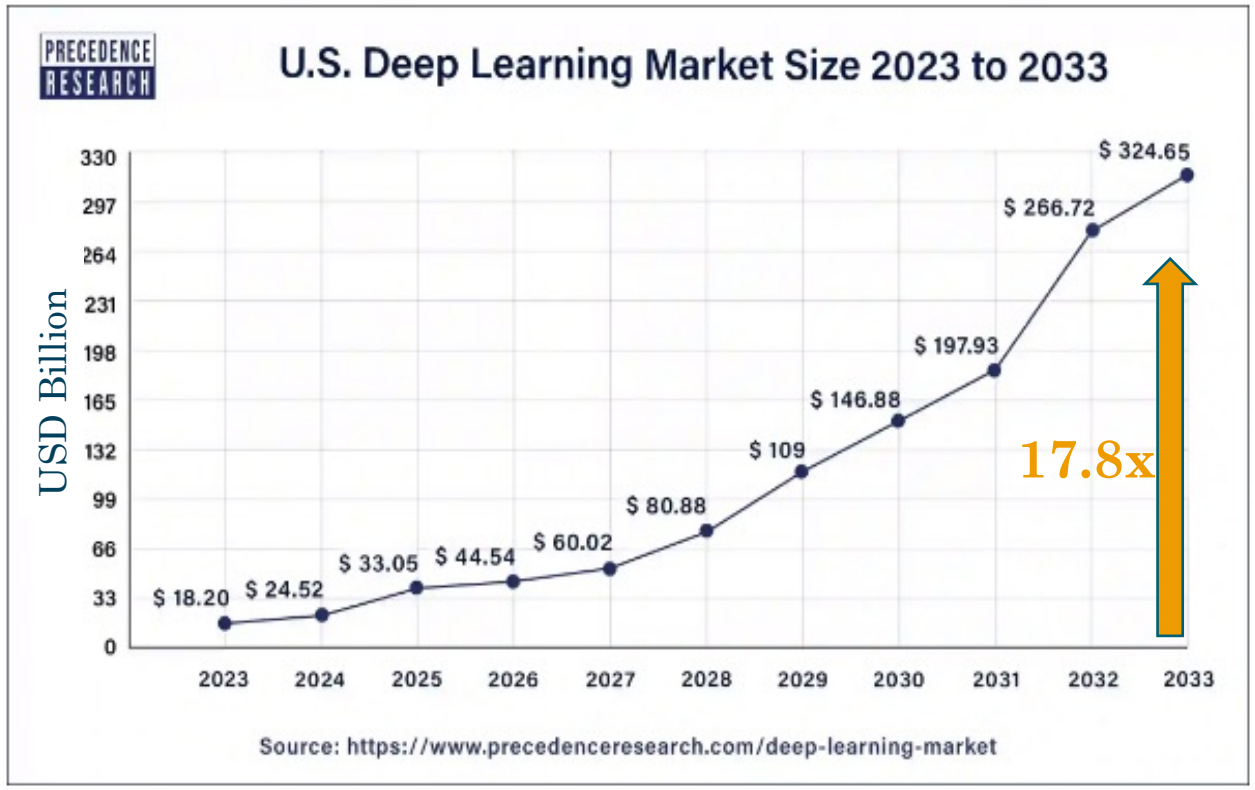
HPC





Deep Learning applications boom with GPU demands





Deep learning applications drive GPU market demand

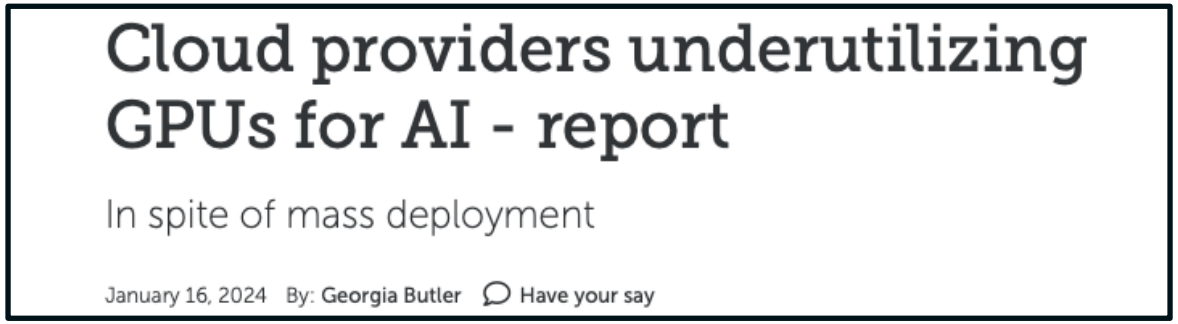
GPUs are underutilized in Datacenter

Big Cloud deploys thousands of GPUs for AI – yet most appear under-utilized

If AWS, Microsoft, Google were anywhere close to capacity, their revenues would be way higher

Tobias Mann

Mon 15 Jan 2024 // 13:29 UTC



Cloud providers underutilizing GPUs for AI - report

In spite of mass deployment

January 16, 2024 By: Georgia Butler Have your say

- Why are GPUs not fully used?
  - DL jobs are diverse. E.g. Light-weight inference jobs not fully utilizing data-center GPUs
  - Cloud providers provision more capacity than the expected usage to plan for peak demands

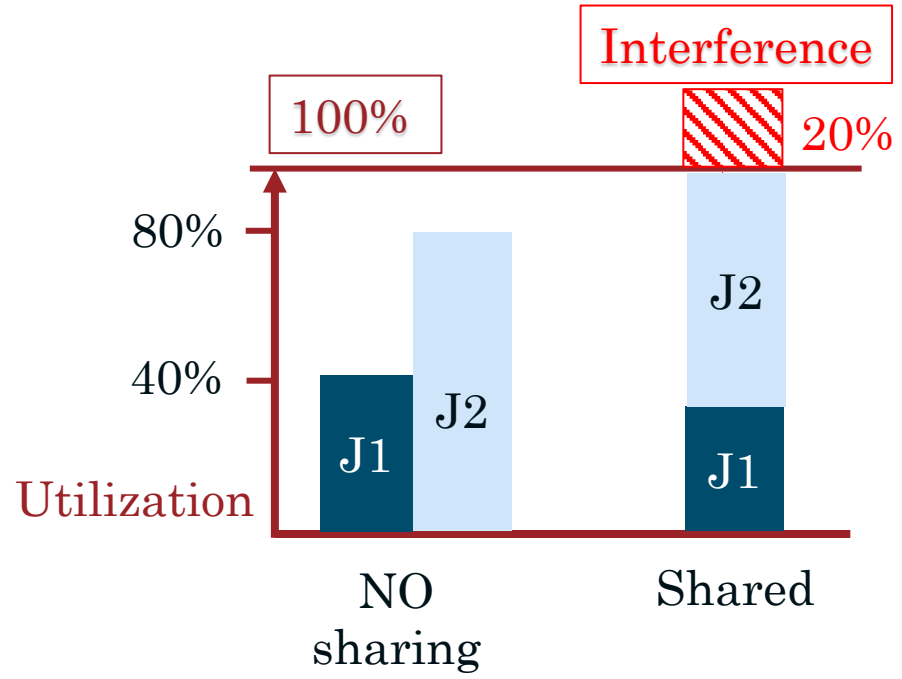
Can we increase GPU utilization?

Sources: https://www.theregister.com/2024/01/15/cloud_providers_gpu_analysis/

Colocating workloads…

- Improves GPU utilization
- Decreases completion time
- Increases throughput

- Any downsides?
  - Utilization cap
  - **Interference caused by colocation**



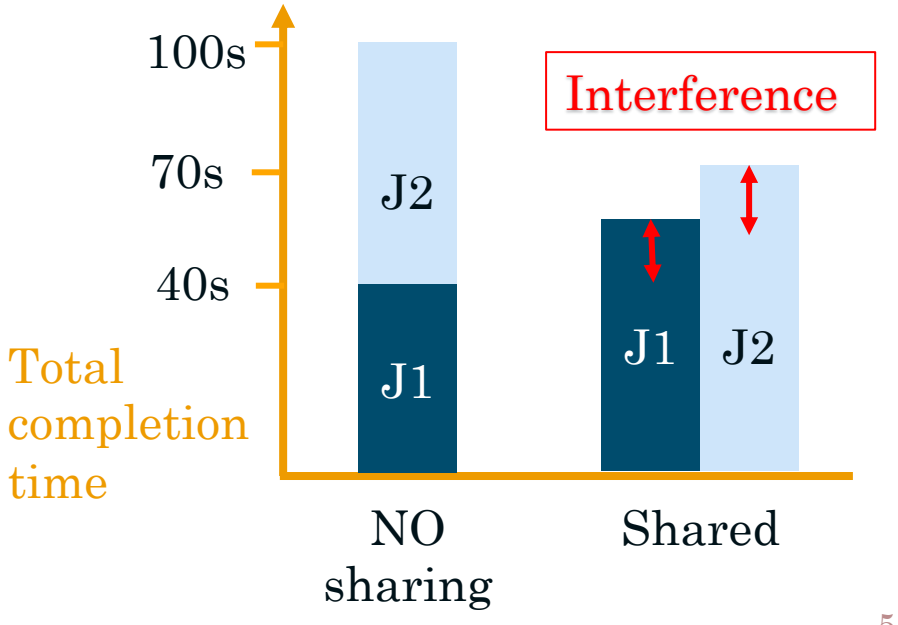
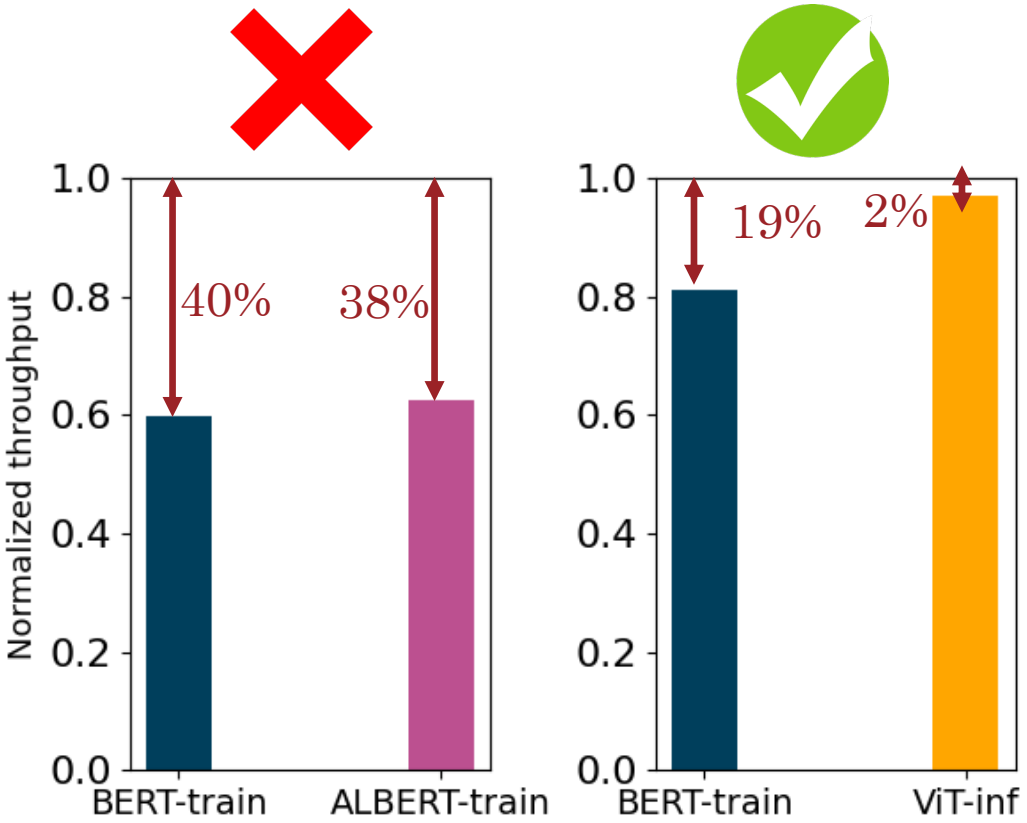


Illustration graph. Not experimental data

Colocating workloads brings unexpected interference

- Performance interference is **workload-dependent**
  - Training or inference
  - Compute intensity per model
  - Data process between host and GPU
- System-level interference factors
  - GPU sharing policies
  - hardware architectures



Could we predict interference for a better colocation pair?

Predicting interference is challenging

- Coarse-grained, overall resource usage pattern are not sufficient for prediction
  - Requires at least **workload-level** metrics to obtain precise colocated performance
  - Workload-specific metrics are not generalizable
- Profiling workload metrics have overheads
  - **Huge search space** to profile all possible colocations
  - **Large training set** for ML-based approach
  - **Long training time** for ML models

Challenges in GPU interference for workload colocation:
1. Minimizing performance degradation
2. Discovering metrics for predicting interference
3. Profile overheads

Problem Statement

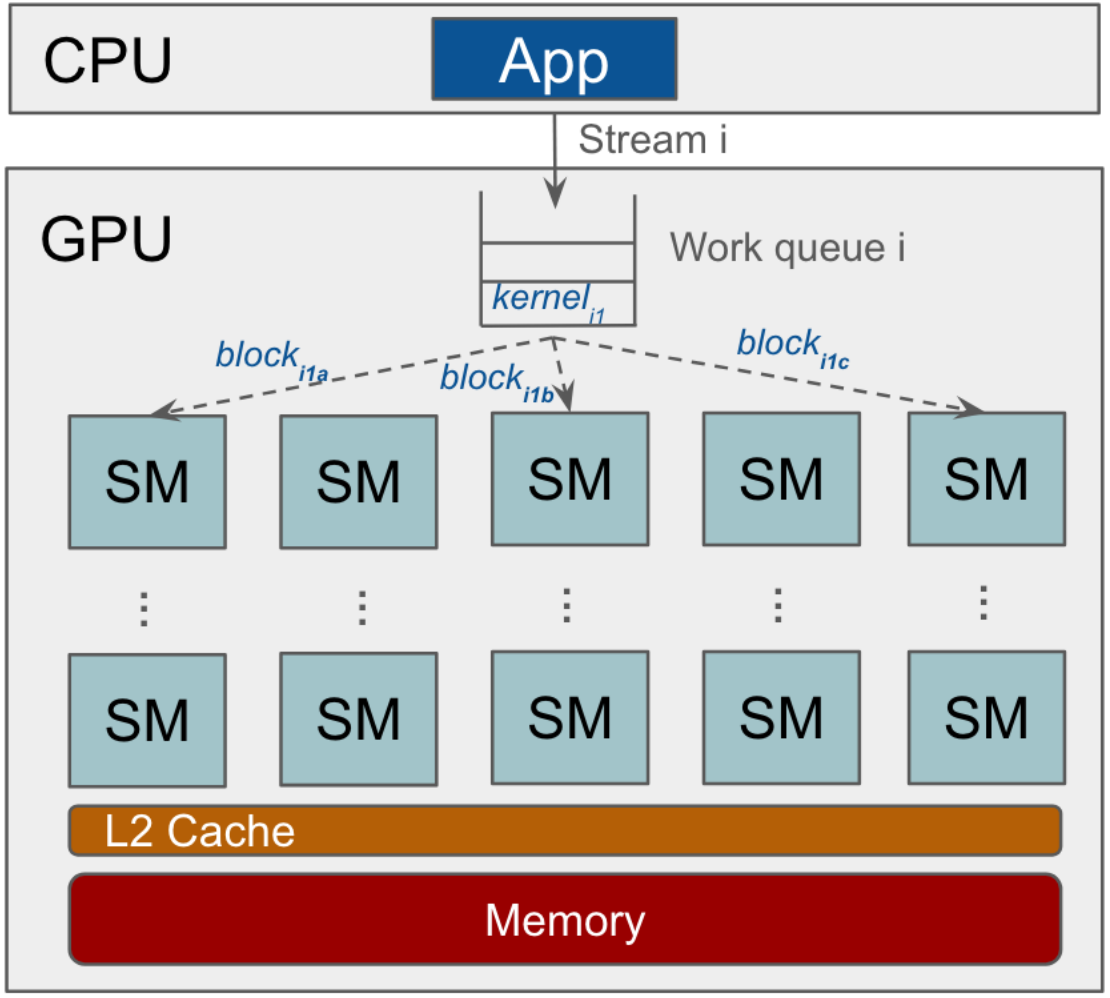
How to choose colocated DL workloads to improve GPU utilization while minimizing performance interference with low profiling overhead?

- Our solution
  - **KACE -** **K**ernel-**A**ware **C**olocation for **E**fficient GPU spatial sharing
  - Propose a lightweight, **prediction-based** approach to effectively colocate GPU workloads
  - Use exclusive **kernel metrics** collected **offline** to eliminate expensive online profiling

Talk outline

- **Background**
  - GPU architecture
  - GPU sharing
  - GPU scheduling flow
- Prior Work
- System Design
- Evaluation
- Conclusion and Future Work

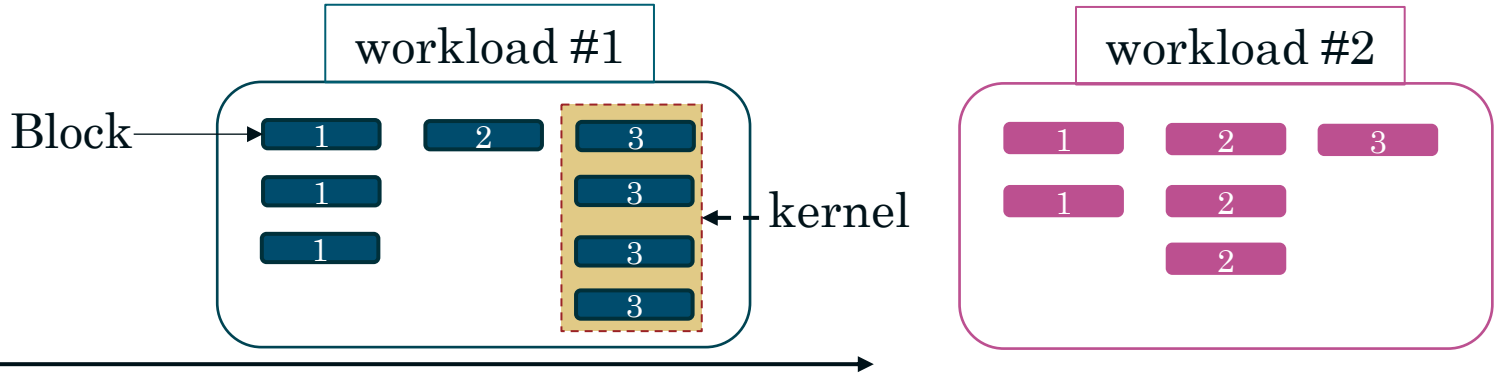
GPU architecture



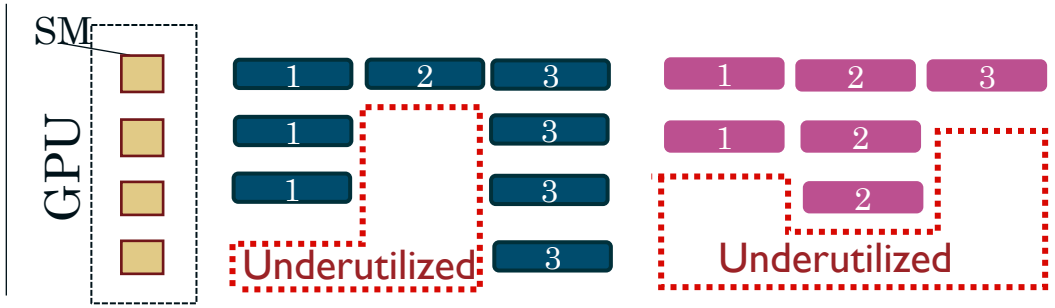
- Streams
  - A queue of GPU operations that are executed in a specific order
- Kernel
  - A **function** meant to execute on GPU
- Block
  - A group of threads that can be run in parallel
- SM – Streaming Multiprocessor
  - Basic compute unit. **"Core"** in CPU

[Eurosys'24] STRATI, F., MA, X., AND KLIMOVIC, A. Orion: Interference-aware, Fine-grained GPU Sharing for ML Applications, p. 1075–1092.

GPU sharing

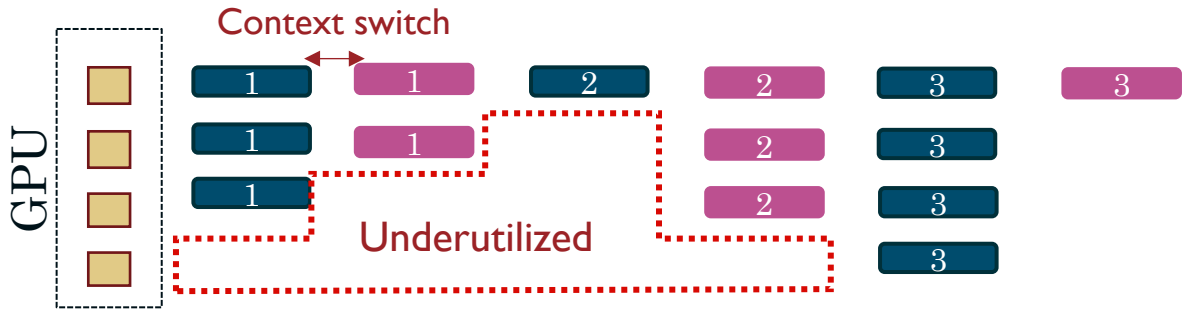


- No sharing



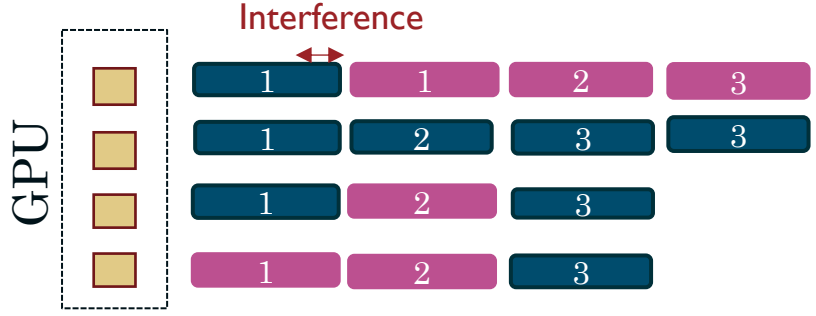
- Execute workload sequentially
- Underutilization

- Temporal sharing



- Fairness for multiple users
- Context switch cost
- No concurrency – GPU still underutilize

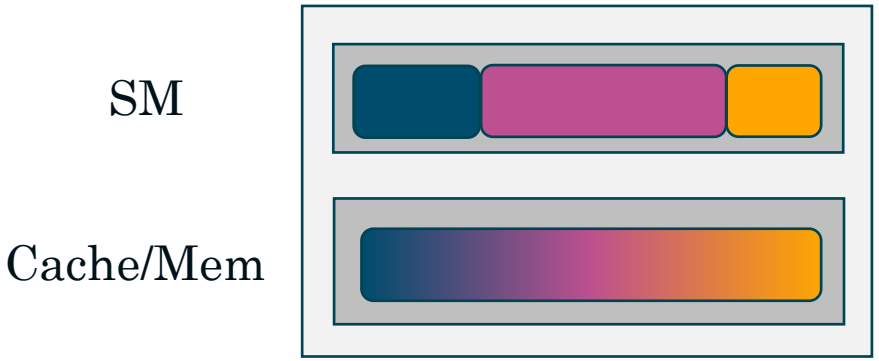
- Spatial sharing



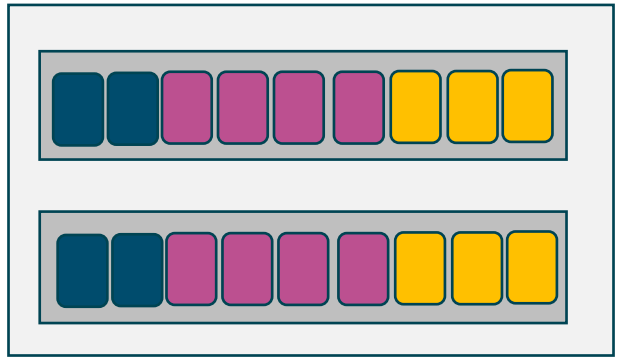
- Higher utilization due to concurrent block execution
- Interference from non-isolated resource.

Spatial sharing - MPS and MIG

workload1 workload2 workload3



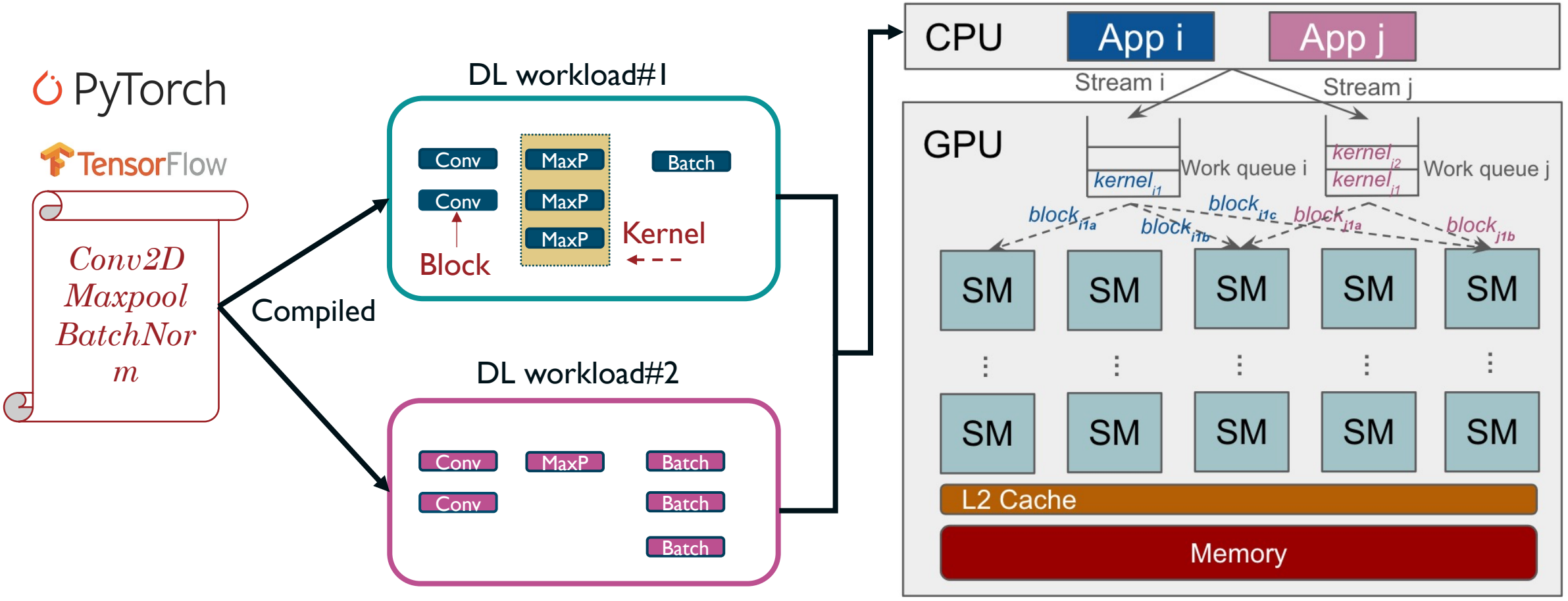
- Nvidia MPS
  - Processes launch simultaneously by sharing their schedule resources
  - **Compute** isolation with workload SM assignment through **logic** partitioning
  - Memory not isolated for workloads



- Nvidia MIG
  - **Compute and memory** isolation through **physical** partitioning
  - Limited partition combinations
    - 19 combinations for A100
  - Only available on new datacenter GPUs
  - Repartition overhead

We use MPS for its wider HW support and flexible configurations

GPU scheduling flow



Talk outline

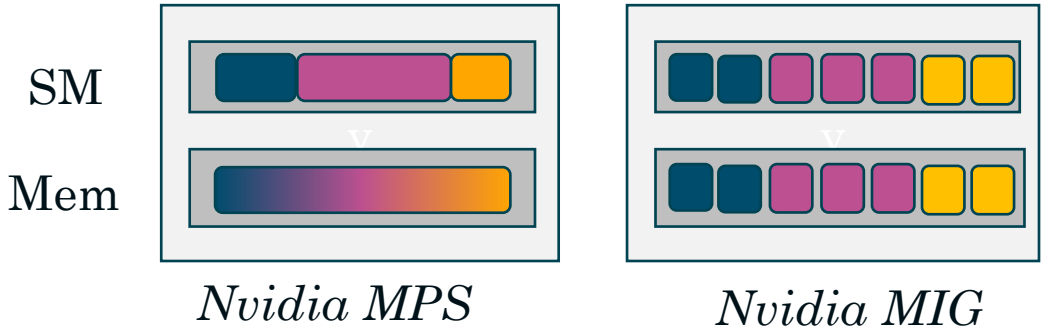
- Background
- **Prior work**
  - **Colocation prediction**
    - **GPU**
    - **Non-GPU**
  - **GPU Scheduling for DL workloads**
    - **Temporal sharing**
    - **Spatial sharing**
    - **No sharing**
- System design
- Evaluation
- Conclusion

Prior work – colocation predictions

	Profile type	GPU Share type	Profiled metrics	Train/ inference	Difference vs our work
MISO, SOCC'22	Online	Spatial	Completion Time	Train	Excessive online profiling, HW support, train only
Xu et al, Hotcloud'19	Offline	Temporal	Kernel	Both	Temporal prediction
Horus, PDS'22	Offline	Temporal	DL graph	Train	Temporal, need DL semantics, train only
This RPE	**Offline**	**Spatial**	**Kernel**	**Both**	**Spatial prediction, offline, train+inf**

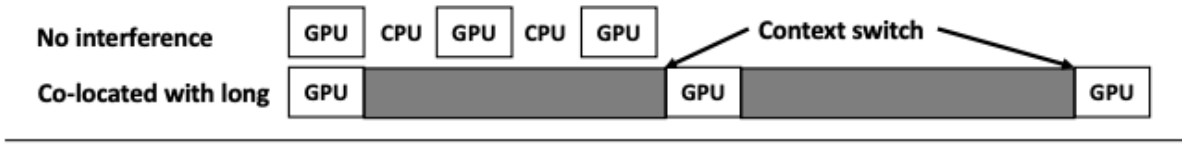
Our work focuses on accurate colocation predictions under spatial-sharing with both training and inference

Colocation predictions



- MISO, SOCC'22
  - Predicts optimal GPU MIG partitions by profiling colocation **training** speedups with MPS
    - Profile with MPS to avoid MIG hardware reset overheads (4s each colocation)
  - **Excessive online profiling**. Require multiple dry-runs for each colocation
    - Each colocation needs 3 MPS profile-runs to obtain best MIG partition
  - Lesson - Predicting interference is feasible. Should try on offline profile methods.

Colocation predictions



- Xu et al, Hotcloud'19
  - Predicts interference on **temporal-shared** VMs with individual kernel and system profiles
  - Highlights the distribution of kernel duration since long kernels block short ones from executing within a context window
  - Use Random Forest for prediction. Mean square error < 0.2.
  - The solution is not applicable for spatial-sharing. It is specified for **context switches.**
  - Lesson – simple regression model is sufficient for predicting colocations

Colocation predictions

- Horus, PDS'22
  - Extracts **DL computation graph metrics** (MaxPool, Conv, Relu counts), FLOPs, and memory to predict GPU utilization with classification models
  - Does not apply on **DL inference** with more unpredictable pattern and latency requirements
  - Lesson – application-specific metrics are helpful but comes with limitation

Our work focuses on predicting interference of **both DL train and inference** under spatial sharing

GPU scheduling

	GPU Share type	Schedule granularity	Train /Inference	Control knob
Antman, OSDI'20	Temporal	Iteration	Train	Memory states
Gandiva, OSDI'18	Temporal	Iteration	Train	Memory states
TGS, NSDI'23	Temporal	Kernel	Train	Kernel ingestion
Gavel, OSDI'20	Temporal	Workload	Both	Schedule policy, heterogenous HW
Reef, OSDI'22	Spatial	Kernel	Inference	Kernel preemption
Salus, MLSys'24	Spatial	Iteration	Train	Memory state
Orion, Eurosys'24	Spatial	Kernel	Both	Kernel types
This RPE	Spatial	Workload	Both	Workload profile, system throughput

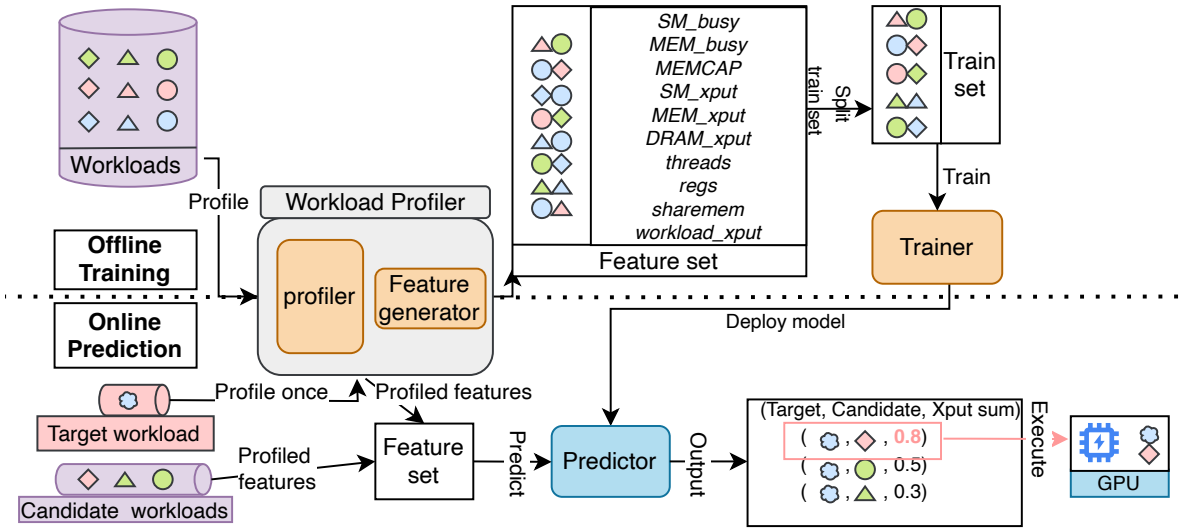
Prior work – GPU scheduling with spatial sharing

- Reef
  - Improves DL **inference** serving by enabling **kernel-level preemptions** on the GPU based on the idempotency of inference matrix multiplication
  - Only support AMD GPUs due to runtime and Apache TVM compiler open-source support
- Salus
  - **Saves memory space** at the **iteration-level** for training tasks by quickly releasing intermediate outputs while persisting model parameters.
  - Not applicable to inference due to training's iteration periodic pattern.
- Orion
  - Colocates **memory and compute-intensive kernels** to reduce interference.
  - Maintain two-level GPU streams to prioritize latency-sensitive workloads.
  - Requires advanced CUDA support for kernel-level scheduling with customed streams.

Fine-grained spatial sharing requires hardware support

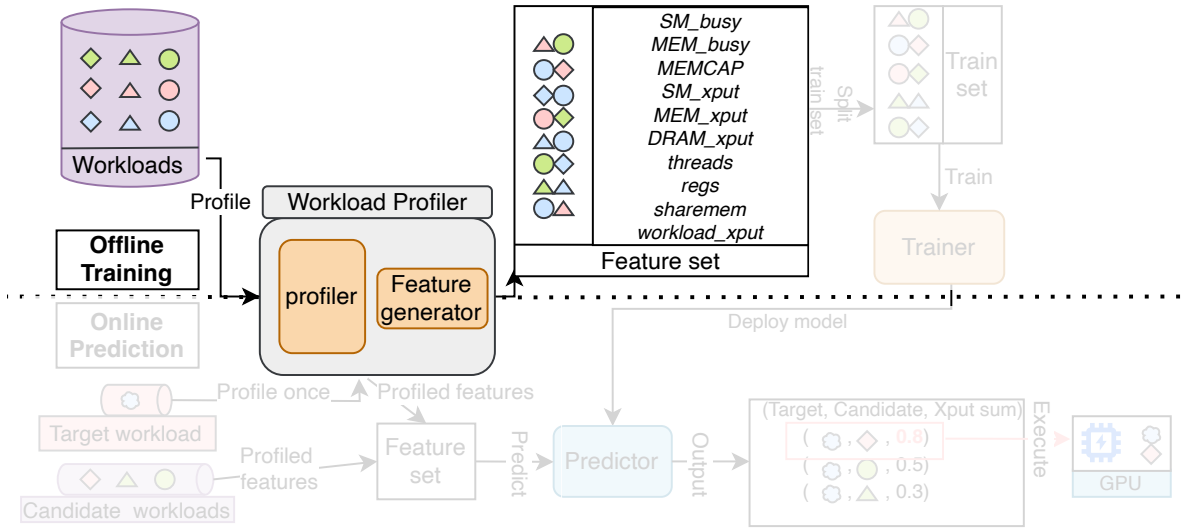
- Introduction
- Background
- Prior work
- **System design**
  - **Overview**
  - **Profiler**
  - **Trainer**
  - **Predictor**
- Evaluation
- Conclusion

System design of KACE



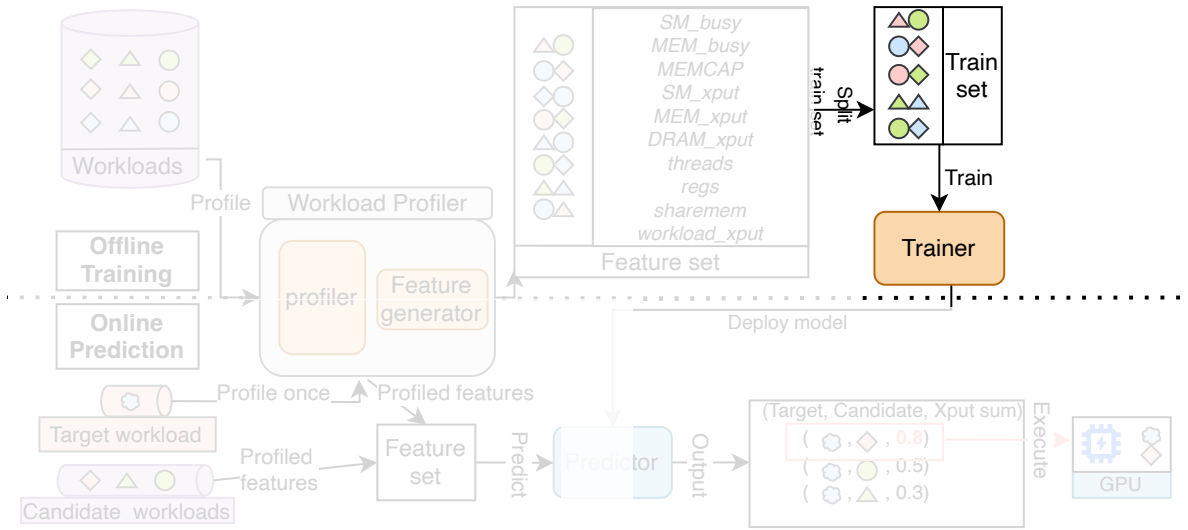
- Offline training
  - Profile workload features
  - Model training
- Online predictions
  - Combine target and candidate features
  - Predicts workload pairs with highest throughput sum

System design - Profiler



- Profile **individual** workloads.
- GPU system metrics (*nvidia-smi*)
  - Quick overview, not kernel-specific
  - Utilization (SM busy rate)
  - Memory busy rate
  - Memory capacity
- Kernel metrics (*Nsight Compute*)
  - Fine-grained profile, requires profile tool
  - Memory throughput
  - Dram throughput
  - Num of Threads
  - Num of Registers
  - Shared memory
- Individual workload throughput

System design – Trainer



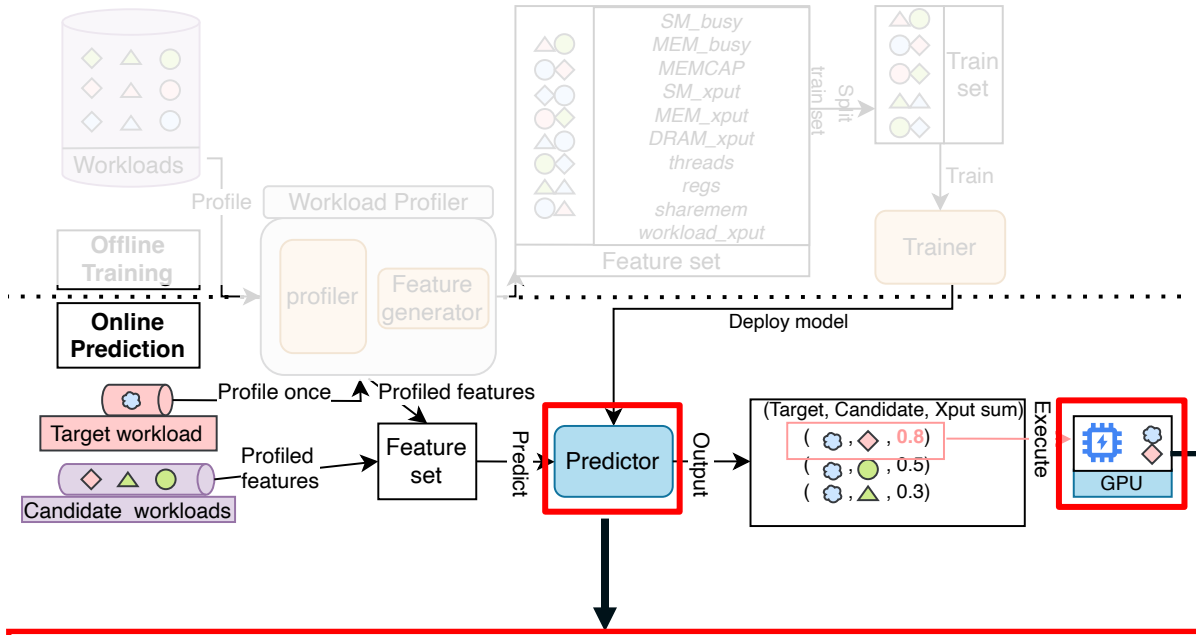
- Train regression model to predict the **colocated throughput sum**

$$\hat{y}_{\text{throughput sum}} = f(\vec{x}_{w_1} + \vec{x}_{w_2})$$

Aggregate features of two workloads

- sum – metrics with absolute value, e.g. Number of threads
- average – metrics with %, e.g. SM busy rate

System design – Predictor



- Extract from test set with all target workload combinations
- Use trained model to predict the colocated pair with **maximum throughput sum**
- Execute predicted pair on GPU

$$\hat{y}_{\text{throughput sum}} = \max(f(\vec{x}_{target} + \vec{x}_i))$$

$$i \in \text{Candidate workloads}$$

$$\text{selected candidate} = \arg\max(\hat{y})$$

Talk outline

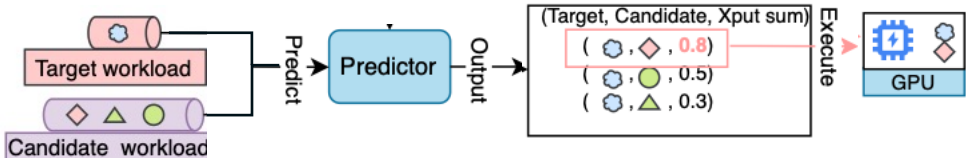
- Background
- Prior work
- System design
- **Evaluation**
  - **Methodology**
  - **Workload colocation**
- Conclusion and Future Work
- Future work

Evaluation - methodology

- Setup
  - Single node server in Chameleon Cloud. Hardware - 2 Intel Xeon Gold 6230 CPUs, 128GB RAM, and a 32GB NVIDIA V100 GPU. Software - PyTorch 1.13, CUDA12.3
- Prediction Models
  - **Linear Regression**, Random Forest, Neural Network, AutoML

Workload	Batch	Application	SM busy (%)	Mem busy (%)	Memcap (GB)
BERT-train	2,8,16	Recommendation	97.0	45.2	5.1
ViT-train		Image classification	97.2	37	17.6
ALBERT-train		Recommendation	97.2	45.1	7.1
BERT-inf		Recommendation	95.1	38.6	1.4
ViT-inf		Image classification	28.5	5.4	3.2
Whisper-inf		Speech recognition	44.2	19.6	11.7
Wav2vec2-inf		Speech recognition	18.9	6.6	12.2

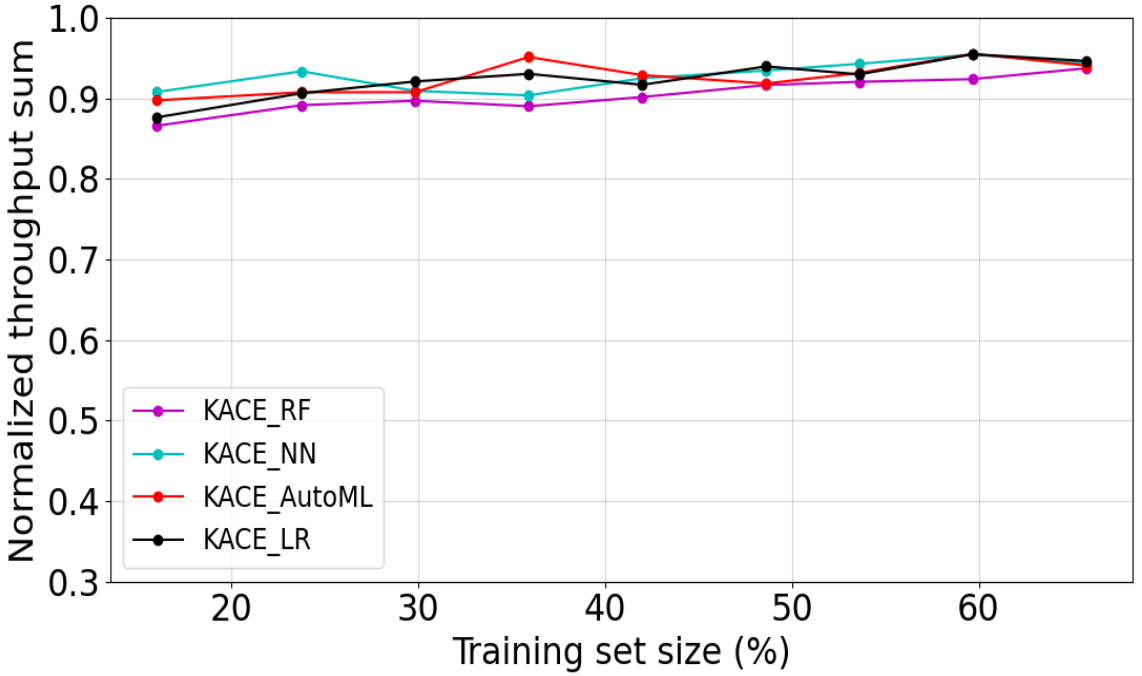
Evaluation - Workload colocation with tested ML models



$$\text{Normalized throughput sum} = \frac{X_{target} + X_i}{X_{Oracle}}$$

$i \in$ Workload candidates

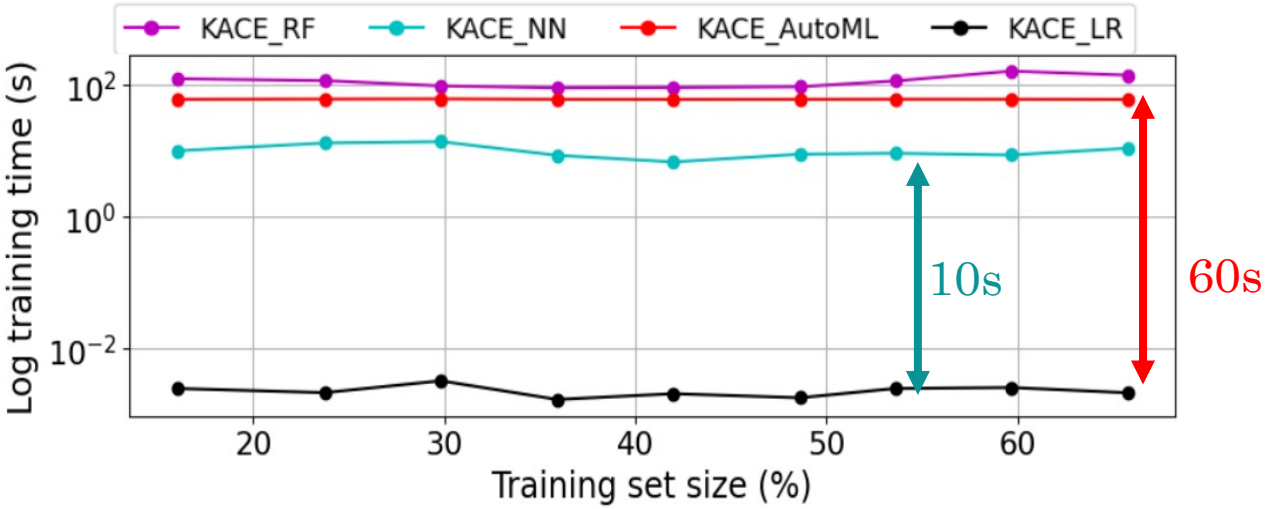
Oracle = workload pair with highest throughput sum



- Demonstrate how actual throughput is impacted with the predicted pair being executed
- Compared throughput sum with Oracle. Averaged with all 21 workloads as target.
- LR achieves similar throughput sum compared with other ML technique

KACE achieves strong performance across various models with small training size

Evaluation - Workload colocation with tested ML models



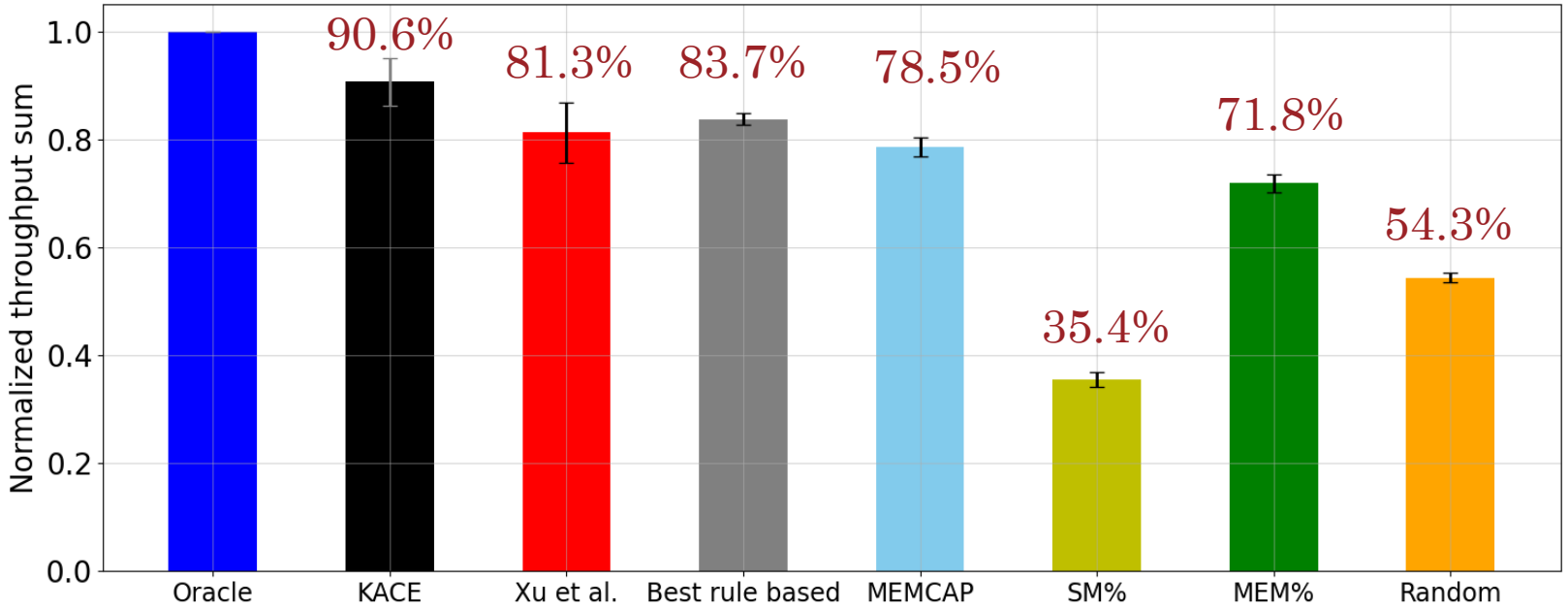
- RF requires grid search for better performance
- AutoML analyze a set of regression and NN models within time budget
- NN updates model parameters with back propagation
- LR requires least training time without parameter searches

We select Linear Regression for its accuracy and fast training time

Evaluation - baseline policies

- Baseline policies for workload colocations
  - Oracle – Impractical, optimal policy that selects workload pair with highest throughput sum
  - Xu et al – Predicts colocation under temporal-sharing with kernel metrics
  - Rule-based approaches with system metrics
    - MEMCAP – choose colocated workload with smallest GPU memory footprint
    - SM% – choose colocated workload with highest SM busy rate
    - MEM% – choose colocated workload with lowest GPU memory busy rate
    - Best rule based – best value of MEMCAP, SM%, MEM%
  - Random – mean of throughput sums of all possible colocations

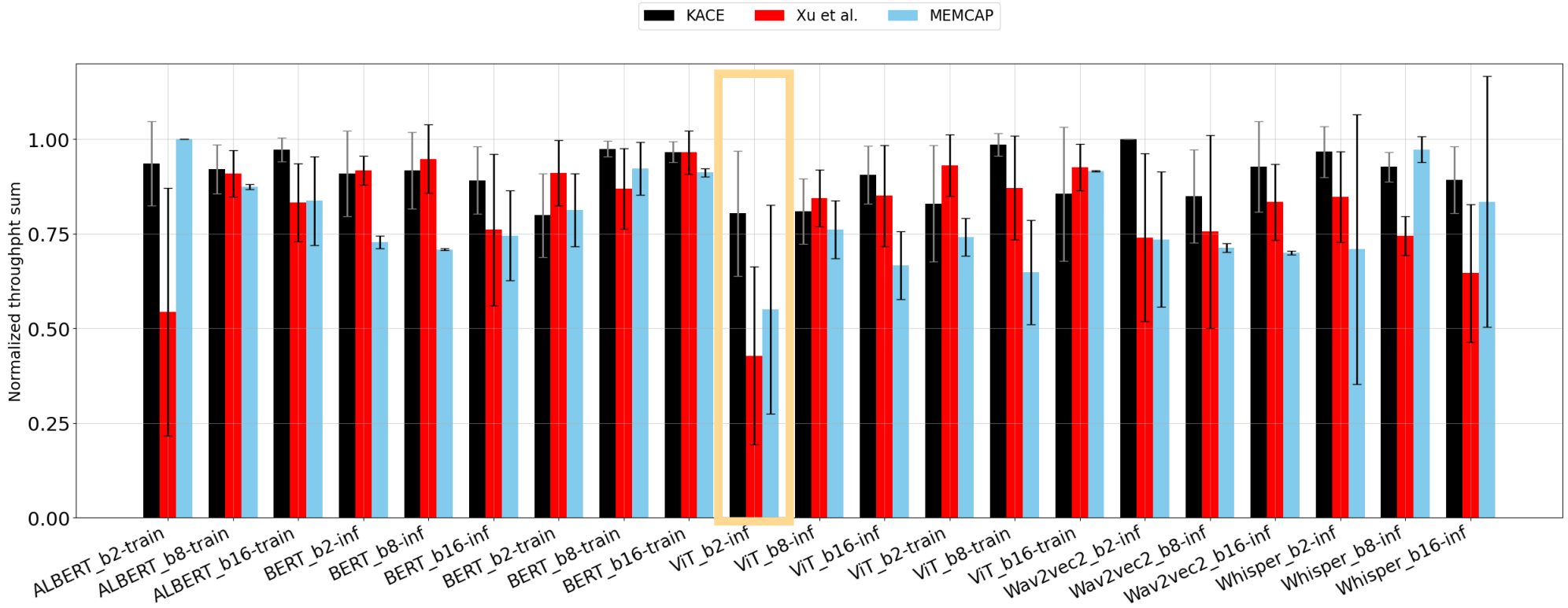
Evaluation - Workload colocation vs baseline policies



- Random performs poorly
- Rule-based approaches do not encapsulate kernel information
- Xu et al. targets temporal sharing
- KACE achieves 90% throughput sum compared with Oracle

KACE outperforms other baseline policies

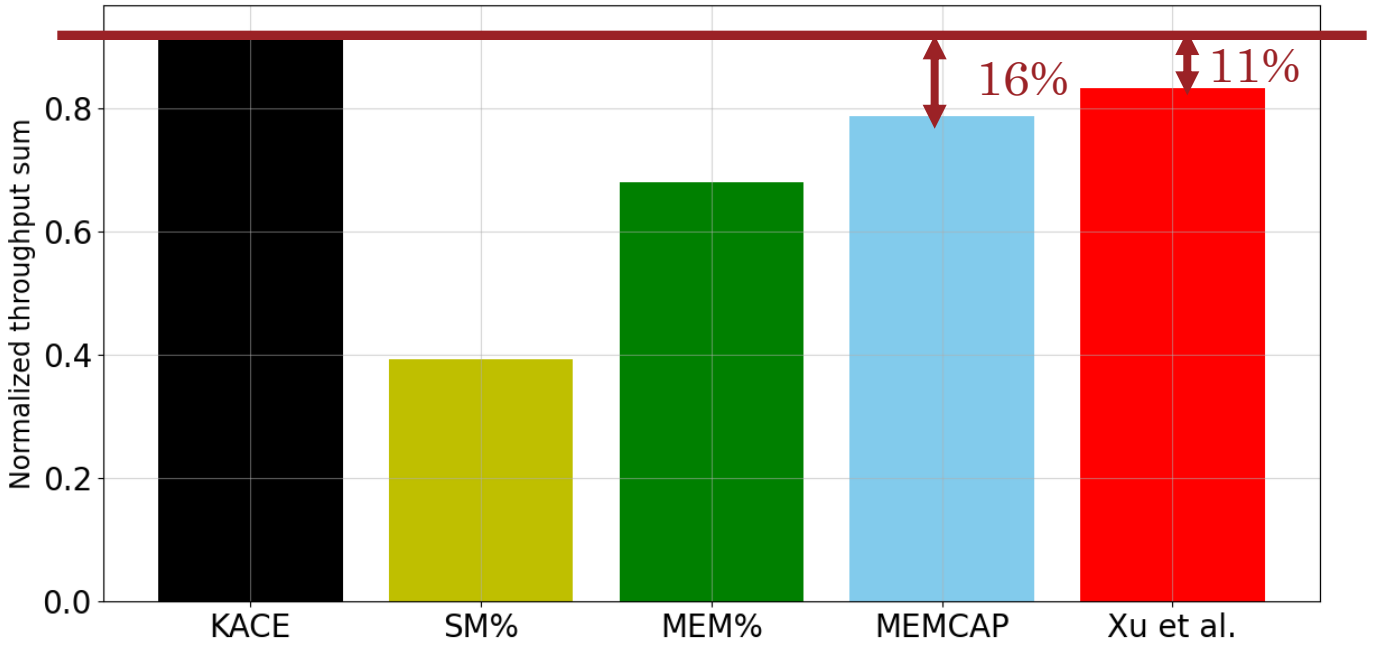
Evaluation - Colocation with different target workload



- KACE is the best policy in 13 cases out of 21 test sets
- Xu et al. ignores batch effect to throughput, causing weaker performance
- MEMCAP selects BERT-batch2-train with minimum footprint, which is compute heavy (92 SM%)

KACE is more consistent than other baselines

Evaluation - Colocation with unseen workloads



- An important evaluation since new workload can be encountered at runtime or the training set size is limited
- With given target unseen in training set for all batch sizes, predict max throughput sum when colocating target with 1 candidate workload
- KACE outperformed the other approaches

KACE can be transformed into a fully-online scheduler

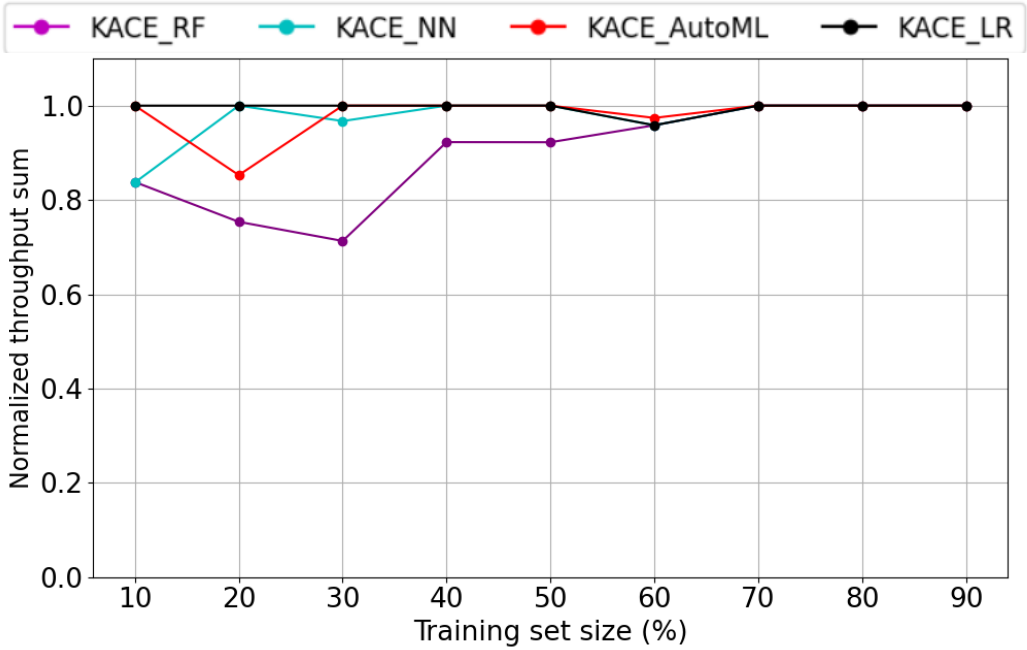
Talk outline

- Introduction
- Background
- Prior work
- System design
- Evaluation
- **Conclusion and Future Work**
  - **Conclusion**
  - **future work**

Conclusion

- We present KACE, which selects a set of **kernel and system metrics** that provide valuable information to predict colocated performance
- Use **limited offline** and exclusive profiling of **individual workloads**, eliminating the need for costly online profiling
- Identify a simple LR model that provides adequate colocation performance prediction accuracy with **minimal training time** and a **small training dataset**
- Evaluate KACE experimentally over multiple **training and inference** workloads and against various baselines

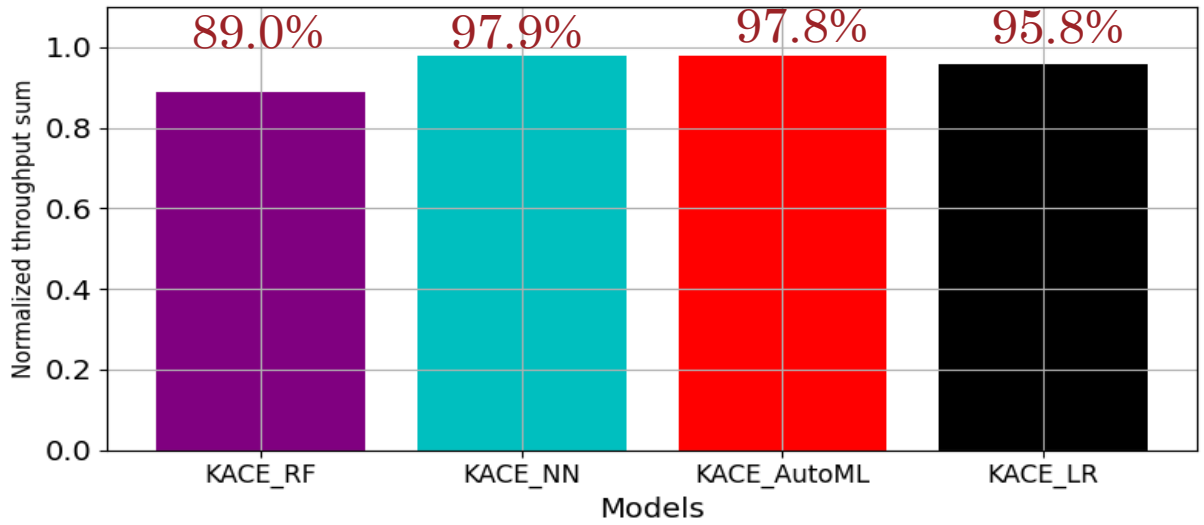
Short term future work - multi-workload prediction



- Select 3 workloads and predict max throughput sum out of all colocations (560)
  - **LR** remains consistent across all train set sizes
  - Require more training data to fit
- Should accommodate multiple workloads until resources are exhausted or service requirements are violated

Transform KACE into a fully-online scheduler

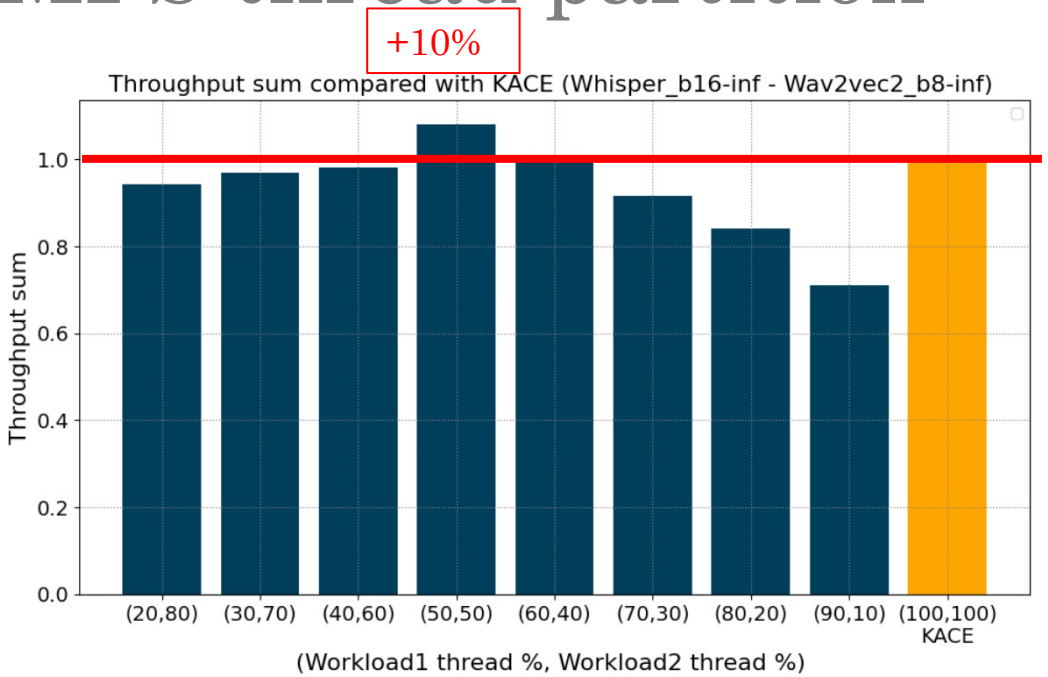
Future work - unseen multi-workloads prediction

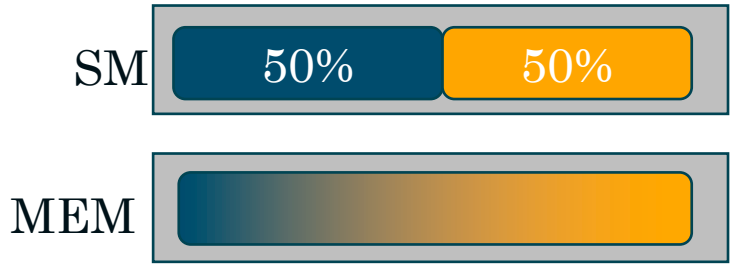


- With given target unseen in training set, predict max throughput sum when colocating target with 2 additional workloads
- Better predicted results since more high-throughput candidates are presented
- Consider other colocation strategies
  - Latency-sensitive + best-effort tasks
  - Evaluate latency and completion times

KACE can predict multiple-workload colocations

Future work – MPS thread partition



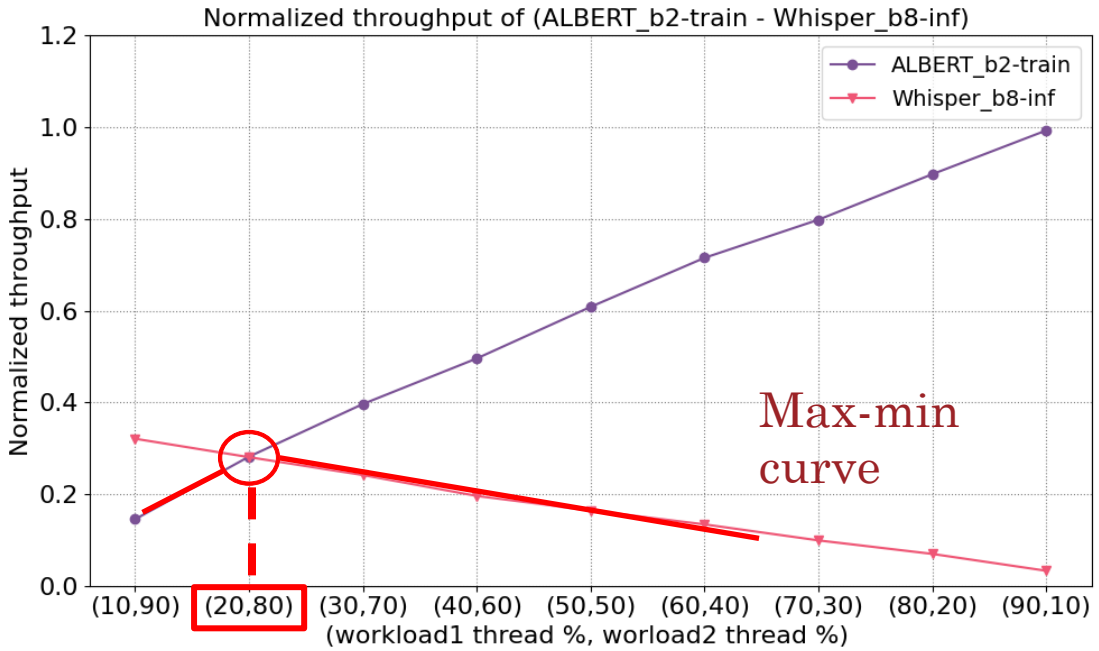


Nvidia MPS

- KACE does not partition specific SM for each process, which is enabled by MPS
- Thread partitioning allows better performance for light-memory pairs
- Tradeoff – limits the compute power for each workload
  - Need to select optimal partition to outperform default allocation

Thread partitions enable performance tuning

Future work – Fairness under MPS thread partitions

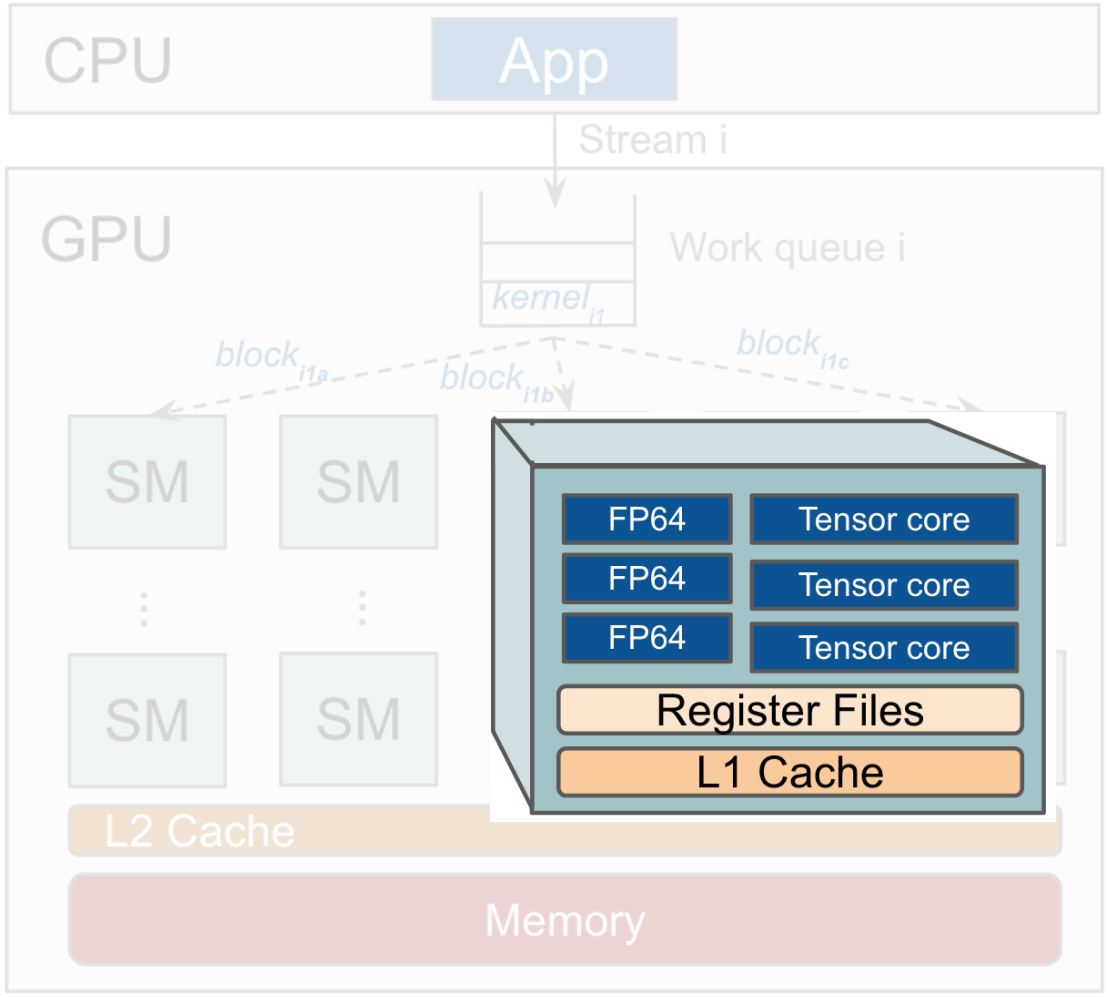


- How to choose the optimal thread partition for fair scheduling?
  - E.g. max-min policy selects maximum throughput out of the min workload curve
  - Workloads have different resource sensitivity, thus optimal point shifts
  - How to expand to multiple workloads?
    - 1 primary + x secondary workloads

Q & A

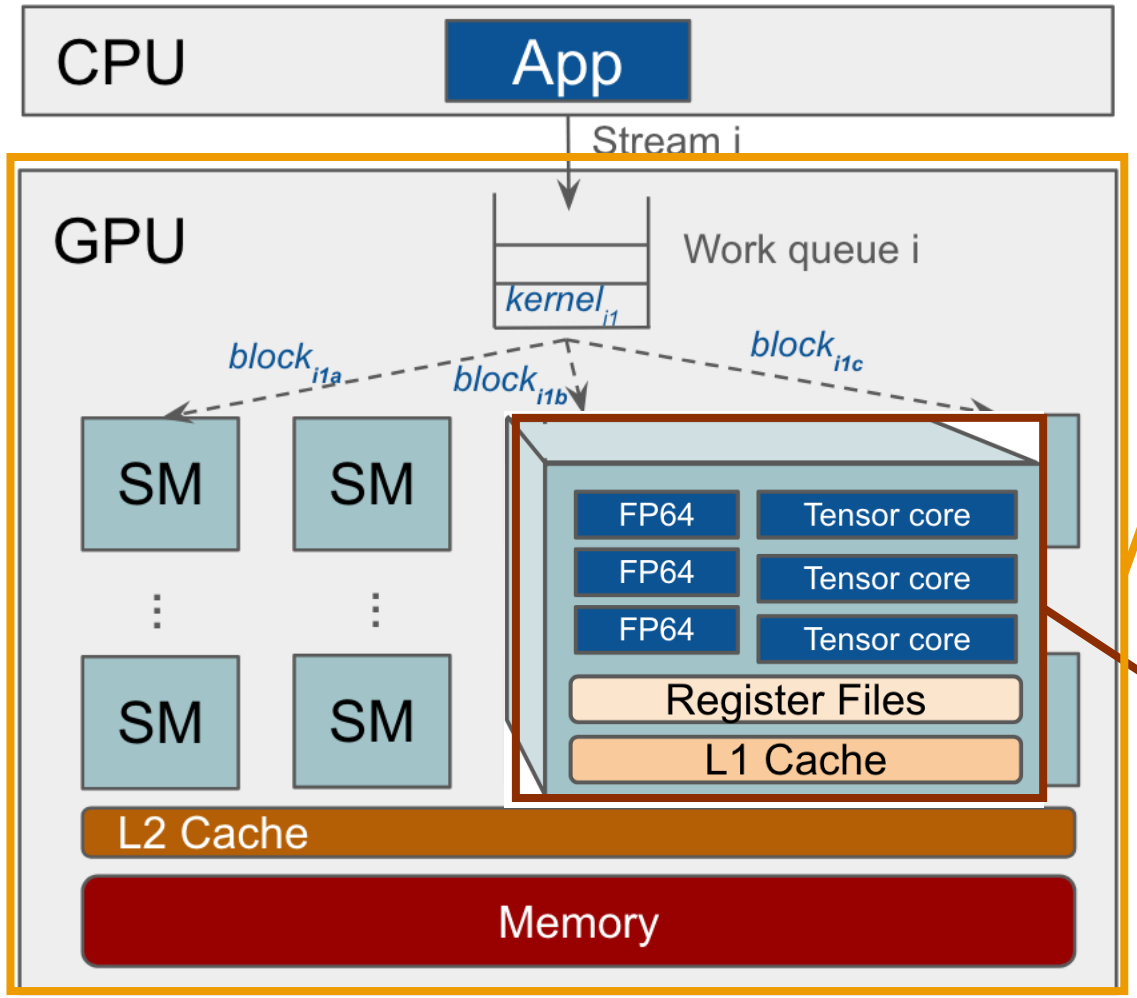
Appendix

GPU architecture



- FP unit
  - Unit to process floating point
- Tensor core
  - Specialized core that enabled mixed precision training
- Register
  - fastest on-chip memories that are used to store operands for the operations

GPU Performance monitoring



- Overall GPU metrics
  - Easy and fast to obtain
  - Quick system view, not kernel specific
    - GPU utilization (SM busy rate)
    - Memory busy rate
    - Memory footprints

- Kernel metrics
  - Fine-grained profile within SM
  - Long profile time, requires profile tool
    - Stream Multiprocessor (SM) Throughput
    - Max Memory throughput across units
    - Registers
    - Shared memory usage