ACM SoCC'24

KACE: Kernel-Aware Colocation for Efficient GPU Spatial Sharing

Bing-Shiun Han, Tathagata Paul, Zhenhua Liu, Anshul Gandhi
Stony Brook University

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

How to increase GPU utilization?

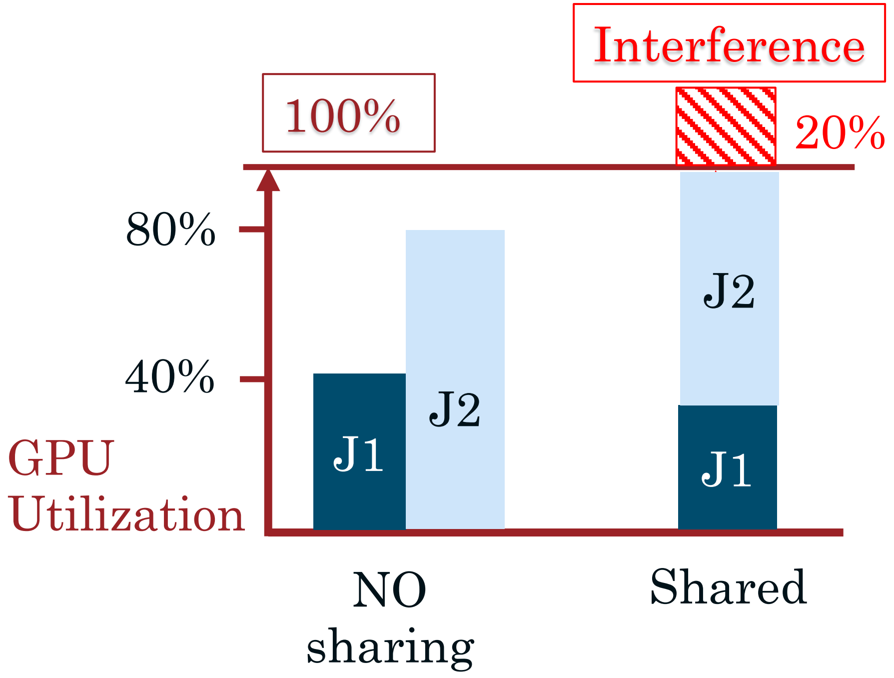
Sources: https://www.theregister.com/2024/01/15/cloud_providers_gpu_analysis/

Solution- Colocating workloads

😃 Colocation improves GPU utilization, decreases completion time, increases throughput

🙁 Challenges?

- **Interference between colocated workload**
  - Workload dependent
  - GPU share policy, architecture
- **Minimize interference via prediction**
  - Requires **per-workload** metrics to predict precise colocated performance
    - **Huge search space to profile all possible colocations**
  - DL approach requires **large training set and time**



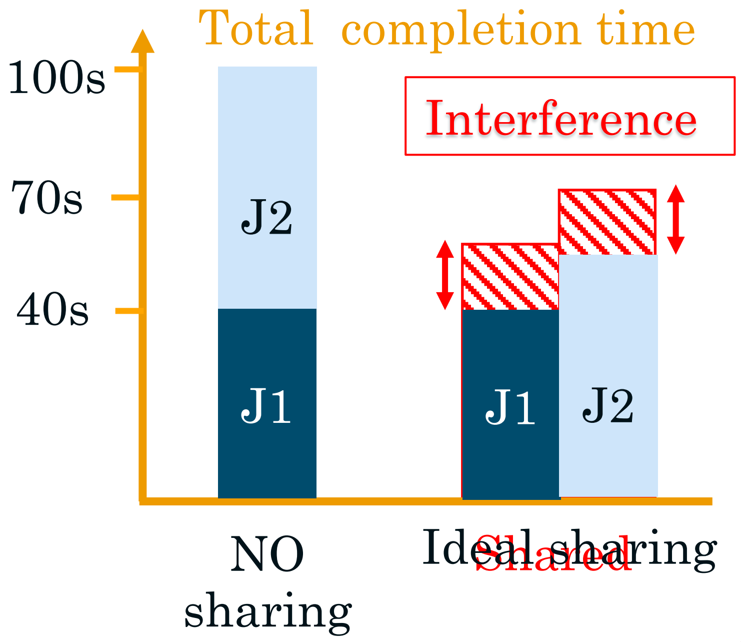


Illustration graph. Not experimental data.

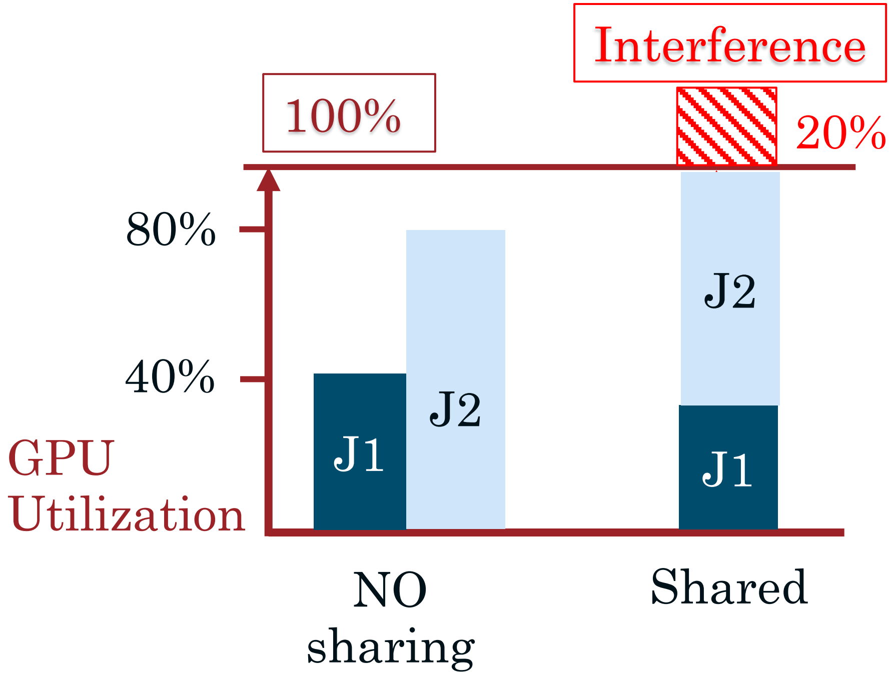
Solution- Colocating workloads

😃 Colocation improves GPU utilization, decreases completion time, increases throughput

☹️ **Challenges?**

- **Interference between colocated workload**
  - Workload dependent
  - GPU share policy, architecture
- **Minimize interference via prediction**

Challenges in GPU interference for workload colocation:
1. Minimizing performance degradation
2. Metrics for predicting interference
3. Profiling overheads



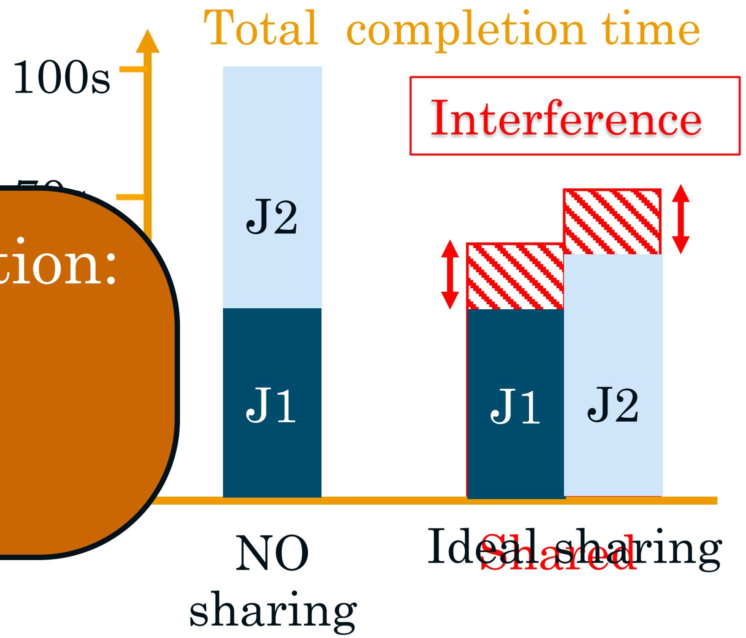


Illustration graph. Not experimental data.

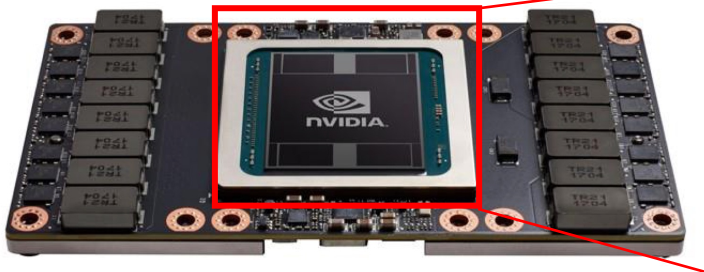
Problem Statement

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

GPU workloads can execute concurrently

- 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

Background- GPU Performance metrics



Conv

MaxP

Batch

- Overall GPU system 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 of each GPU unit
  - Long profile time, requires profile tool
    - Stream Multiprocessor (SM) Throughput
    - Max Memory throughput across units
    - Registers
    - Shared memory usage

NVIDIA Volta Architecture Whitepaper, https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

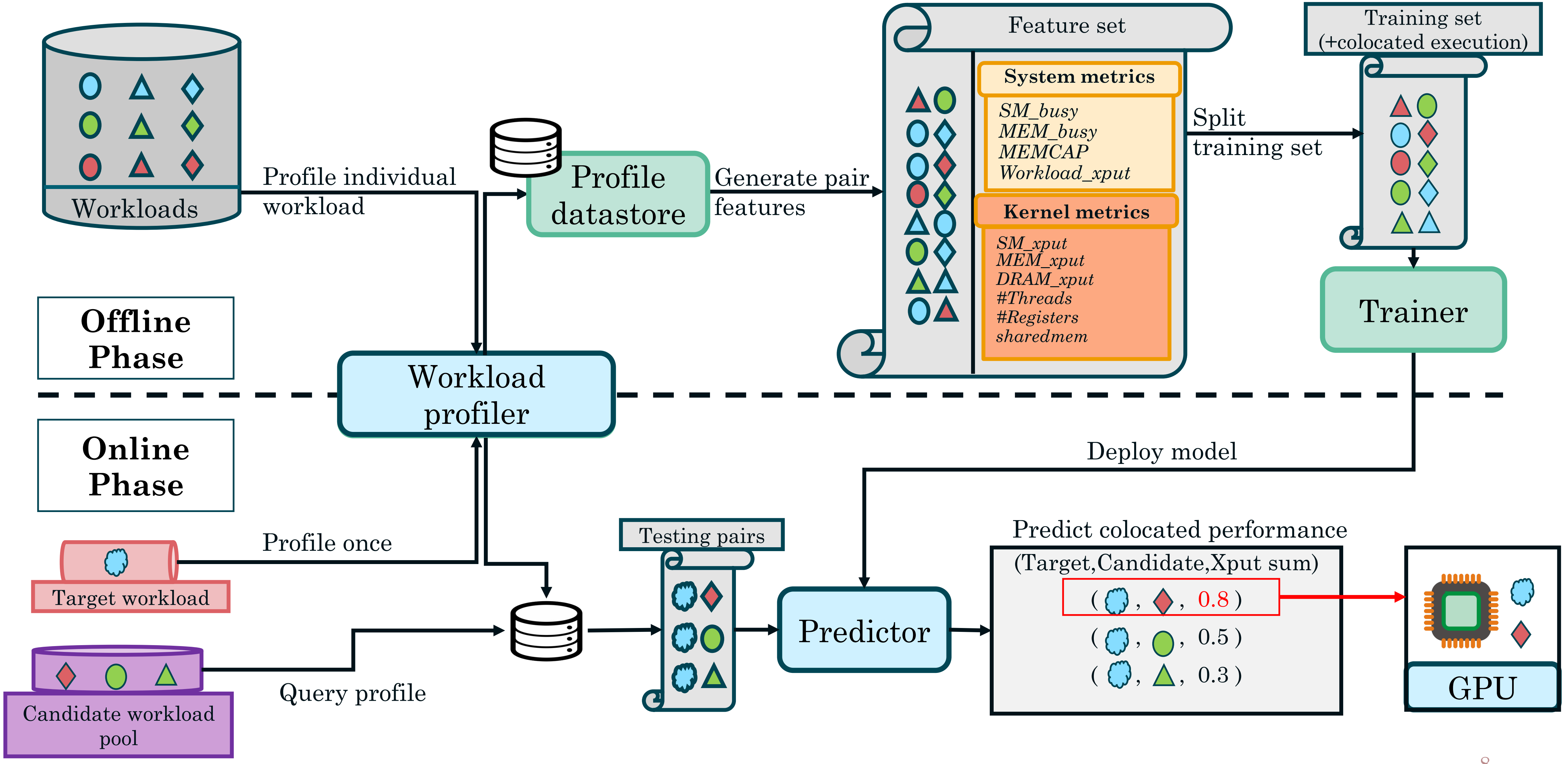
Related work

	Profile type	GPU Share type	Profiled metrics	Train/ inference	Prediction	Difference vs our work
MISO, SOCC'22	Online	Spatial	Completion Time	Train	V	Excessive online profiling, HW support, train only
Xu et al, Hotcloud'19	Offline	Temporal	Kernel	Both	V	Temporal prediction
Horus, PDS'22	Offline	Temporal	DL graph	Train	V	Temporal, need DL semantics, train only
Orion, Eurosys'24	Offline	Spatial	Kernel	Both	X	No performance prediction
KACE	**Offline**	**Spatial**	**Kernel**	**Both**	**V**	**Spatial prediction, offline, train+inf**

Our work focuses on accurate colocation predictions under spatial-sharing with offline kernel profiling

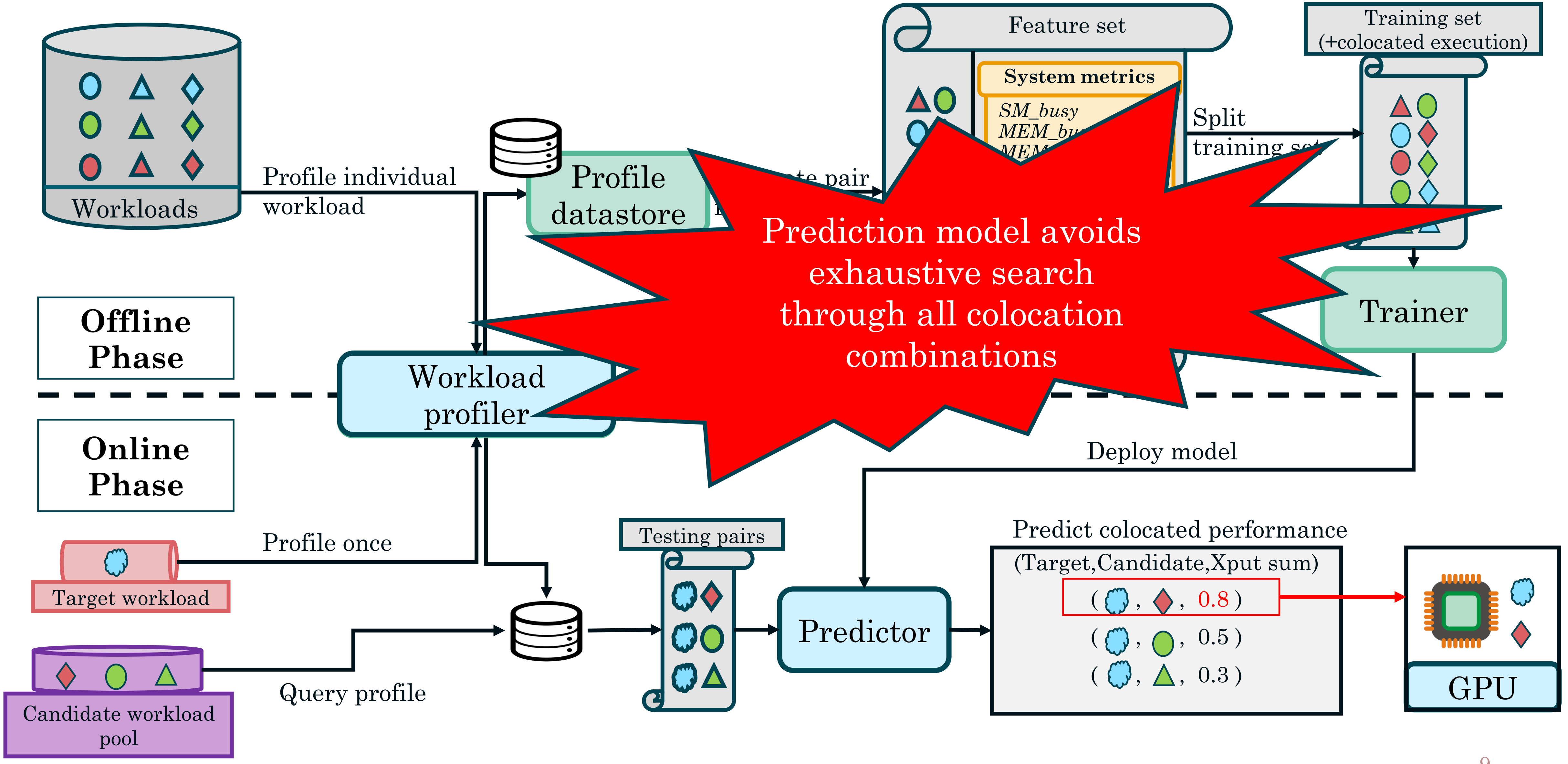
KACE System design

How to use individual workload's offline profile to predict online colocated performance?



KACE System design

How to use individual workload's offline profile to predict online colocated performance?

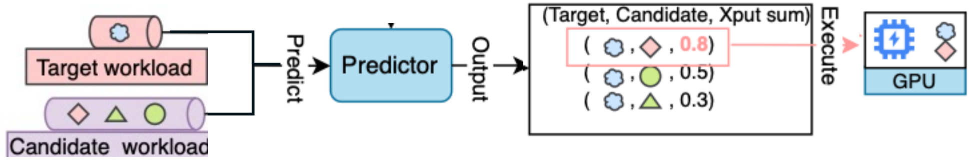


Evaluation - methodology

- 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

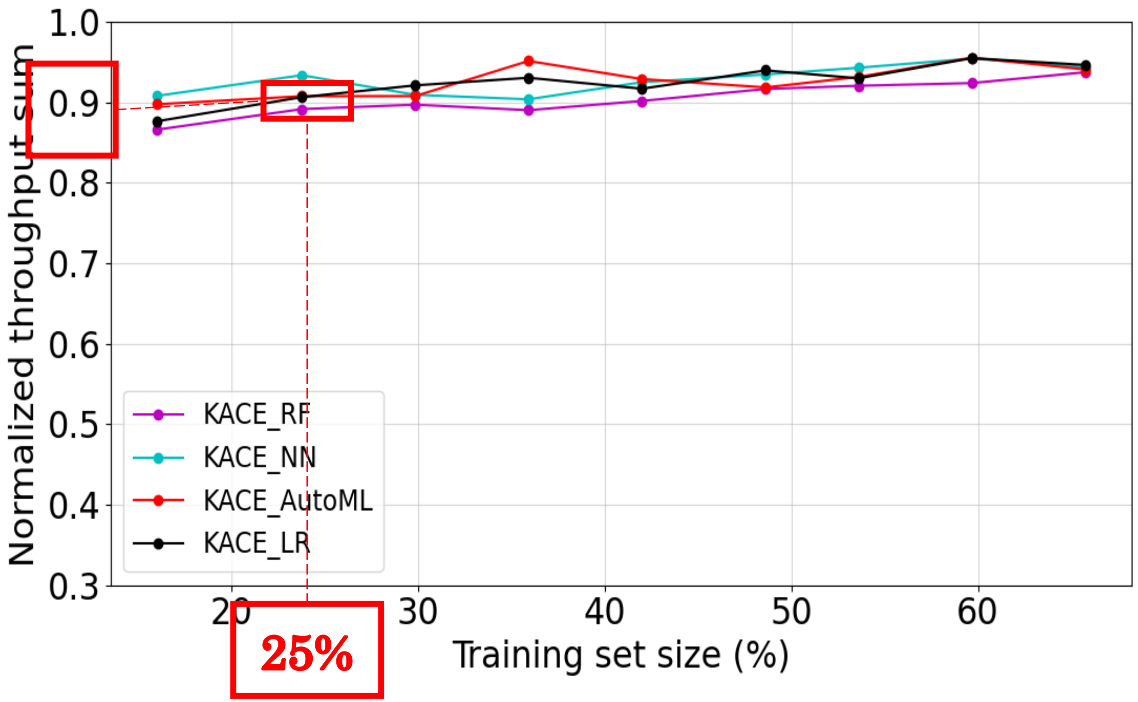
Evaluation - Workload colocation with tested ML models



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

$i \in$ Workload candidates

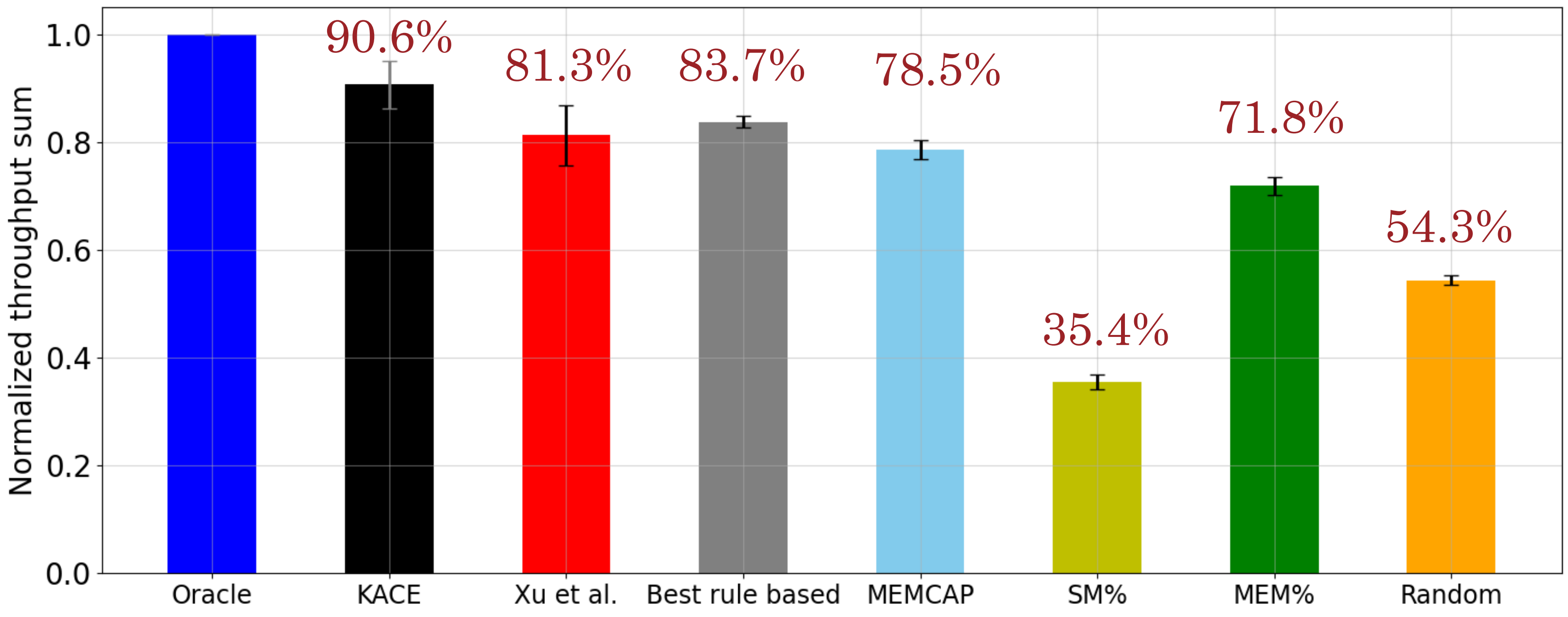
Oracle = workload pair with highest throughput sum (Impractical)



- 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

We use LR for its accuracy and fast training time

Evaluation - Workload colocation vs baseline policies



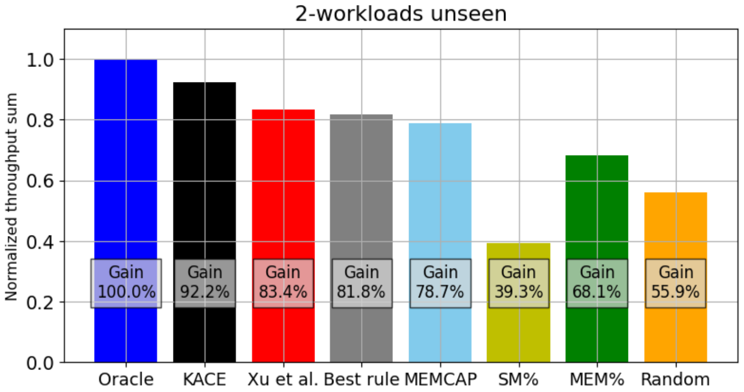
- Random performs poorly
- Rule-based approaches do not encapsulate kernel information
- Xu et al. targets temporal sharing

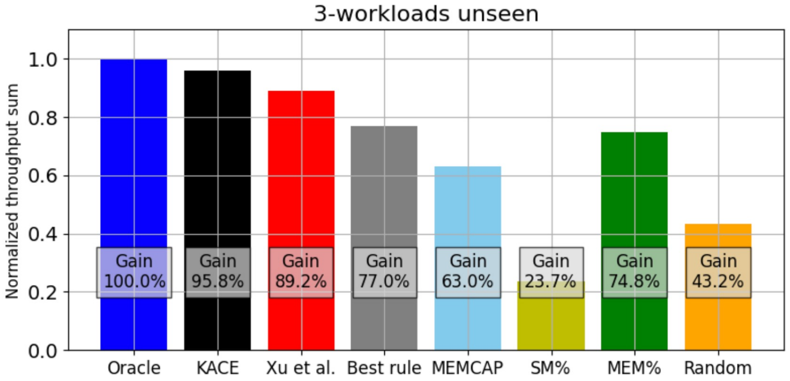
KACE achieves **90%** throughput sum compared with Oracle

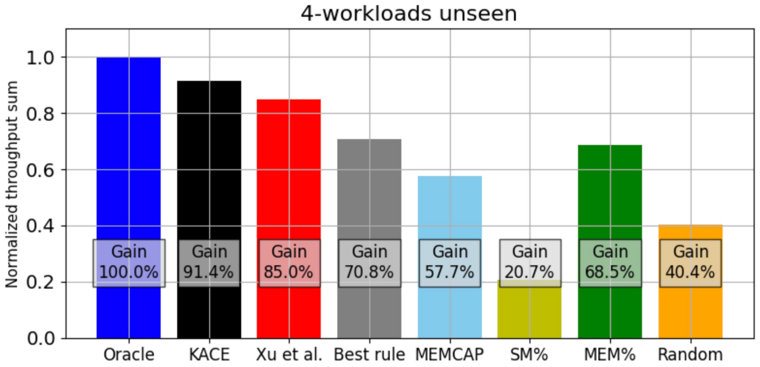
Evaluation- unseen and autoregressive workloads

- Unseen workload

Achieves 92% / 96% / 91% of Oracle sum

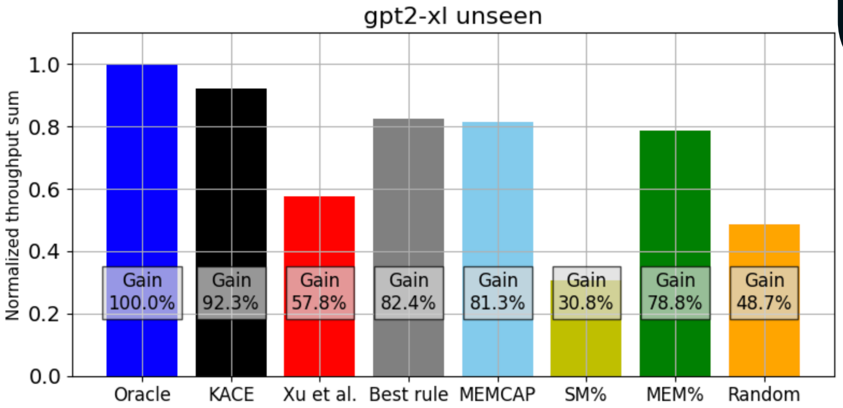






- Autoregressive workload (unseen)

Achieves 92% of Oracle sum
10% better than next-best policy



Summary

KACE is a ML system framework that accurately predicts GPU spatial-sharing interference, achieving over **90%** of Oracle's throughput sum.

- Selects key kernel & system metrics for lightweight performance prediction
- Limited offline profiling of individual workloads
- Simple LR model for accurate colocation prediction with minimal training time and small dataset

Questions?

GitHub

https://github.com/nba556677go/KACE-artifact

System design- Profiler

How to generate individual workload profile?

Workload profiler

Nvidia-smi

Duration - 100 steps

System metrics

SM_busy
MEM_busy
MEMCAP
Workload_xput

NVIDIA Nsight™ Compute

Duration -
1 forward + backward pass (train),
1 forward pass (inference)

Kernel functions	**sgemm**	+	**elementwise**	+	**Softmax**	=	**Aggregated**
SM_xput	50		80		20		**50**
MEM_xput	50		80		10		**50**
DRAM_xput	30		20		50		**33**
#Threads	500000		800000		200000		**1500000**
#Registers	5000000		3000000		1000000		**9000000**
Sharedmem	5000		2000		2000		**90000**

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-Profiler

How to generate individual workload profile and paired feature set?