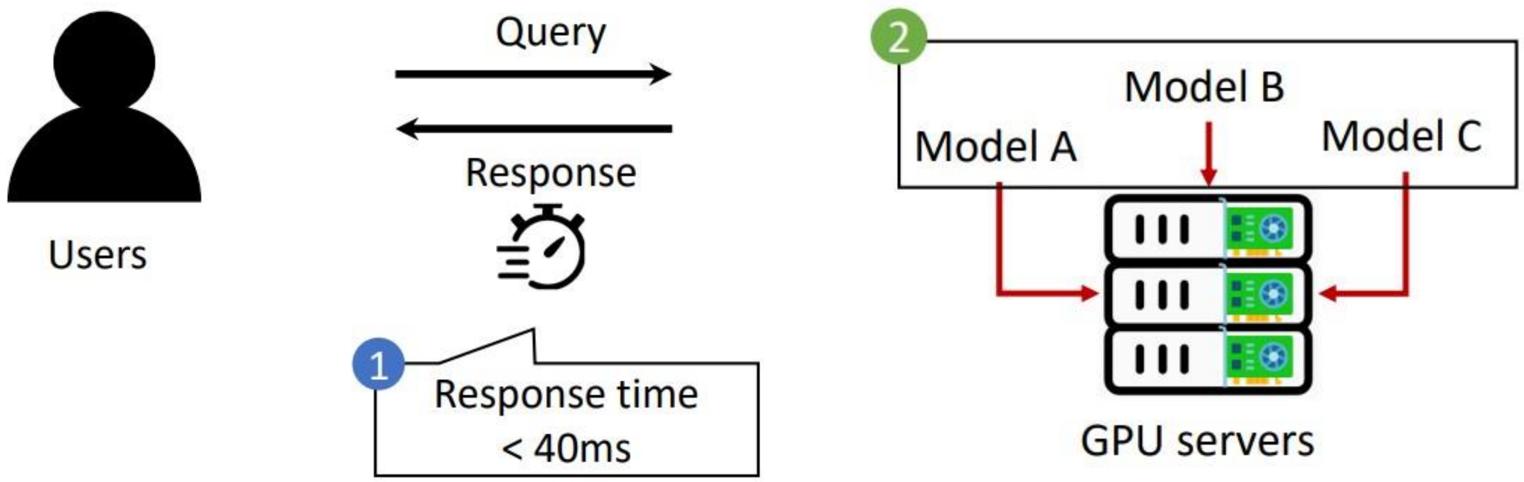
Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing

Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh, KAIST

USENIX ATC'22

Machine Learning Inference in GPUs

- GPU-based servers are widely used for ML Inference.
- Two requirements for ML inference request
  - For each request must provide a bounded latency to support a service-level objective(SLO)
  - Serve multiple heterogeneous ML models in a system

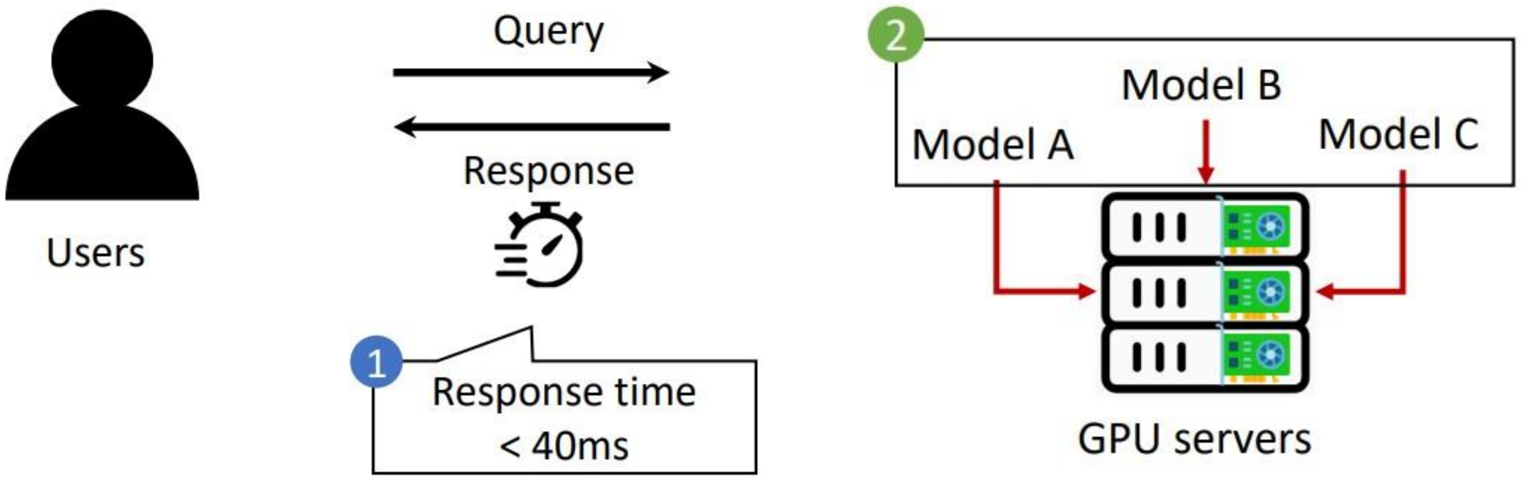


GPU underutilized

Training computation vs **Inference computation**

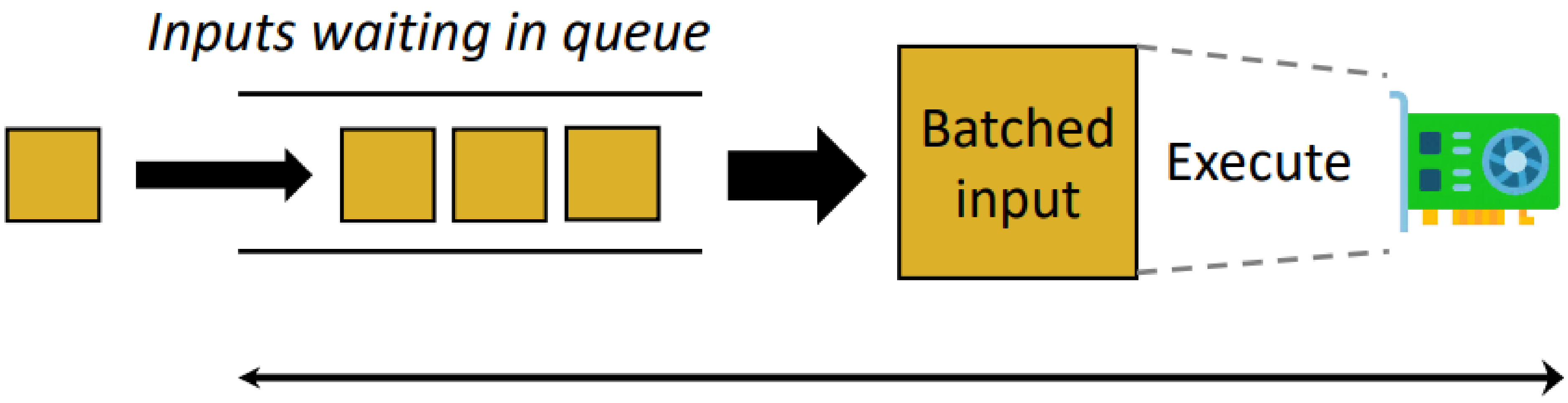
Batch Processing

Online Request Rate



Existing Work – Batch-Aware ML Inference

- **Merge request inputs to a single large input**
  - **Improves utilization of GPU**
  - **Batch size could not be huge due to SLO**



Waiting time + Batch Prepare time + Batch Exec time < every SLO

Existing Work – Temporal Scheduling

- **Round-based interleaved execution of batches**
  - **Increase utilization by reducing idle time on GPU**

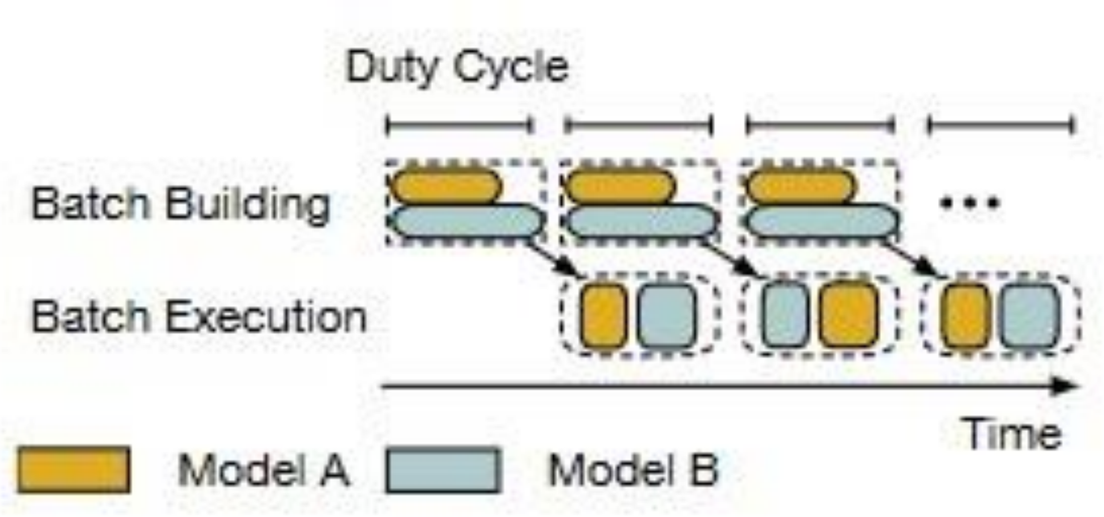


Figure 2: Round-based execution of SBP for two models consolidated on a GPU. *Duty cycle* is the interval for a round.

Existing Work – Spatial Sharing

- **Splits a GPU resource into multiple pieces**
- **Improves GPU utilization allowing high throughput without SLO violation.**

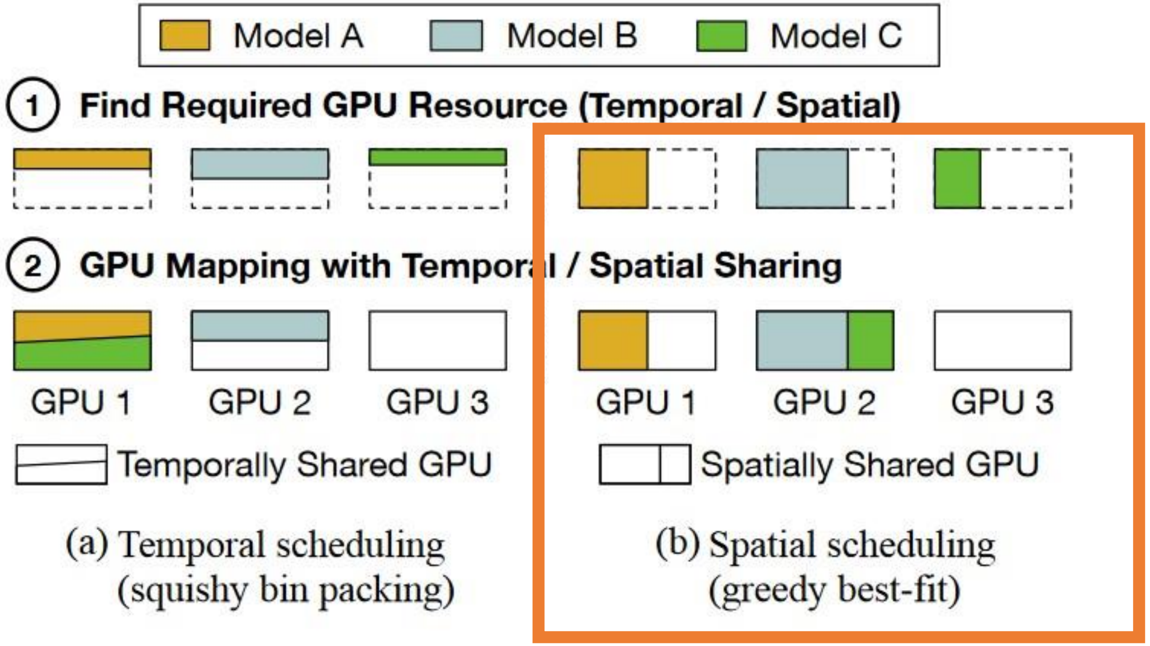


Figure 3: Temporal scheduling (SBP) vs. spatial scheduling (greedy best-fit partitioning).

Motivation – Optimal Batch Size and Partition

- Opportunities for improving performance with better resource utilization
- Large batch size
  - The latency significantly drops as more resource is added.
- Small batch size
  - The latency is not largely affected by the amount of GPU resources

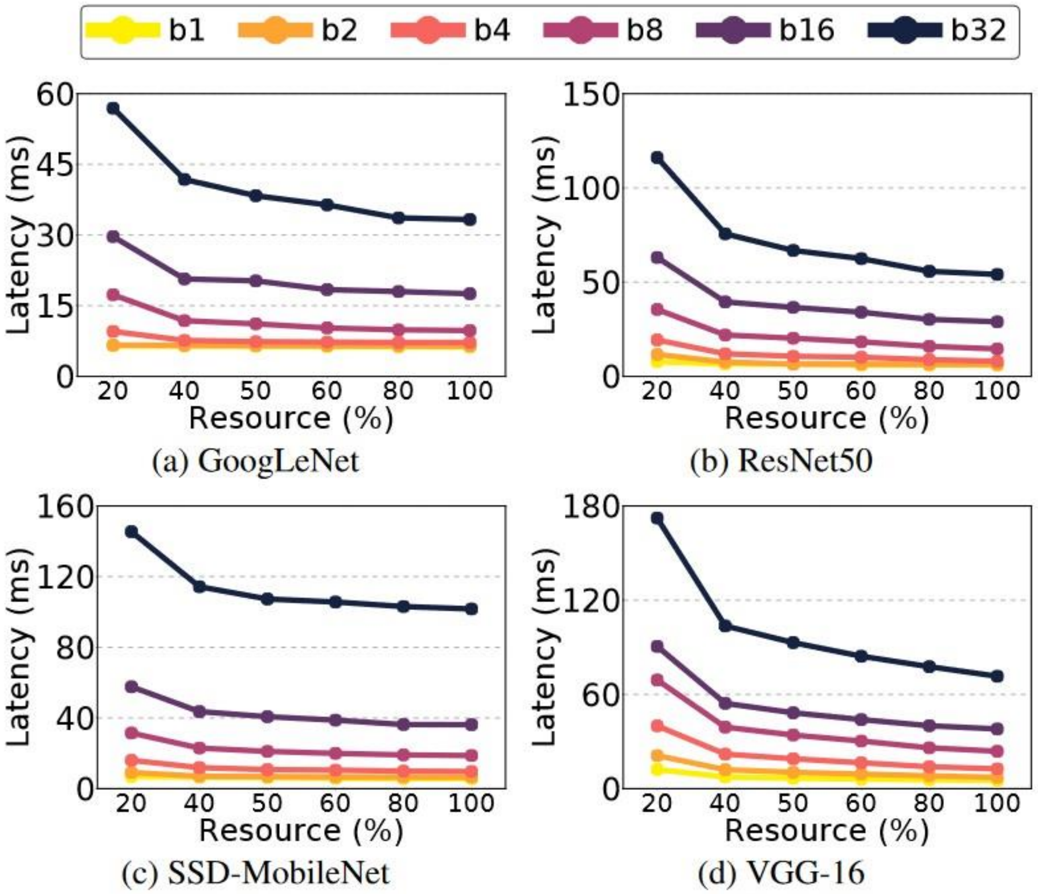


Figure 4: Batch inference latency as the fraction of computing resource assigned to the model inference changes from 20% to 100%, for the four ML models. Each curve represents a different batch size, and bn is a batch size of n.

Motivation – Schedulability and GPU Partitioning

- Add GPU Partitioning to Temporal-Sharing
- GPU partitioning is capable of putting wasted GPU compute power to use, enabling higher resource utilization

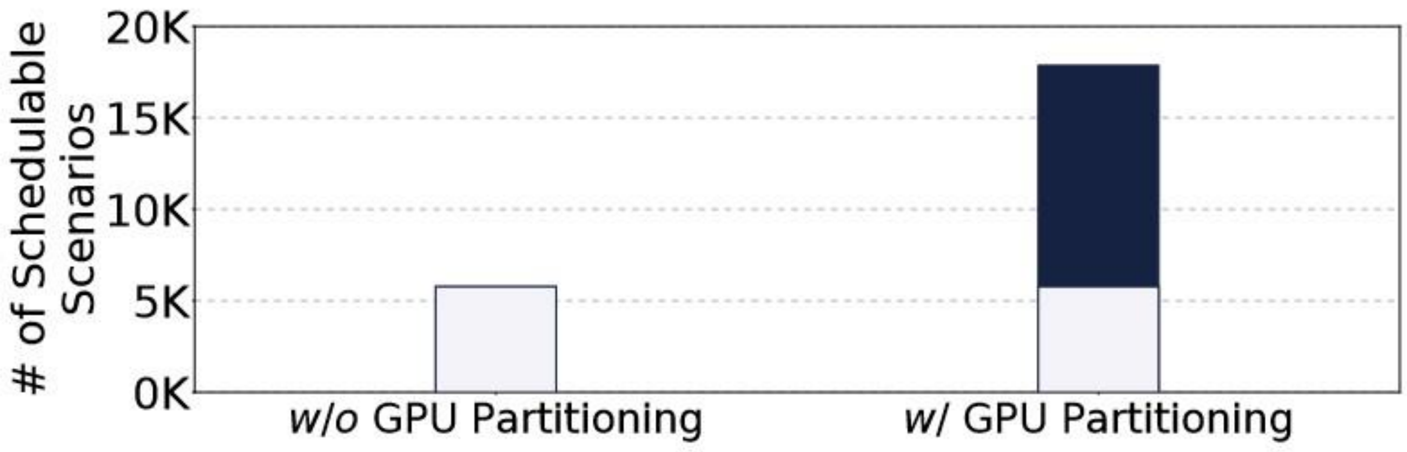
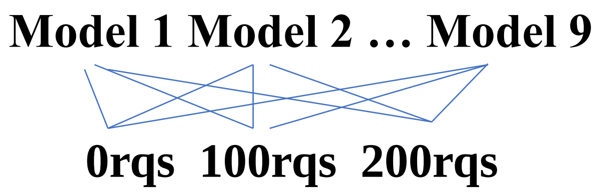


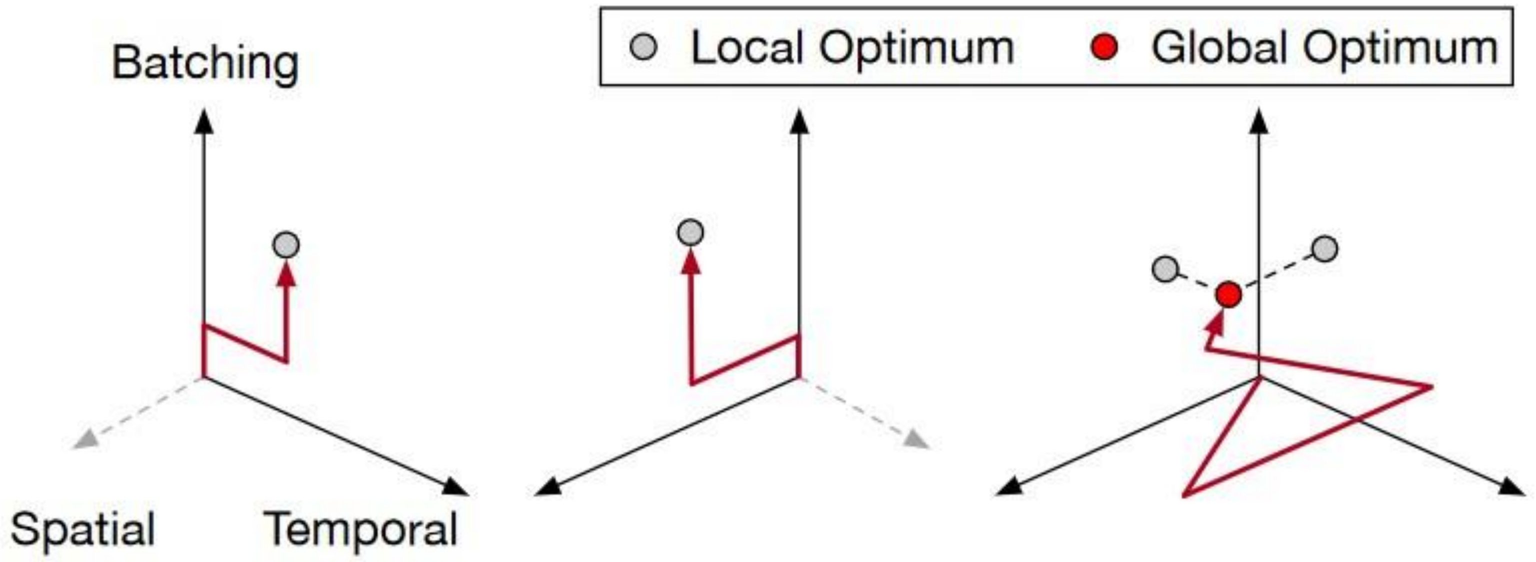
Figure 5: Number of schedulable scenarios when the SBP algorithm performs the scheduling *without* (left) and *with* (right) a fixed 1:1 GPU partitioning scheme.



3^9 -1 scenarios

Motivation – Effective Partitioning

- **Demonstrate the performance of** **cost-effective partitioning scheme**



(a) Temporal scheduling with batching (b) Spatial scheduling with batching (c) Temporal and spatial scheduling with batching

Figure 1: Multi-dimensional search space for providing globally optimal performance.

Motivation – Effective Partitioning

- Demonstrate the performance of **cost-effective partitioning scheme**

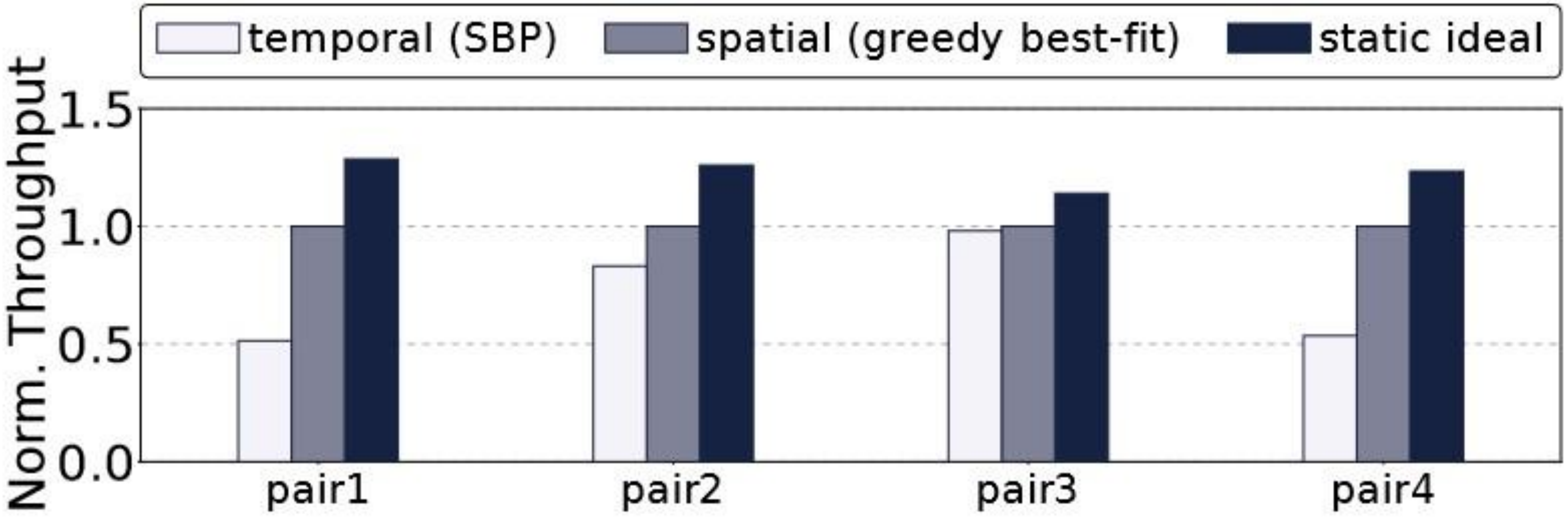


Figure 6: Comparison of SLO preserved throughput for temporal (SBP), spatial (greedy best-fit), and static ideal scheduling, normalized to spatial scheduling.

Motivation – Interference in Consolidated Executions

- The performance interference caused by multiple inference executions concurrently running on a GPU.

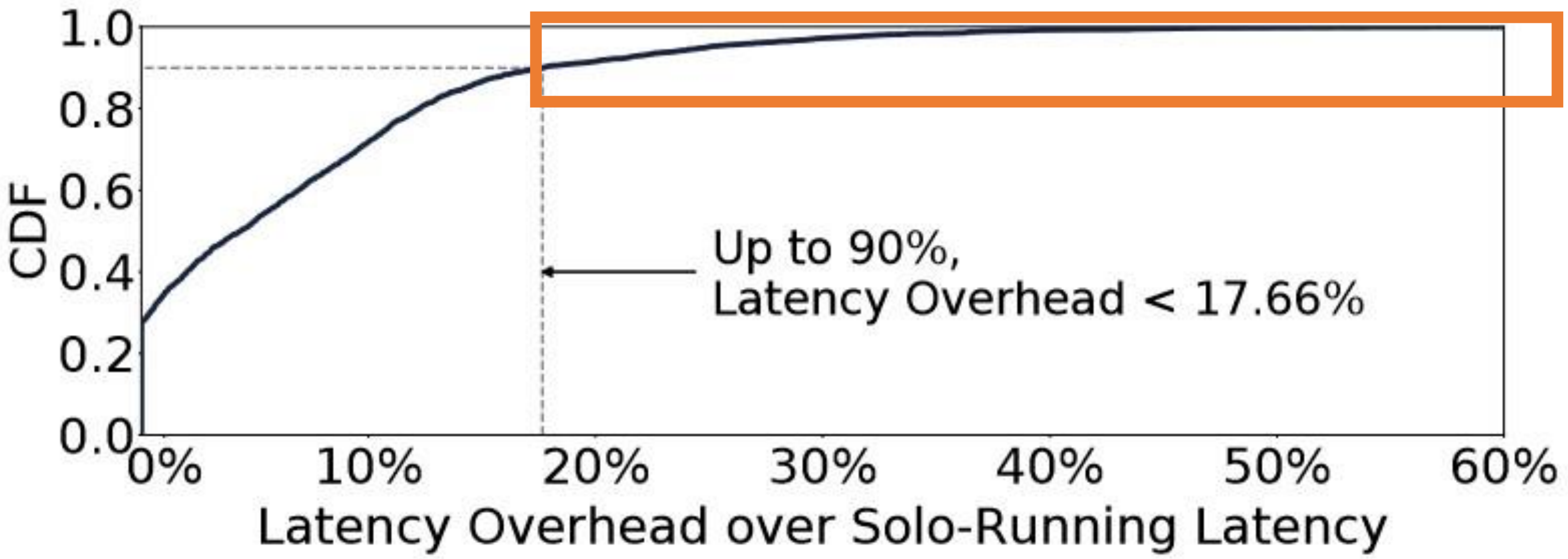
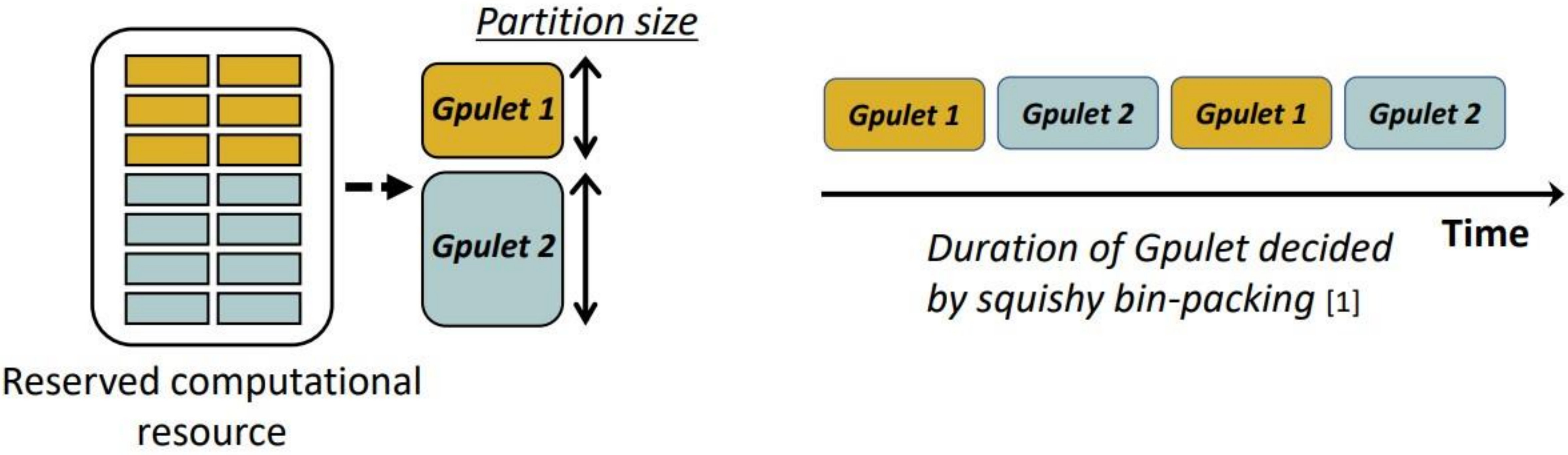


Figure 7: Cumulative distribution of latency overhead when pairs of inference executions are consolidated on a GPU.

Design – *Gpulet*

- **Gpulet: A share of spatial/temporal partition of GPU resource**



Design – *Overview*

- **Gpulet Scheduler**: Decides and sends batched requests to the backend servers.
- **Request Monitor**

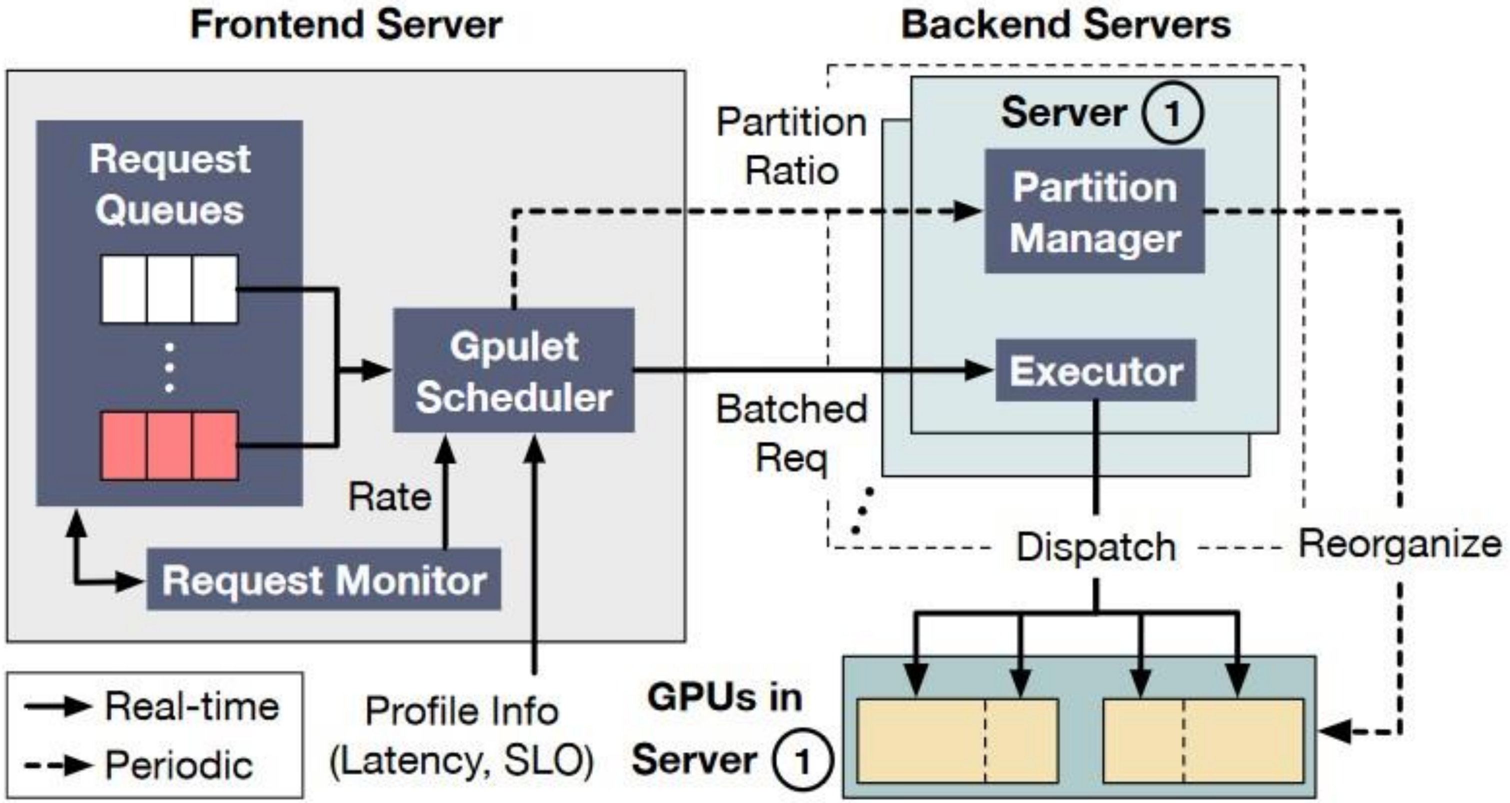


Figure 8: Overview of the scheduling framework with gpulets.

Design – *Overview*

- **Executor**: Dispatches requests to GPUs
- **Partition Manager**: prepares the partitions on the GPUs

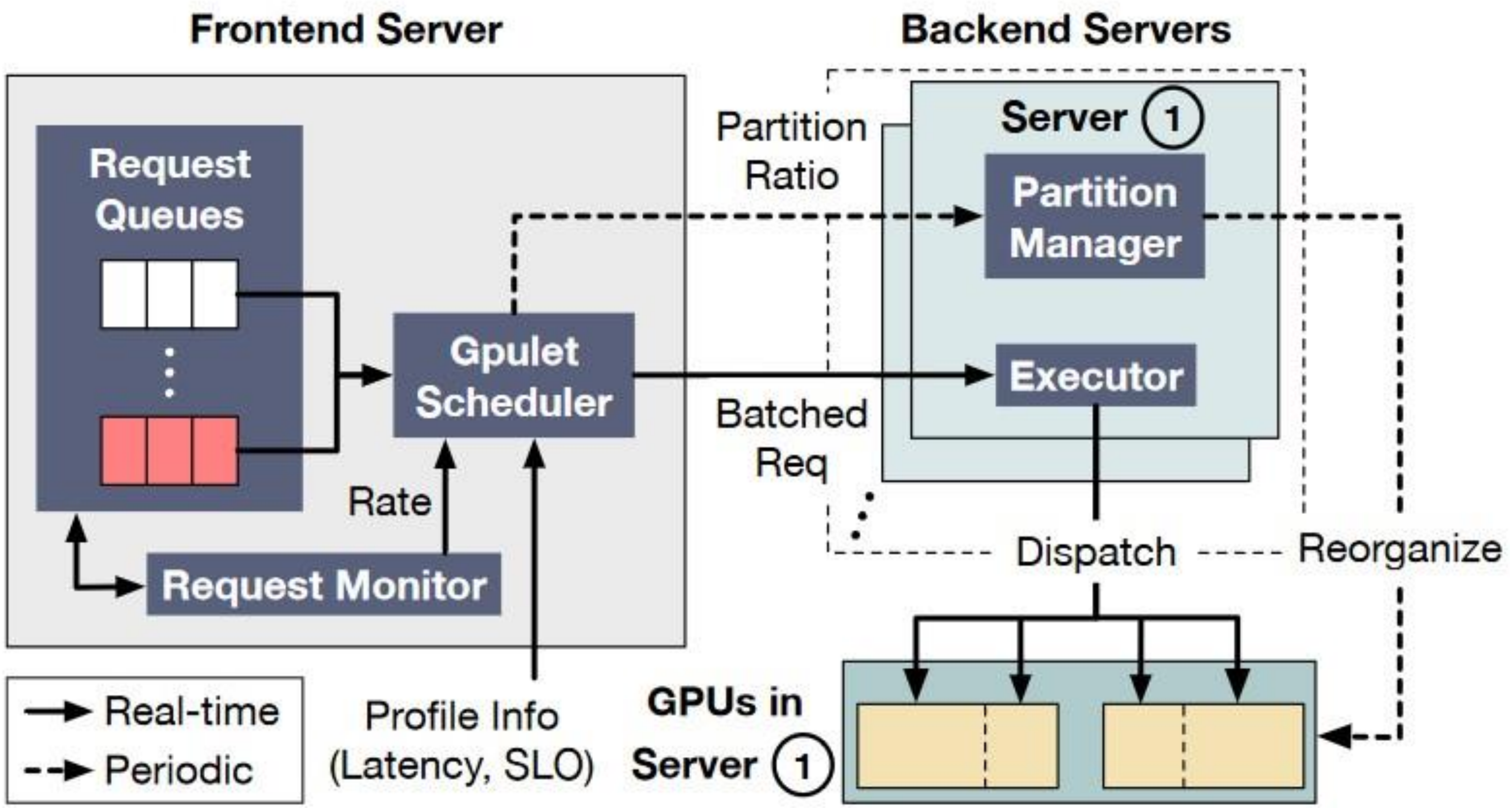


Figure 8: Overview of the scheduling framework with gpulets.

Design – *Cost-effective scheduling*

- **Maximize** **Performance &** **Minimize** **Resource Usage**

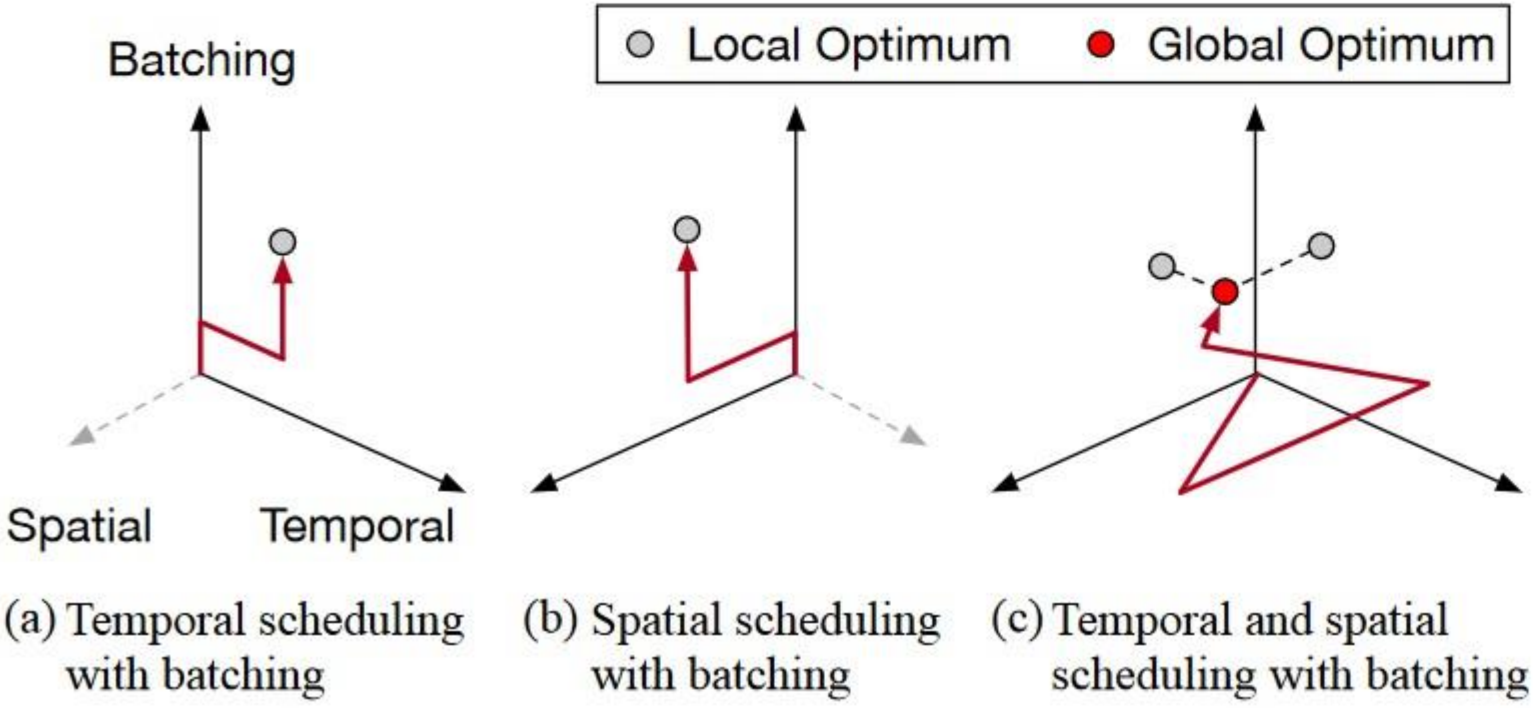


Figure 1: Multi-dimensional search space for providing globally optimal performance.

Design – *Cost-effective scheduling*

- **Maximize** **Performance &** **Minimize** **Resource Usage**

Name	Description
$L(b,p)$	Latency function of batch size b and partition size p
$int\,f$	Interference overhead function
SLO_i	SLO (in latency) of model i
$gpulet.size$	Actual partition size of gpulet

Table 1: Definitions of variables for Algorithm 1.

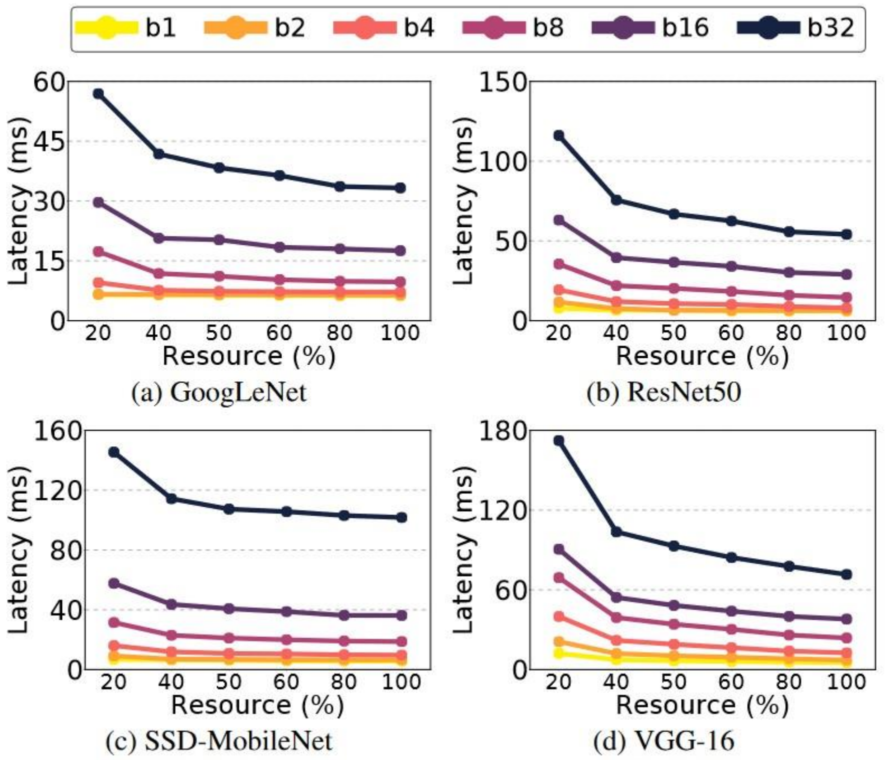


Figure 4: Batch inference latency as the fraction of computing resource assigned to the model inference changes from 20% to 100%, for the four ML models. Each curve represents a different batch size, and bn is a batch size of n.

Design – *Cost-effective scheduling*

- **Maximize** Performance & **Minimize** Resource Usage

Algorithm 1: Gpulet Scheduling Algorithm

ElasticPartitioning($L(b,p)$, $intf$, SLO):

for each period **do** // If rescheduling is required

  Sort every model by $rate_m \times SLO_m$ in ascending order

  for each model m **do**

    while IsRemainRate() and IsRemaingpulet() **do**

      $rate \leftarrow$ Remaining rate of model m

      $p_{\text{eff}} \leftarrow$ MaxEfficientPartition()

      $p_{\text{req}} \leftarrow$ MinRequiredPartition($rate$)

      $p_{\text{ideal}} \leftarrow$ Min(p_{eff}, p_{req})

      $gpulet \leftarrow$ FindBestFit(p_{ideal}, SLO_m, $intf$)

      Apply $gpulet$ to system

    end

  end

end

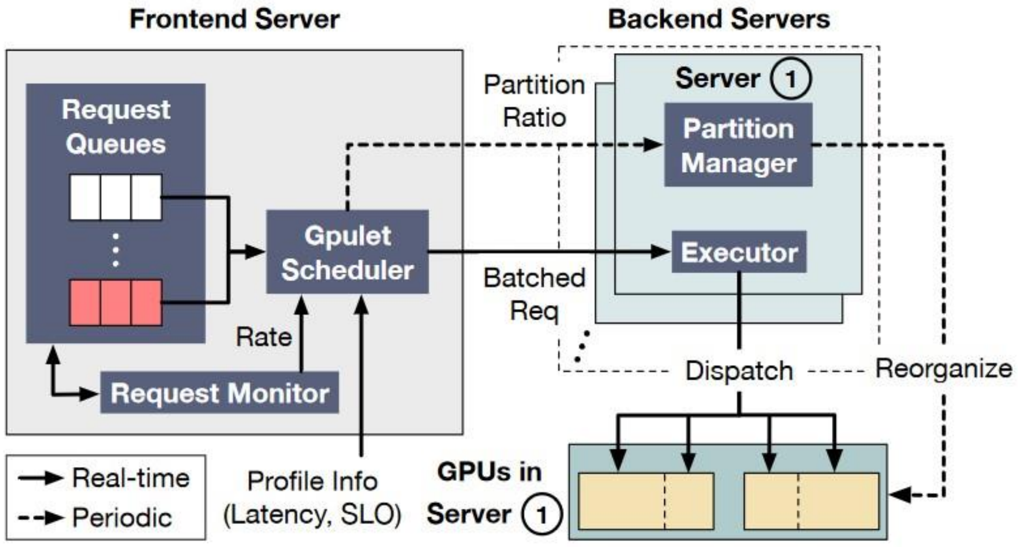


Figure 8: Overview of the scheduling framework with gpulets.

Design – *Cost-effective scheduling*

- **Maximize Performance & Minimize Resource Usage**

Algorithm 1: Gpulet Scheduling Algorithm

ElasticPartitioning($L(b,p)$, $intf$, SLO):

for each period **do** // If rescheduling is required

  Sort every model by $rate_m \times SLO_m$ in ascending order

  for each model m **do**

    while IsRemainRate() and IsRemaingpulet() **do**

      $rate \leftarrow$ Remaining rate of model m

      $p_{\mathrm{eff}} \leftarrow$ MaxEfficientPartition()

      $p_{\mathrm{req}} \leftarrow$ MinRequiredPartition($rate$)

      $p_{\mathrm{ideal}} \leftarrow$ Min(p_{eff}, p_{req})

      $gpulet \leftarrow$ FindBestFit(p_{ideal}, SLO_m, $intf$)

      Apply $gpulet$ to system

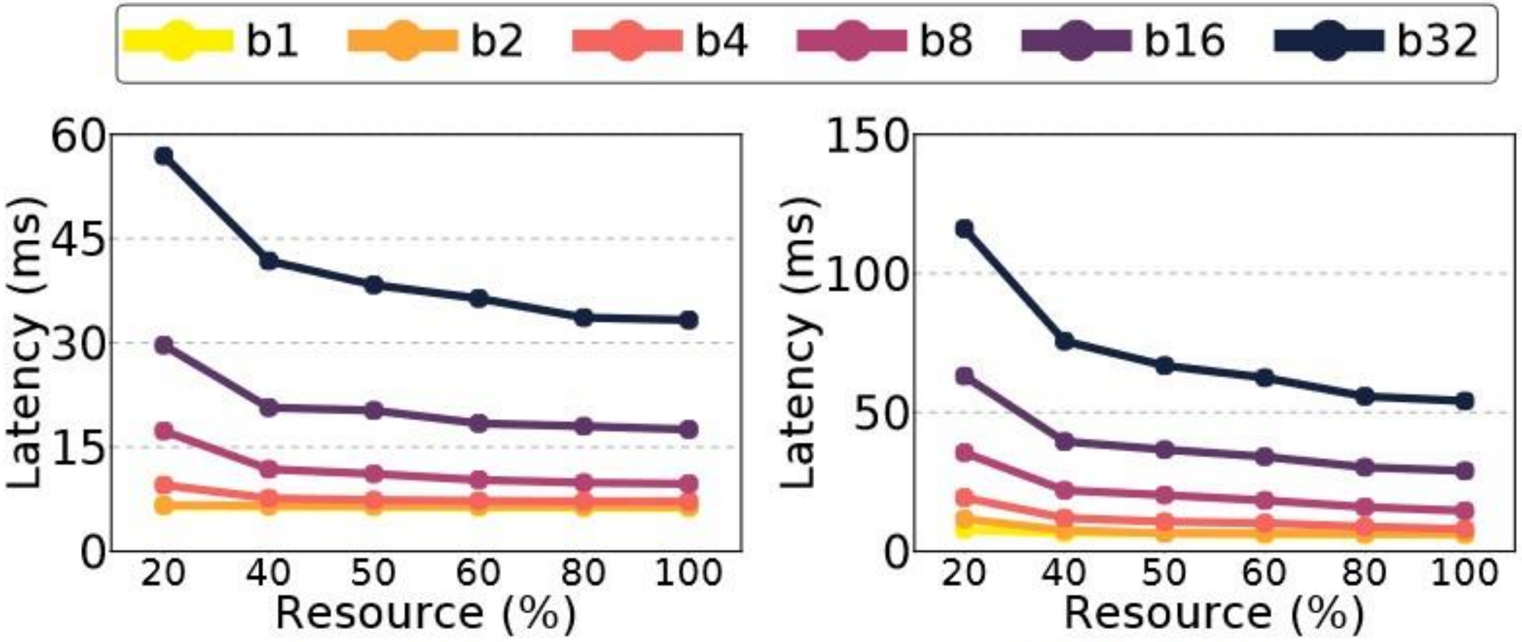
    end

  end

end

Design – *Cost-effective scheduling*

- **Peff : for each partition size**
  - **Get maximum batch size b with Latency(b) < SLO**
  - **Get throughput = b / Latency(b)**



(a) GoogLeNet (b) ResNet50

Design – *Cost-effective scheduling*

- **Maximize** **Performance &** **Minimize** **Resource Usage**

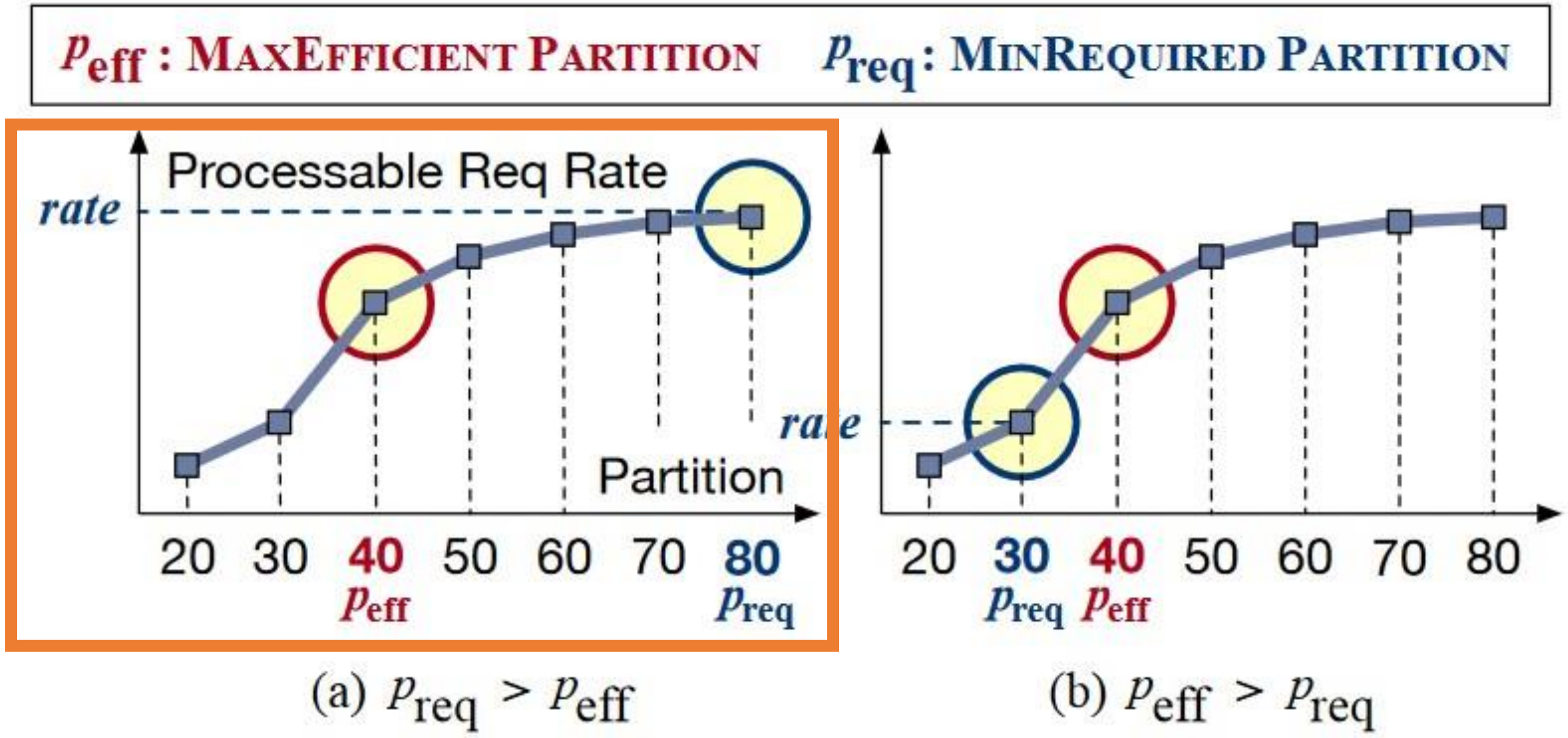
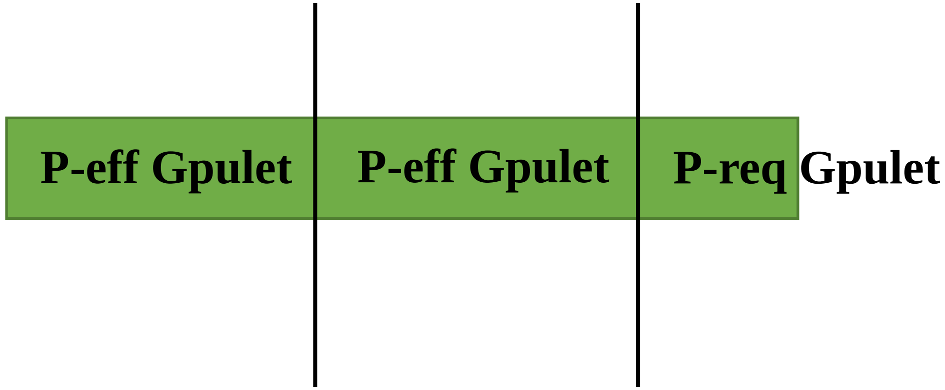


Figure 9: Max efficient partition (p_{eff}) and min required partition (p_{req}).

Design – *Cost-effective scheduling*

- As **much** as many **cost-effective partitions** within rate
- One **minimum partition for remaining rate**



Model A request

Algorithm 1: Gpulet Scheduling Algorithm

ElasticPartitioning($L(b,p)$, $intf$, SLO):

for each period **do** // If rescheduling is required

  Sort every model by $rate_m \times SLO_m$ in ascending order

  for each model m **do**

    while IsRemainRate() and IsRemaingpulet() **do**

      $rate \leftarrow$ Remaining rate of model m

      $p_{\mathrm{eff}} \leftarrow$ MaxEfficientPartition()

      $p_{\mathrm{req}} \leftarrow$ MinRequiredPartition($rate$)

      $p_{\mathrm{ideal}} \leftarrow$ Min(p_{eff}, p_{req})

      $gpulet \leftarrow$ FindBestFit(p_{ideal}, SLO_m, $intf$)

      Apply $gpulet$ to system

    end

  end

end

Design – *Cost-effective scheduling*

- **Maximize Performance & Minimize Resource Usage**

Algorithm 1: Gpulet Scheduling Algorithm

ELASTICPARTITIONING($L(b,p)$, $intf$, SLO):

for each period **do** // If rescheduling is required

    Sort every model by $rate_m \times SLO_m$ in ascending order

    for each model m **do**

        while ISREMAINRATE() and ISREMAINGPULET() **do**

            $rate \leftarrow$ Remaining rate of model m

            $p_{\text{eff}} \leftarrow$ MAXEFFICIENTPARTITION()

            $p_{\text{req}} \leftarrow$ MINREQUIREDPARTITION($rate$)

            $p_{\text{ideal}} \leftarrow$ MIN(p_{eff}, p_{req})

            $gpulet \leftarrow$ FINDBESTFIT(p_{ideal}, SLO_m, $intf$)

            Apply $gpulet$ to system

        end

    end

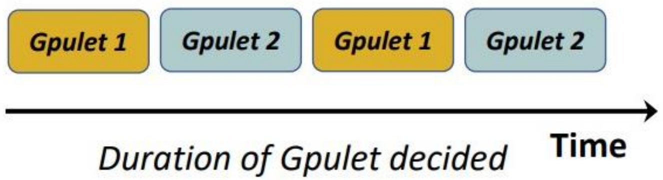
end

Design – *Cost-effective scheduling*

- **Maximize Performance & Minimize Resource Usage**

FINDBESTFIT(p_{ideal}, SLO_m, $intf$):

Sort every remaining gpulets by size in ascending order
for *gpulet* in GETREMAINGPULETS() **do**
  if $gpulet.size \geq p_{\text{ideal}}$ **then**
    if *gpulet is unpartitioned* **then**
      Split and allocate *gpulet* to p_{ideal} size partition
    end
    $b \leftarrow \text{argmax}_{k \in \mathbb{N}}(L(k, gpulet.size) + intf \leq SLO)$
    if b exists **then**
      $\boxed{\text{TEMPORALSCHEDULING}(gpulet)}$
      return *gpulet*
    end
  end
end



Duration of Gpulet decided by squishy bin-packing [1]

Design – *Cost-effective scheduling*

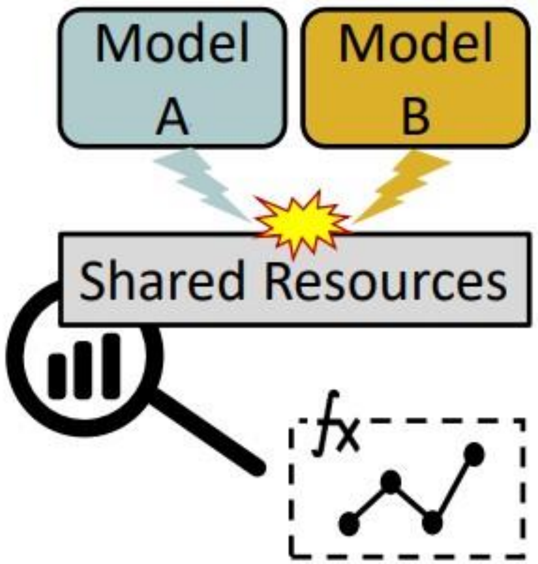
- **Scaling GPUs for request rate changes**

Algorithm 2: GPU Scaling Algorithm

```
SCALING(GPU_LIMIT):
for each period do
    N ← The number of used GPUs in previous period
    result ← ELASTICPARTITIONING with N GPUs
    while result is fail and N < GPU_LIMIT do
        N ← N + 1
        result ← ELASTICPARTITIONING with N GPUs
    end
    if result is fail then
        Report an unschedulable event
    end
end
```

Design – *Modeling Interference*

Interference prediction

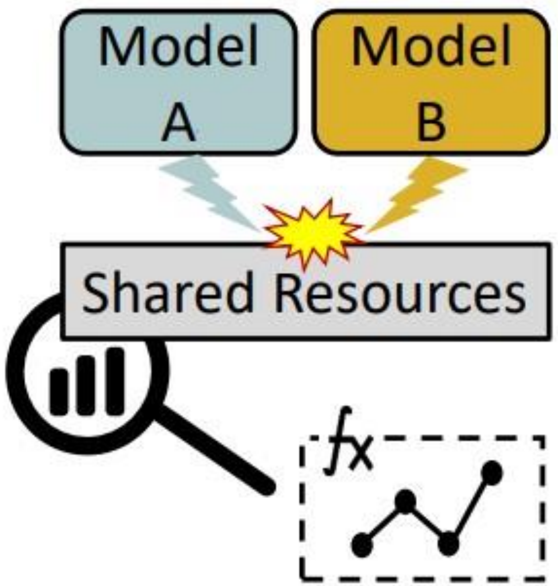


- **L2 utilization**
- **DRAM bandwidth utilization**

$$interference\_factor = c_1 \times L2_{m_1} + c_2 \times L2_{m_2} + c_3 \times mem_{m_1} + c_4 \times mem_{m_2} + c_5$$

Design – *Modeling Interference*

Interference prediction



$$interference\_factor = c_1 \times L2_{m_1} + c_2 \times L2_{m_2} + c_3 \times mem_{m_1} + c_4 \times mem_{m_2} + c_5$$

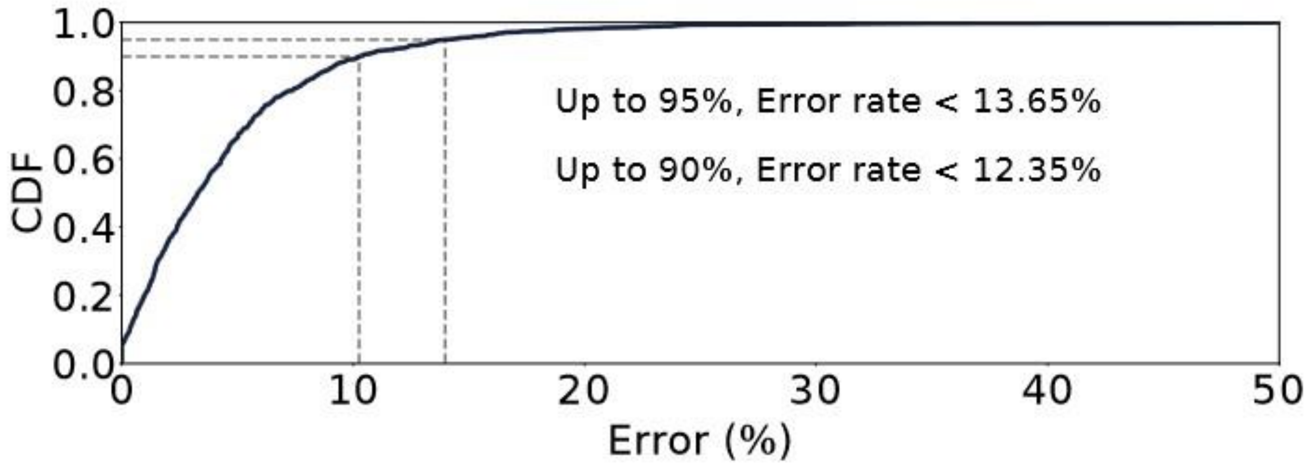
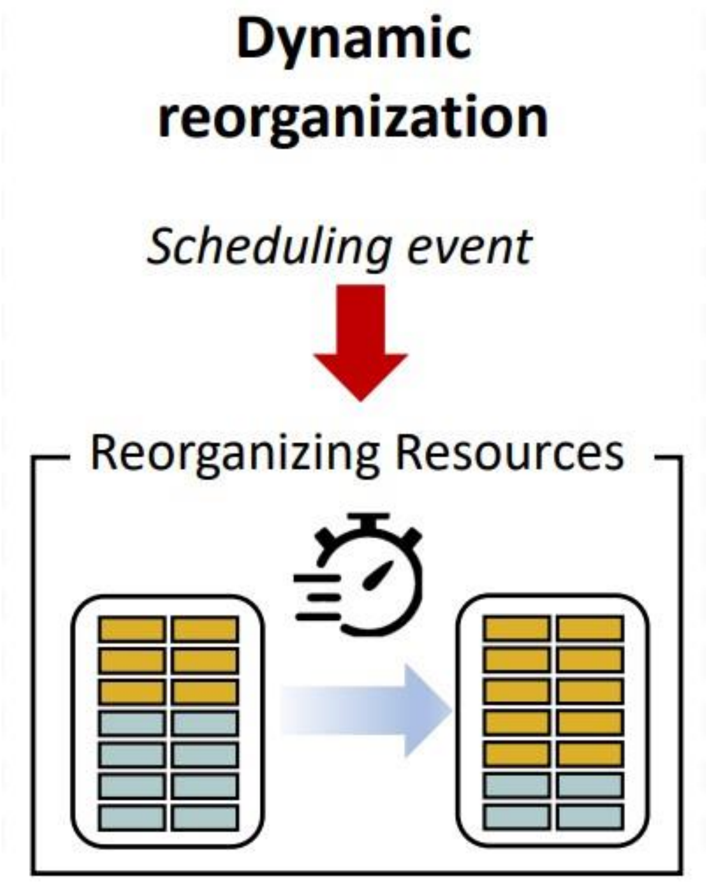
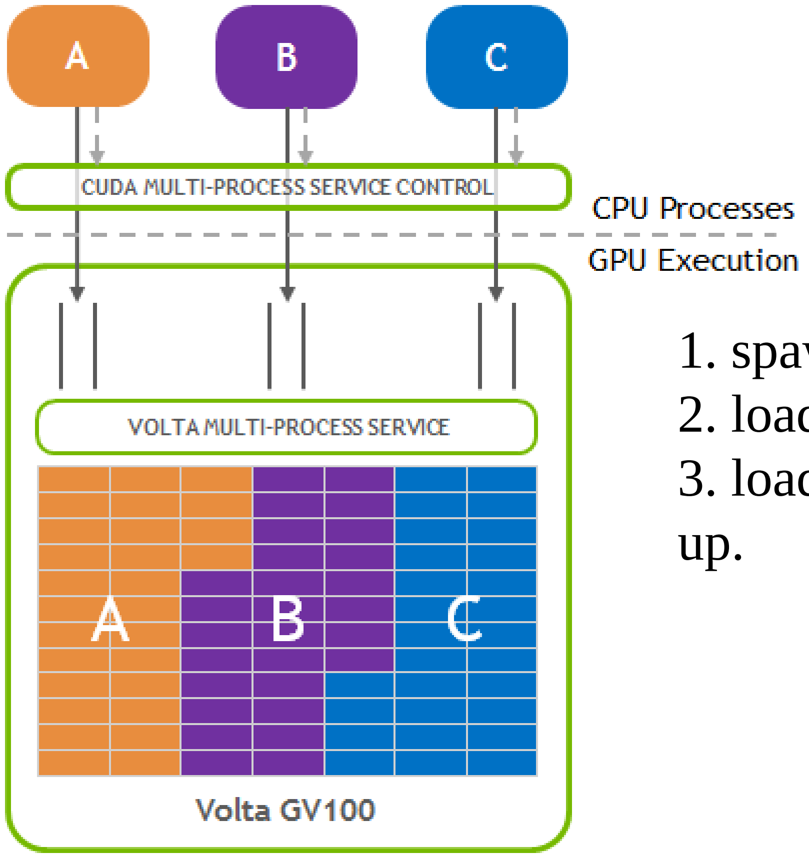


Figure 10: Cumulative distribution of relative error rate. Proposed analytical model can predict up-to 95% of cases with less than 13.98 % error rate.

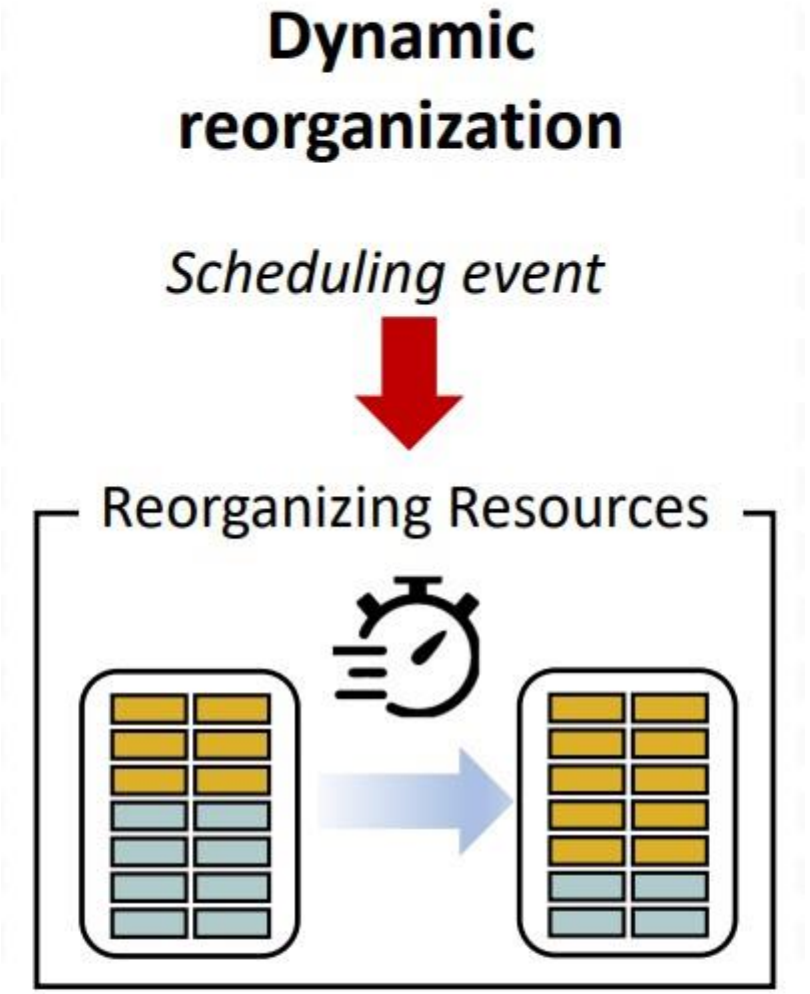
Design – *Dynamic reorganization*





1. spawning a new process
2. loading kernels used by PyTorch
3. loading required models, and warming up.

Design – *Dynamic reorganization*



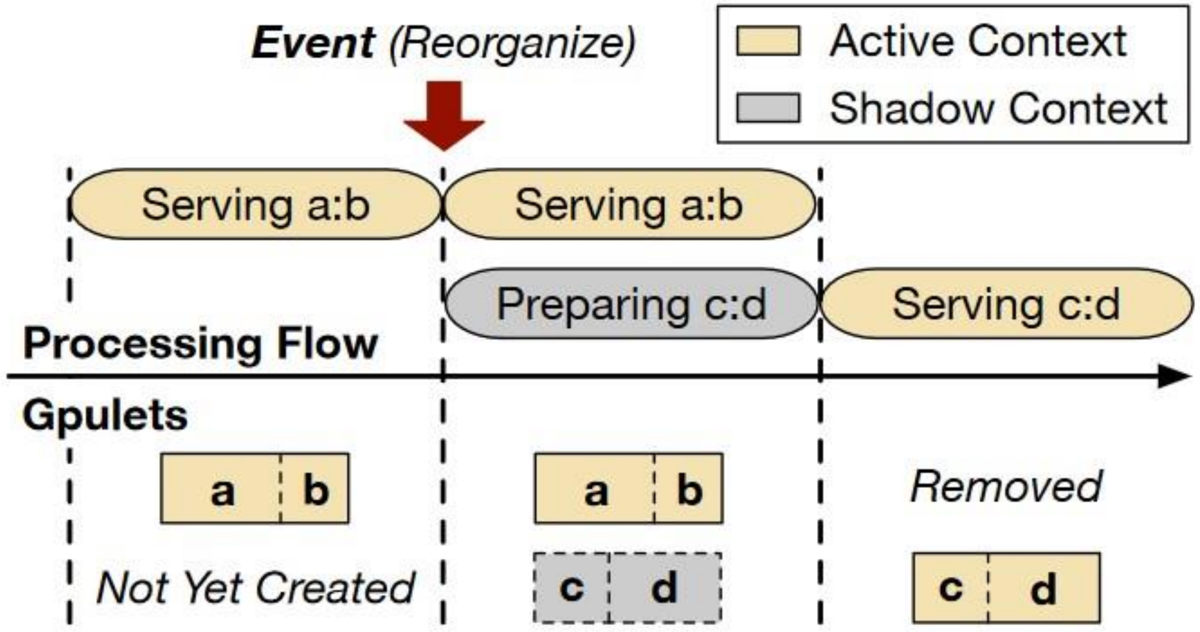


Figure 11: Illustration of dynamic partition reorganization.

Implementation

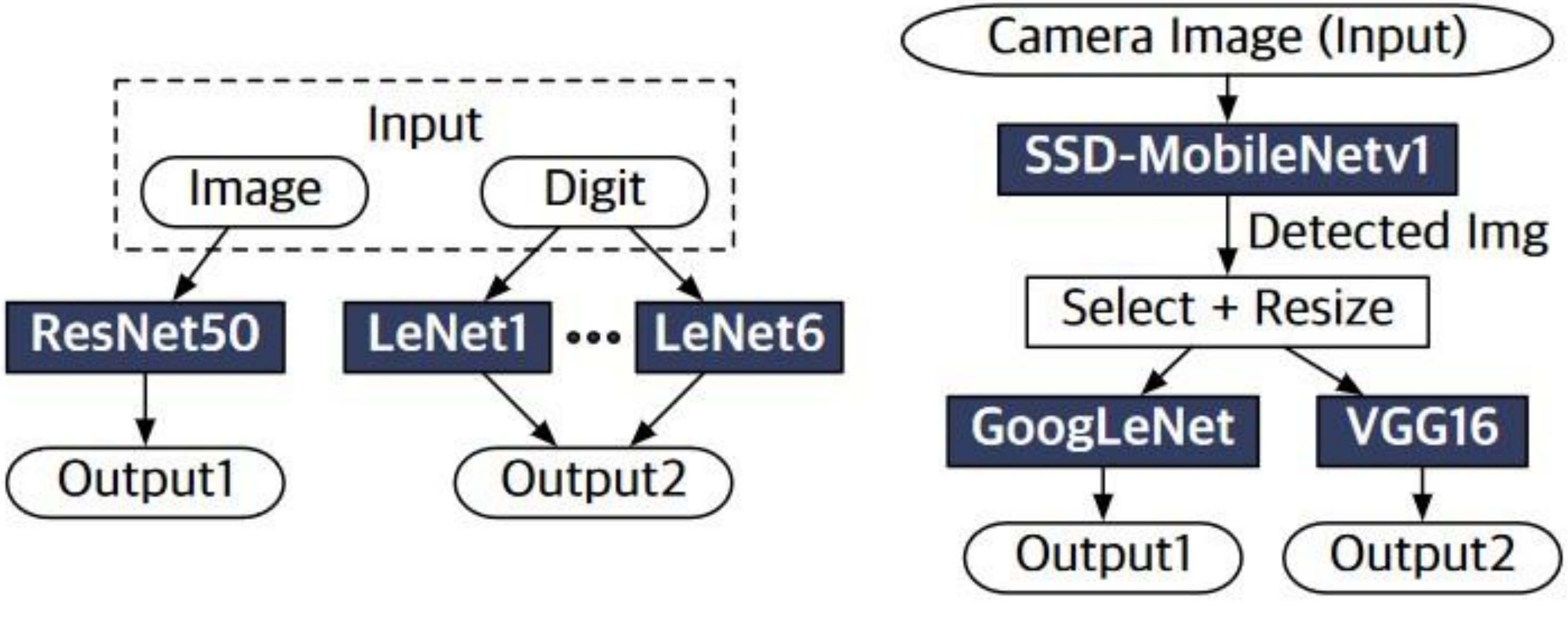
- **System Overview**

System Overview	
CPU	20-core, Xeon E5-2630 v4
GPU	2 × RTX 2080 Ti
Memory Capacity	192 GB DRAM
Operating System	Ubuntu 18.04
CUDA	10.2
NVIDIA Driver	440.64
ML framework	PyTorch 1.10
GPU Specification	
CUDA cores	4,352
Memory Capacity	11 GB GDDR6
Memory Bandwidth	616 GB/sec

Table 2: The evaluated system specifications.

Implementation

- **2 Applications**



(a) Scenario *game* (b) Scenario *traffic*

Figure 12: Two multi-model applications: *game* and *traffic*.

Implementation

- **ML models used in the evaluation**

Model	Input Data (Dimension)	SLO (ms)
GoogLeNet (goo)	ImageNet (3x224x224))	66
LeNet (le)	MNIST (1x28x28)	5
ResNet50 (res)	ImageNet (3x224x224)	108
SSD-MobileNet (ssd)	Camera Data (3x300x300)	202
VGG-16 (vgg)	ImageNet (3x224x224)	142
MnasNet (nas)	ImageNet (3x224x224)	62
Mobilenet_v2 (mob)	ImageNet(3x224x224)	64
DenseNet (den)	ImageNet(3x224x224)	202
Base Bert (be)	Rand. Index Vector(1x14)	22

Table 3: List of ML models used in the evaluation.

Implementation

- **5 scenarios**

Name	Group Composition by Memory Footprint		
	<1GB	1GB - 2GB	>2GB
scen1	mob,be	nas,goo	-
scen2	-	den	vgg
scen3	mob	res	vgg
scen4	ssd	nas,den	-
scen5	le	ssd,nas	vgg

Table 4: Five request scenarios, each of which represents a particular composition of multiple models based on memory footprint. The amount of requests per model in a group is equal across the models in the group.

Evaluation – *SLO preserved max throughput*

- **SLO violate < 1 %**
- **Best performance when both time and spatial scheduling enabled**
- **throughput increased by an average 61.7% than time-share**

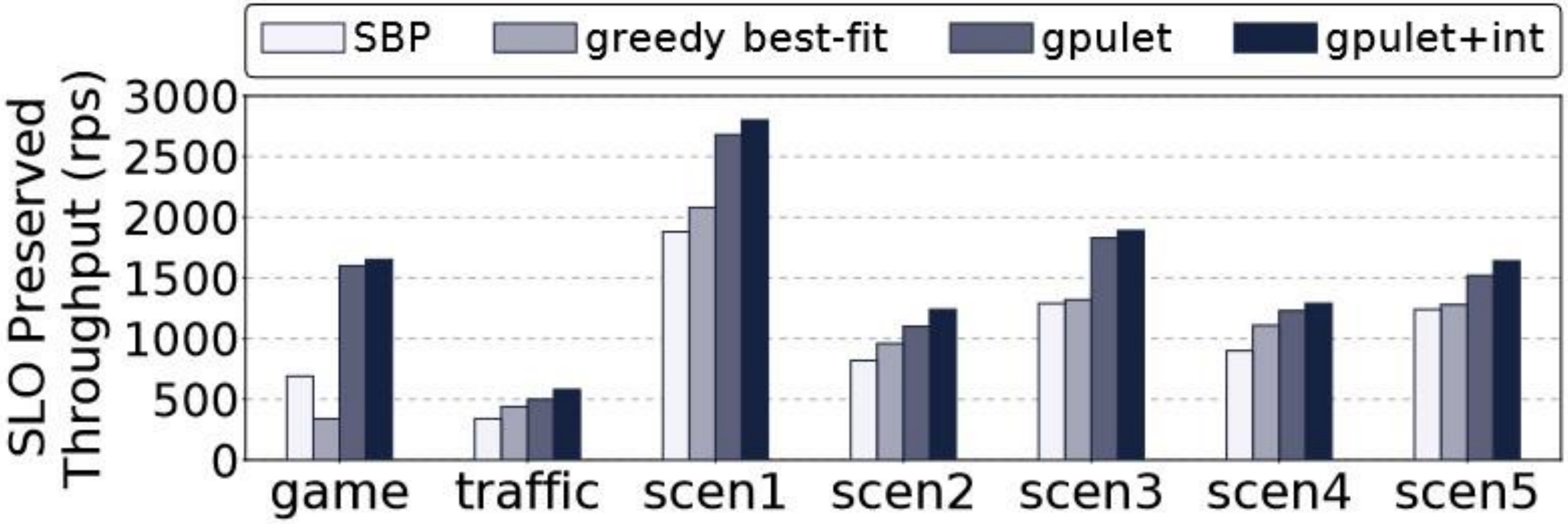


Figure 13: SLO preserved max throughput of the two multi-model applications (game and traffic) and five scenarios.

Evaluation – *The effect of interference model*

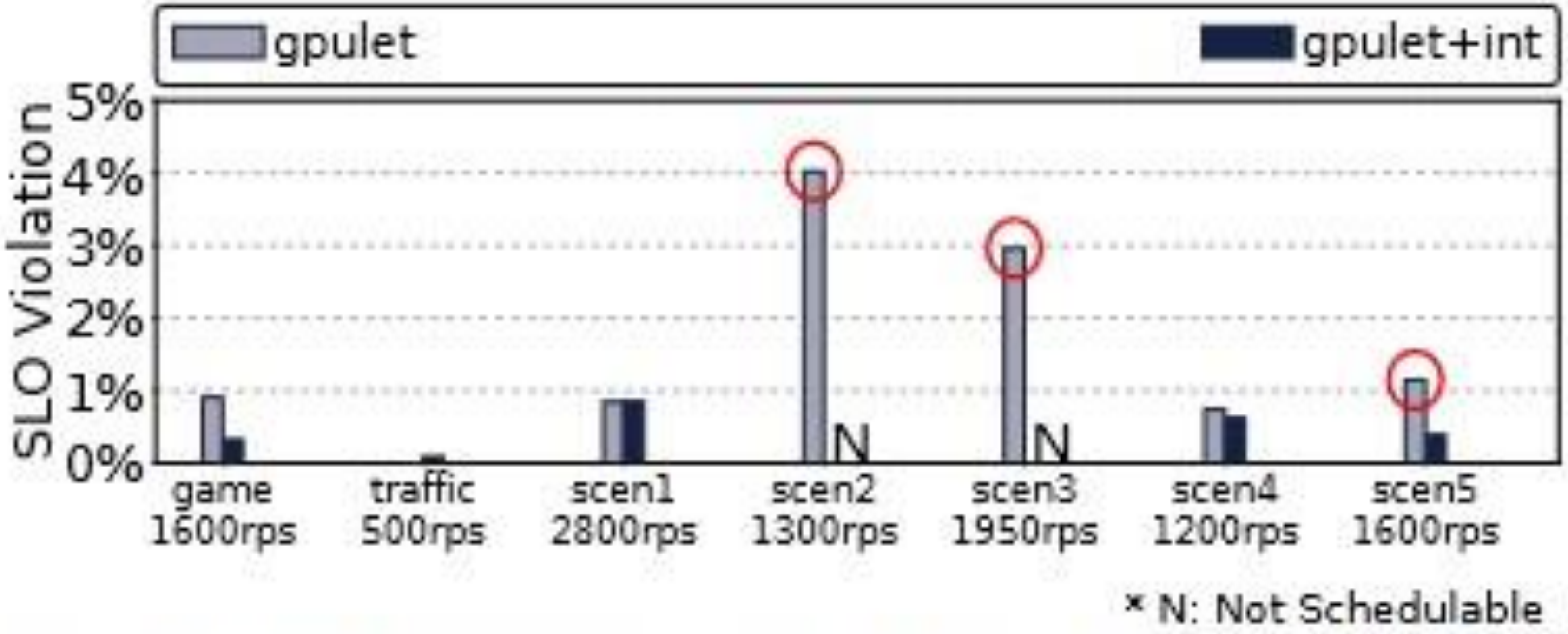
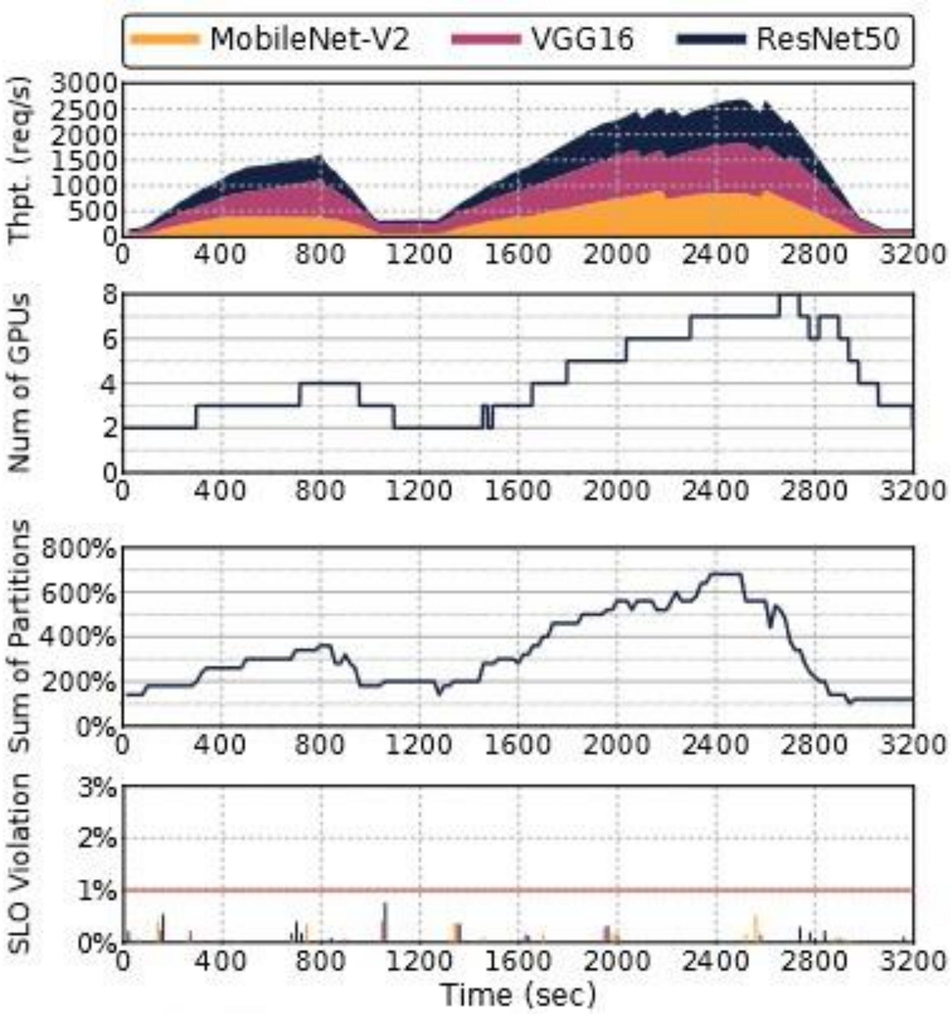


Figure 14: SLO violation rates of two multi-model applications and five scenarios. Request rates are increased until both gpulet and gpulet +int concluded the rate to be *Not Schedulable*.

Evaluation – *Evaluation of scalability*

scen3	mob	res	vgg



Minimize Resource Usage

Evaluation – *Comparison to the ideal scheduler*

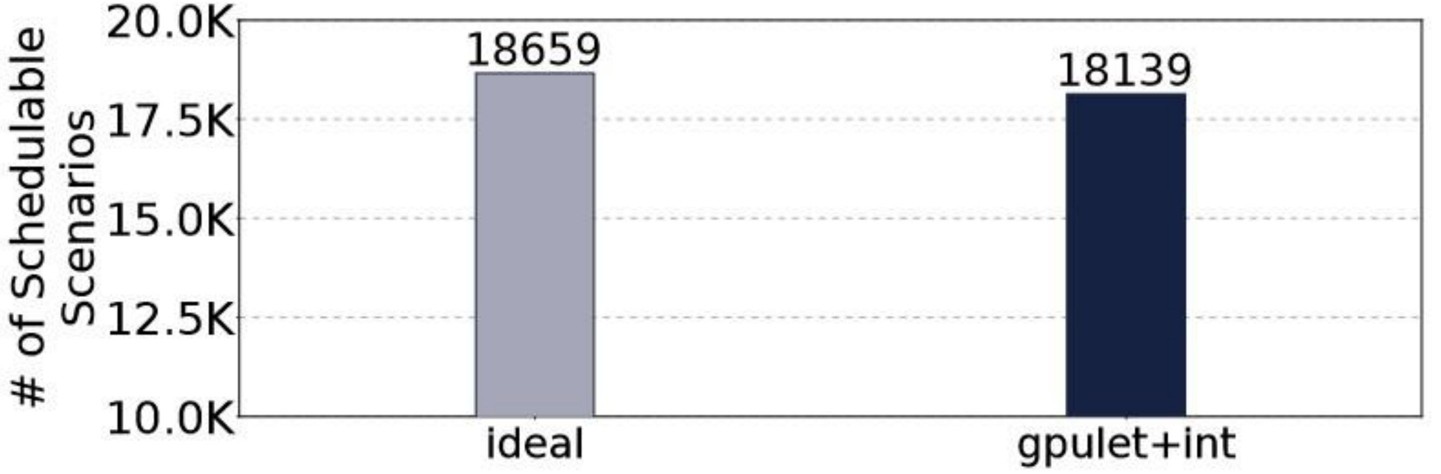
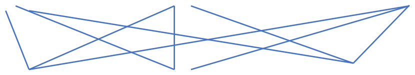


Figure 16: Comparison of the numbers of schedulable scenarios between the ideal scheduler and gpulet +int scheduler.

Model 1 Model 2 … Model 9



0rqs 100rqs 200rqs

3^9 -1 scenarios

Advantage

- Improve the performance of ML inference by spatial-temporal scheduling.
- Predicting interference effect when scheduling.
- Scaling resource efficiently.

Features	Batch Tuning	Multi Model	GPU Scaling	Temporal Schedule	Spatial Schedule	Interference Prediction
Clipper [15]	✓	✓	✓	✓	✗	✗
MArk [45]	✓	✗	✓	✗	✗	✗
INFaaS [36]	✓	✓	✓	✓	✗	✗
Nexus [38]	✓	✓	✓	✓	✗	✗
GSLICE [17]	✓	✓	✗	✗	✓	✗
Gpulet	✓	✓	✓	✓	✓	✓

Table 5: Comparison with prior work.

Disadvantage

- Heterogeneous hardware environments are not taken into account
- Not enough considerations for interference model.