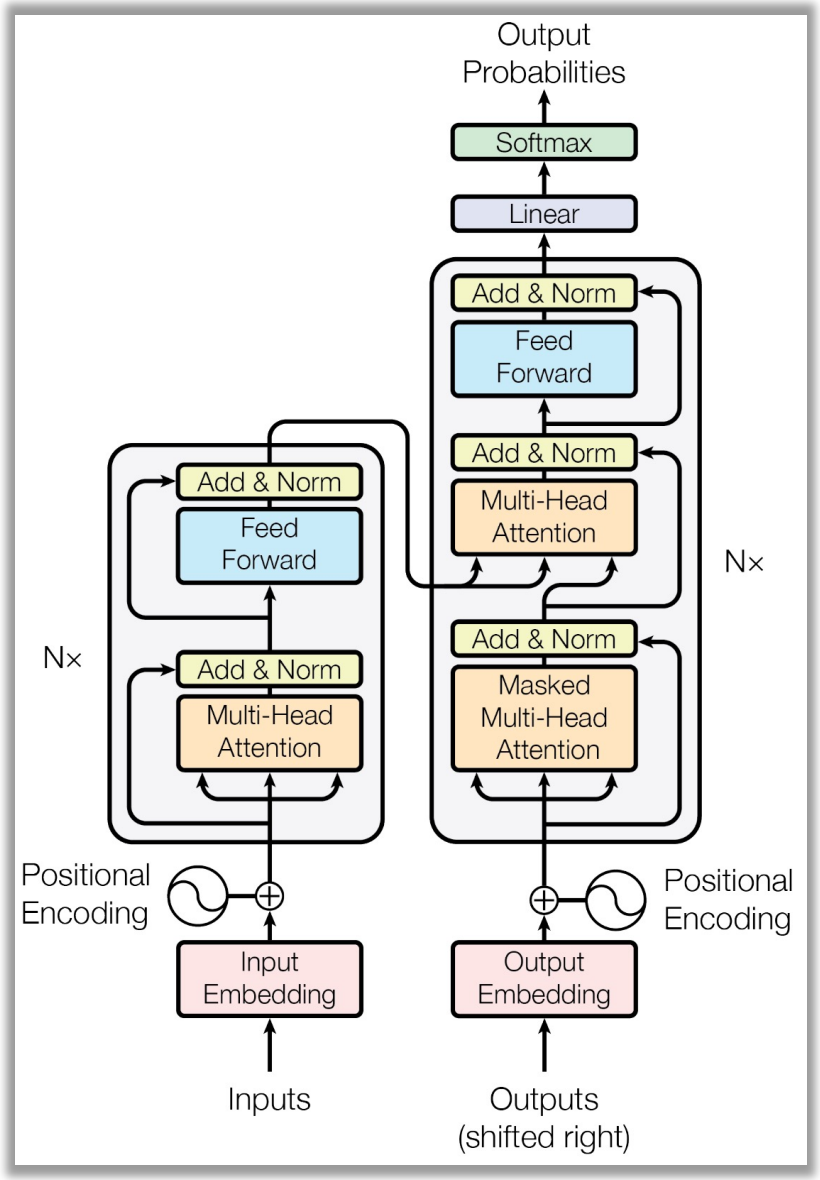
PetS: A Unified Framework for Parameter-Efficient Transformers Serving

Zhe Zhou[1], Xuechao Wei[1,2], Jiejing Zhang[2], Guangyu Sun[1]
[1]Peking University
[2]Alibaba Group

北京大学
PEKING UNIVERSITY

We Are in the Transformers Era!



Language Modeling on WikiText-103

Outperform RNNs!

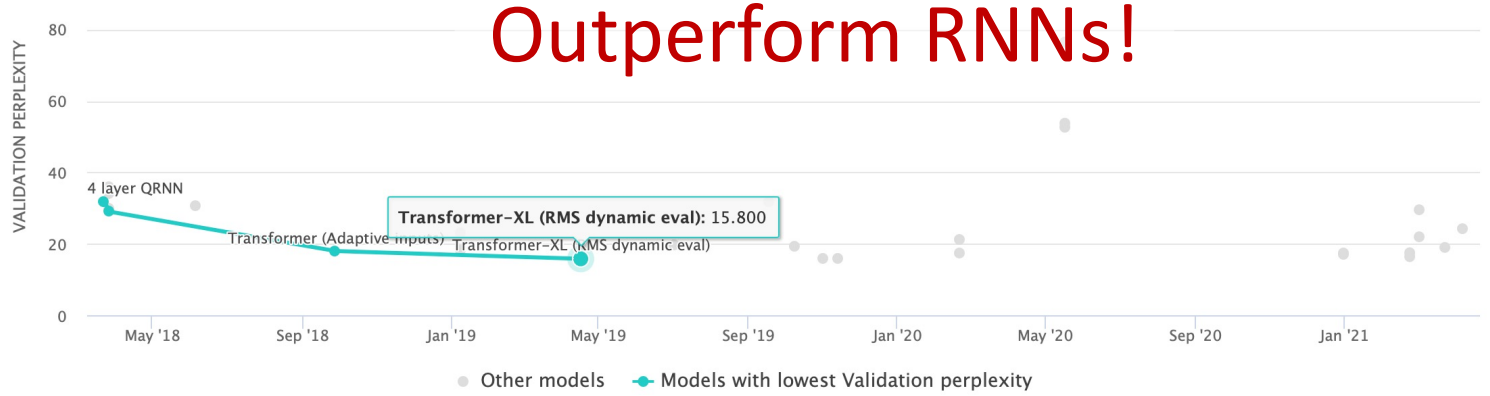
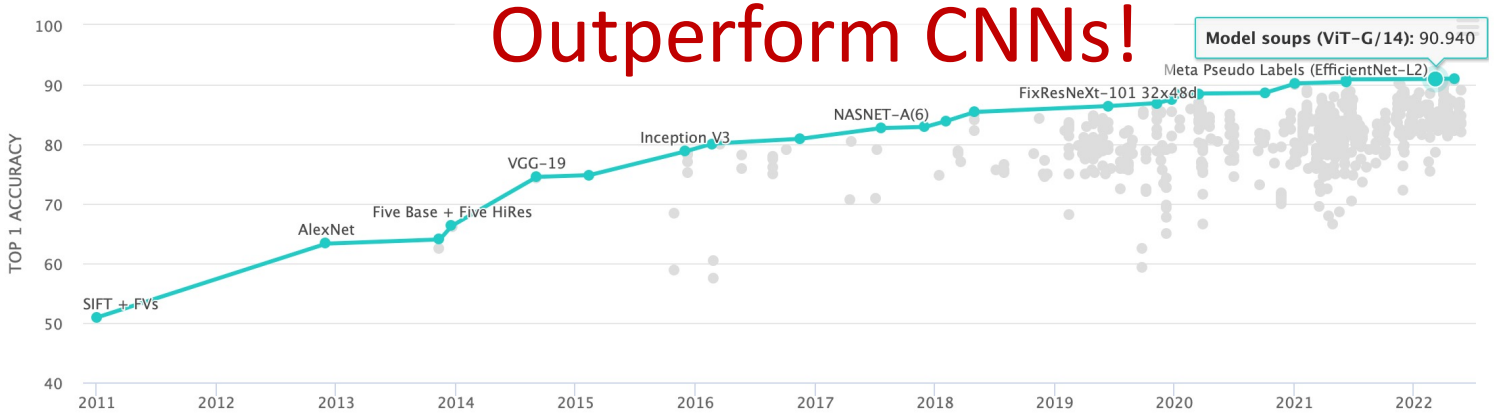


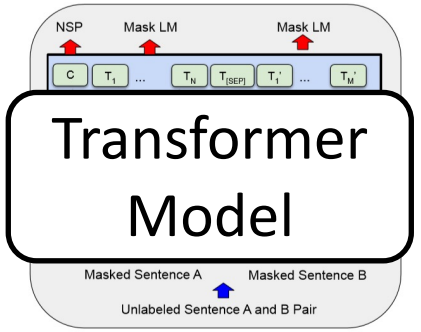
Image Classification on ImageNet

Outperform CNNs!



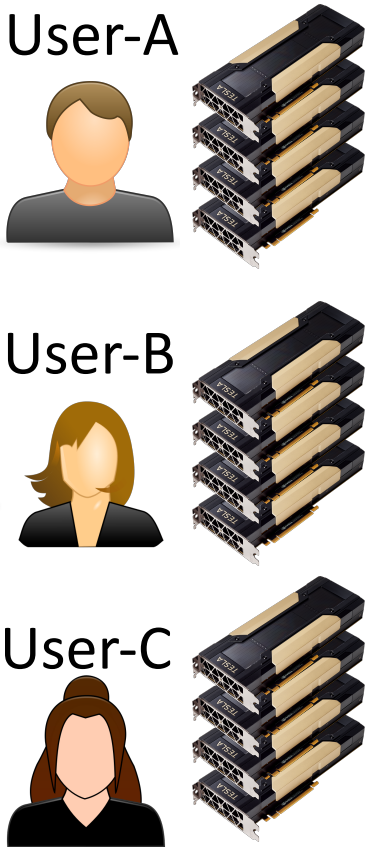
https://paperswithcode.com/sota/

Pretrain-then-finetune Paradigm

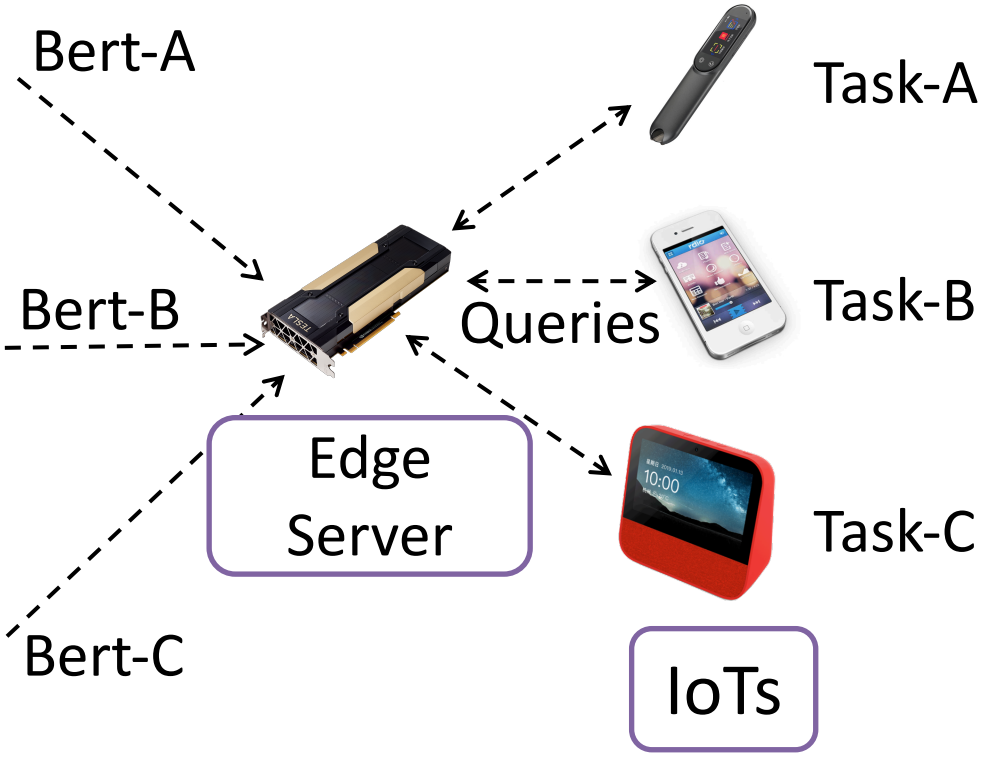


TPU/GPU Cluster

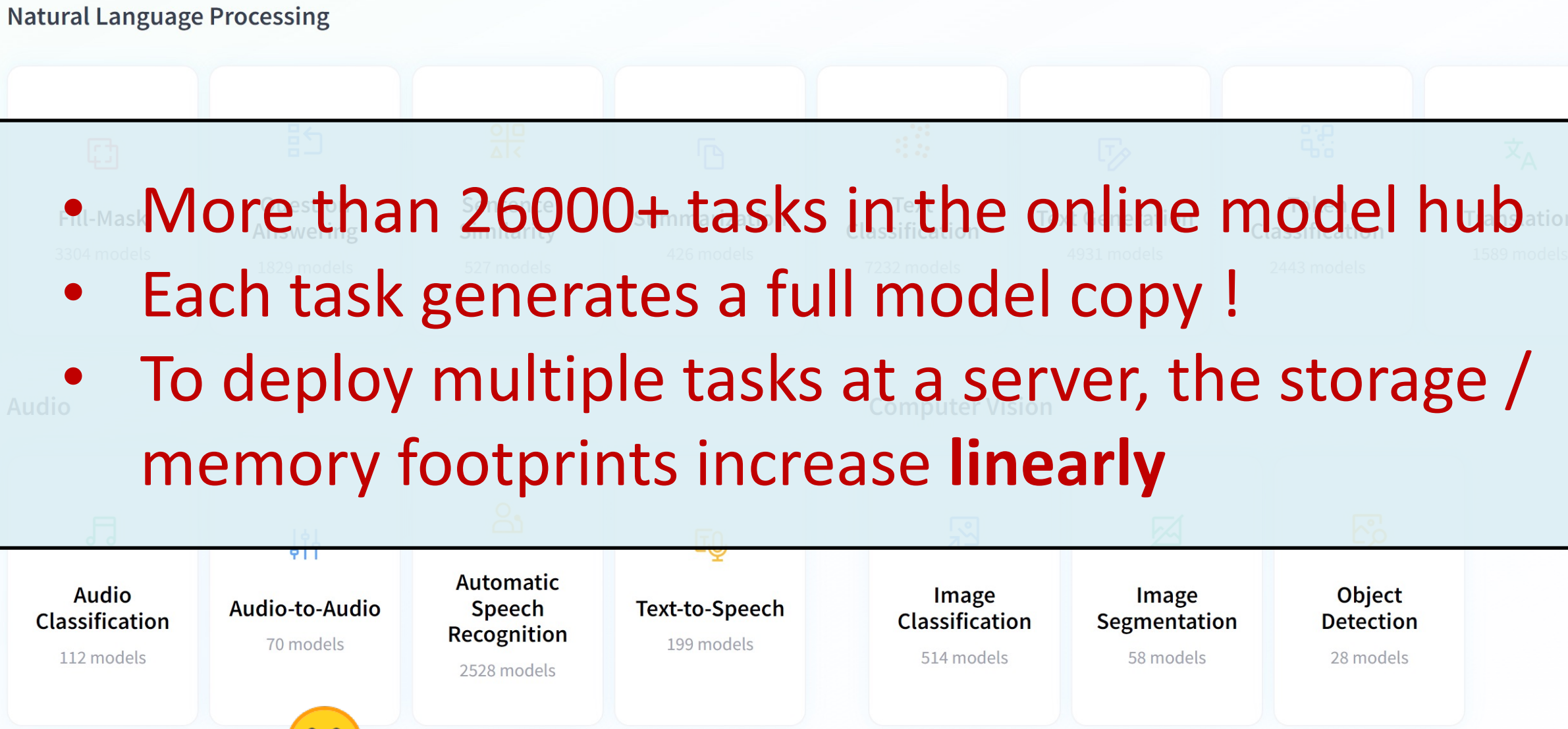
Bert



GPU Servers



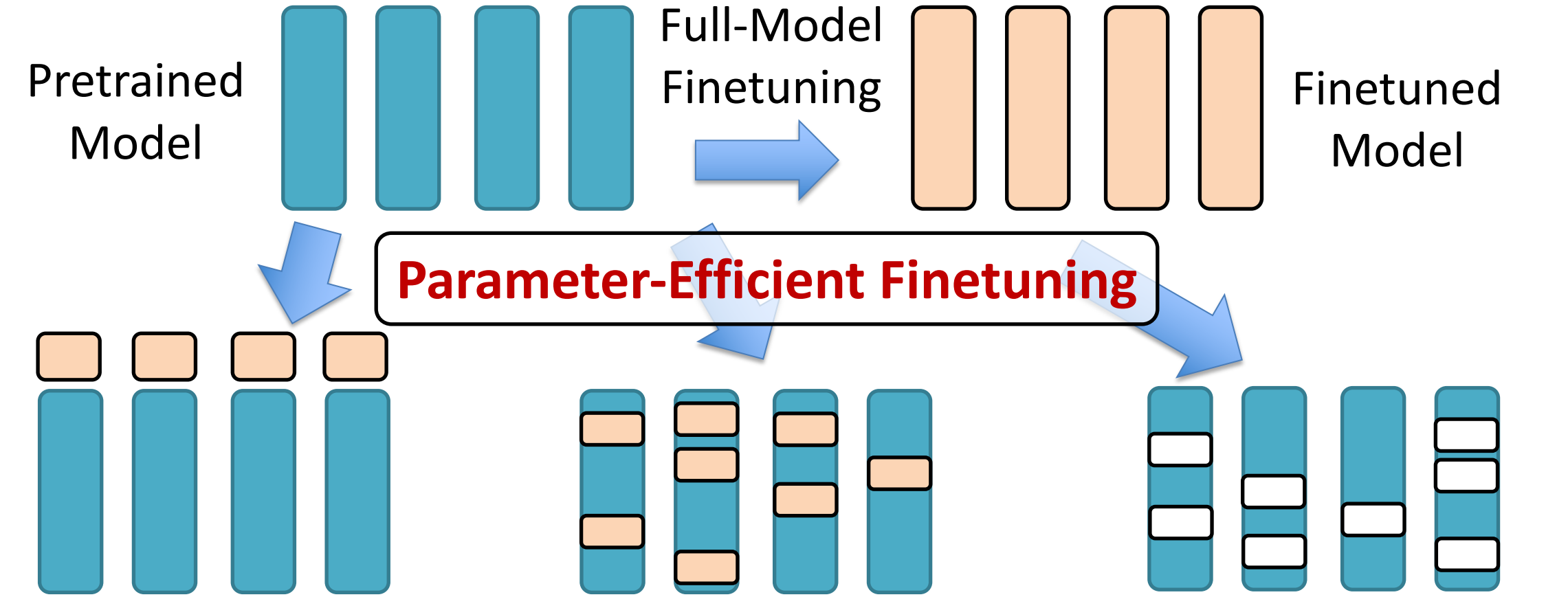
Explosion of Down-stream Tasks



- More than 26000+ tasks in the online model hub
- Each task generates a full model copy !
- To deploy multiple tasks at a server, the storage / memory footprints increase **linearly**

Hugging Face https://huggingface.co/tasks

Parameter-Efficient Transformers (PETs)



Challenges of PETs Serving

- 1. How to support various PETs in one framework?
  - Downstream tasks may favor different PETs
  - Application developers usually choose their best PETs.
- 2. How to mitigate the GPU-memory footprint?
  - Existing serving frameworks like *TurboTransformers*, *FairSeq*, etc, should feed into full model copies.
- 3. How to improve the system throughput?
  - Queries of different tasks can hardly be batched due to the model parameter / algorithm differences.

Unified Representation of PETs

①	②	③	④
Adapters	MaskBert	Diff-Pruning	BitFit

Multiple PET Tasks = Shared Operations + PET-specific Operations

①: $\boldsymbol{Y_t = \sigma((X_t \cdot W + b) \cdot W_{down}) \cdot W_{up}}$

②: $\boldsymbol{Y_t = X_t \cdot (M_t \odot W) + b}$

③: $\boldsymbol{Y_t = X_t \cdot (W + \delta_t) + b + b_t}$

④: $\boldsymbol{Y_t = X_t \cdot W + b + b_t}$

$\boldsymbol{Y_t = X_t \cdot W + b}$

$\boldsymbol{Y_t = X_t \cdot W + b}$

$\boldsymbol{Y_t = X_t \cdot W + b}$

$\boldsymbol{Y_t = X_t \cdot W + b}$

$\boldsymbol{\sigma(\cdot W_{down})W_{up}}$

$\boldsymbol{- X_t \cdot (W \odot \bar{M}_t)}$

$\boldsymbol{+ X_t \cdot \delta_t + b_t}$

$\boldsymbol{+ b_t}$

Original Representations

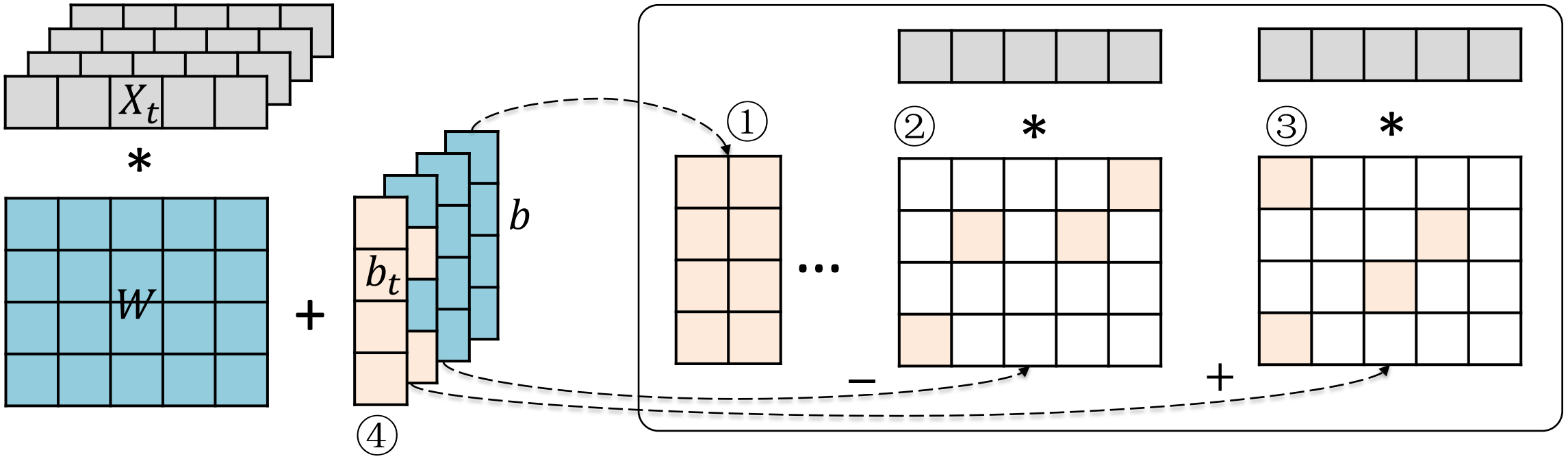
Unified Representation

Unified Representation of PETs

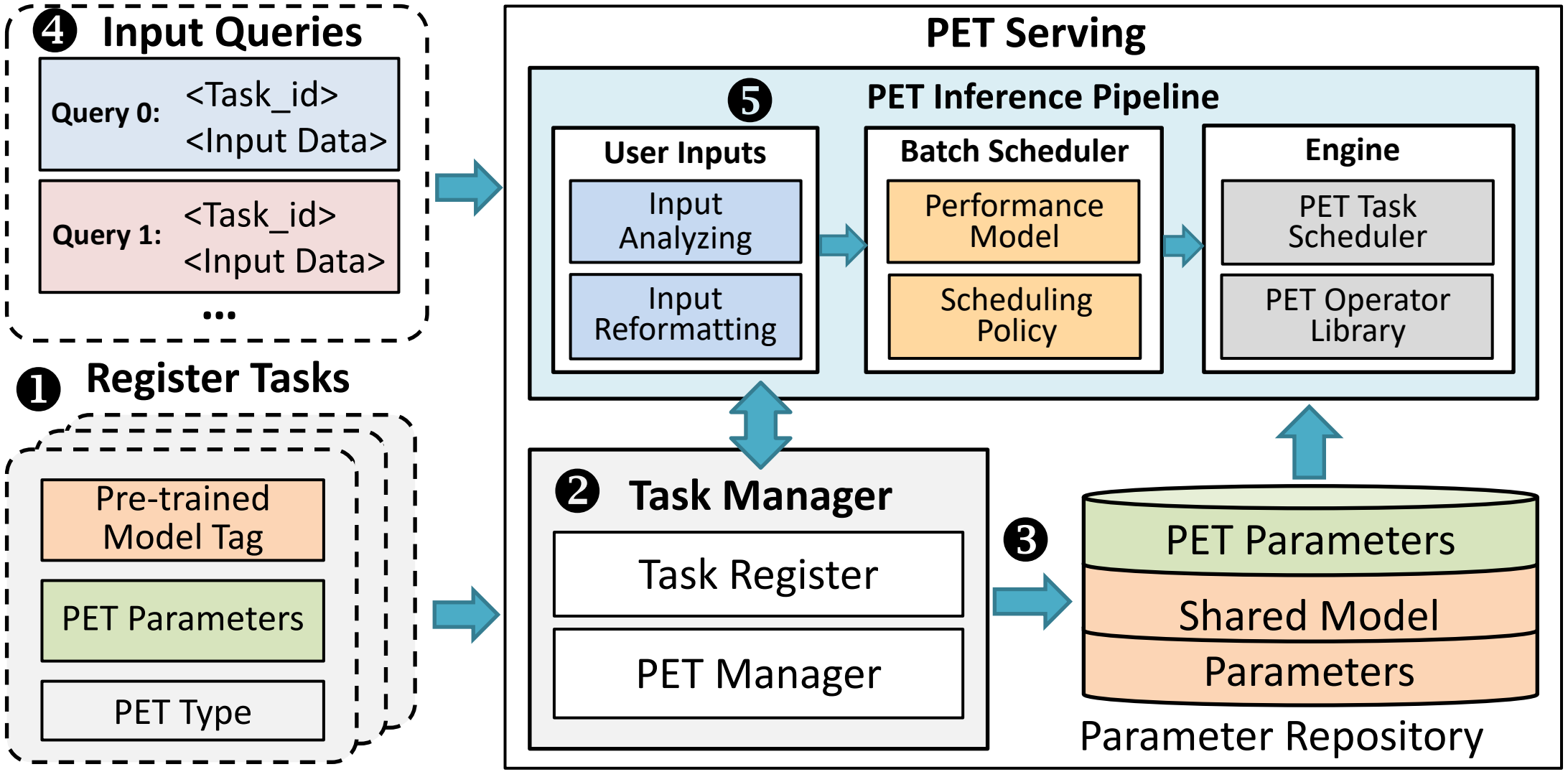
①	②	③	④
Adapters	MaskBert	Diff-Pruning	BitFit

Shared operations can be batched among tasks

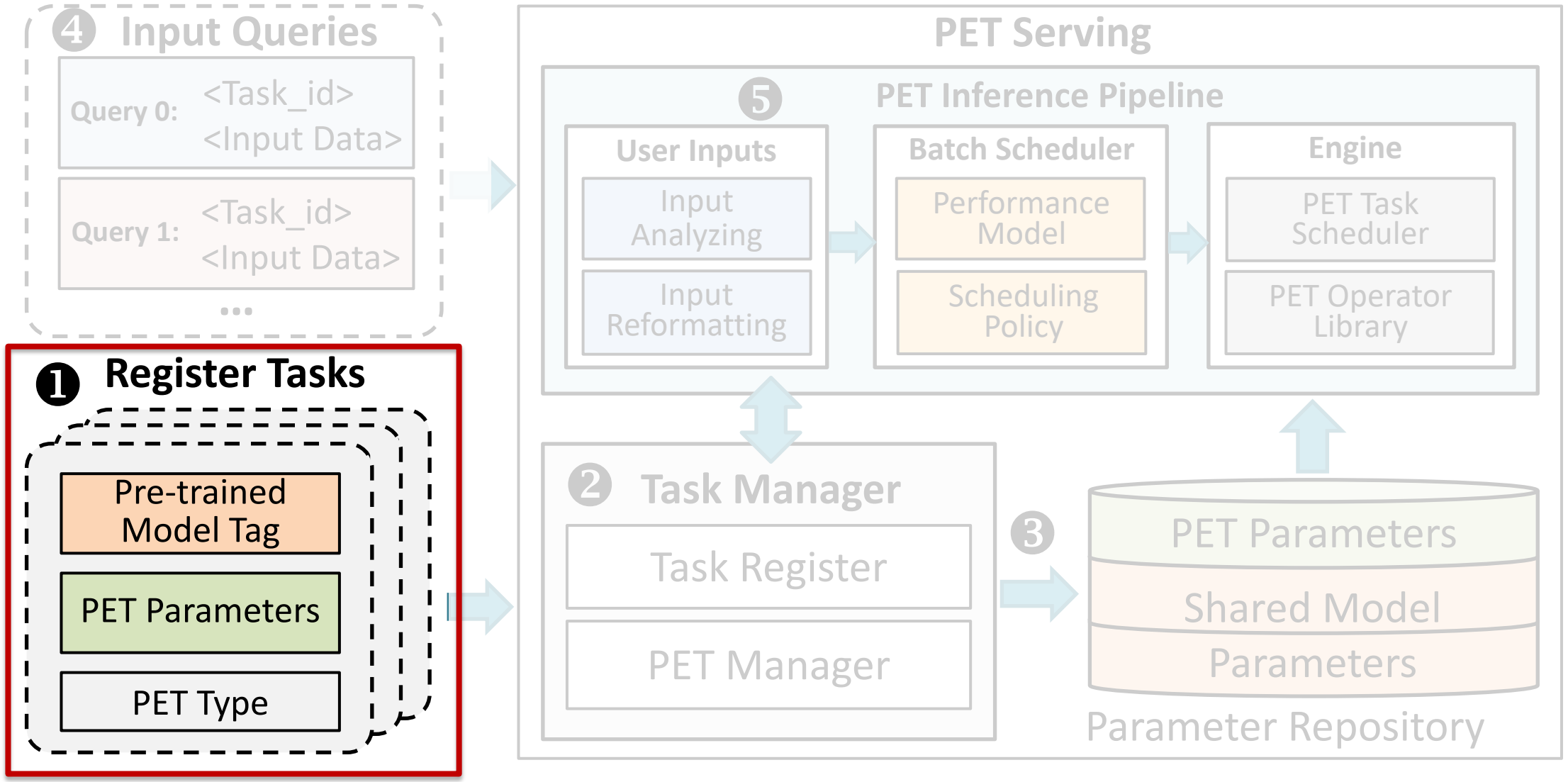
PET Operations can be computed with light-weighted operators



PetS Overview

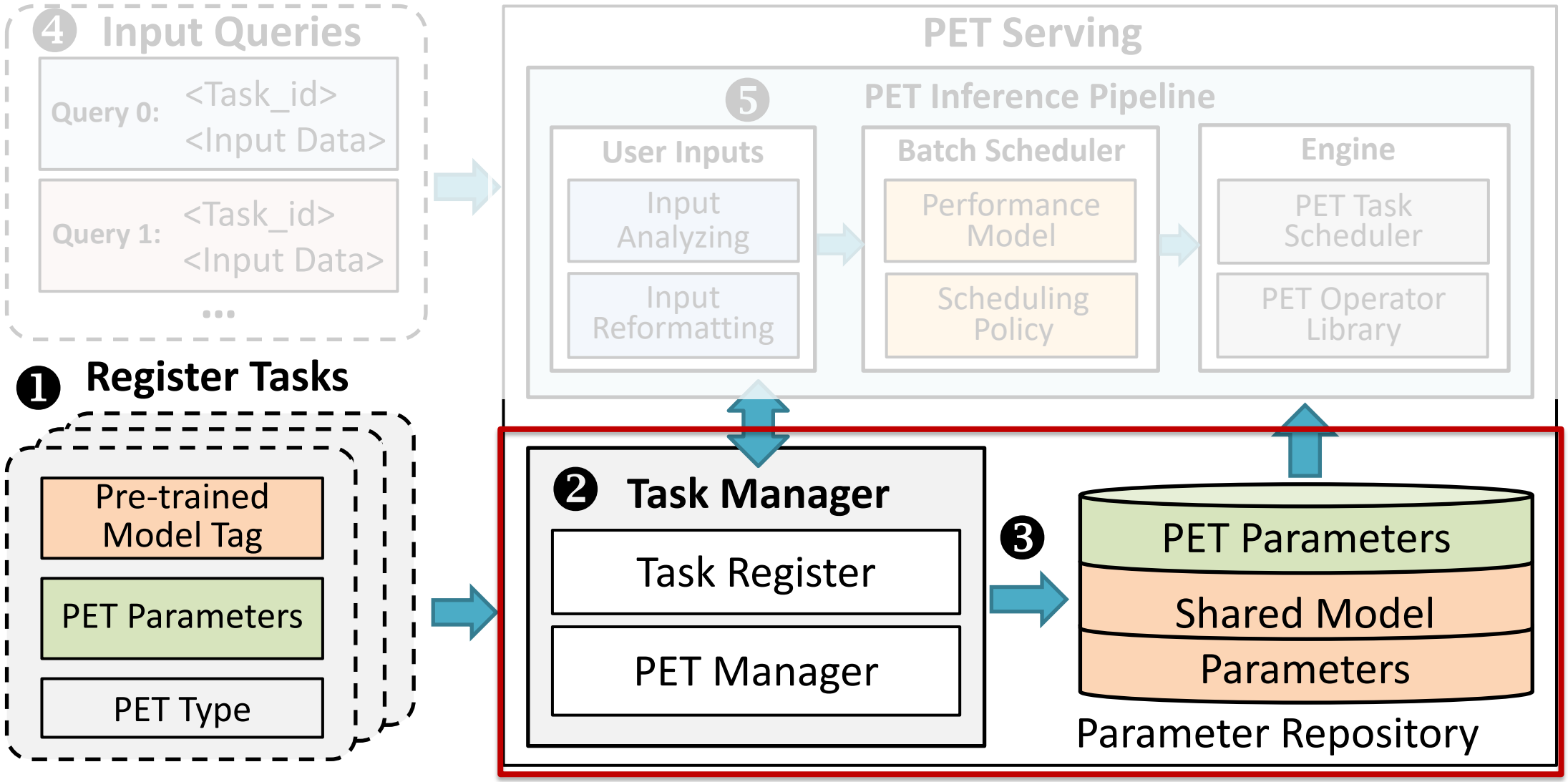


PetS Overview



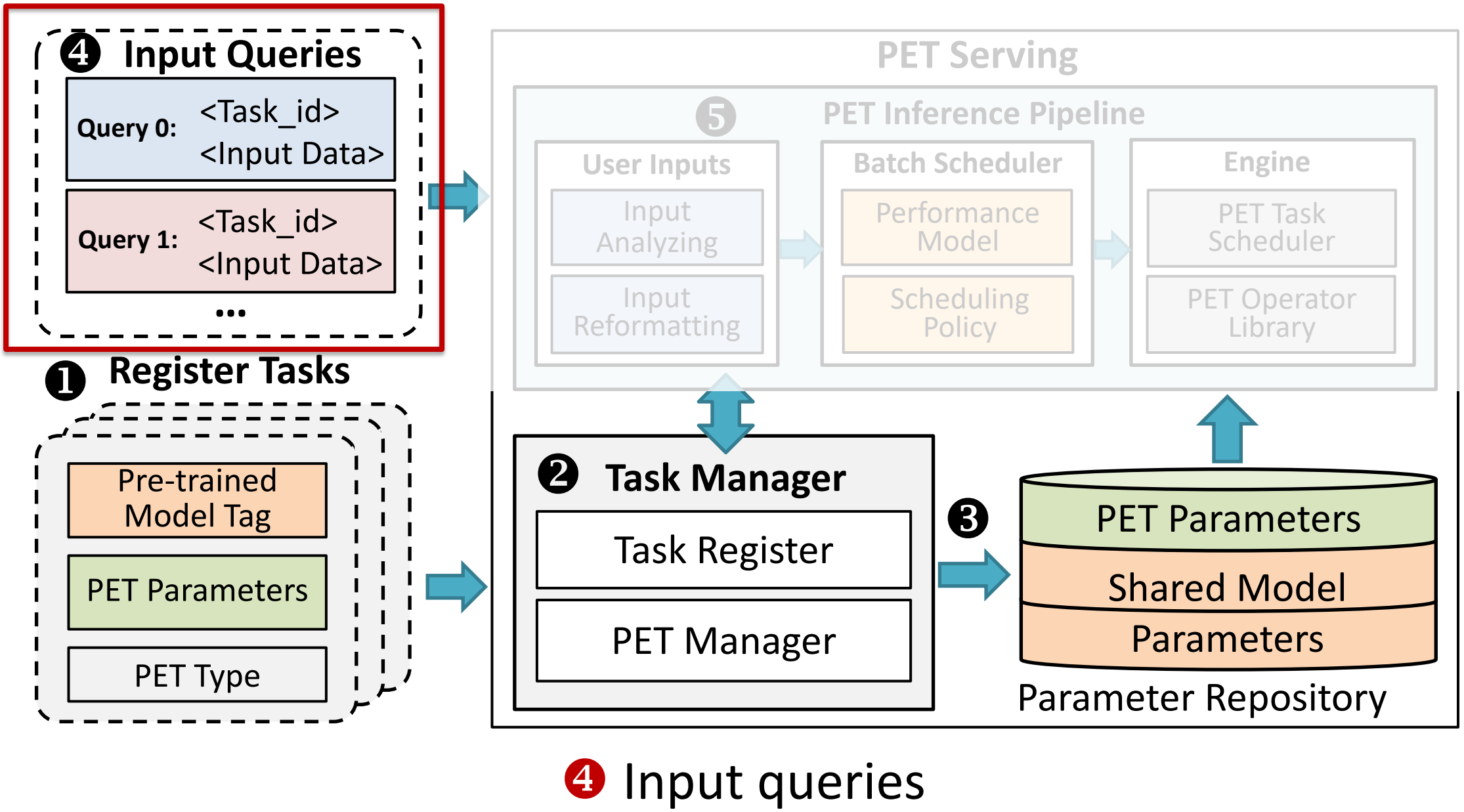
❶ Users register tasks

PetS Overview

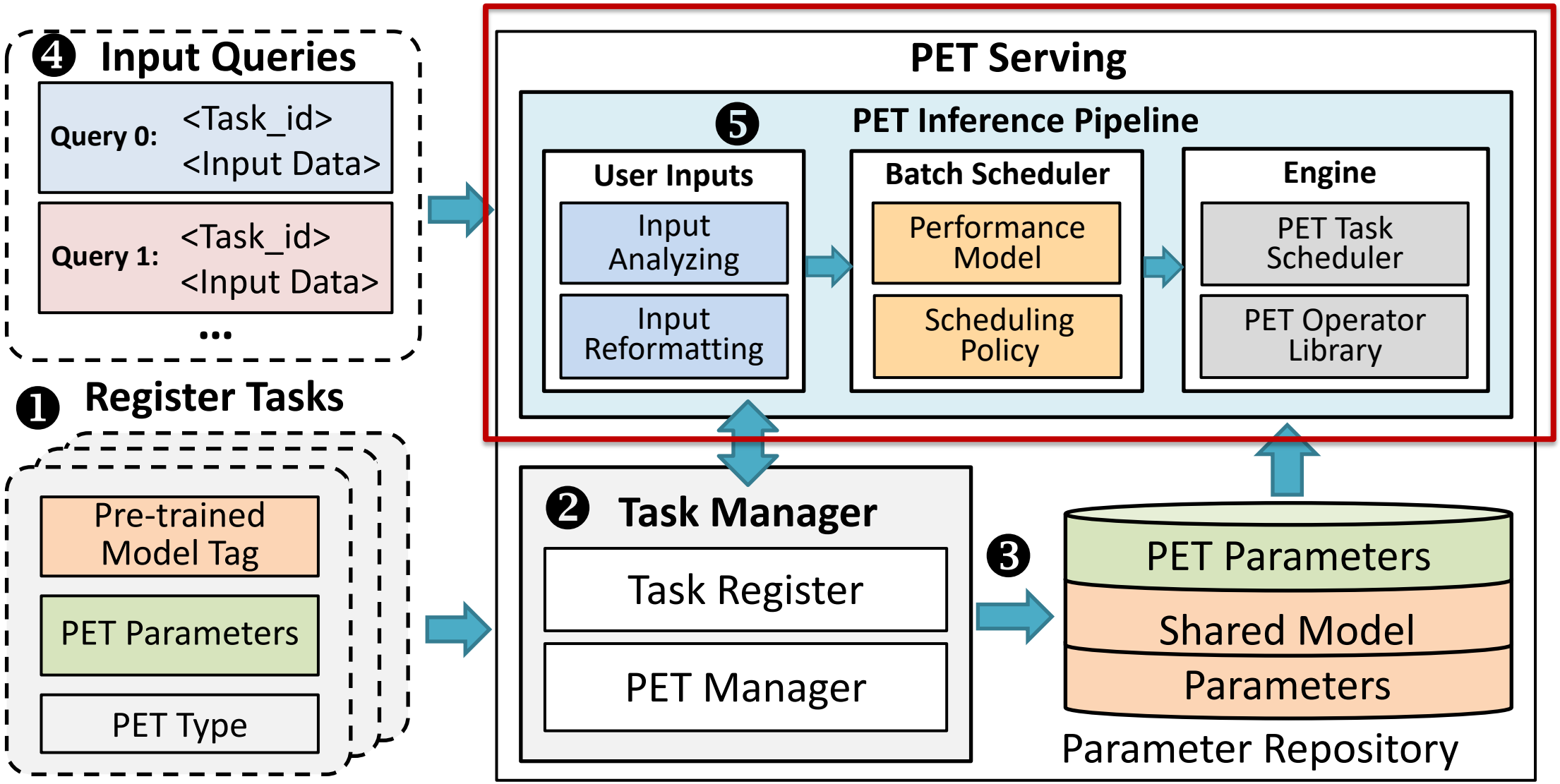


❷-❸ Task Manager manages the registered tasks

PetS Overview



PetS Overview



5 Scheduling + Execution with PIE

User Interface

```
server = PetS() # create a PET server
# Register PET tasks
server.register_task("Adapter", "bert-base", pet_param_url_0)
server.register_task("MaskBert", "bert-base", pet_param_url_1)
# Register other PET tasks ...
# Load shared model parameters and PET tasks
server.load_shared_model("bert-base")
server.load_pet_tasks(pet_task_ids)
# Fetch queries from input query queue and run inference.
queries = server.fetch(input_query_queue)
results = server.inference(queries)
```

User Interface

```
server = PetS() # create a PET server
# Register PET tasks
server.register_task("Adapter", "bert-base", pet_param_url_0)
server.register_task("MaskBert", "bert-base", pet_param_url_1)
# Register other PET tasks ...
# Load shared model parameters and PET tasks
server.load_shared_model("bert-base")
server.load_pet_tasks(pet_task_ids)
# Fetch queries from input query queue and run inference.
queries = server.fetch(input_query_queue)
results = server.inference(queries)
```

User Interface

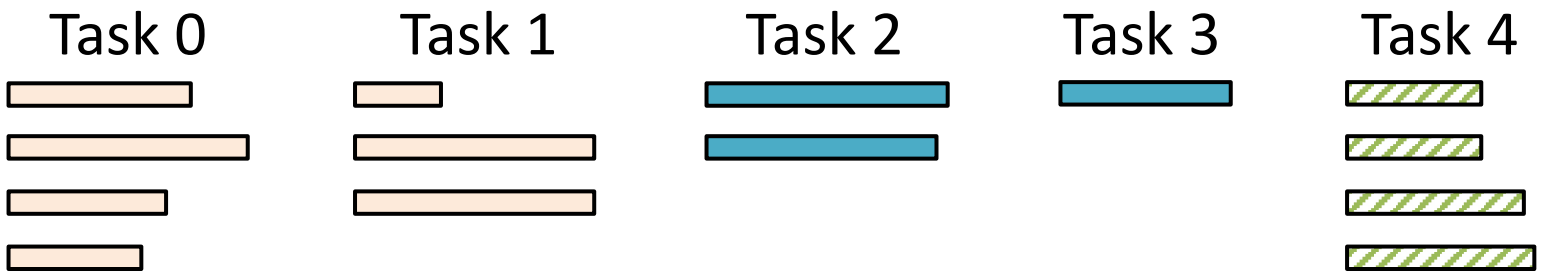
```
server = PetS() # create a PET server
# Register PET tasks
server.register_task("Adapter", "bert-base", pet_param_url_0)
server.register_task("MaskBert", "bert-base", pet_param_url_1)
# Register other PET tasks ...
# Load shared model parameters and PET tasks
server.load_shared_model("bert-base")
server.load_pet_tasks(pet_task_ids)
# Fetch queries from input query queue and run inference.
queries = server.fetch(input_query_queue)
results = server.inference(queries)
```

Optimization Strategies

- **Challenge 1:**
  - The input queries have variable lengths
  - The invoked PET operators are also different
  - How to split the queries into batches to maximum throughput?
    - Coordinated batch scheduling
- **Challenge 2:**
  - The PET operators of different tasks are not batched.
  - How to improve the execution efficiency further?
    - PET operator scheduling

Coordinated Batch Scheduling

Task-specific Queries



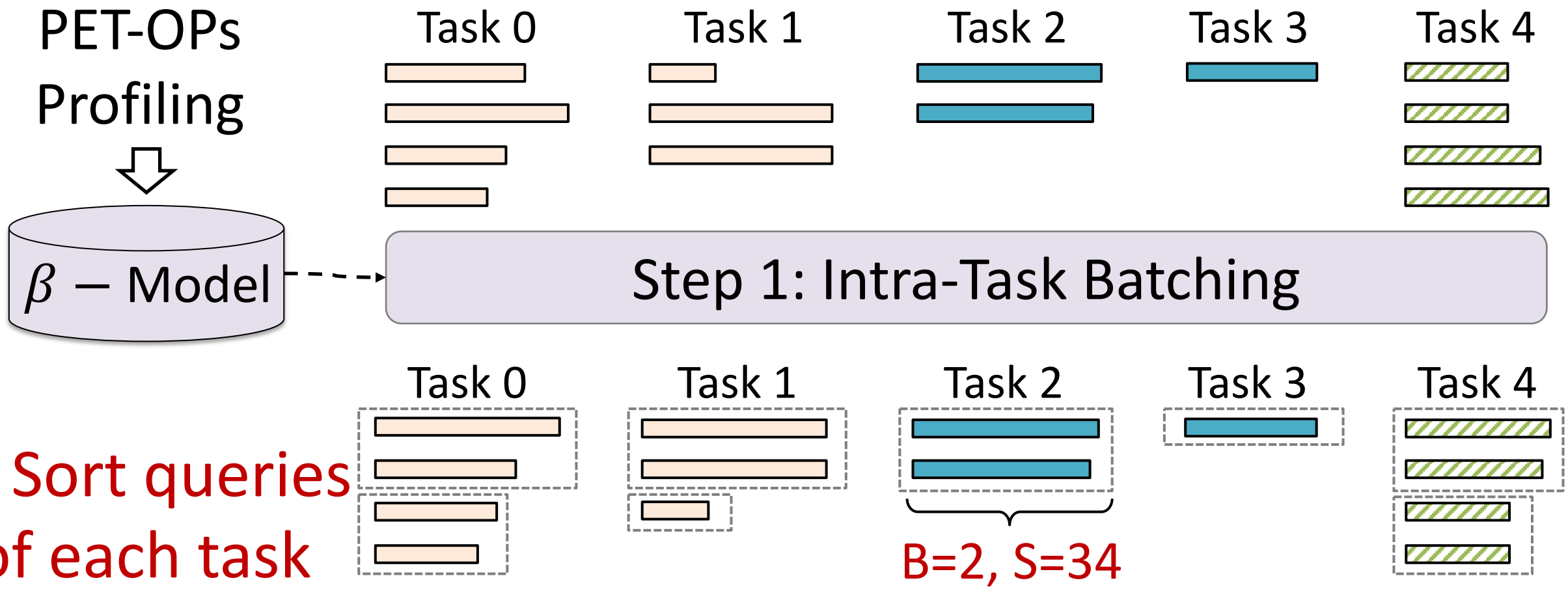
Goal: find an efficient way to batch the task queries

$$Batch\_Latency(B_i) = \alpha[N_i][L_i] + \sum_{j=0}^{t_i-1} \beta[pt_{ij}][n_{ij}][l_{ij}]$$

Shared Op Latency

PET-Op Latency

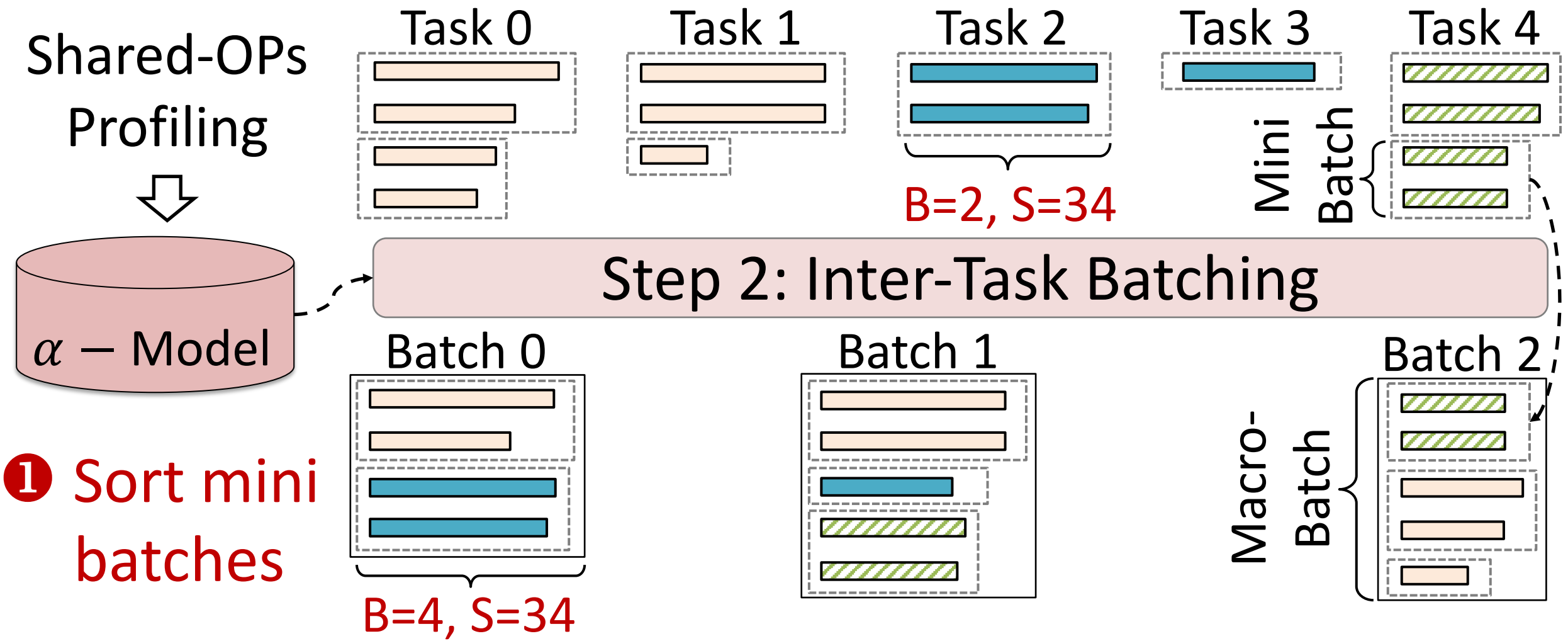
Coordinated Batch Scheduling



❶ Sort queries of each task

❷ Split to mini batches using β model and a DP algorithm

Coordinated Batch Scheduling

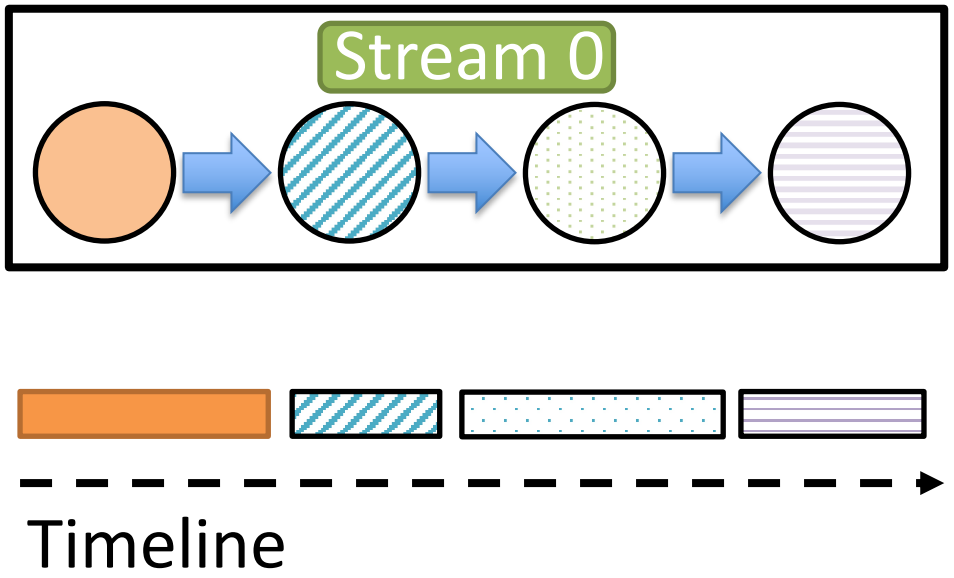


❶ Sort mini batches

❷ Split to macro batches using the α model and DP

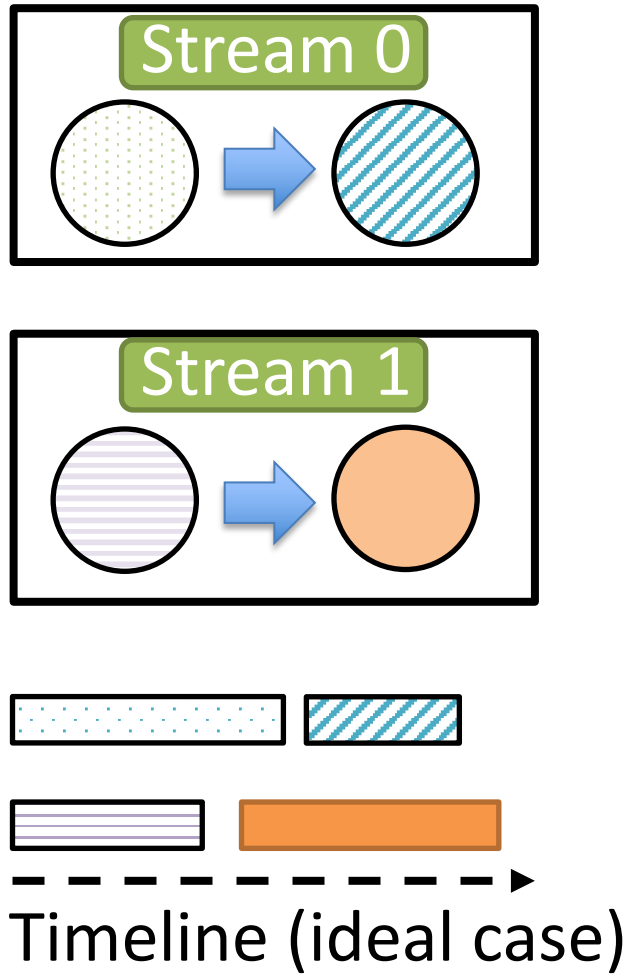
PET Operator Scheduling

Sequential Execution of PET Ops



$$Op\_intensity = \frac{op.FLOPs}{\beta(op) * \omega(op)}$$

Sort & Evenly assign to multiple CUDA streams



Evaluation Setup

H/W Platforms	Edge	Desktop	Server
GPU	Jetson TX2 (8GB)	GTX-1080Ti (11GB)	Tesla V100 (32GB)
CPU	ARM	Xeon E5-2690	Xeon 5220
Driver/Firmware	Jetpack 4.4.1	CUDA-11.3	CUDA-10.1

Model Type	# Layer	Hidden Size	Inter-Size	# Params
DistilBert	6	768	3072	66M
Bert-base	12	768	3072	110M
Bert-large	24	1024	4096	340M

Model Type	Config
Adapter	Bottleneck = 64
MaskBert	95% Sparsity
Diff-pruning	99.5% Sparsity
Bitfit	N/A

Codebase: TurboTransformers

Maximum Number of Supported Tasks

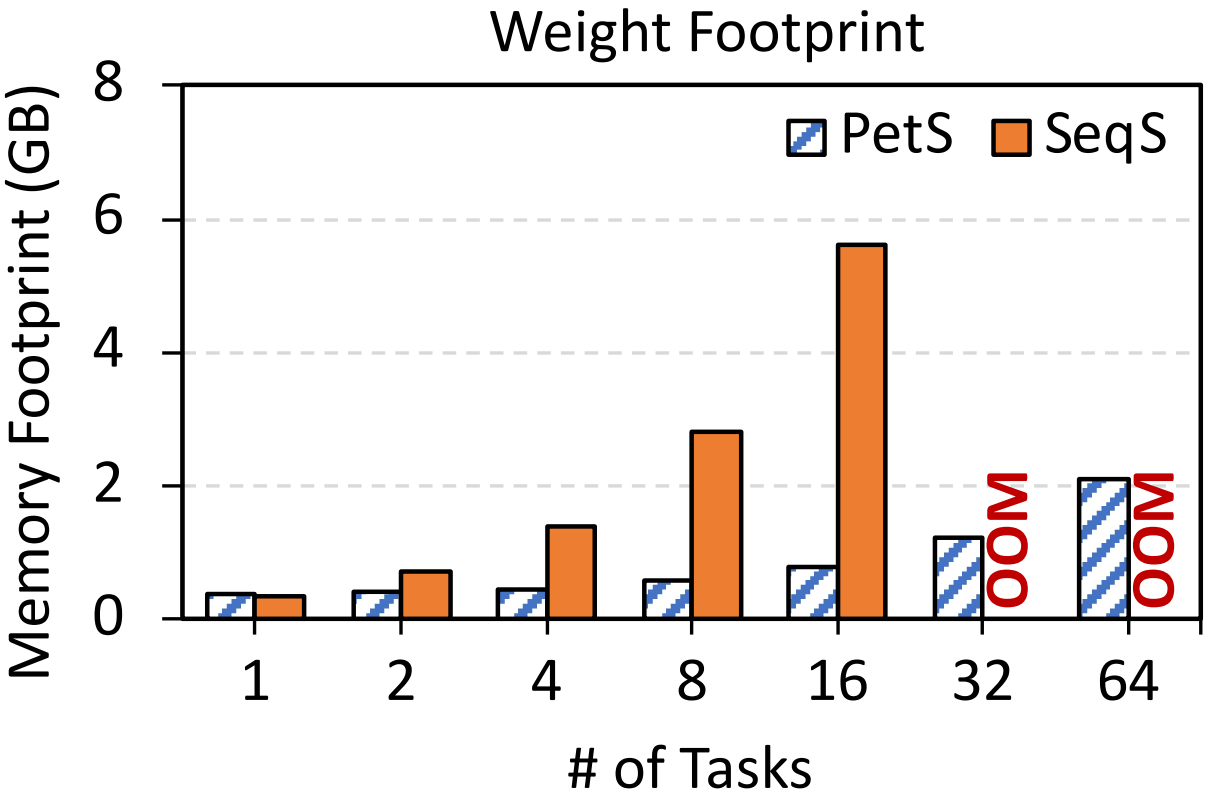
- Compared to the sequential serving baseline (SeqS[1]), PetS supports $4\times$ to $26\times$ more concurrent tasks.

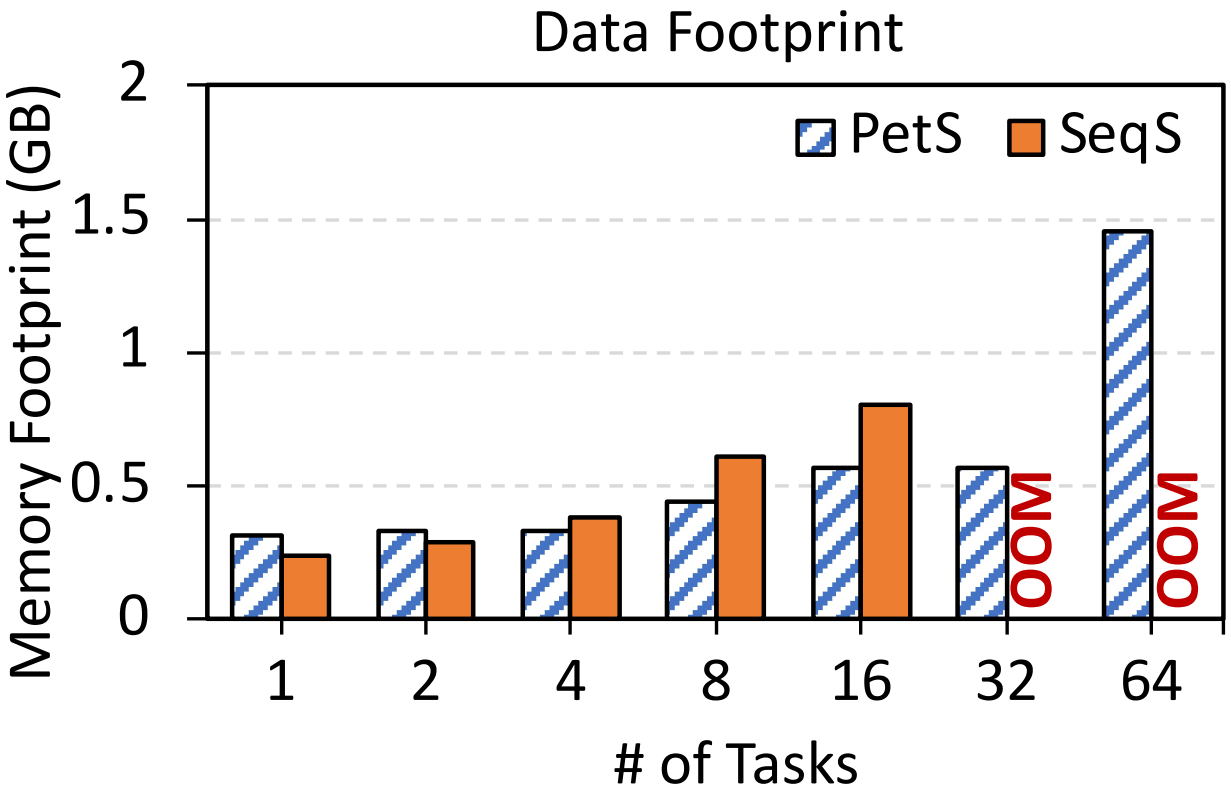
Platform	Device Memory	Shared Models	DistilBert	Bert-base	Bert-large
			SeqS/**PetS**	SeqS/**PetS**	SeqS/**PetS**
Jetson TX2	8GB[2]	**Supported Tasks**	34 / **504**	17 / **180**	3 / **12**
1080Ti	11GB		56 / **1336**	28 / **588**	7 / **126**
V100	32GB		170 / **4344**	85 / **2164**	25 / **560**

[1] The unmodified TurboTransformers [2] Shared by CPU and GPU

GPU Memory Footprint Comparison

- PetS consumes much lower memory for storing weights

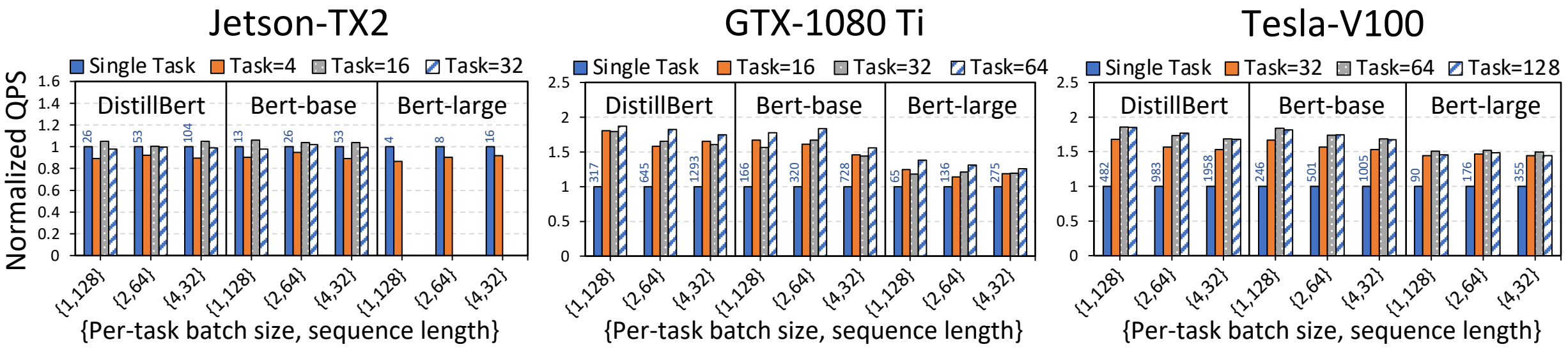




Platform: Desktop GPU

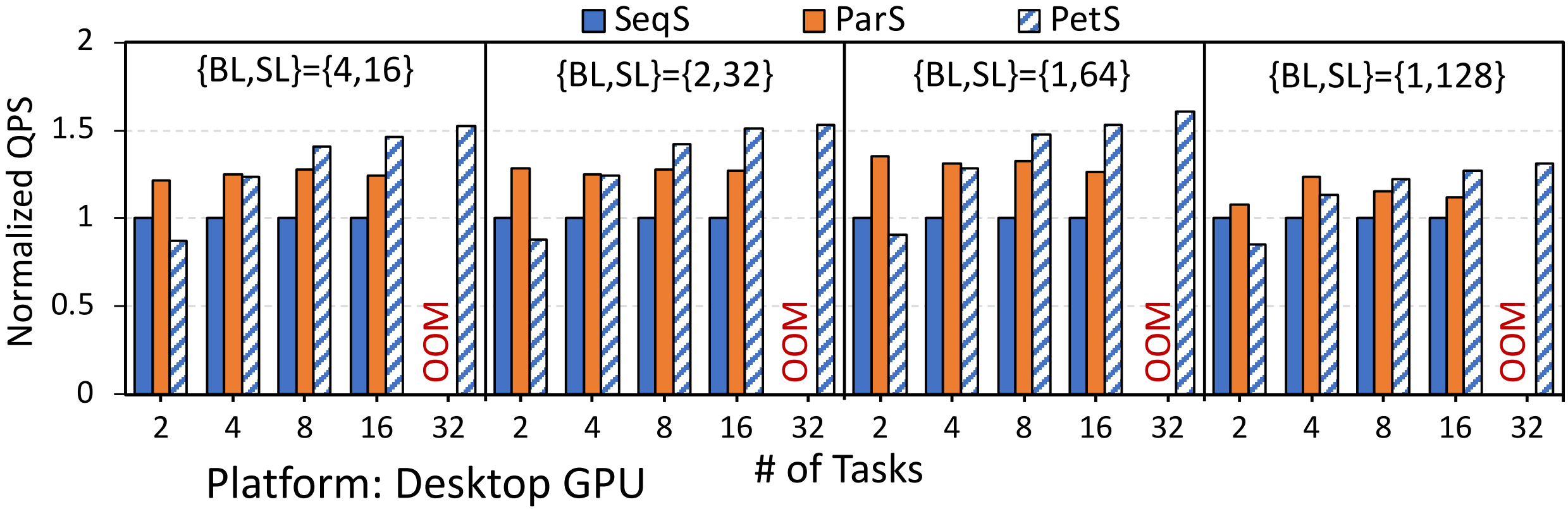
Serving Throughput with Fixed-Size Inputs

- 1.63× higher throughput (average) on V100
- 1.53× on GTX-1080Ti
- No throughput improvement on TX2 due to its limited hardware parallelism



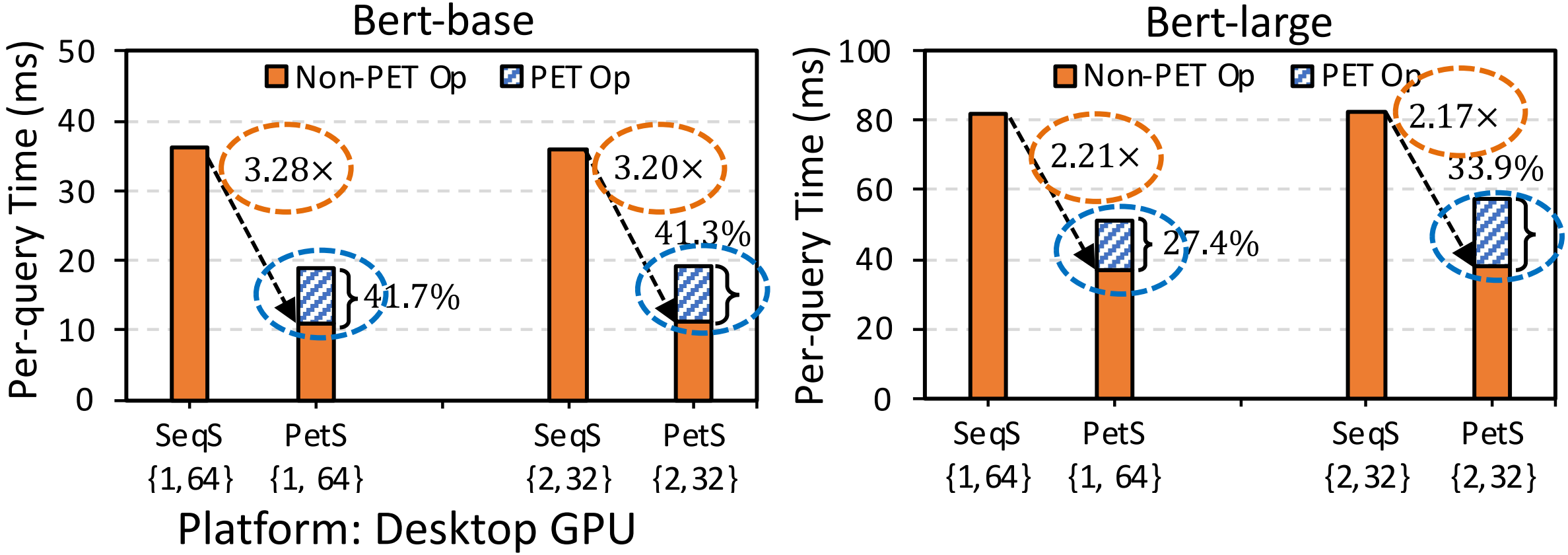
Comparison with SeqS and ParS

- ParS (Parallel Serving) has the better performance when the number of tasks is limited.
- PetS has the better scalability than ParS and SeqS



Execution Time Breakdown

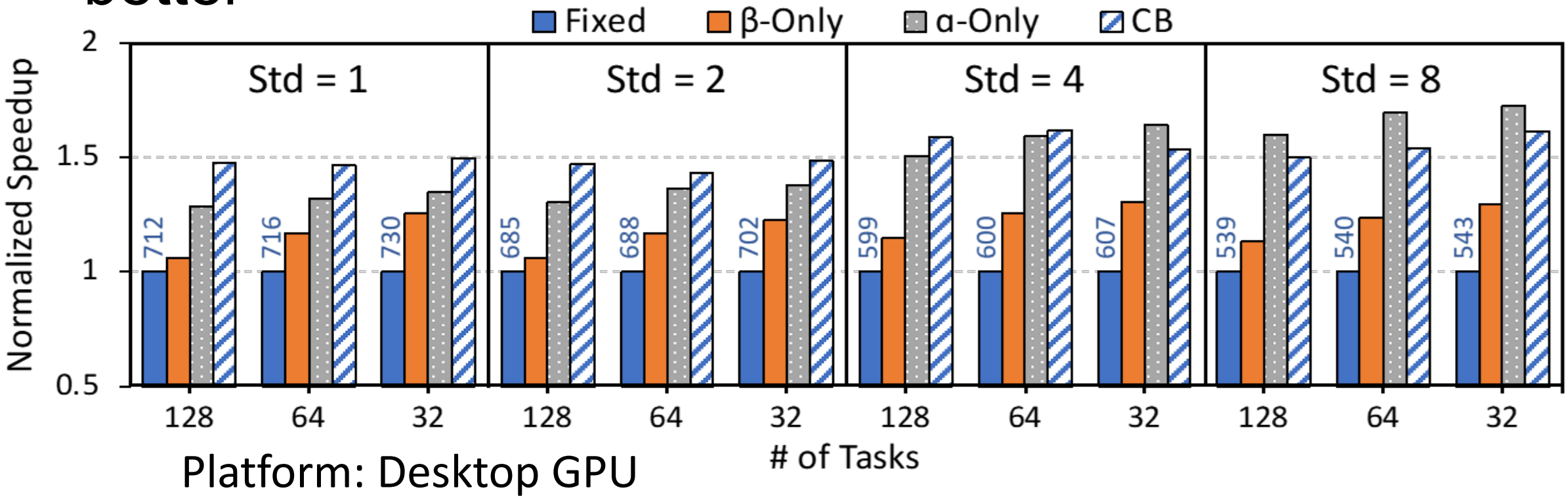
- Batched execution greatly reduces the non-PET Ops.
- The PET operators only take up a small portion of execution time.



Platform: Desktop GPU

Batch Scheduling Performance

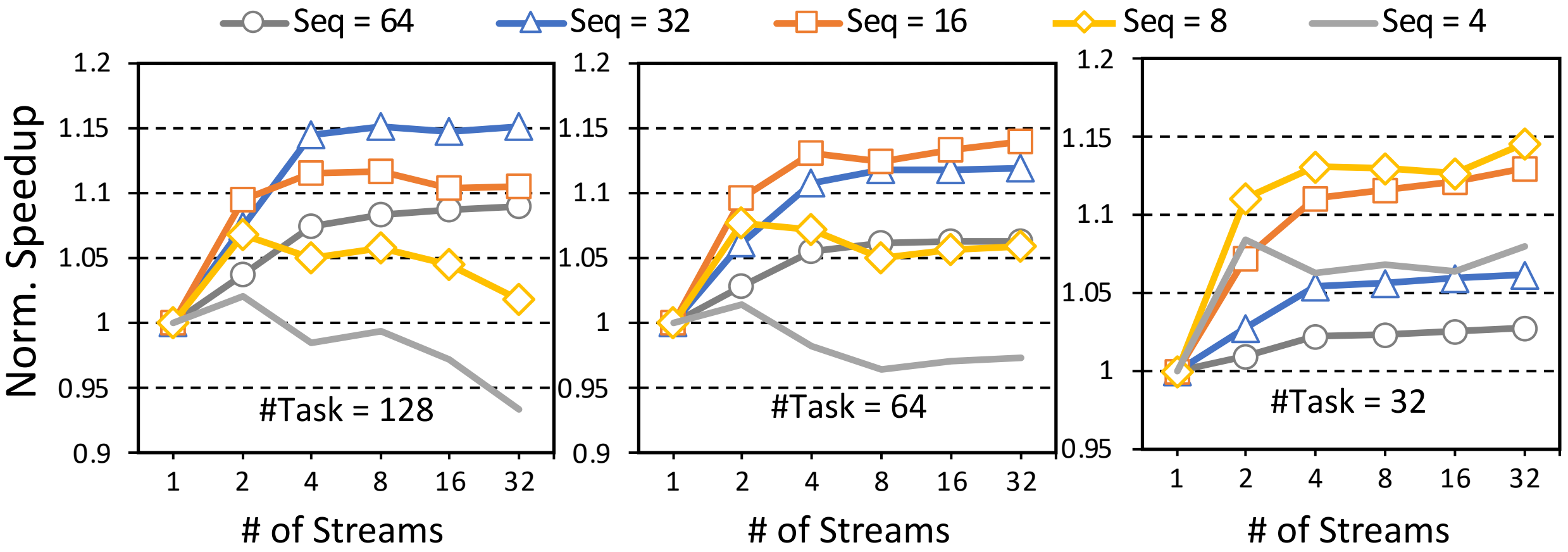
- Coordinated Batching is suitable for queries with small variance in query lengths.
- For queries with large variance, α-Only Batching is better



Platform: Desktop GPU

PET Operator Scheduling Performance

- More effective on long input sequences



Platform: Desktop GPU

Discussion

- Current Limitations:
  - Optimized purely for throughput
  - Did not consider latency-critical tasks
- How to support a new PET algorithm
  - Identify the PET operations of the algorithm
  - Decouple them from the shared operations
  - Implement new PET operators if necessary
  - Extend the model loading/managing APIs accordingly

Summary of Contributions

- A unified representation for various PET algorithms
- The PetS framework for efficient multi-task PETs serving
- Two optimization strategies:
  - Coordinated batch scheduling
  - PET operator scheduling
- Evaluated on Edge/Desktop/Server platforms:
  - Supports up to 27x more tasks, 1.53x and 1.63x higher throughput on *Desktop* and *Server* GPUs

Future Plan

- **PetS** will be integrated to Alibaba's **HIE** framework
  - **HIE** is a high-performance inference serving framework
  - **HIE** is scheduled to be released at this August
- **PetS** is planed to support more PET models trained by existing PET training frameworks such as **AdapterHub** and **OpenDelta**

AdapterHub

OpenDelta

Thanks For Your Listening

The Artifact of PetS:
https://doi.org/10.5281/zenodo.6534753

Contact: zhou.zhe@pku.edu.cn