PetS: A Unified Framework for Parameter-Efficient Transformers Serving

Zhe Zhou¹, Xuechao Wei¹,², Jiejing Zhang², Guangyu Sun¹
¹Peking University
²Alibaba Group
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

Dataset

Transformer Model

User-A

User-B

User-C

Bert

Bert-A

Bert-B

Bert-C

Edge Server

Queries

TPU/GPU Cluster

GPU Servers

Task-A

Task-B

Task-C

IoTs
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)

Pretrained Model

Full-Model Finetuning

Finetuned Model

Add new trainable modules

Finetune a subset of pre-trained params

Remove a subset of pre-trained params

Parameter-Efficient Finetuning
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

<table>
<thead>
<tr>
<th>Adapters</th>
<th>MaskBert</th>
<th>Diff-Pruning</th>
<th>BitFit</th>
</tr>
</thead>
</table>

### Multiple PET Tasks

1. \[ Y_t = \sigma((X_t \cdot W + b) \cdot W_{down}) \cdot W_{up} \]
2. \[ Y_t = X_t \cdot (M_t \odot W) + b \]
3. \[ Y_t = X_t \cdot (W + \delta_t) + b + b_t \]
4. \[ Y_t = X_t \cdot W + b + b_t \]

### Shared Operations

\[ Y_t = X_t \cdot W + b \]

### PET-specific Operations

\[ \sigma(W_{down})W_{up} \]

\[ -X_t \cdot (W \odot \bar{M}_t) \]

\[ +X_t \cdot \delta_t + b_t \]

\[ +b_t \]
Unified Representation of PETs

<table>
<thead>
<tr>
<th>①</th>
<th>②</th>
<th>③</th>
<th>④</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapters</td>
<td>MaskBert</td>
<td>Diff-Pruning</td>
<td>BitFit</td>
</tr>
</tbody>
</table>

Shared operations can be batched among tasks

PET Operations can be computed with light-weighted operators
PetS Overview

1. Register Tasks
   - Pre-trained Model Tag
   - PET Parameters
   - PET Type

2. Task Manager
   - Task Register
   - PET Manager

3. Parameter Repository
   - PET Parameters
   - Shared Model Parameters

4. Input Queries
   - Query 0: <Task_id> <Input Data>
   - Query 1: <Task_id> <Input Data>

5. PET Serving
   - PET Inference Pipeline
     - User Inputs: Input Analyzing, Input Reformatting
     - Batch Scheduler: Performance Model, Scheduling Policy
     - Engine: PET Task Scheduler, PET Operator Library

- Query 0: <Task_id> <Input Data>
- Query 1: <Task_id> <Input Data>
PetS Overview

1. Users register tasks

Register Tasks
- Pre-trained Model Tag
- PET Parameters
- PET Type

Input Queries
- Query 0: <Task_id> <Input Data>
- Query 1: <Task_id> <Input Data>

PET Serving
- PET Inference Pipeline
  - User Inputs
    - Input Analyzing
    - Input Reformatting
  - Batch Scheduler
    - Performance Model
    - Scheduling Policy
  - Engine
    - PET Task Scheduler
    - PET Operator Library

Task Manager
- Task Register
- PET Manager

Parameter Repository
- PET Parameters
- Shared Model Parameters
- Parameter Repository
PetS Overview

1. Register Tasks
   - Pre-trained Model Tag
   - PET Parameters
   - PET Type

2. Task Manager
   - Task Register
   - PET Manager

3. Parameter Repository
   - PET Parameters
   - Shared Model Parameters

4. Input Queries
   - Query 0: <Task_id> <Input Data>
   - Query 1: <Task_id> <Input Data>

5. PET Serving
   - PET Inference Pipeline
     - User Inputs
     - Batch Scheduler
     - Engine
       - PET Task Scheduler
       - PET Operator Library
     - Performance Model
     - Scheduling Policy

Task Manager manages the registered tasks
PetS Overview

1. Register Tasks
   - Pre-trained Model Tag
   - PET Parameters
   - PET Type

2. Task Manager
   - Task Register
   - PET Manager

3. PET Parameters
   - Shared Model Parameters
   - Parameter Repository

4. Input Queries
   - Query 0: <Task_id> <Input Data>
   - Query 1: <Task_id> <Input Data>

5. Scheduling + Execution with PIE

- PET Serving
  - PET Inference Pipeline
    - User Inputs
      - Input Analyzing
      - Input Reformatting
    - Batch Scheduler
      - Performance Model
      - Scheduling Policy
    - Engine
      - PET Task Scheduler
      - PET Operator Library

- Performance
  - Model
  - Batch Scheduler
  - Scheduling Policy

- Task Register
- PET Manager
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)
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)
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")
s_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

Goal: find an efficient way to batch the task queries

\[
Batch_{-}Latency(B_i) = \alpha [N_i][L_i] + \sum_{j=0}^{t_i-1} \beta [p_{tij}][n_{ij}][l_{ij}]
\]

Shared Op Latency

PET-Op Latency
Coordinated Batch Scheduling

1. Sort queries of each task
2. Split to mini batches using $\beta$ model and a DP algorithm

PET-OPs Profiling

$\beta$ - Model

Step 1: Intra-Task Batching

Task 0
Task 1
Task 2
Task 3
Task 4

B=2, S=34
Coordinated Batch Scheduling

Step 2: Inter-Task Batching

1. Sort mini batches
   - Batch 0: B=4, S=34
   - Batch 1: B=4, S=34
   - Batch 2: B=2, S=34

2. Split to macro batches using the $\alpha$ model and DP
PET Operator Scheduling

Sequential Execution of PET Ops

\[ Op\_intensity = \frac{op\cdot FLOPs}{\beta(op) \times \omega(op)} \]

Sort & Evenly assign to multiple CUDA streams

Timeline (ideal case)
## Evaluation Setup

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

<table>
<thead>
<tr>
<th>Model Type</th>
<th># Layer</th>
<th>Hidden Size</th>
<th>Inter-Size</th>
<th># Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>DistilBert</td>
<td>6</td>
<td>768</td>
<td>3072</td>
<td>66M</td>
</tr>
<tr>
<td>Bert-base</td>
<td>12</td>
<td>768</td>
<td>3072</td>
<td>110M</td>
</tr>
<tr>
<td>Bert-large</td>
<td>24</td>
<td>1024</td>
<td>4096</td>
<td>340M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adapter</td>
<td>Bottleneck = 64</td>
</tr>
<tr>
<td>MaskBert</td>
<td>95% Sparsity</td>
</tr>
<tr>
<td>Diff-pruning</td>
<td>99.5% Sparsity</td>
</tr>
<tr>
<td>Bitfit</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Codebase: TurboTransformers
### Maximum Number of Supported Tasks

- Compared to the sequential serving baseline (SeqS\(^1\)), **PetS** supports **4× to 26× more concurrent tasks**.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Device Memory</th>
<th>Shared Models</th>
<th>DistilBert</th>
<th>Bert-base</th>
<th>Bert-large</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SeqS</strong></td>
<td><strong>PetS</strong></td>
<td></td>
<td><strong>SeqS</strong>/<strong>PetS</strong></td>
<td><strong>SeqS</strong>/<strong>PetS</strong></td>
<td><strong>SeqS</strong>/<strong>PetS</strong></td>
</tr>
<tr>
<td>Jetson TX2</td>
<td>8GB(^2)</td>
<td></td>
<td>34 / 504</td>
<td>17 / 180</td>
<td>3 / 12</td>
</tr>
<tr>
<td>1080Ti</td>
<td>11GB</td>
<td><strong>Supported Tasks</strong></td>
<td>56 / 1336</td>
<td>28 / 588</td>
<td>7 / 126</td>
</tr>
<tr>
<td>V100</td>
<td>32GB</td>
<td></td>
<td>170 / 4344</td>
<td>85 / 2164</td>
<td>25 / 560</td>
</tr>
</tbody>
</table>

\(^1\) 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

#### Single Task

**Jetson-TX2**

<table>
<thead>
<tr>
<th>{Per-task batch size, sequence length}</th>
<th>DistillBert</th>
<th>Bert-base</th>
<th>Bert-large</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,128)</td>
<td>1.63</td>
<td>1.53</td>
<td>1.63</td>
</tr>
<tr>
<td>(2,64)</td>
<td>1.63</td>
<td>1.53</td>
<td>1.63</td>
</tr>
<tr>
<td>(4,32)</td>
<td>1.63</td>
<td>1.53</td>
<td>1.63</td>
</tr>
<tr>
<td>(11,128)</td>
<td>1.63</td>
<td>1.53</td>
<td>1.63</td>
</tr>
<tr>
<td>(12,64)</td>
<td>1.63</td>
<td>1.53</td>
<td>1.63</td>
</tr>
<tr>
<td>(11,128)</td>
<td>1.63</td>
<td>1.53</td>
<td>1.63</td>
</tr>
<tr>
<td>(2,64)</td>
<td>1.63</td>
<td>1.53</td>
<td>1.63</td>
</tr>
<tr>
<td>(4,32)</td>
<td>1.63</td>
<td>1.53</td>
<td>1.63</td>
</tr>
</tbody>
</table>

**GTX-1080 Ti**

<table>
<thead>
<tr>
<th>{Per-task batch size, sequence length}</th>
<th>DistillBert</th>
<th>Bert-base</th>
<th>Bert-large</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,128)</td>
<td>2.63</td>
<td>2.53</td>
<td>2.63</td>
</tr>
<tr>
<td>(2,64)</td>
<td>2.63</td>
<td>2.53</td>
<td>2.63</td>
</tr>
<tr>
<td>(4,32)</td>
<td>2.63</td>
<td>2.53</td>
<td>2.63</td>
</tr>
<tr>
<td>(11,128)</td>
<td>2.63</td>
<td>2.53</td>
<td>2.63</td>
</tr>
<tr>
<td>(12,64)</td>
<td>2.63</td>
<td>2.53</td>
<td>2.63</td>
</tr>
<tr>
<td>(11,128)</td>
<td>2.63</td>
<td>2.53</td>
<td>2.63</td>
</tr>
<tr>
<td>(2,64)</td>
<td>2.63</td>
<td>2.53</td>
<td>2.63</td>
</tr>
<tr>
<td>(4,32)</td>
<td>2.63</td>
<td>2.53</td>
<td>2.63</td>
</tr>
</tbody>
</table>

**Tesla-V100**

<table>
<thead>
<tr>
<th>{Per-task batch size, sequence length}</th>
<th>DistillBert</th>
<th>Bert-base</th>
<th>Bert-large</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,128)</td>
<td>3.63</td>
<td>3.53</td>
<td>3.63</td>
</tr>
<tr>
<td>(2,64)</td>
<td>3.63</td>
<td>3.53</td>
<td>3.63</td>
</tr>
<tr>
<td>(4,32)</td>
<td>3.63</td>
<td>3.53</td>
<td>3.63</td>
</tr>
<tr>
<td>(11,128)</td>
<td>3.63</td>
<td>3.53</td>
<td>3.63</td>
</tr>
<tr>
<td>(12,64)</td>
<td>3.63</td>
<td>3.53</td>
<td>3.63</td>
</tr>
<tr>
<td>(11,128)</td>
<td>3.63</td>
<td>3.53</td>
<td>3.63</td>
</tr>
<tr>
<td>(2,64)</td>
<td>3.63</td>
<td>3.53</td>
<td>3.63</td>
</tr>
<tr>
<td>(4,32)</td>
<td>3.63</td>
<td>3.53</td>
<td>3.63</td>
</tr>
</tbody>
</table>
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.

Platform: Desktop GPU

<table>
<thead>
<tr>
<th># of Tasks</th>
<th>SeqS</th>
<th>ParS</th>
<th>PetS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Normalized QPS

\{BL,SL\} = \{4,16\}  \{BL,SL\} = \{2,32\}  \{BL,SL\} = \{1,64\}  \{BL,SL\} = \{1,128\}
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, $\alpha$-Only Batching is better
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 planned to support more PET models trained by existing PET training frameworks such as **AdapterHub** and **OpenDelta**
Thanks For Your Listening

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

Contact: zhou.zhe@pku.edu.cn