MuxTune: Efficient Multi-Task LLM Fine-Tuning in Multi-Tenant Datacenters via Spatial-Temporal Backbone Multiplexing

Chunyu Xue[¶*], Yi Pan[¶*], Weihao Cui[¶‡], Quan Chen[¶], Shulai Zhang[¶], Bingsheng He[‡], Minyi Guo[¶]

[¶]*Shanghai Jiao Tong University,* [‡]*National University of Singapore*

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

NUS
National University of Singapore

[*]*Equal contribution*

Customizing LLMs using Public Fine-Tuning APIs

User-level task creation

OpenAI Gemini together.ai

```
# together-style fine-tuning API
from together import Together
client = Together(api_key=API_KEY)
# upload dataset
train_file = client.files.upload(
    “coqa_prepared_train.jsonl”,
    purpose=“fine-tune”)
# create task
ft_task = client.fine_tuning.create(
    training_file=train_file.id,
    model=“meta-llama/Meta-Llama-8B”,
    n_epochs=3,
    batch_size=64,
    lora=True)
```

[1] https://docs.together.ai/docs/fine-tuning-quickstart

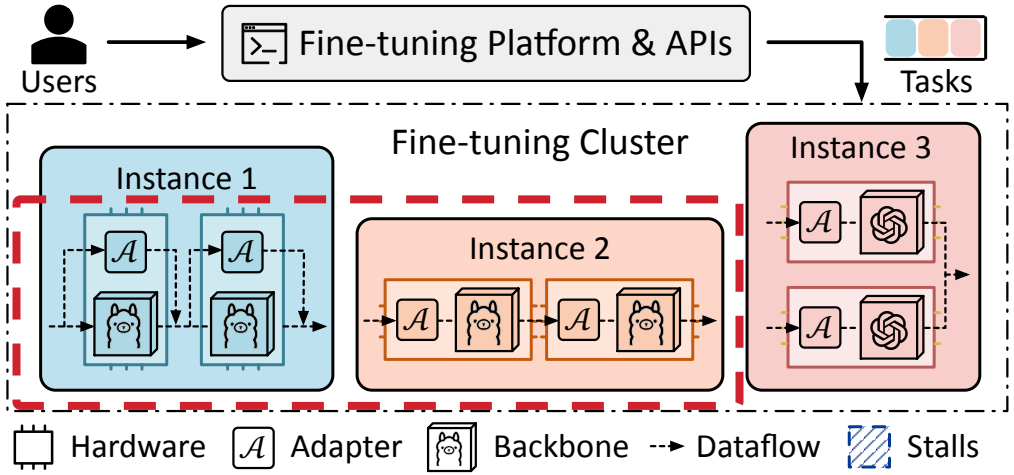
Customizing LLMs using Public Fine-Tuning APIs

User-level task creation

OpenAI Gemini together.ai

```
# together-style fine-tuning API
from together import Together
client = Together(api_key=API_KEY)
# upload dataset
train_file = client.files.upload(
    “coqa_prepared_train.jsonl”,
    purpose=“fine-tune”)
# create task
ft_task = client.fine_tuning.create(
    training_file=train_file.id,
    model=“meta-llama/Meta-Llama-8B”,
    n_epochs=3,
    batch_size=64,
    lora=True)
```

Cluster-level task scheduling & execution



One parameter-efficient fine-tuning (PEFT) task / instance

[1] https://docs.together.ai/docs/fine-tuning-quickstart

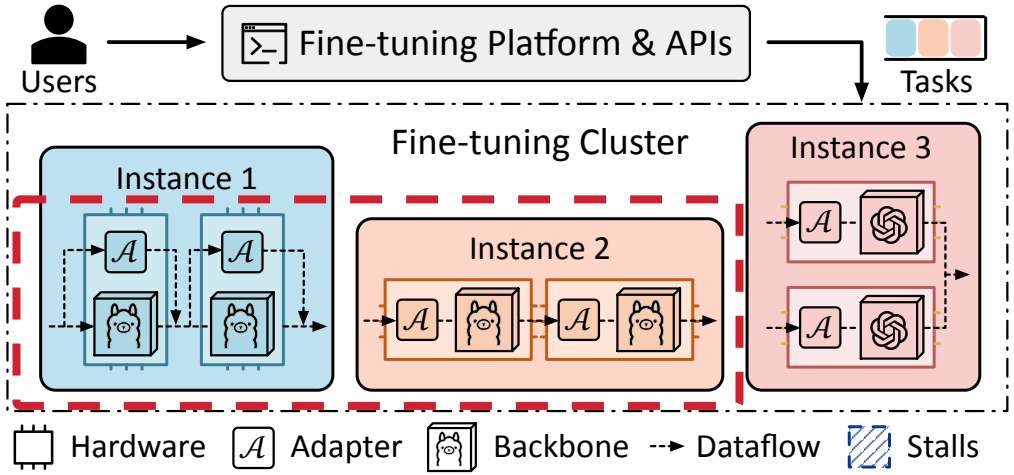
Customizing LLMs using Public Fine-Tuning APIs

User-level task creation

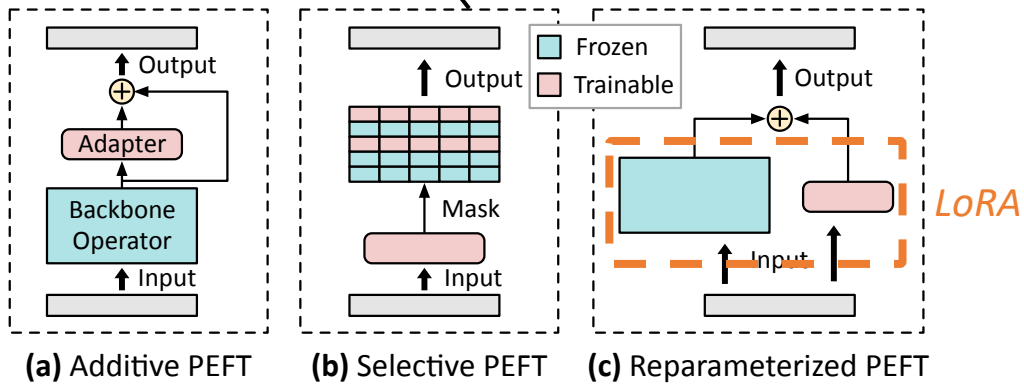
OpenAI Gemini together.ai

```
# together-style fine-tuning API
from together import Together
client = Together(api_key=API_KEY)
# upload dataset
train_file = client.files.upload(
    "coqa_prepared_train.jsonl",
    purpose="fine-tune")
# create task
ft_task = client.fine_tuning.create(
    training_file=train_file.id,
    model="meta-llama/Meta-Llama-8B",
    n_epochs=3,
    batch_size=64,
    lora=True)
```

Cluster-level task scheduling & execution

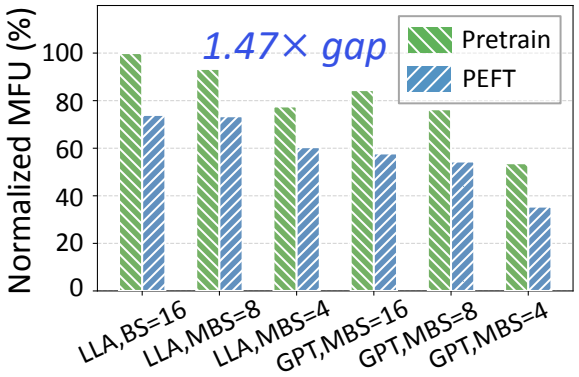


One parameter-efficient fine-tuning (PEFT) task / instance



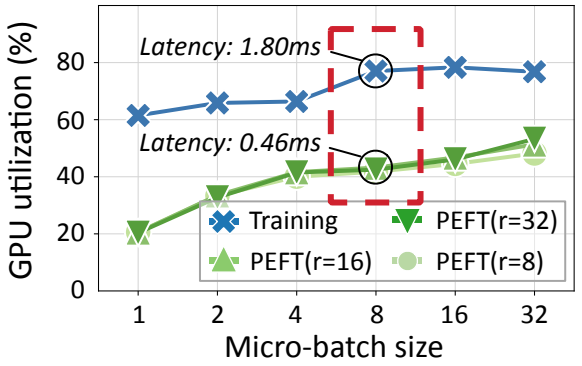
[1] https://docs.together.ai/docs/fine-tuning-quickstart

Inefficiencies of PEFT Workloads



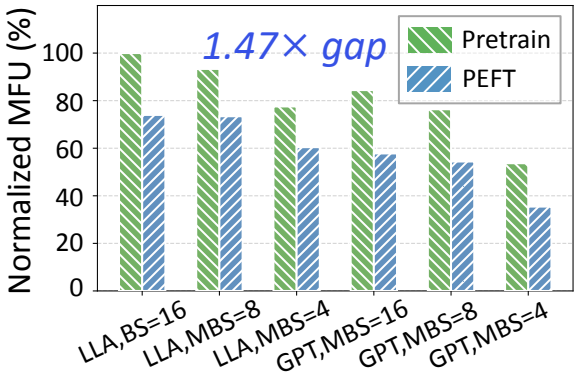
(a) End-to-end (1-GPU)

Lower util. yet non-negligible latency…



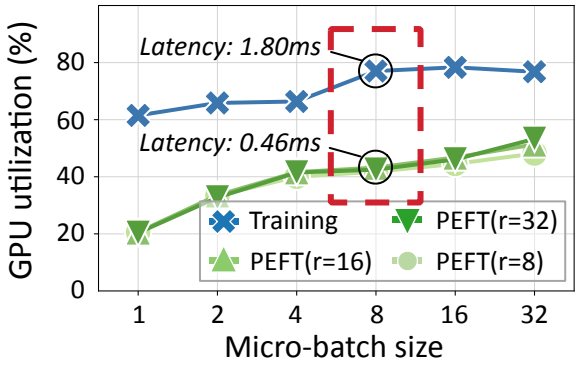
(b) Single GEMM operator

Inefficiencies of PEFT Workloads

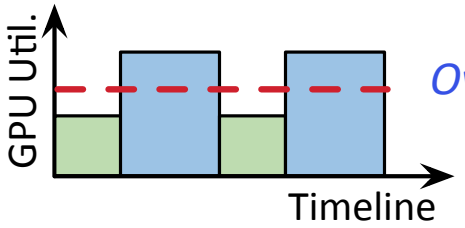


(a) End-to-end (1-GPU)

Lower util. yet non-negligible latency…

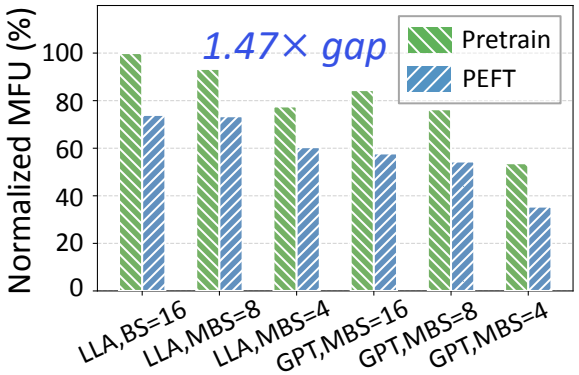


(b) Single GEMM operator



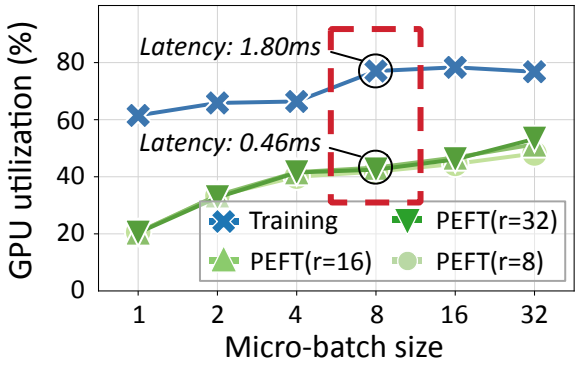
Overall utilization is degraded…

Inefficiencies of PEFT Workloads

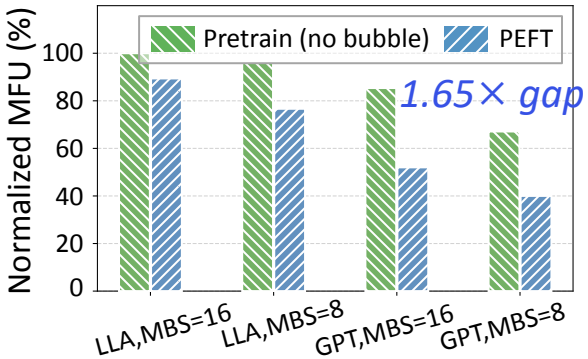


(a) End-to-end (1-GPU)

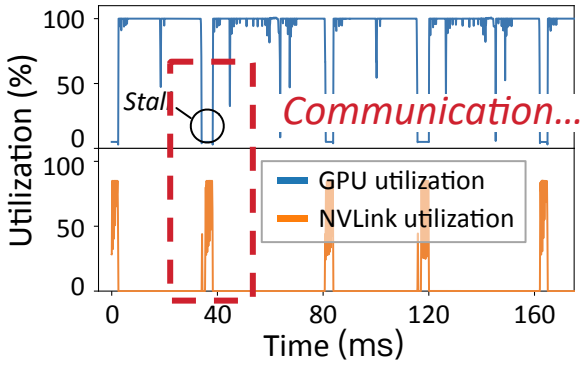
Lower util. yet non-negligible latency…



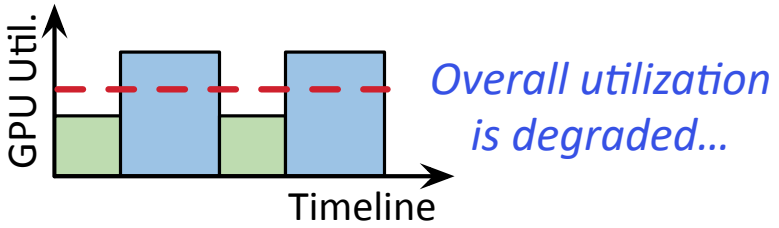
(b) Single GEMM operator



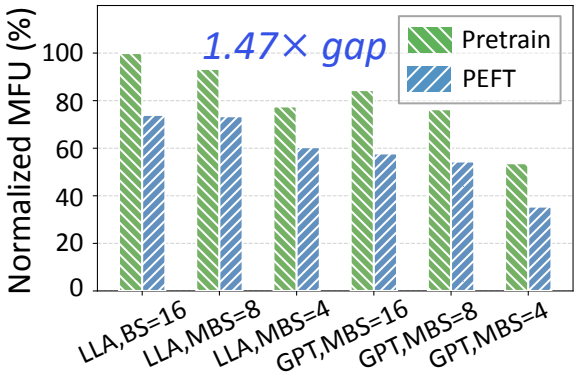
(c) End-to-end (4-GPU pipeline)



(d) Utilization breakdown (4-GPU TP)

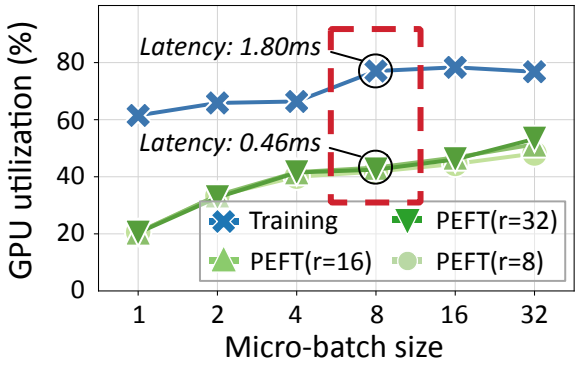


Inefficiencies of PEFT Workloads

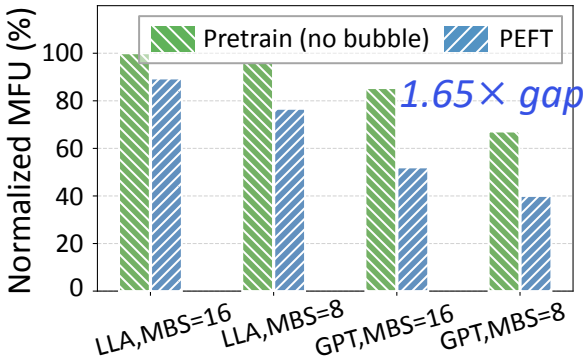


(a) End-to-end (1-GPU)

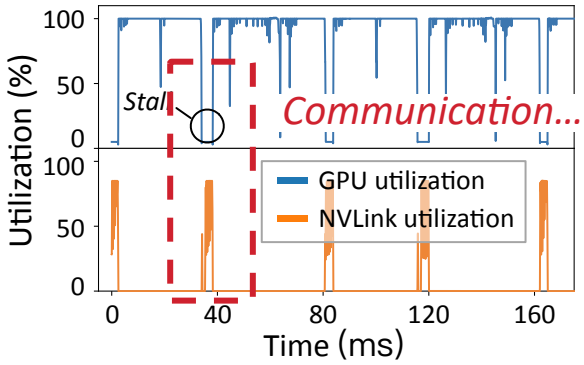
Lower util. yet non-negligible latency…



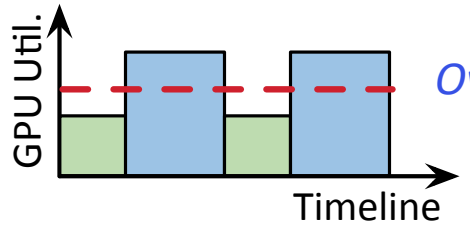
(b) Single GEMM operator



(c) End-to-end (4-GPU pipeline)



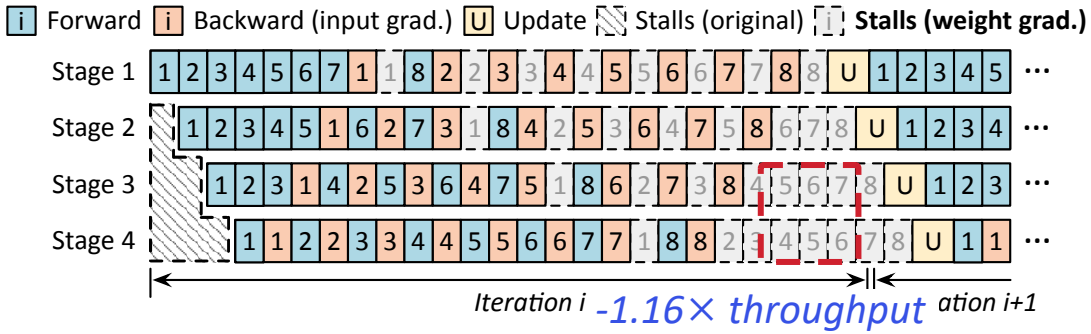
(d) Utilization breakdown (4-GPU TP)



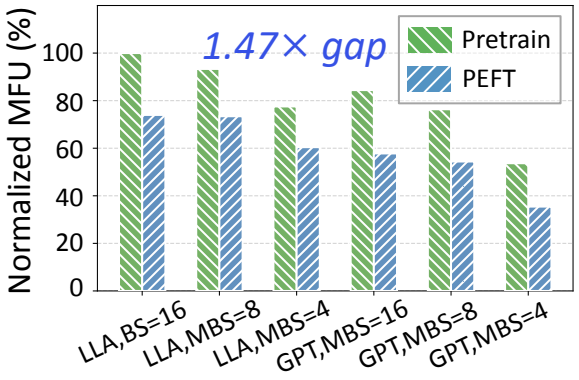
Overall utilization is degraded…

❶ Pipeline stalls (bubbles) ← *Zero-bubble pipeline*

Backbone has no weight gradient in PEFT ☹

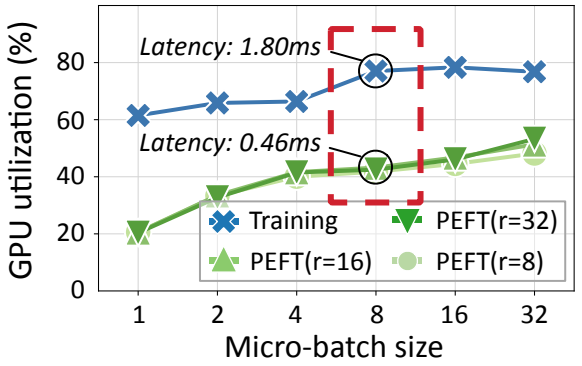


Inefficiencies of PEFT Workloads

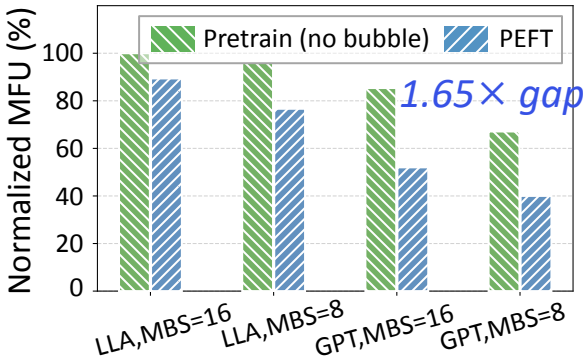


(a) End-to-end (1-GPU)

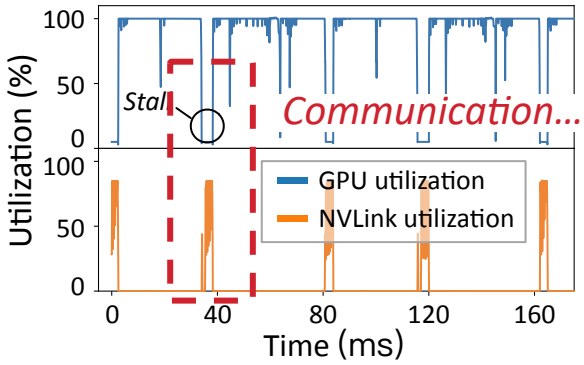
Lower util. yet non-negligible latency…



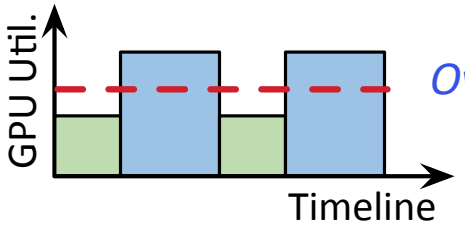
(b) Single GEMM operator



(c) End-to-end (4-GPU pipeline)



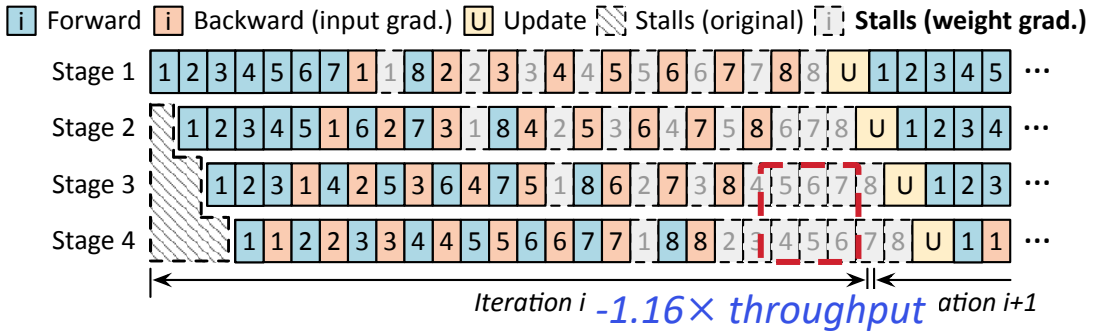
(d) Utilization breakdown (4-GPU TP)



Overall utilization is degraded…

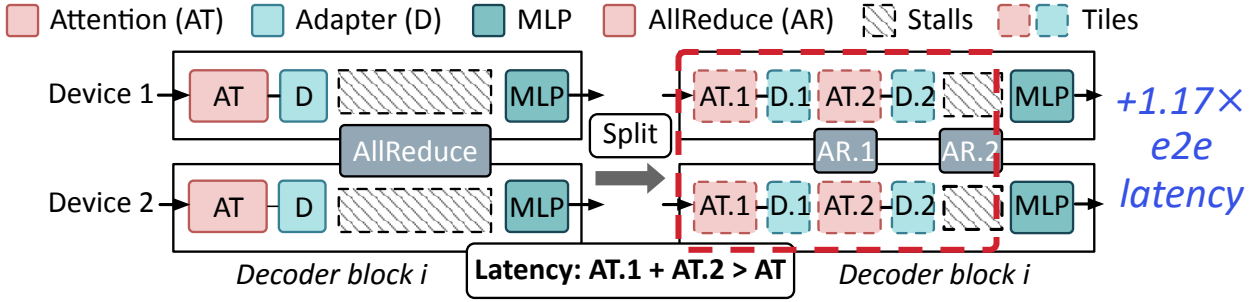
❶ Pipeline stalls (bubbles) ✕ *Zero-bubble pipeline*

Backbone has no weight gradient in PEFT ☹



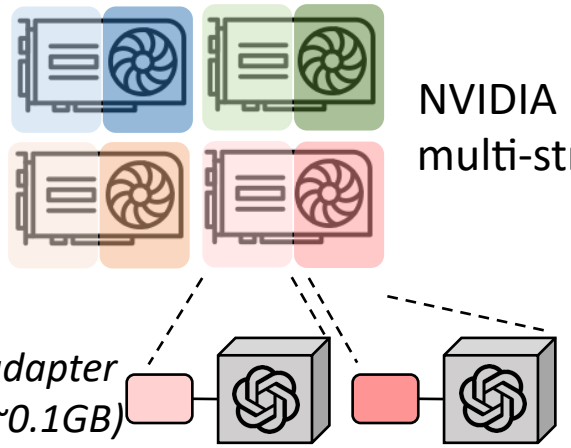
❷ Intra-stage stalls (e.g., TP) ✕ *Communication overlap*

PEFT has shorter samples / smaller batch size ☹



Why Intuitive Approaches are Inefficient?

Coarse-grained spatial multiplexing



Memory bottleneck ☹

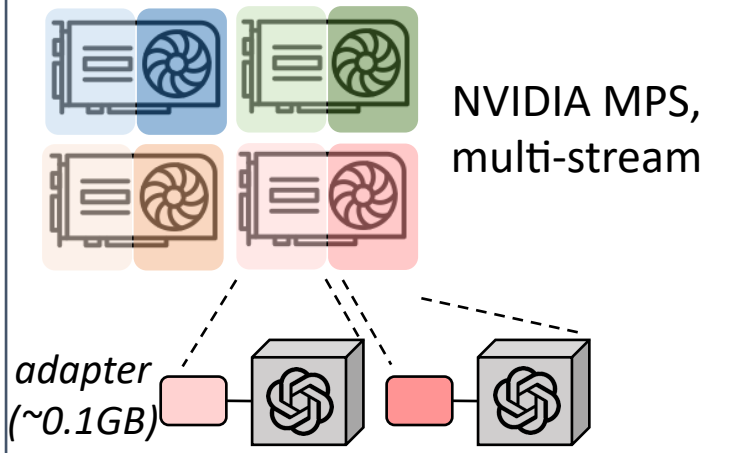
Cannot scale to larger models…

Inter-task interference ☹

Lack of fine-grained operator control…

Why Intuitive Approaches are Inefficient?

Coarse-grained spatial multiplexing



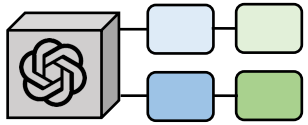
Memory bottleneck ☹

Cannot scale to larger models…

Inter-task interference ☹

Lack of fine-grained operator control…

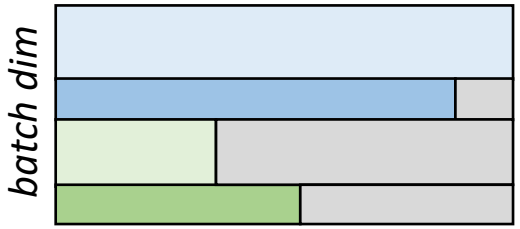
Batching-based spatial multiplexing



Multi-LoRA serving systems (e.g. S-LoRA)

Concatenated batch

sequence dim



Fine-tuning differs from inference!

Diminish returns, enlarged stalls ☹

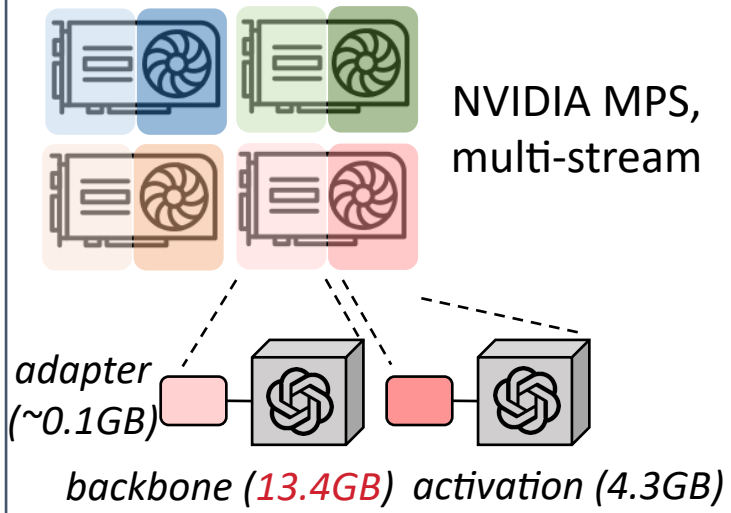
Excessive batching inflates compute latency, inducing larger stalls…

Padding wastes computation ☹

Different datasets by batched tasks…

Why Intuitive Approaches are Inefficient?

Coarse-grained spatial multiplexing



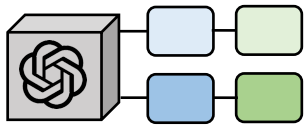
Memory bottleneck ☹

Cannot scale to larger models…

Inter-task interference ☹

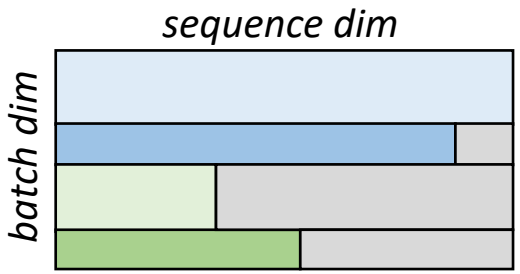
Lack of fine-grained operator control…

Batching-based spatial multiplexing



Multi-LoRA serving systems (e.g. S-LoRA)

Concatenated batch



Fine-tuning differs from inference!

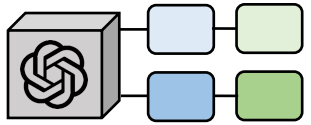
Diminish returns, enlarged stalls ☹

Excessive batching inflates compute latency, inducing larger stalls…

Padding wastes computation ☹

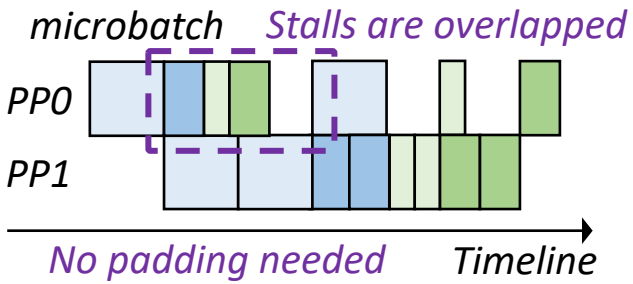
Different datasets by batched tasks…

Temporal multiplexing and overlapping



Still share a single backbone

Taking pipeline as an example…

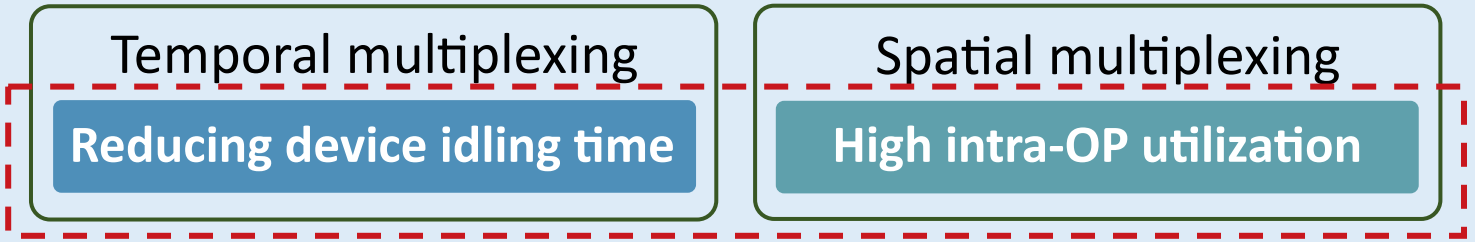


Intra-OP utilization not improved ☹

PEFT has limited input size…

Our Solution: *Spatial-Temporal Backbone Multiplexing*

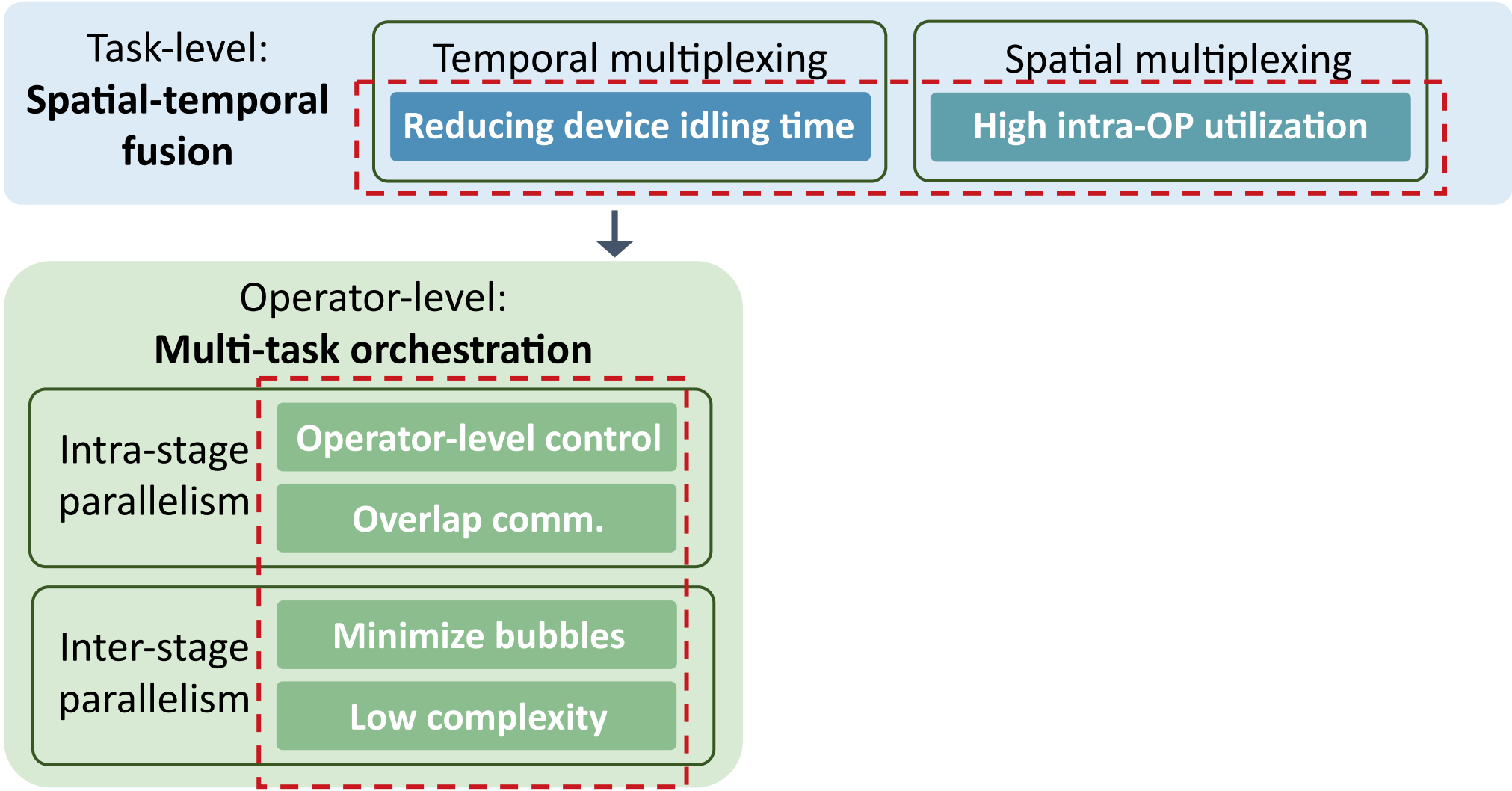
Task-level:
Spatial-temporal fusion



C1. Navigating the tradeoff

Cost modeling…
Efficient algorithm…

Our Solution: *Spatial-Temporal Backbone Multiplexing*



C1. Navigating the tradeoff

Cost modeling…

Efficient algorithm…

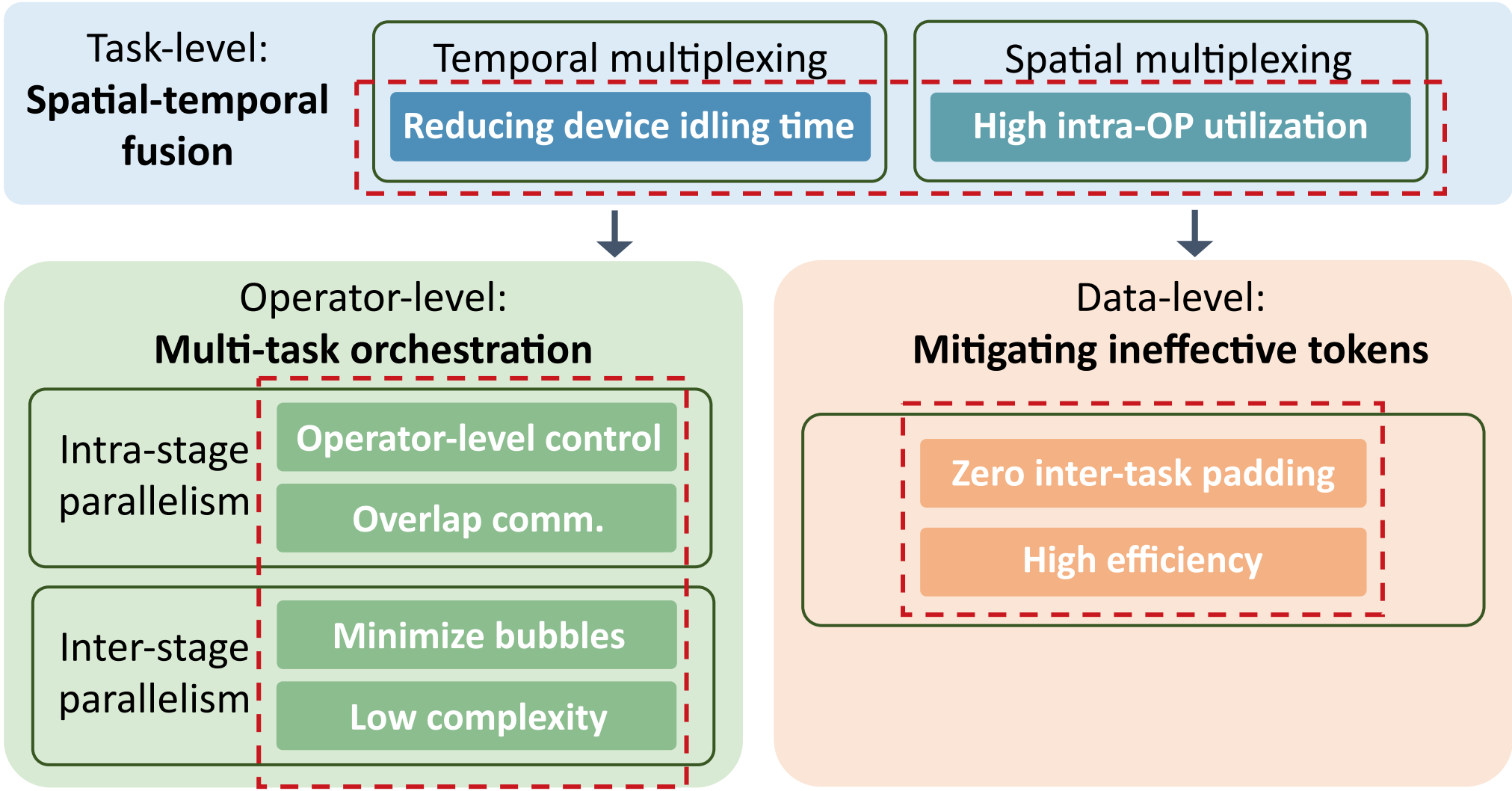
C2. Stall-free execution

Two-tiered schedule…

Low overhead…

Fine-grained coordination…

Our Solution: *Spatial-Temporal Backbone Multiplexing*



C1. Navigating the tradeoff

Cost modeling…
Efficient algorithm…

C2. Stall-free execution

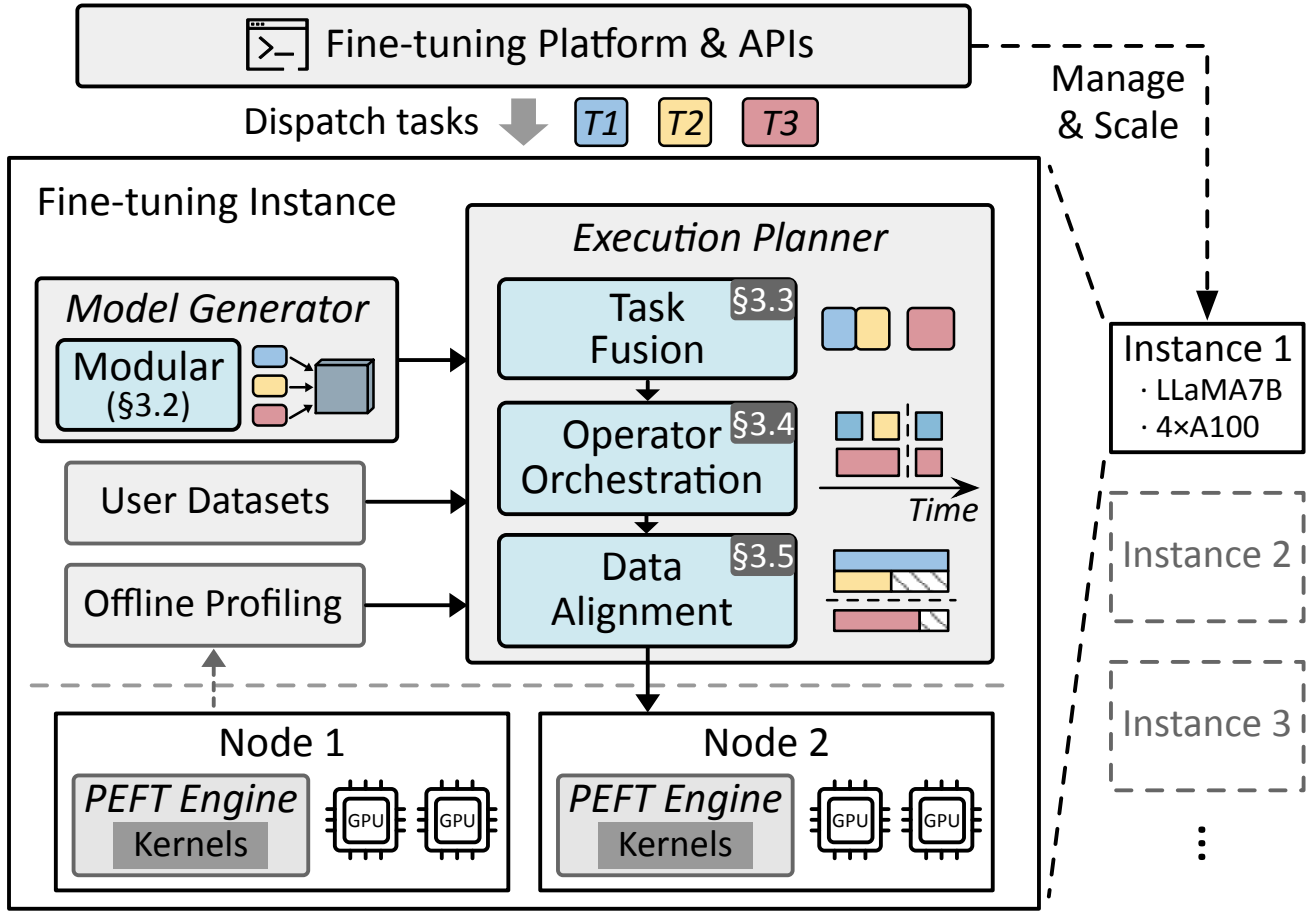
Two-tiered schedule…
Low overhead…
Fine-grained coordination…

C3. Data alignment

Naïve batching or seq packing ✗

System Design Overview

MuxTune: An efficient multi-task PEFT system as the backend of fine-tuning APIs, by **multiplexing the shared backbone** in a **spatial-temporal** manner to enhance resource efficiency.



Phase 0: Model Generation

- Handling on-the-fly task arrivals
- **Modularizing** PEFT workflows

Phase 1: Task Fusion

- **Hybrid task** as spatial-temporal abstraction
- Cost model + DP to **optimize fusing tradeoff**

Phase 2: Operator Orchestration

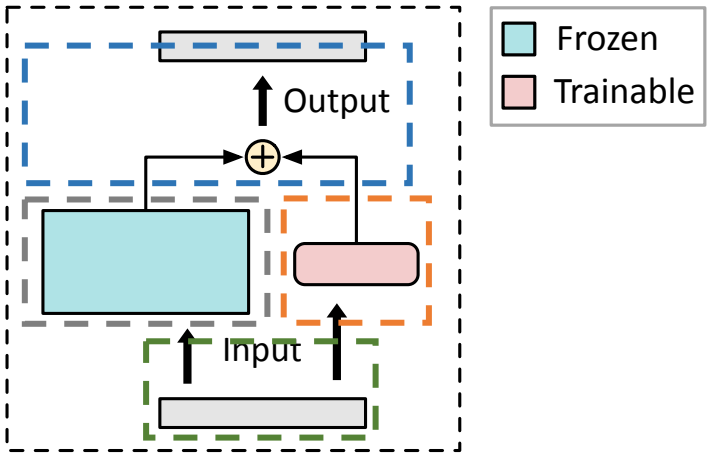
- Disaggregating into two tiers
- **Structured pipeline**, **subgraph scheduling**

Phase 3: Data Alignment

- Slicing per-task sequences into chunks
- **Chunk-based alignment**

Sharing Backbone across Tasks

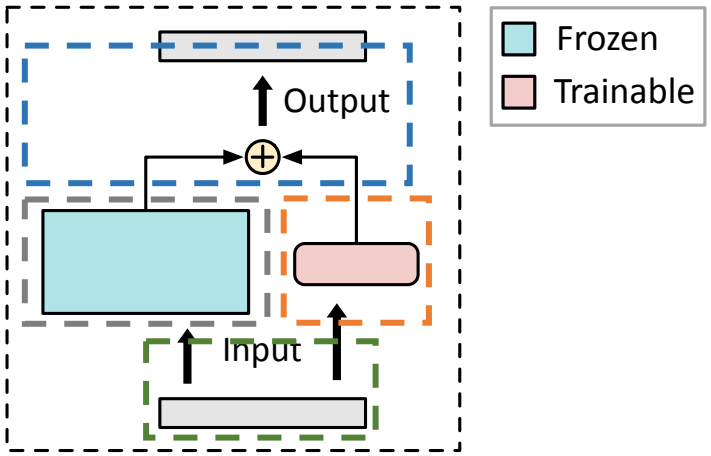
PEFT modularization



- ***BaseOp:*** an operator of backbone
- ***Adapter:*** PEFT algorithm
- ***Dispatch:*** prepares multi-task inputs
- ***Aggregate:*** merges output tensors

Sharing Backbone across Tasks

PEFT modularization



- ***BaseOp:*** an operator of backbone
- ***Adapter:*** PEFT algorithm
- ***Dispatch:*** prepares multi-task inputs
- ***Aggregate:*** merges output tensors

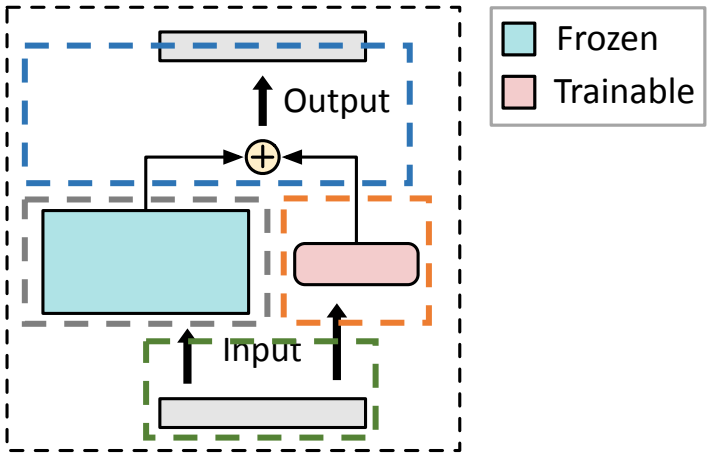
User interface comparison

```
class PEFTLinear(nn.Linear):# Nested wrap ✗
  def __init__(self, peft_cls, *cfg):
    nn.Linear.__init__(*cfg)
    self.adapter = peft_cls(*cfg)
  def forward(self, x):
    # BaseOp
    base_out = self.weight(x)
    # Adapter
    adapter_out = self.adapter(base_out)

class LLMBackbone:                    Statically attached ✗
  def __init__(self, peft_cls, *cfg):
    self.qkv = PEFTLinear(peft_cls, *cfg)
    ...
  def forward(self, x):
    qkv_out = self.qkv(x) # with adapter
    ··· # Other non-BaseOp layers
```

(a) Static nested adapter implementation

Sharing Backbone across Tasks

PEFT modularization



- ***BaseOp:*** an operator of backbone
- ***Adapter:*** PEFT algorithm
- ***Dispatch:*** prepares multi-task inputs
- ***Aggregate:*** merges output tensors

User interface comparison

```
class PEFTLinear(nn.Linear):# Nested wrap ✗
  def __init__(self, peft_cls, *cfg):
    nn.Linear.__init__(*cfg)
    self.adapter = peft_cls(*cfg)
  def forward(self, x):
    # BaseOp
    base_out = self.weight(x)
    # Adapter
    adapter_out = self.adapter(base_out)

class LLMBackbone:                    Statically attached ✗
  def __init__(self, peft_cls, *cfg):
    self.qkv = PEFTLinear(peft_cls, *cfg)
    ...
  def forward(self, x):
    qkv_out = self.qkv(x) # with adapter
    ... # Other non-BaseOp layers
```

(a) Static nested adapter implementation

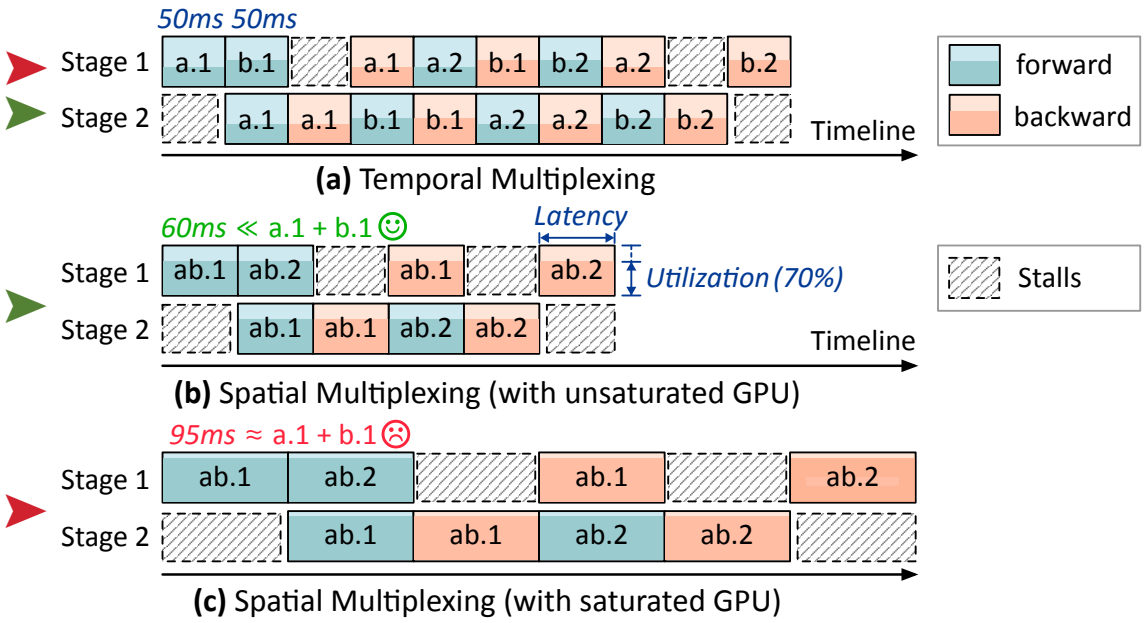
```
class LLMBackbone:# Original LLM class ✓
  def __init__(self, *cfg):
    self.qkv = nn.Linear(*cfg)
    ...
  def forward(self, x):
    qkv_out = self.qkv(x) # with adapter
    ... # Other non-BaseOp layers

def register_tasks(llm, tasks: List[..]):
  # On-the-fly task registry to backbone
  for task in tasks:
    for t_op in task.target:
      base_op = get_llm_op(llm, t_op)
      adapter = get_decouple_adapter(task)
      register_one_peft_hook( Dynamically
        base_op, adapter,       attached ✓
        dispatch_rule=task.dispatch,
        aggregate_rule=task.aggregate)
```

(b) On-the-fly adapter attachment

NOTE: Tasks are isolated for (1) runtime errors (*e.g., semantic error*), (2) forward + backward computation (*i.e., convergence guarantee*)!

Spatial-Temporal Fusing Tasks

Spatial-temporal tradeoff



Option 1: Temporal multiplexing

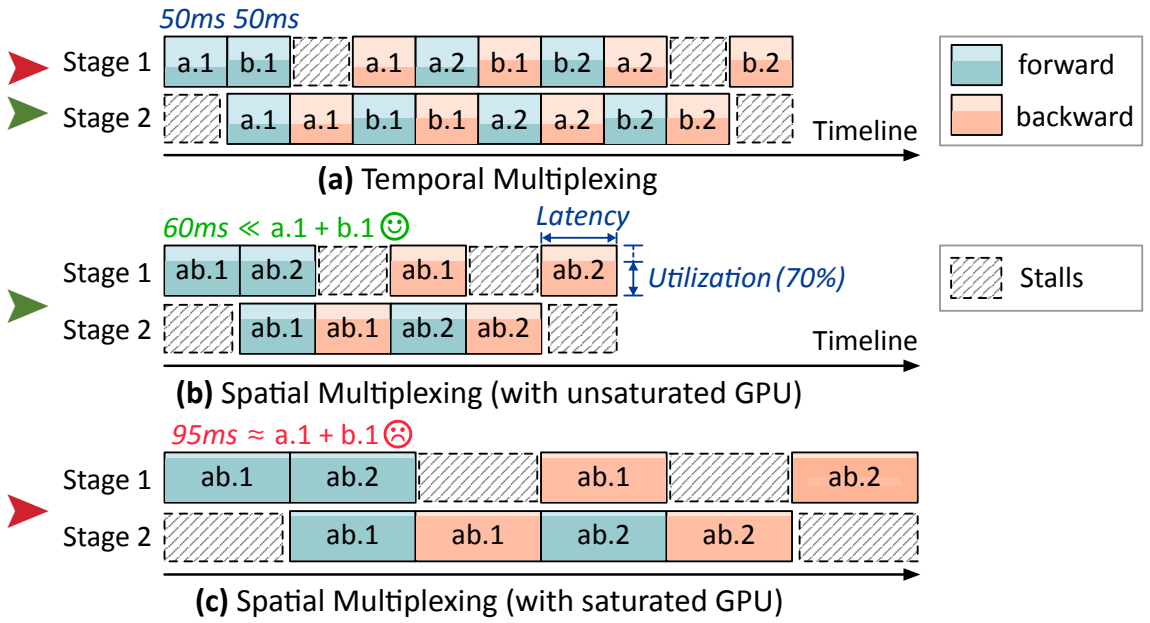
Interleave all task microbatches, **better w. high util.**

Option 2: Spatial multiplexing

Batch as one microbatch across tasks, **better w. low util.**

Spatial-Temporal Fusing Tasks

Spatial-temporal tradeoff



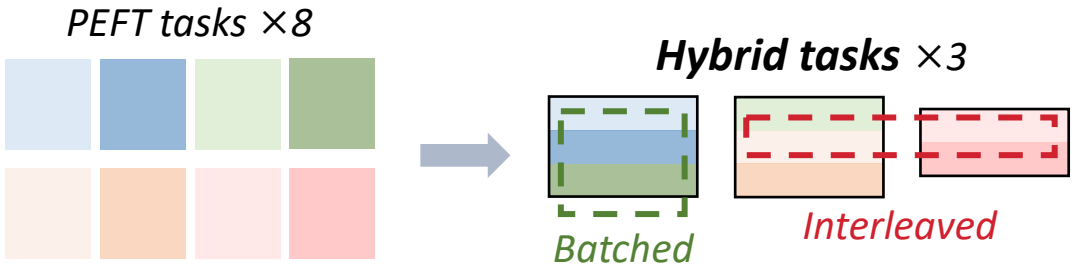
Option 1: Temporal multiplexing

Interleave all task microbatches, **better w. high util.**

Option 2: Spatial multiplexing

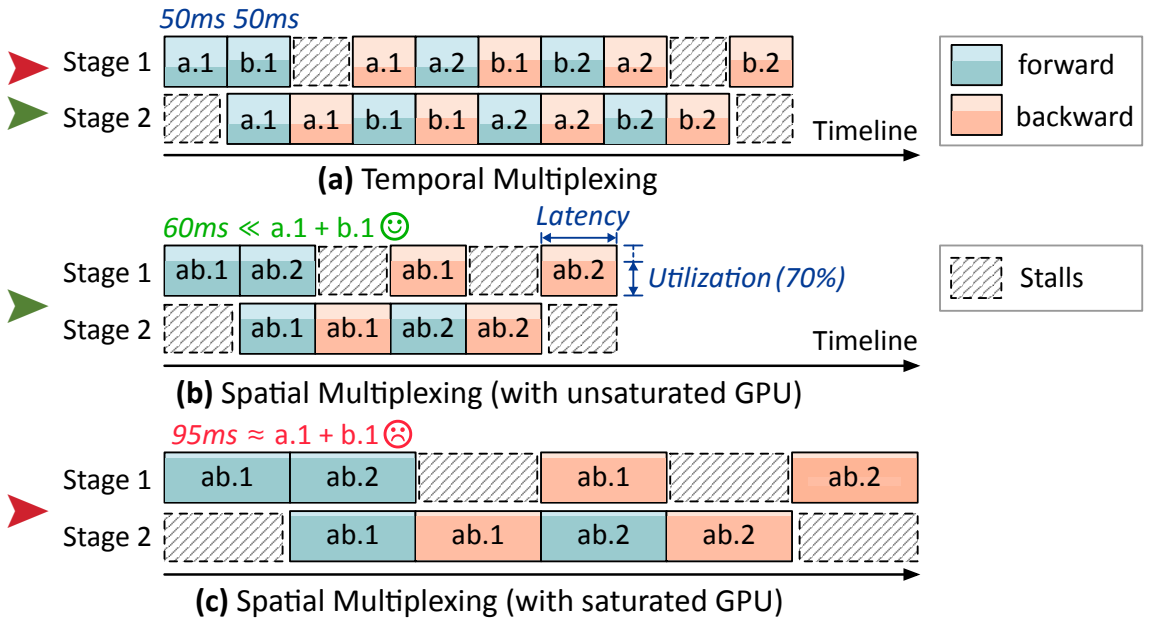
Batch as one microbatch across tasks, **better w. low util.**

Hybrid task (hTask) construction



Spatial-Temporal Fusing Tasks

Spatial-temporal tradeoff



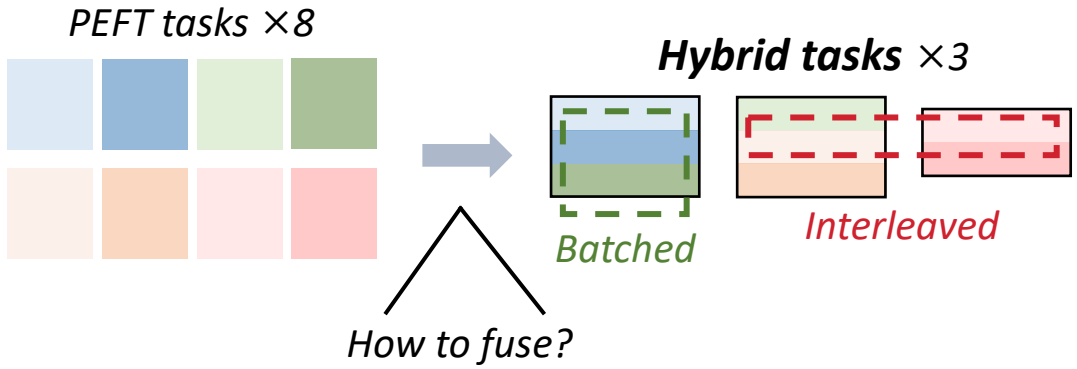
Option 1: Temporal multiplexing

Interleave all task microbatches, **better w. high util.**

Option 2: Spatial multiplexing

Batch as one microbatch across tasks, **better w. low util.**

Hybrid task (hTask) construction



Cost modeling

For $hTask_{i,j} = batch[t_i, t_{i+1}, \ldots, t_j]$, stage s

(stage latency) $lat^s_{i,j} = \sum_t \sum_{op} lat(base\_op) + \sum_t \sum_{op} lat(adapter)$

(end-to-end latency) $e2e\_lat_{i,j} = 2\sum_s lat^s_{i,j} + 2\#mb \max_s lat^s_{i,j}$

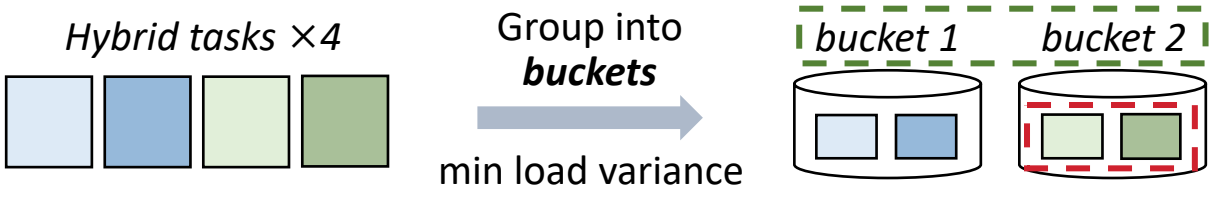
Dynamic programming algorithm

$F(m, n)$: min e2e latency of packing first m tasks into n hTasks

(transition eqn) $F(m, n) = \min_{n-1 \le i \le m} F(i, n-1) + e2e\_lat_{i+1,m} / S$

Orchestrating Operators of Hybrid Tasks

Disaggregating tasks as *two tiers*

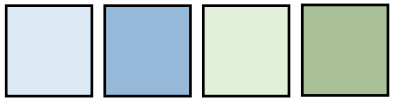


Tasks within the same bucket
→ ***Intra-stage orchestration***

Different buckets
→ ***Inter-stage orchestration***

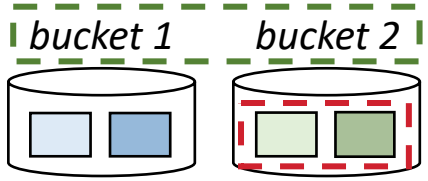
Orchestrating Operators of Hybrid Tasks

Disaggregating tasks as *two tiers*



Group into ***buckets***

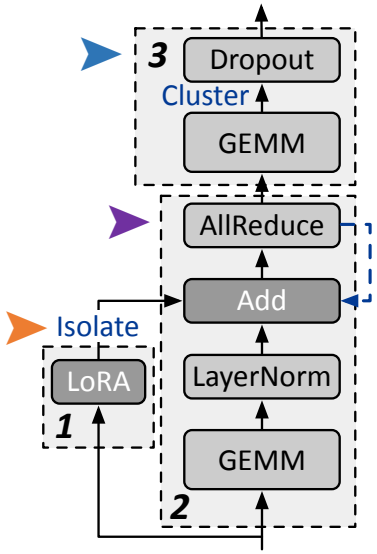
min load variance



Tasks within the same bucket
→ ***Intra-stage orchestration***

Different buckets
→ ***Inter-stage orchestration***

Intra-stage orchestration

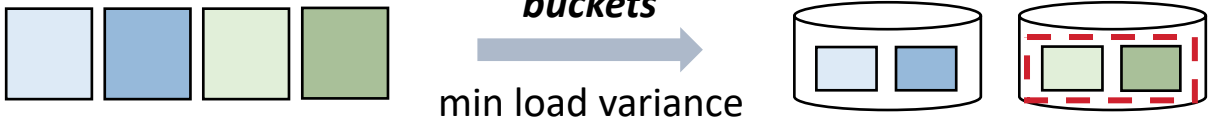


Constructing subgraphs for each hTask:

- Consecutive computation operators are **clustered**
- Communication operators are **appended**
- Adapters are **isolated**
- Assign subgraph **priority** by topo depth → interleaved

Orchestrating Operators of Hybrid Tasks

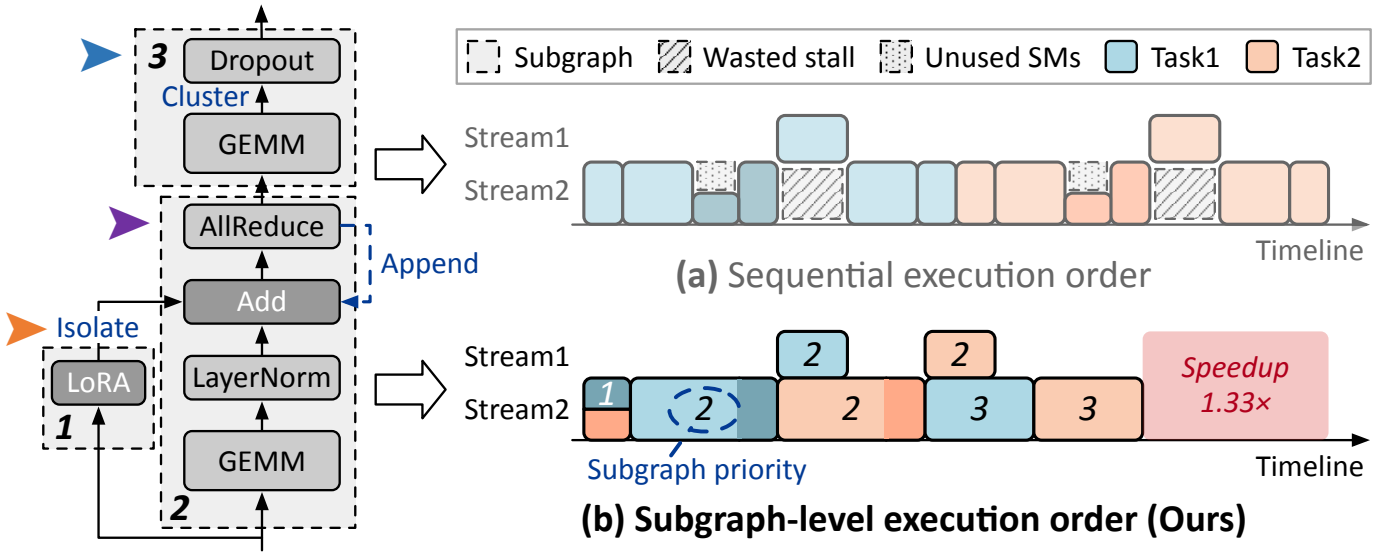
Disaggregating tasks as *two tiers*



Tasks within the same bucket
→ ***Intra-stage orchestration***

Different buckets
→ ***Inter-stage orchestration***

Intra-stage orchestration



Constructing subgraphs for each hTask:

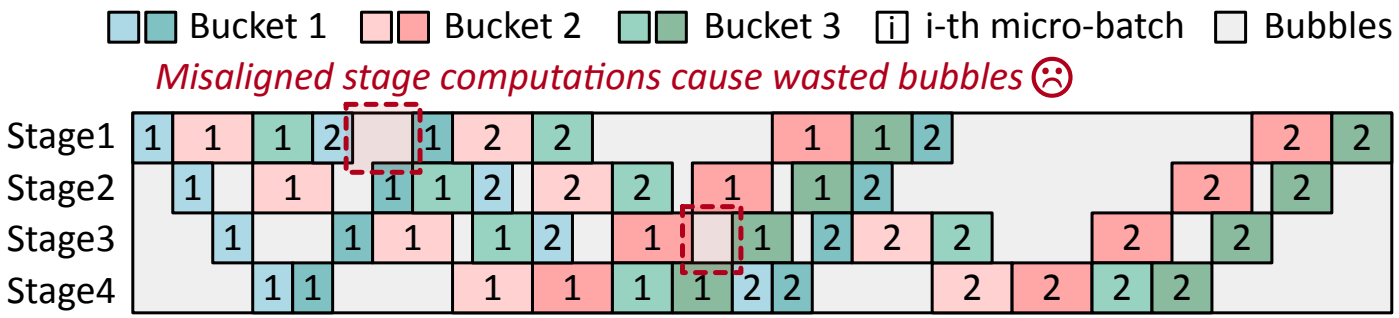
- Consecutive computation operators are **clustered**
- Communication operators are **appended**
- Adapters are **isolated**
- Assign subgraph **priority** by topo depth → interleaved

Scheduling subgraphs of hTasks:

1. Maintain a priority queue to **track zero in-degree subgraphs** in each DAG
2. Iteratively **pop** the subgraph with longest latency
3. **Enqueue** new zero in-degree ones

Orchestrating Operators of Hybrid Tasks

Inter-stage orchestration *(i.e., micro-batch scheduling)*



(a) 1F1B with unordered, interleaved tasks

Computation homogeneity:

- Consistent microbatch shapes across iterations

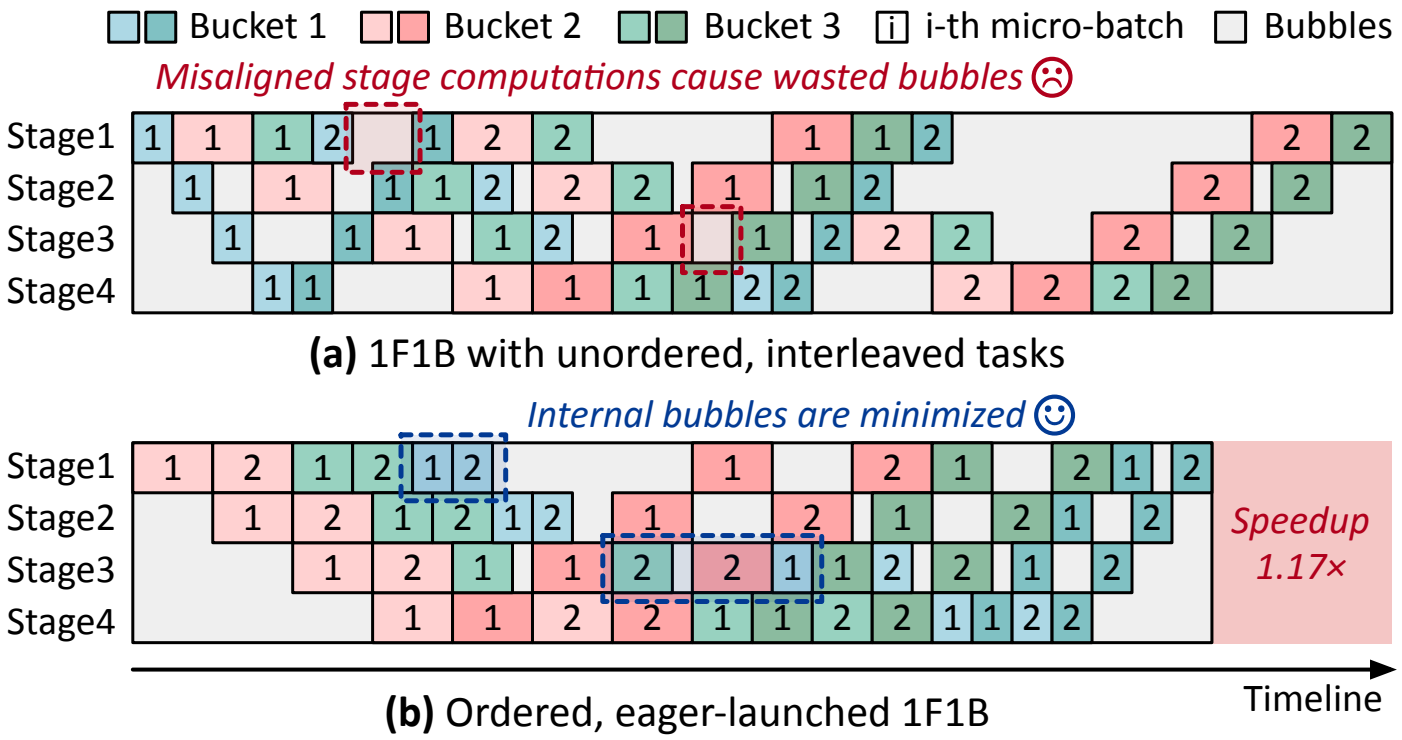
  → ***Pipeline template**, no costly per-iter searching*

- Equal fwd/bwd latency for the same stage

  → ***Interleave fwd/bwd** to minimize bubbles*

Orchestrating Operators of Hybrid Tasks

Inter-stage orchestration *(i.e., micro-batch scheduling)*



Computation homogeneity:

- Consistent microbatch shapes across iterations

  → ***Pipeline template**, no costly per-iter searching*

- Equal fwd/bwd latency for the same stage

  → ***Interleave fwd/bwd** to minimize bubbles*

Structured pipeline template:

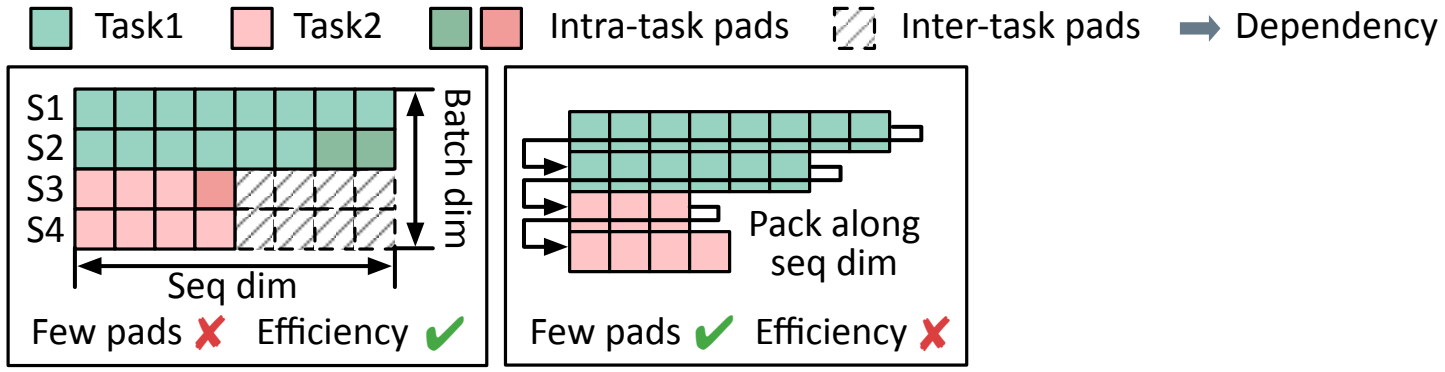
1. Sort buckets **by latency** in descending order
2. **Consecutive** microbatches of the same bucket
3. **Early launching** microbatches in memory limits

(Optimality analysis)

For *1F1B-like* pipeline, near-optimal execution

⇔ **mitigating internal bubbles of the last stage!**

Aligning Data for a Hybrid Task

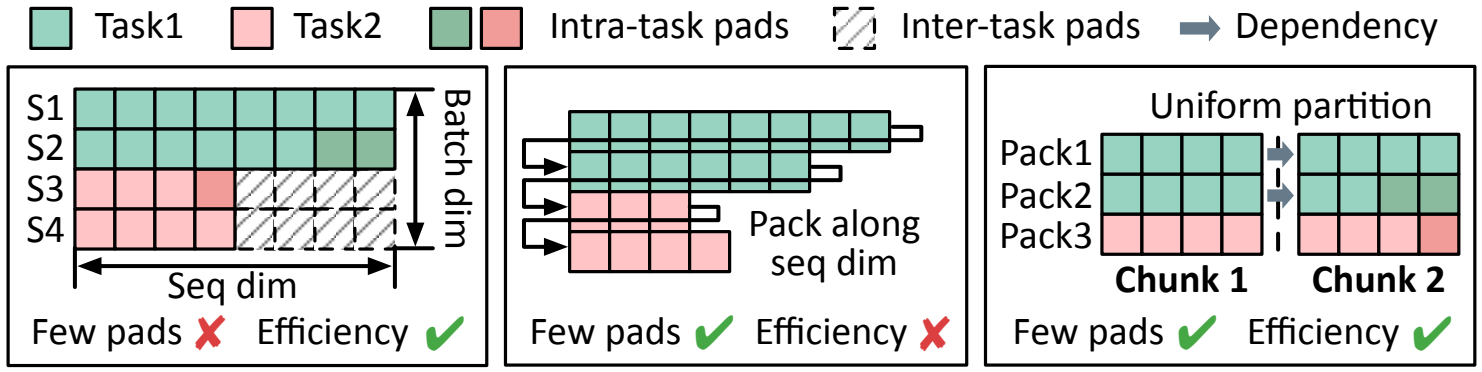


(a) Zero-pad to max length **(b)** Pack sequences

Why common strategies cannot apply?

Strategy 1: Zero-padding	Strategy 2: Packing
Zero-pad a microbatch's sequences to a **unified maximum length**	Pack a micro-batch's sequences into **a longer sequence**
Non-semantic tokens waste compute and memory resources	Wasted attention computation across sequences due to attention mask

Aligning Data for a Hybrid Task



(a) Zero-pad to max length **(b)** Pack sequences **(c)** Chunk-based alignment

Why common strategies cannot apply?

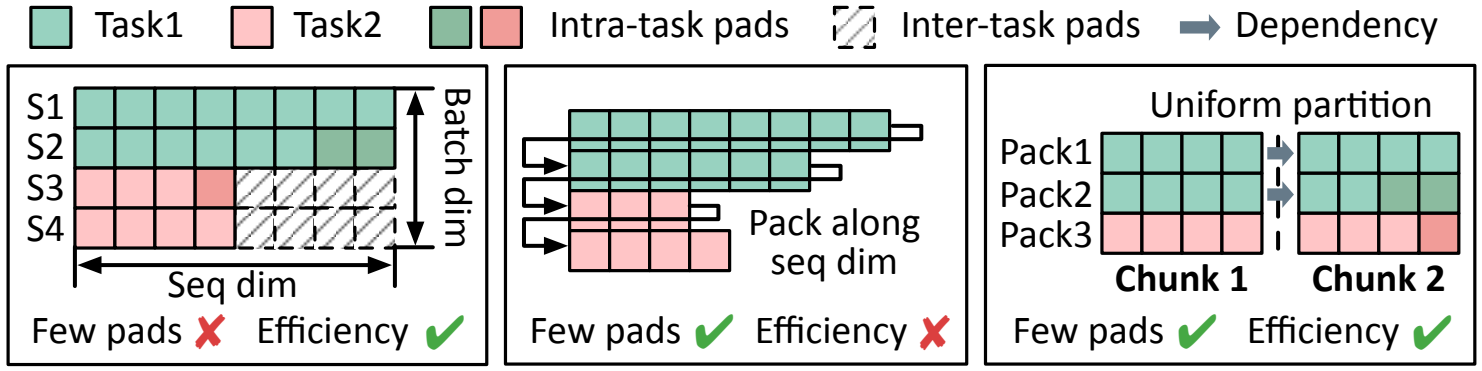
Strategy 1: Zero-padding	Strategy 2: Packing
Zero-pad a microbatch's sequences to a **unified maximum length**	Pack a micro-batch's sequences into **a longer sequence**
Non-semantic tokens waste compute and memory resources	Wasted attention computation across sequences due to attention mask

Our solution: chunk-based alignment

1. For each task, **adaptively pack** a global batch's sequences into longer ones;
2. **Uniformly partition** into **equal-sized chunks**. Scatter long sequences across chunks with KV cache reuse.

Key idea: *Adjust intra-chunk task ratio to balance sequence length difference, without attention waste.*

Aligning Data for a Hybrid Task



(a) Zero-pad to max length **(b)** Pack sequences **(c)** Chunk-based alignment

Why common strategies cannot apply?

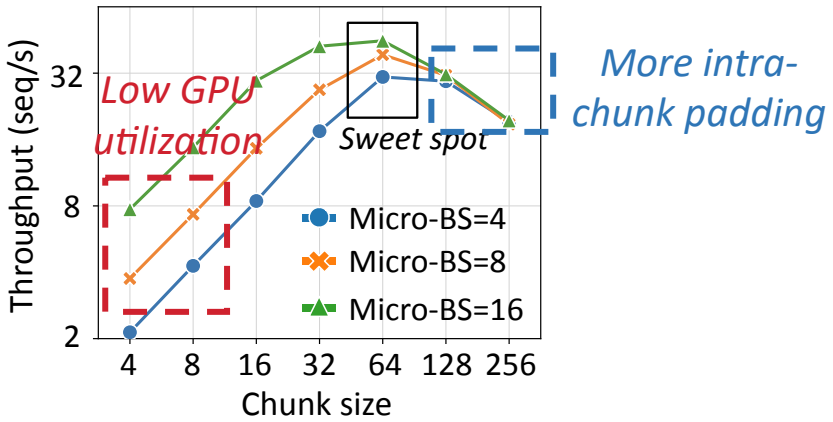
Strategy 1: Zero-padding	Strategy 2: Packing
Zero-pad a microbatch's sequences to a **unified maximum length**	Pack a micro-batch's sequences into **a longer sequence**
Non-semantic tokens waste compute and memory resources	Wasted attention computation across sequences due to attention mask

Our solution: chunk-based alignment

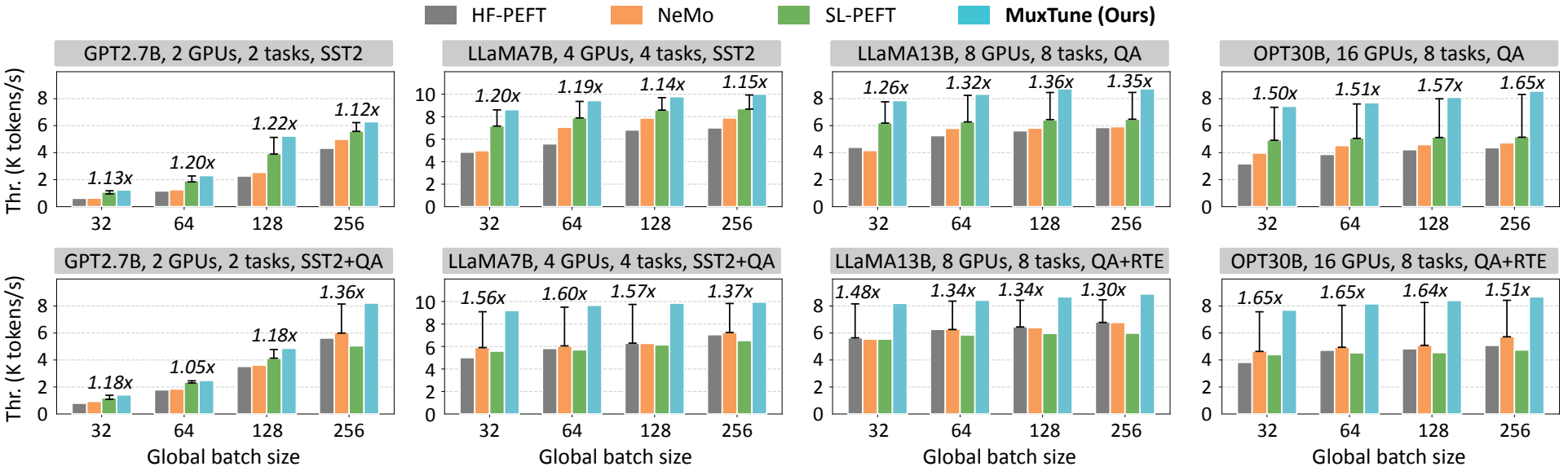
1. For each task, **adaptively pack** a global batch's sequences into longer ones;
2. **Uniformly partition** into **equal-sized chunks**. Scatter long sequences across chunks with KV cache reuse.

Key idea: *Adjust intra-chunk task ratio to balance sequence length difference, without attention waste.*

Chunk size: **efficiency** v.s. **padding reduction**

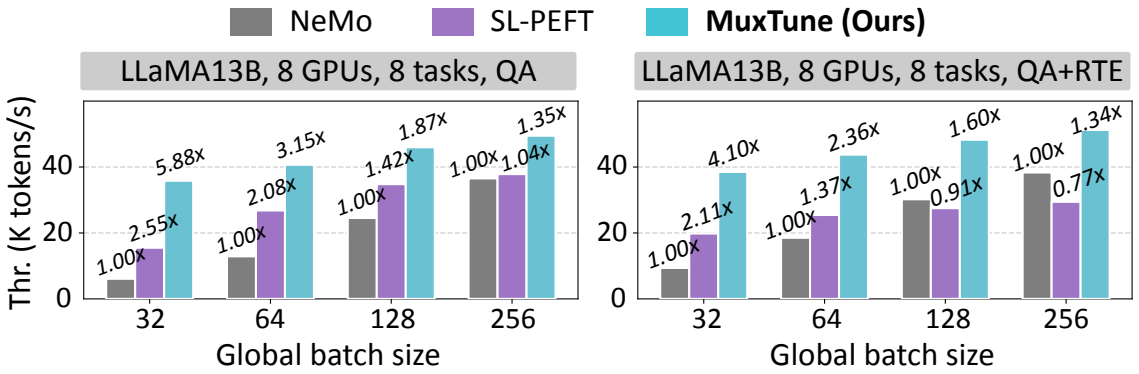


End-to-End Experiments



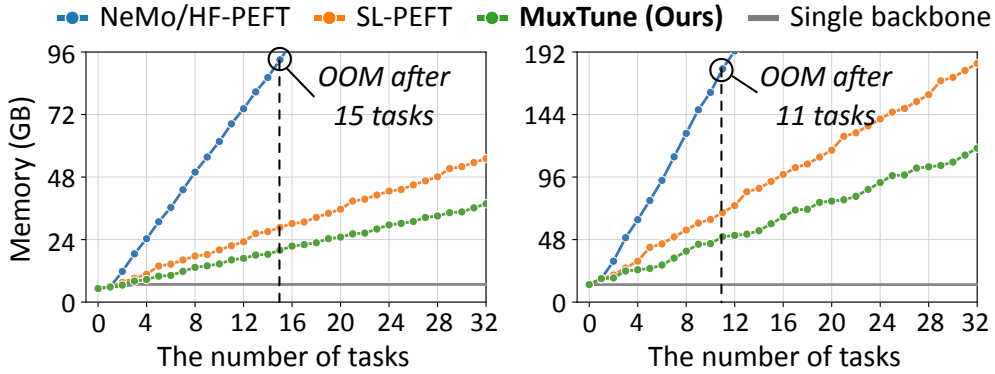
System throughput across input sizes, backbone types, and hardware

+1.05× ~ +2.33× throughput



Throughput on higher-end hardware (H100)

+1.34× ~ +5.88× throughput

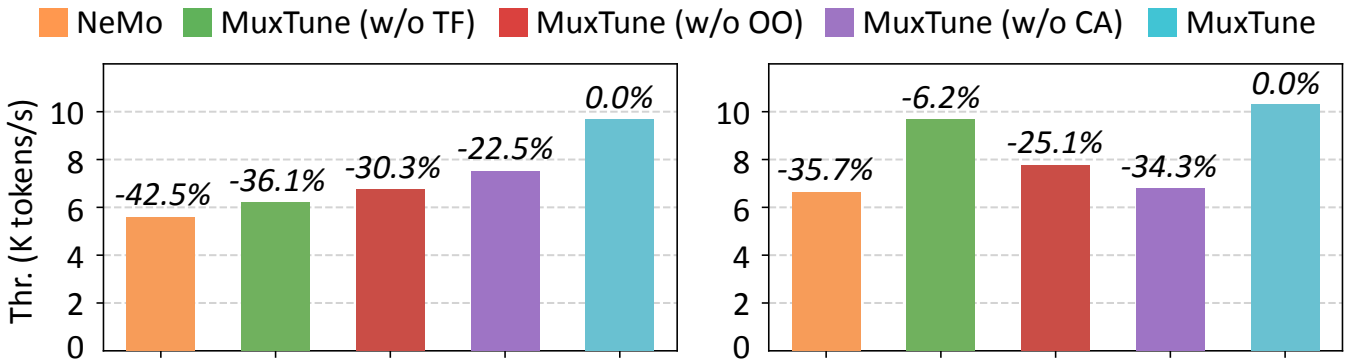


(a) GPT3-2.7B (2-GPU tensor parallel) **(b)** LLaMA2-7B (4-GPU pipeline)

Memory footprint across #tasks

-1.44× ~ -5.29× memory

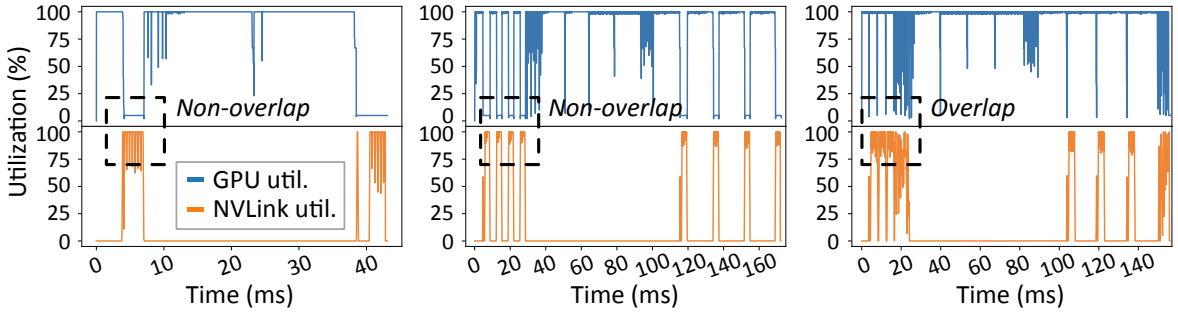
Ablation Studies



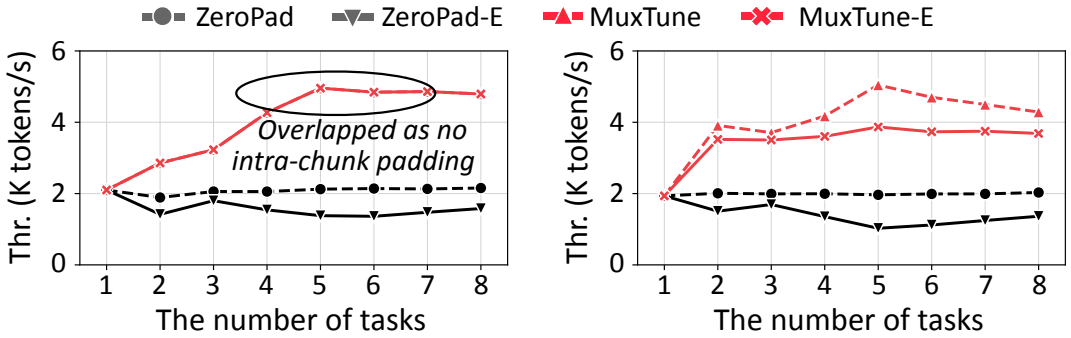
(a) 2 tasks, 4 micro-batches, SST2+QA

(b) 4 tasks, 8 micro-batches, QA+RTE

Computation fusion improves GPU utilization, **communication overlapping** reduces device stalls, **chunk-based alignment** mitigates ineffective tokens



(a) NeMo (1 task) **(b)** Ours (4 tasks, no overlap) **(c)** Ours (4 tasks, overlap)



(a) Workload-A (SST2 + QA) **(b)** Workload-B (SST2 + RTE)

Efficiency of operator orchestration

Device stalls are efficiently overlapped

Effectiveness of data alignment

Chunk-based alignment improves effective throughput

USENIX nsdi'26

Thank you!

Contact: dicardo@sjtu.edu.cn

Summary:

- MuxTune **multiplexes the LLM backbone** across independent tasks in a **spatial-temporal** manner to improve resource efficiency.
- MuxTune improves parameter-efficient fine-tuning (PEFT) throughput by **2.33×,** and reduces memory footprint by **5.29×**.

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

NUS
National University of Singapore