Mist: Efficient Distributed Training of Large Language Models via Memory-Parallelism Co-Optimization

Zhanda Zhu[1, 2, 3], Christina Giannoula[1, 2, 3], Muralidhar Andoorveedu[1], Qidong Su[1, 2, 3], Karttikeya Mangalam[4], Bojian Zheng[1, 2, 3], Gennady Pekhimenko[1, 2, 3]

1 CentML

2 

3 VECTOR INSTITUTE

4 SigIQ.ai

Executive Summary

- Accelerate distributed training by comprehensively co-optimizing *parallelism* and *memory optimizations*.
- Key Challenges
  - **Exploded** and complex configuration search space.
  - **Inaccurate** performance prediction: due to missing overlap and ignoring inter-microbatch imbalance.
- Mist addresses the challenges with
  ① Symbolic-based Performance Analysis
  ② Overlap-Centric Scheduling & Imbalance-aware hierarchical tuning
- Key Result: Up to $\mathbf{1.73}\times$ better vs. Megatron-LM and $\mathbf{2.04}\times$ vs. the automatic method Aceso, respectively.

Large Language Models

- Widely adopted in both open communities and commercial systems.

LLaMA

OpenAI

Claude

- State-of-the-art performance in many applications:

Text Generation

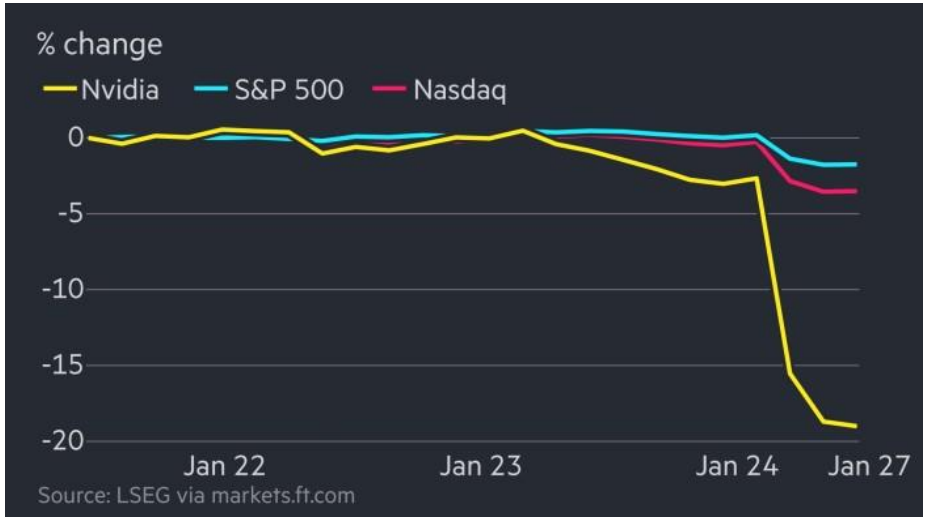
Machine Translation

Speech Recognition

Cost of Training Large Language Models

- Extremely resource-consuming and expensive!

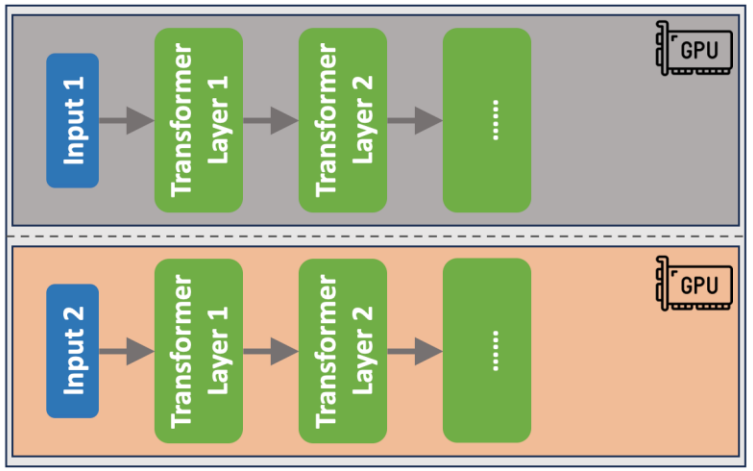
	Training Costs ($)	Gas Emissions (tons CO_2)
Llama 3.1 8B	2.92M	420
Llama 3.1 70B	14.00M	2,040
Llama 3.1 405B	61.68M	8,930
Total	78.60M	11,390



- Strong incentive to reduce the training time of these models.

Training Consumption Source: github.com/meta-llama/llama-models/blob/main/models/llama3_1/MODEL_CARD.md

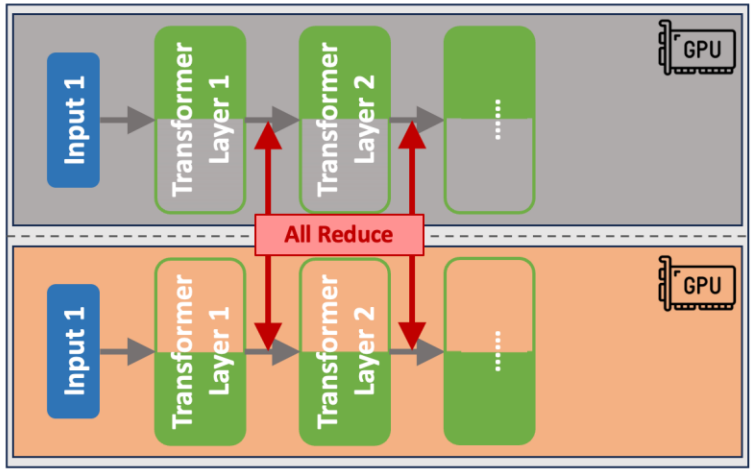
Optimizations – Distributed Parallelism



Data Parallelism

Partition input data

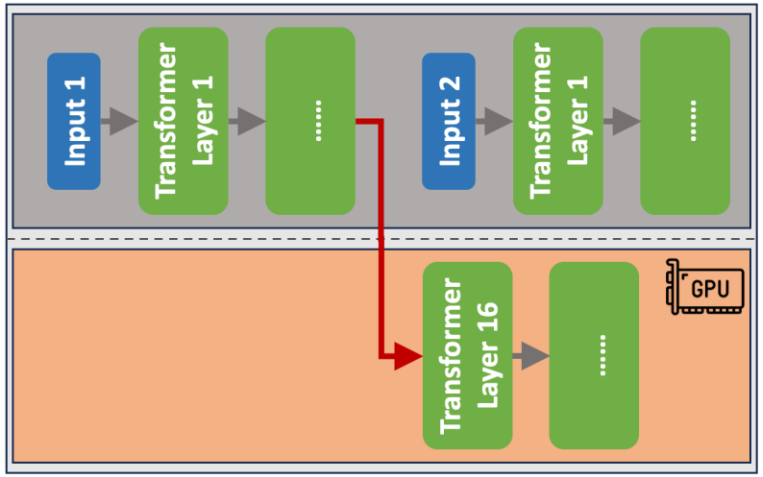
- Computation Utilization: 😀
- Communication Overhead: 😀
- Memory Usage: 😨



Tensor Parallelism

Partition linear layers

- Computation Overhead: 😀
  Communication Overhead: 😨
- Memory Usage: 😀

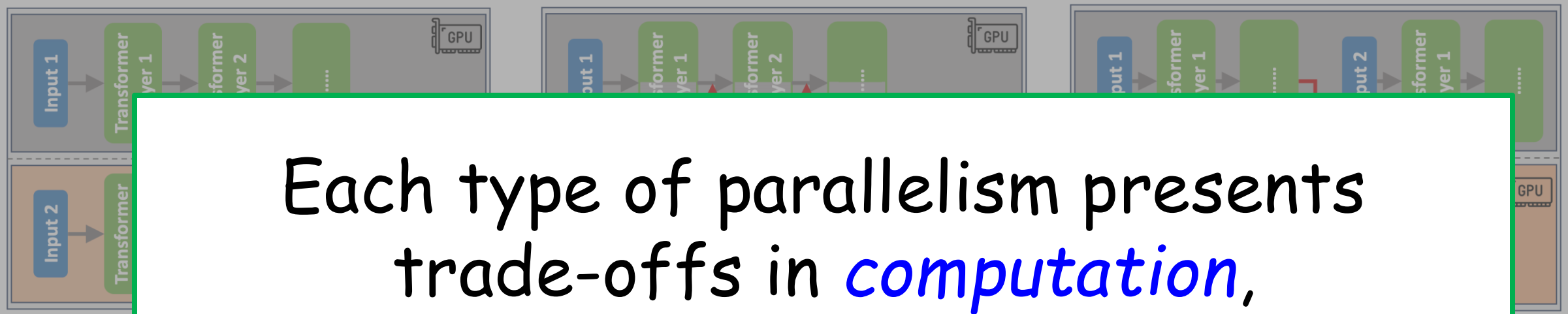


Pipeline Parallelism

Partitions the model into stages

- Computation Utilization: 😨
- Communication Overhead: 😀
- Memory Usage: 😀

Optimizations – Distributed Parallelism



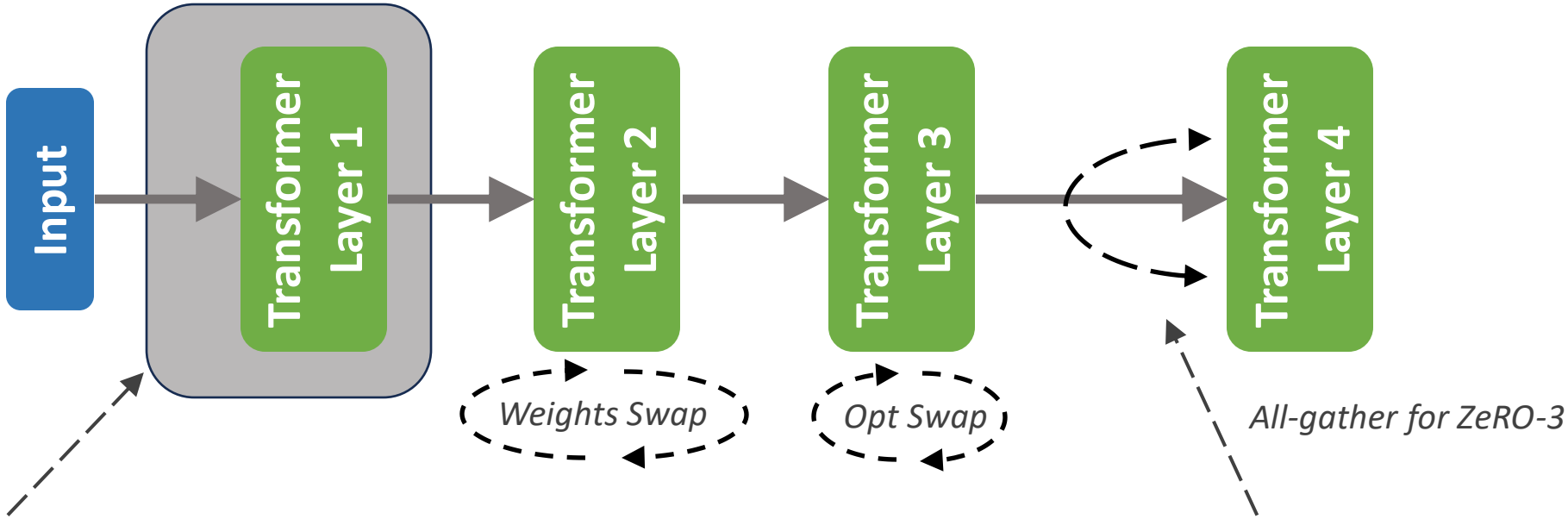
Each type of parallelism presents trade-offs in *computation*, *communication*, and *memory footprint*.

- Computation Utilization: 😀
- Communication Overhead: 😀
- Memory Usage: 😨

- Computation Overhead: 😀
- Communication Overhead: 😨
- Memory Usage: 😀

- Computation Utilization: 😨
- Communication Overhead: 😀
- Memory Usage: 😀

Optimizations - Memory Optimizations



Activation Checkpointing (CKPT)
Recompute activations in BWD

- Computation Utilization: 😀
- Communication Overhead: 😀
- Memory Usage: 😨

Offloading/Swapping
Offload tensors to CPU

- Computation Overhead: 😀
- Communication Overhead: 😨
- Memory Usage: 😀

ZeRO
Redundancy elimination

- Computation Utilization: 😨
- Communication Overhead: 😀
- Memory Usage: 😀

Optimizations - Memory Optimizations

Each type of memory optimizations presents trade-offs in *computation*, *communication*, and *memory footprint*.

Activation

Recomp

- Computation Utilization: 😀
- Communication Overhead: 😀
- Memory Usage: 😨

- Computation Overhead: 😀
- Communication Overhead: 😨
- Memory Usage: 😀

- Computation Utilization: 😨
- Communication Overhead: 😀
- Memory Usage: 😀

Optimizations - Memory Optimizations

Parallelism and *memory footprint reduction techniques* need to be jointly optimized.

Activation

Recomp

- Computation Utilization: 😀
- Communication Overhead: 😀
- Memory Usage: 😨

- Computation Overhead: 😀
- Communication Overhead: 😨
- Memory Usage: 😀

- Computation Utilization: 😨
- Communication Overhead: 😀
- Memory Usage: 😀

The Need for Comprehensive Co-Optimization

GPU Memory Footprint

The Need for Comprehensive Co-Optimization

Aggressive Memory Optimizations

- Apply a higher level of ZeRO

Overhead ↑

GPU Memory Footprint

The Need for Comprehensive Co-Optimization

Aggressive Memory Optimizations

- Apply a higher level of ZeRO
- Apply more CKPT

Overhead ↑ ↑

GPU Memory Footprint

The Need for Comprehensive Co-Optimization

Aggressive Memory Optimizations

- Apply a higher level of ZeRO
- Apply more CKPT
- Apply more offloading

Overhead ↑ ↑ ↑

GPU Memory Footprint

The Need for Comprehensive Co-Optimization

Aggressive Memory Optimizations	Utilize Gained Memory
• Apply a higher level of ZeRO • Apply more CKPT • Apply more offloading	• Reduce TP Size
Overhead ↑ ↑ ↑	**Perf Gain ↑**

The Need for Comprehensive Co-Optimization

Aggressive Memory Optimizations	Utilize Gained Memory
• Apply a higher level of ZeRO • Apply more CKPT • Apply more offloading	• Reduce TP Size • Reduce PP Size
Overhead ↑ ↑ ↑	**Perf Gain ↑ ↑**

The Need for Comprehensive Co-Optimization

Aggressive Memory Optimizations	Utilize Gained Memory
• Apply a higher level of ZeRO • Apply more CKPT • Apply more offloading	• Reduce TP Size • Reduce PP Size • Increase batch size

Overhead ↑ ↑ ↑

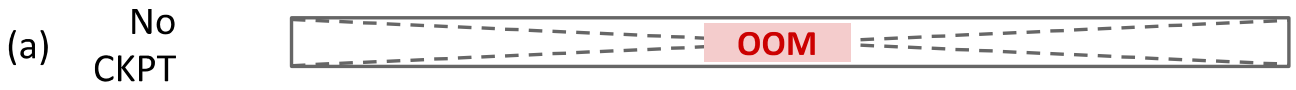
Perf Gain ↑ ↑ ↑

GPU Memory Footprint

Motivational Example

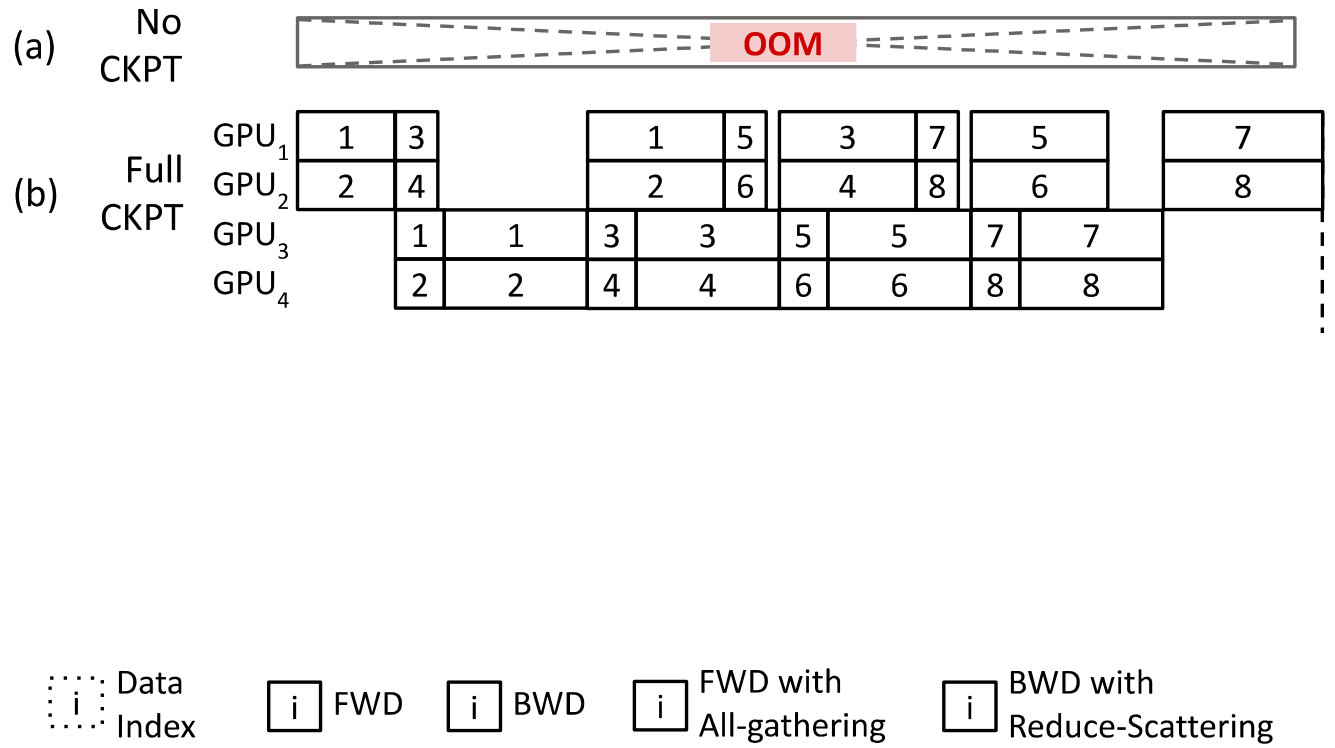
- GPT-3-2.7B model on 4 GPUS, Seq=4096, Bsz_{global}=8

Motivational Example



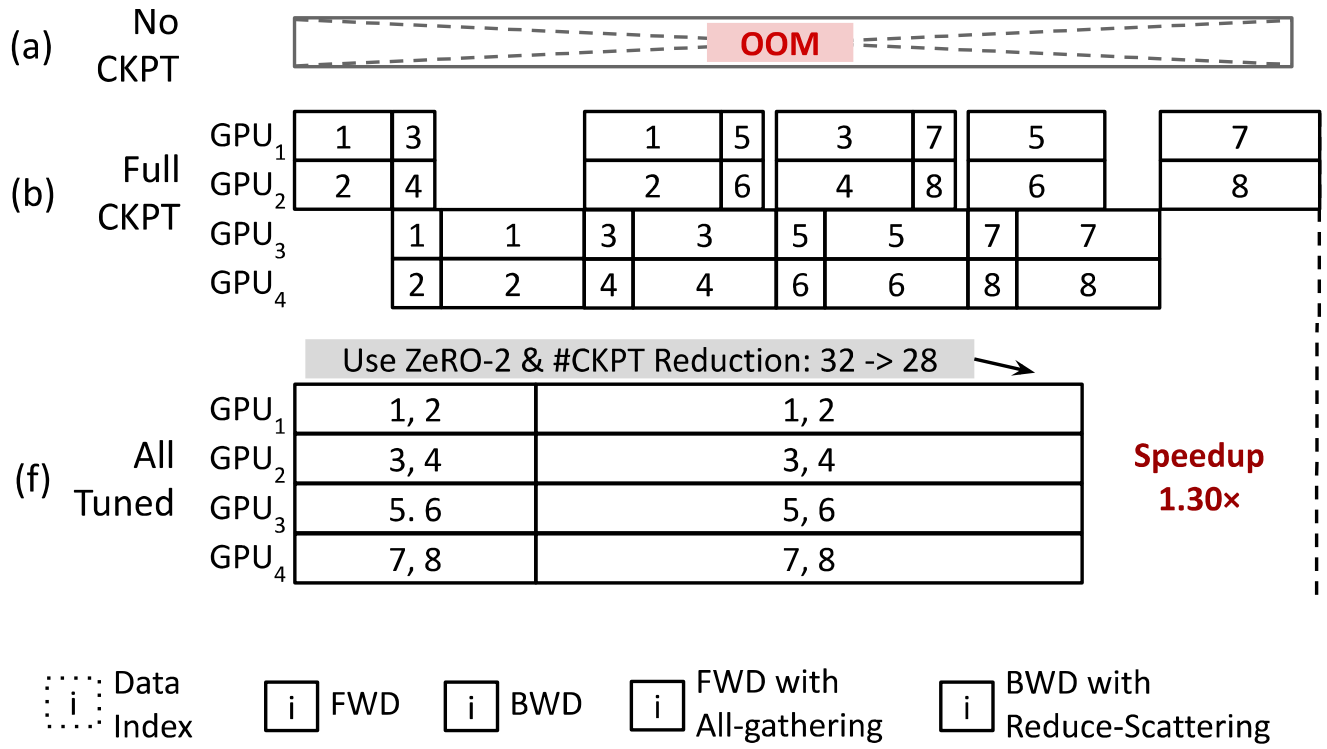
- GPT-3-2.7B model on 4 GPUS, Seq=4096, Bsz_{global}=8

Motivational Example



- GPT-3-2.7B model on 4 GPUS, Seq=4096, Bsz_{global}=8

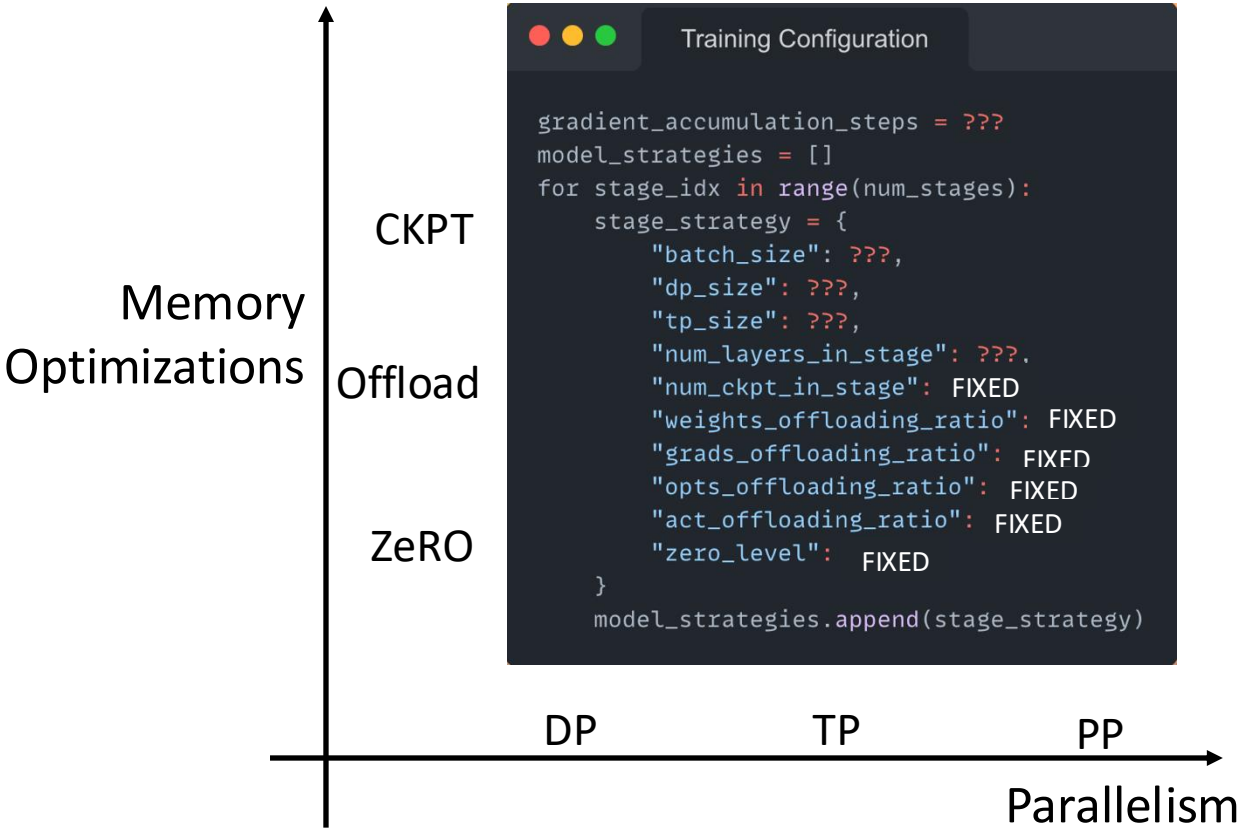
Motivational Example

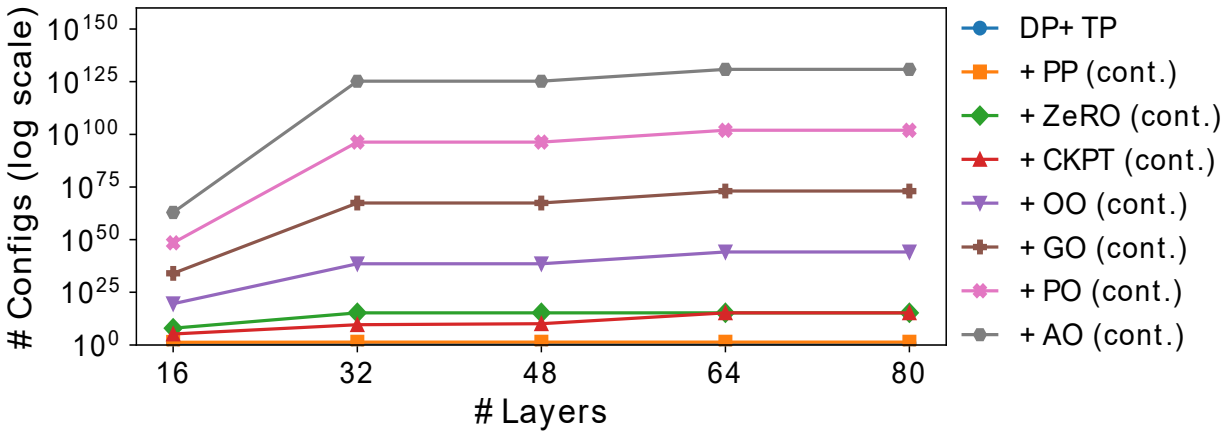


- GPT-3-2.7B model on 4 GPUS, Seq=4096, Bsz_{global}=8

Why do existing systems fail to co-optimize?

- Shortcoming #1: Unable to navigate the exploded search space.



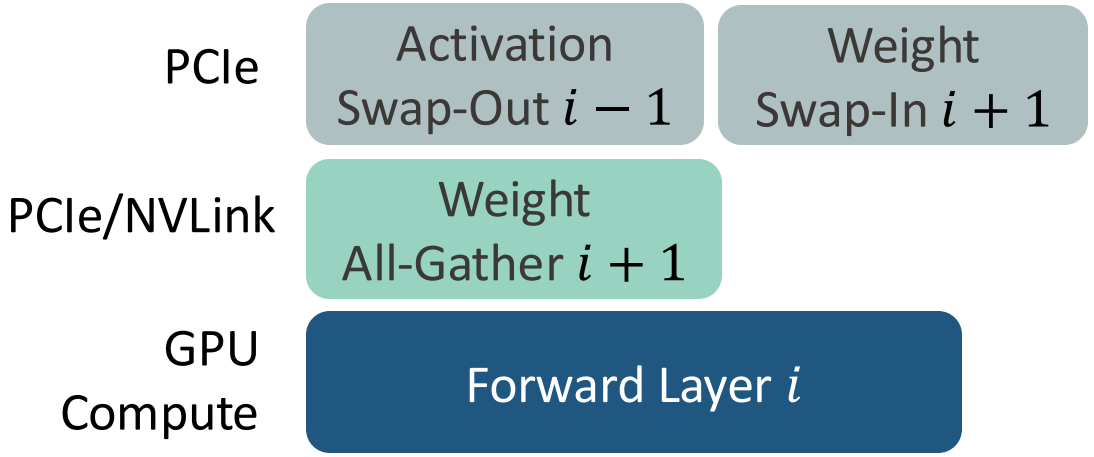


Why do existing systems fail to co-optimize?

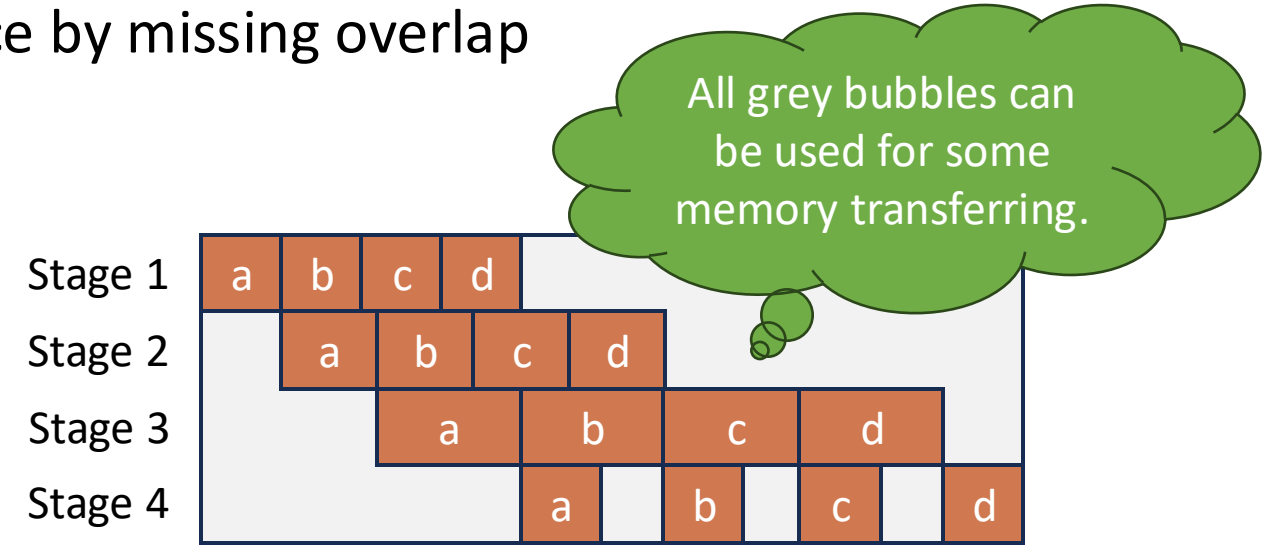
- Shortcoming #1: Unable to navigate the exploded search space.

Why do existing systems fail to co-optimize?

- Shortcoming #1: Unable to navigate the exploded search space.
- Shortcoming #2: Inaccurate performance prediction.
  - #2.1 Underestimate performance by missing overlap



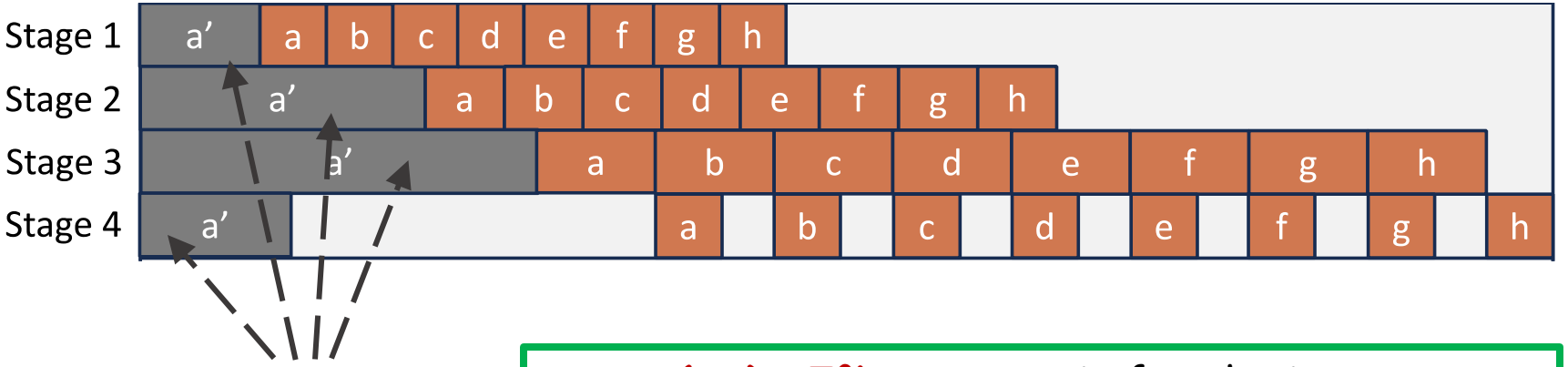
Mitigate the Overhead of Offloading and ZeRO



Overlap Opportunities in Pipeline Parallelism

Why do existing systems fail to co-optimize?

- Shortcoming #1: Unable to navigate the exploded search space.
- Shortcoming #2: Inaccurate performance prediction.
  - #2.1 Under-estimate performance by missing overlap
  - #2.2 Mispredict performance by ignoring inter-microbatch imbalance



The first and last micro-batches cost more time.

extra (+-) ~7% error ratio for the inter-stage performance prediction, leading to up to **extra 15% performance degradation**.

Outline

Background of distributed training and its optimizations

Shortcomings of existing distributed training systems

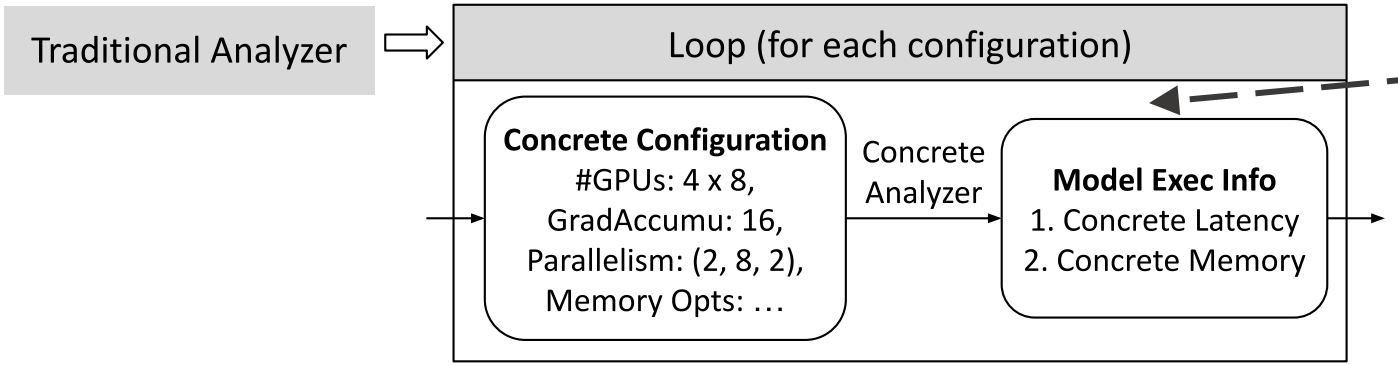
Key Ideas of Mist to address these shortcomings

Evaluation on state-of-the-art workloads

Mist's Key Ideas

① Symbolic-Based Efficient Performance Prediction.
② Fine-Grained Overlap-Centric Scheduling.
③ Imbalance-Aware Hierarchical Tuning via Pareto Frontier Sampling.

① Symbolic-Based Efficient Performance Prediction



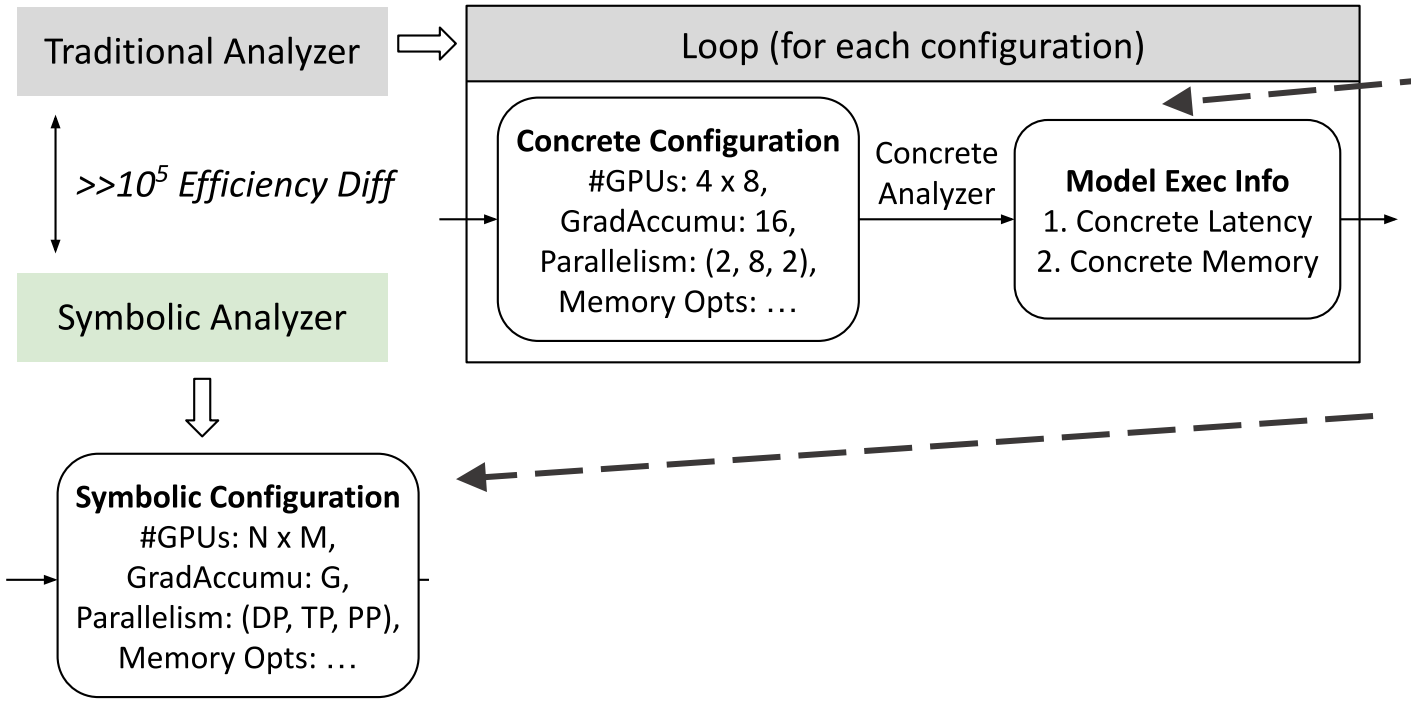
several seconds for each configuration

Insight: Repeated simulation is doing redundant work

Mist's Key Ideas

① Symbolic-Based Efficient Performance Prediction.
② Fine-Grained Overlap-Centric Scheduling.
③ Imbalance-Aware Hierarchical Tuning via Pareto Frontier Sampling.

① Symbolic-Based Efficient Performance Prediction



several seconds for each configuration

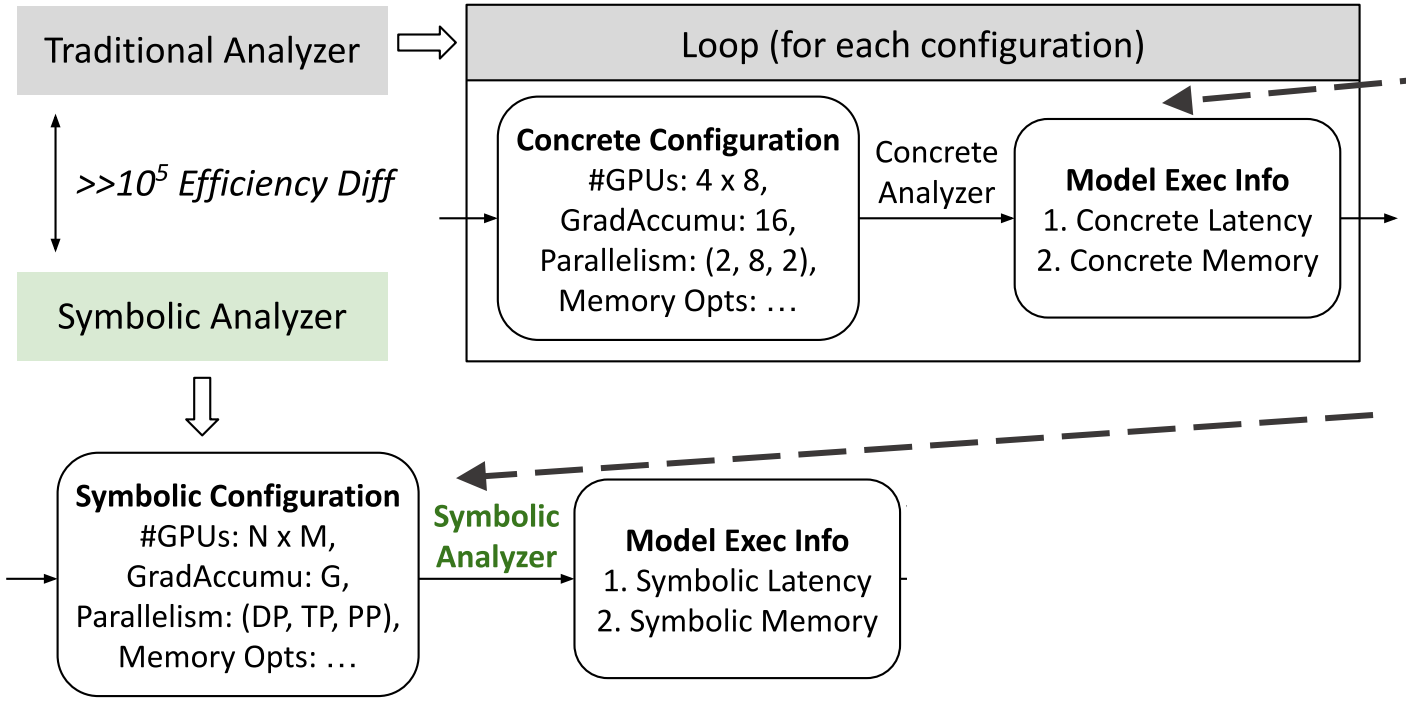
```python
from mist import global_symbol_manager as gsm
# Define symbols
b, s, h, d, tp = gsm.symbols("b s h d tp", (4, 128, 12, 64, 8),
      integer=True, positive=True)
# Initialize the model configuration using symbolic parameters
config = GPT2Config(n_embd=h*d, n_head=h, tp=tp, ...)
# Construct the GPT-2 model with symbolic configuration
model = GPT2LMHeadModel(config)
# Create symbolic input tensors
input_ids = torch.randint(0, V, (b, s), dtype=torch.long)
# Execute the model with symbolic inputs
logits = model(input_ids).logits
```

```
>>> logits
symbolic_tensor((b, s, V), concrete_shape=(4, 128, 50257), ...)
```

Mist's Key Ideas

① Symbolic-Based Efficient Performance Prediction.
② Fine-Grained Overlap-Centric Scheduling.
③ Imbalance-Aware Hierarchical Tuning via Pareto Frontier Sampling.

① Symbolic-Based Efficient Performance Prediction



several seconds for each configuration

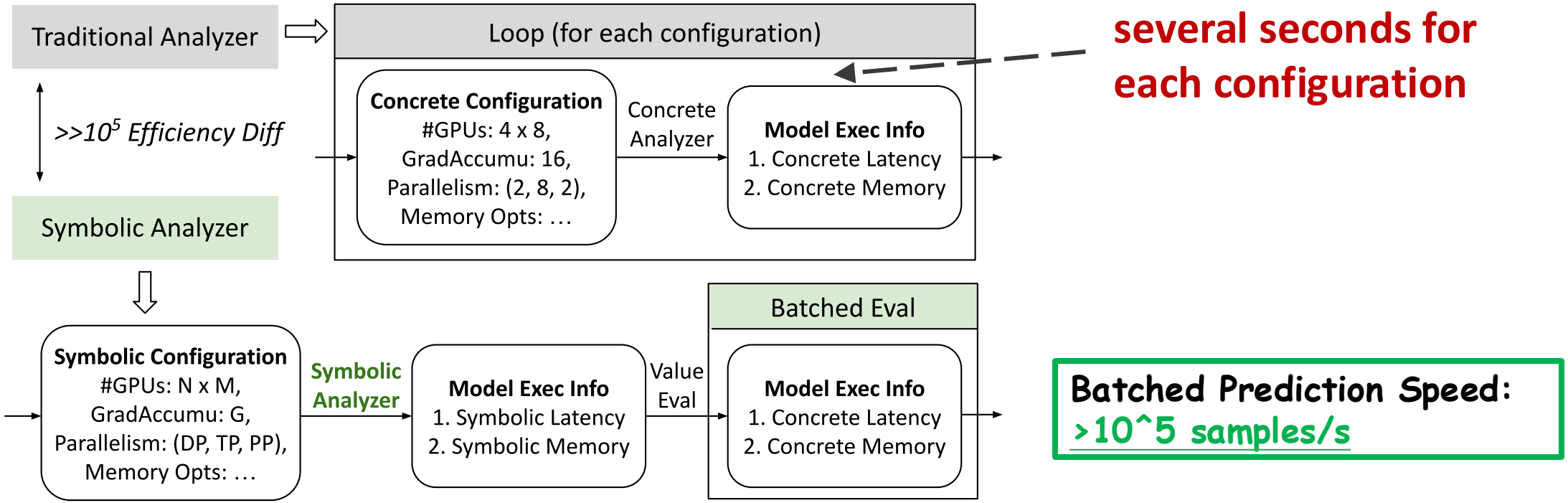
```python
from mist import global_symbol_manager as gsm
# Define symbols
b, s, h, d, tp = gsm.symbols("b s h d tp", (4, 128, 12, 64, 8),
      integer=True, positive=True)
# Initialize the model configuration using symbolic parameters
config = GPT2Config(n_embd=h*d, n_head=h, tp=tp, ...)
# Construct the GPT-2 model with symbolic configuration
model = GPT2LMHeadModel(config)
# Create symbolic input tensors
input_ids = torch.randint(0, V, (b, s), dtype=torch.long)
# Execute the model with symbolic inputs
logits = model(input_ids).logits
```

```
>>> logits
symbolic_tensor((b, s, V), concrete_shape=(4, 128, 50257), ...)
```

Mist's Key Ideas

① Symbolic-Based Efficient Performance Prediction.
② Fine-Grained Overlap-Centric Scheduling.
③ Imbalance-Aware Hierarchical Tuning via Pareto Frontier Sampling.

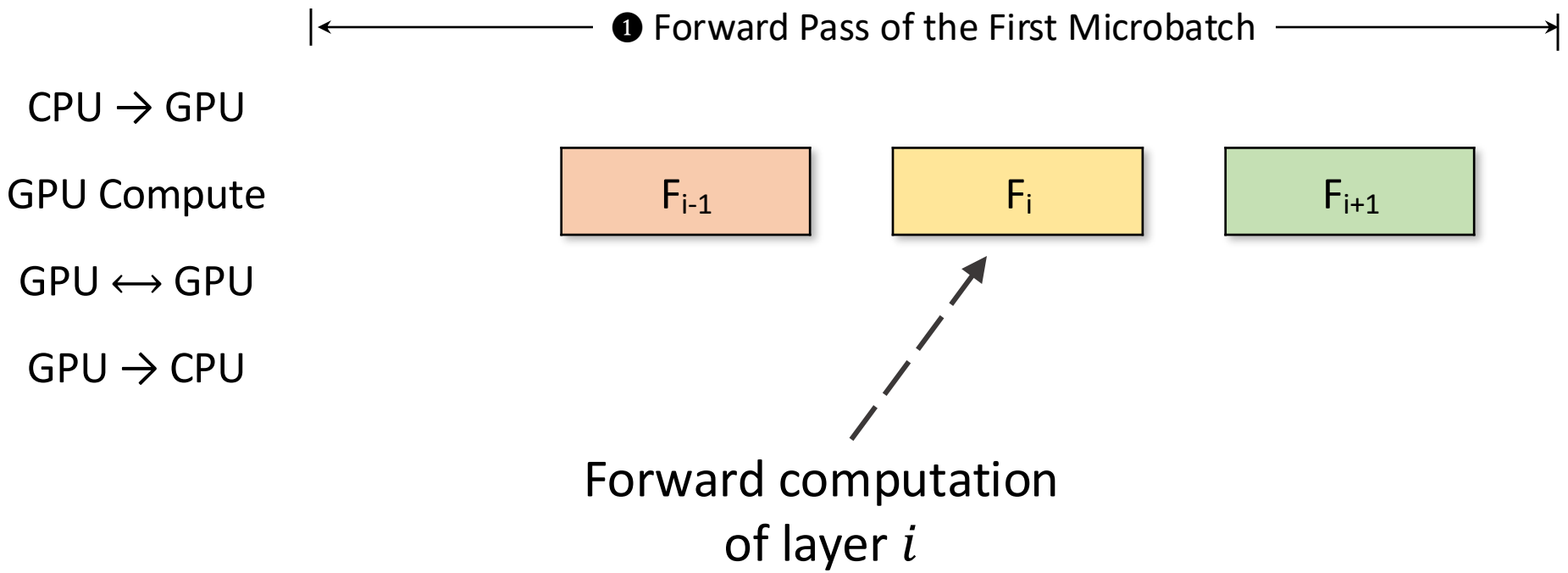
① Symbolic-Based Efficient Performance Prediction



Mist's Key Ideas

① Symbolic-Based Efficient Performance Prediction.
② Fine-Grained Overlap-Centric Scheduling.
③ Imbalance-Aware Hierarchical Tuning via Pareto Frontier Sampling.

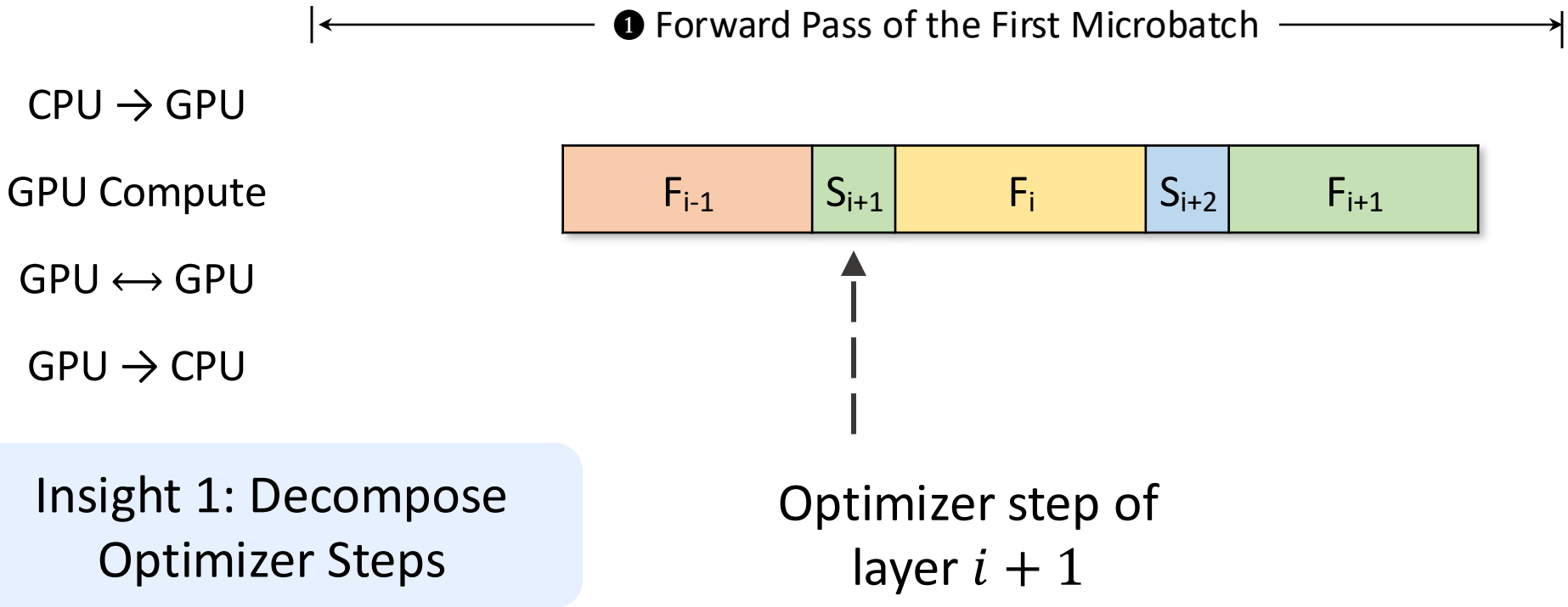
② Fine-Grained Overlap-Centric Scheduling.



Mist's Key Ideas

① Symbolic-Based Efficient Performance Prediction.
② Fine-Grained Overlap-Centric Scheduling.
③ Imbalance-Aware Hierarchical Tuning via Pareto Frontier Sampling.

② Fine-Grained Overlap-Centric Scheduling.

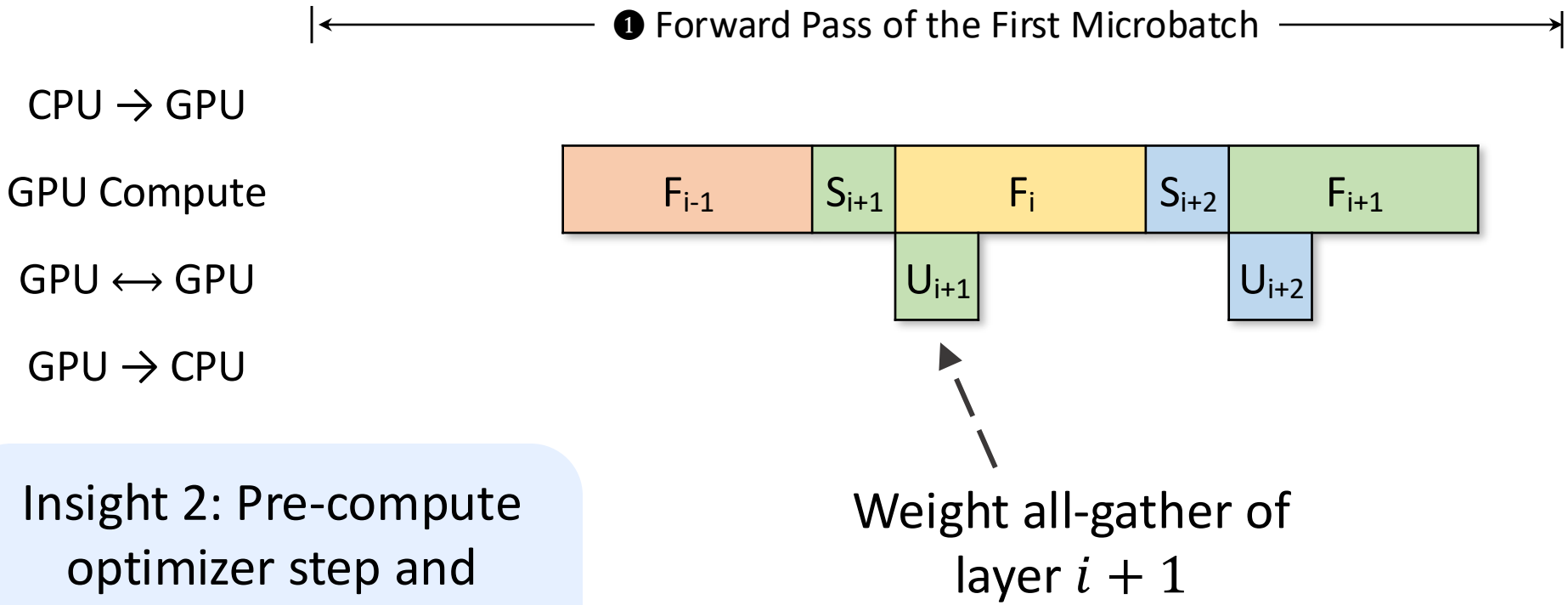


Insight 1: Decompose Optimizer Steps

Mist's Key Ideas

① Symbolic-Based Efficient Performance Prediction.
② Fine-Grained Overlap-Centric Scheduling.
③ Imbalance-Aware Hierarchical Tuning via Pareto Frontier Sampling.

② Fine-Grained Overlap-Centric Scheduling.

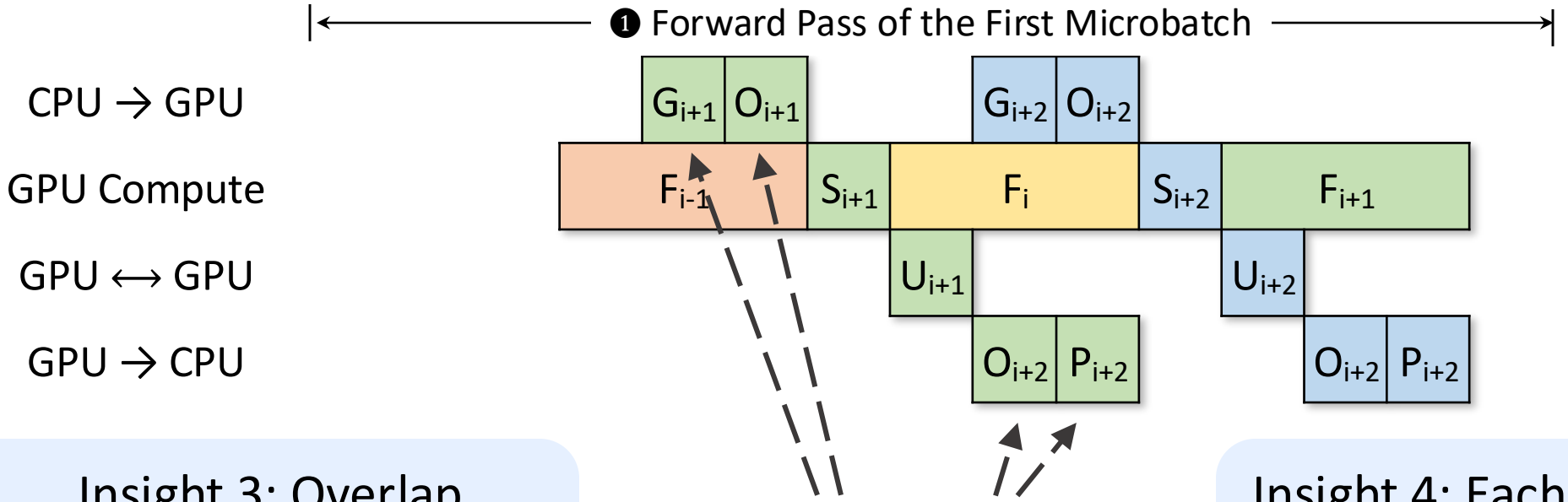


Insight 2: Pre-compute optimizer step and overlap weight all-gather with computation

Mist's Key Ideas

① Symbolic-Based Efficient Performance Prediction.
② Fine-Grained Overlap-Centric Scheduling.
③ Imbalance-Aware Hierarchical Tuning via Pareto Frontier Sampling.

② Fine-Grained Overlap-Centric Scheduling.



Insight 3: Overlap offloading for optimizer step related tensors

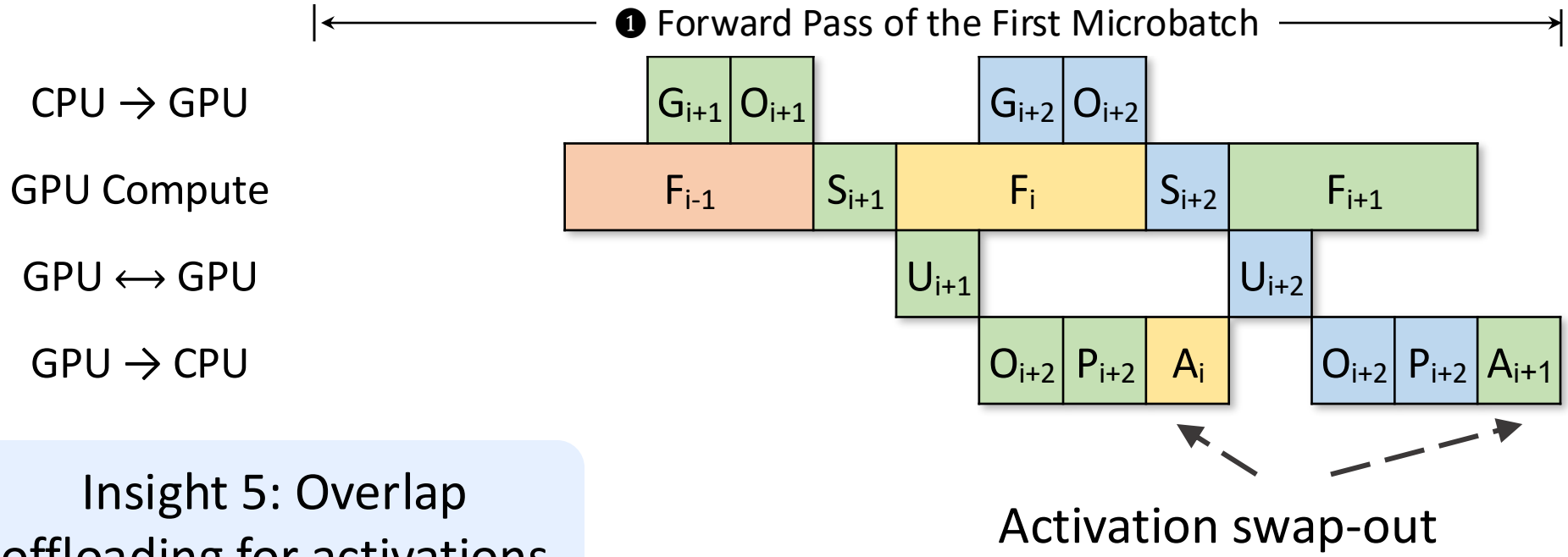
Gradients swap-in
Opt states swap-in
Opt states swap-out
Weights swap-out

Insight 4: Each offloading type can be set with a ratio from 0 to 1.

Mist's Key Ideas

① Symbolic-Based Efficient Performance Prediction.
② Fine-Grained Overlap-Centric Scheduling.
③ Imbalance-Aware Hierarchical Tuning via Pareto Frontier Sampling.

② Fine-Grained Overlap-Centric Scheduling.



Insight 5: Overlap offloading for activations

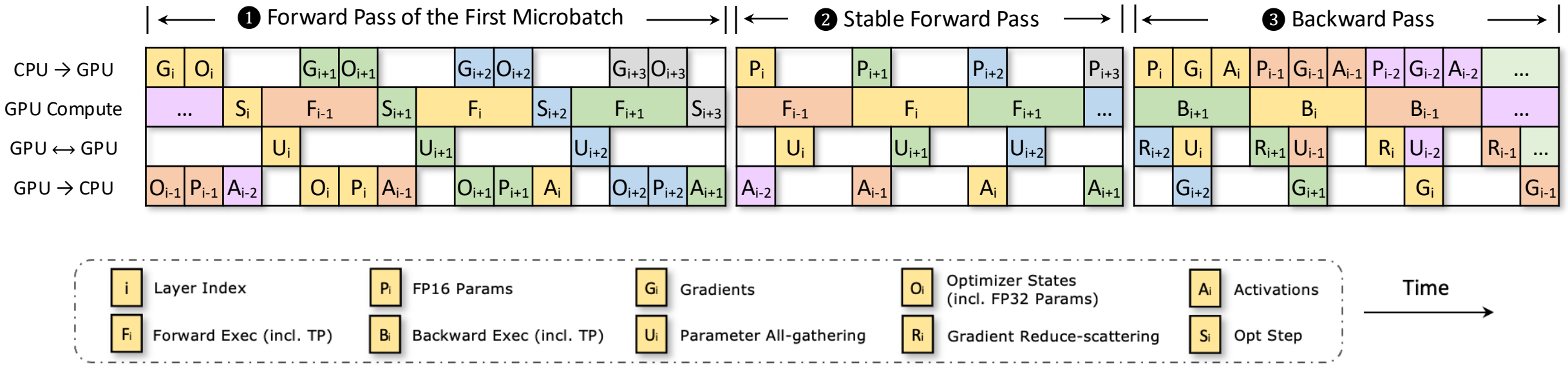
Mist's Key Ideas

① Symbolic-Based Efficient Performance Prediction.

② Fine-Grained Overlap-Centric Scheduling.

③ Imbalance-Aware Hierarchical Tuning via Pareto Frontier Sampling.

② Fine-Grained Overlap-Centric Scheduling.



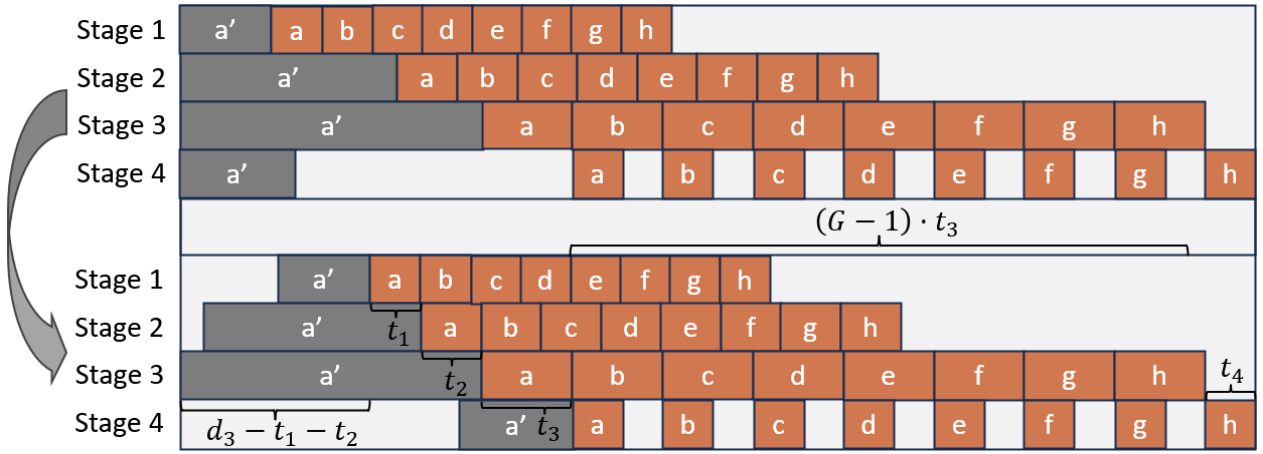
Mist's Key Ideas

① Symbolic-Based Efficient Performance Prediction.
② Fine-Grained Overlap-Centric Scheduling.
③ Imbalance-Aware Hierarchical Tuning via Pareto Frontier Sampling.

③ Imbalance-Aware Hierarchical Tuning via Pareto Frontier Sampling.

- Formulize inter-stage tuning as a MILP problem.

However, t_i and d_i are correlated.



t_i: Time of stable micro batch in stage i
d_i: Time difference between the first microbatch and t_i in stage i

Inter-Stage

$$\min_{\substack{1\le i\le S \\ l_i,f_i,(n_i,m_i)}} \left\{ \underbrace{(G-1)\cdot \max_{1\le i\le S}\{t_i\} + \sum_{i=1}^{S} t_i}_{\text{Normal pipeline time calculation}} + \underbrace{\max_{1\le i\le S}\left(d_i - \sum_{1\le j<i} t_i\right)}_{\text{Consider microbatch differences}} \right\} \quad (2)$$

Intra-Stage

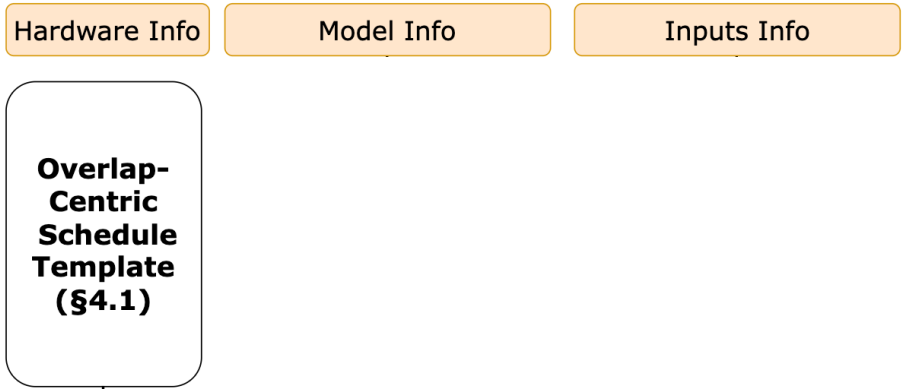
Sample (t_i, d_i) from the pareto frontier of intra stage

$$\min_{p,z,o} \alpha \cdot G \cdot t_{p,z,o} + (1-\alpha)\cdot d_{p,z,o}$$

$$i.e.\ \max\left(Mem_{peak}^{(fwd)}, Mem_{peak}^{(bwd)}\right) \le Mem_{Budget} \quad (4)$$

Mist Overview

An automatic distributed training ***performance analysis*** and ***tuning*** system with ***overlapped execution engine***.



Outline

Background of distributed training and its optimizations

Shortcomings of existing distributed training systems

Key Ideas of Mist to address these shortcomings

Evaluation on state-of-the-art workloads

Methodology

Hardware

NVIDIA L4 GPU

NVIDIA A100 GPU

Platform	GPU	GPU#	Mem.	PCIe Spec	NVLink	Interconnect
GCP	L4	[2, 4, 8, 16, 32]	24GB	Gen3@16x	✗	100Gbps
AWS	A100	[2, 4, 8, 16, 32]	40GB	Gen4@16x	✓	400Gbps

Methodology

Hardware

NVIDIA L4 GPU

NVIDIA A100 GPU

Workloads

GPU	Models	Param# (billion)	Global Batch Size	Seq Len
L4	GPT, Llama, Falcon	[1.3, 2.6, 6.7, 13, 22]	[32, 64, 128, 256, 512]	2048
A100	GPT, Llama, Falcon	[1.3, 2.6, 6.7, 13, 22]	[32, 64, 128, 256, 512]	4096

Methodology

Hardware

NVIDIA L4 GPU

NVIDIA A100 GPU

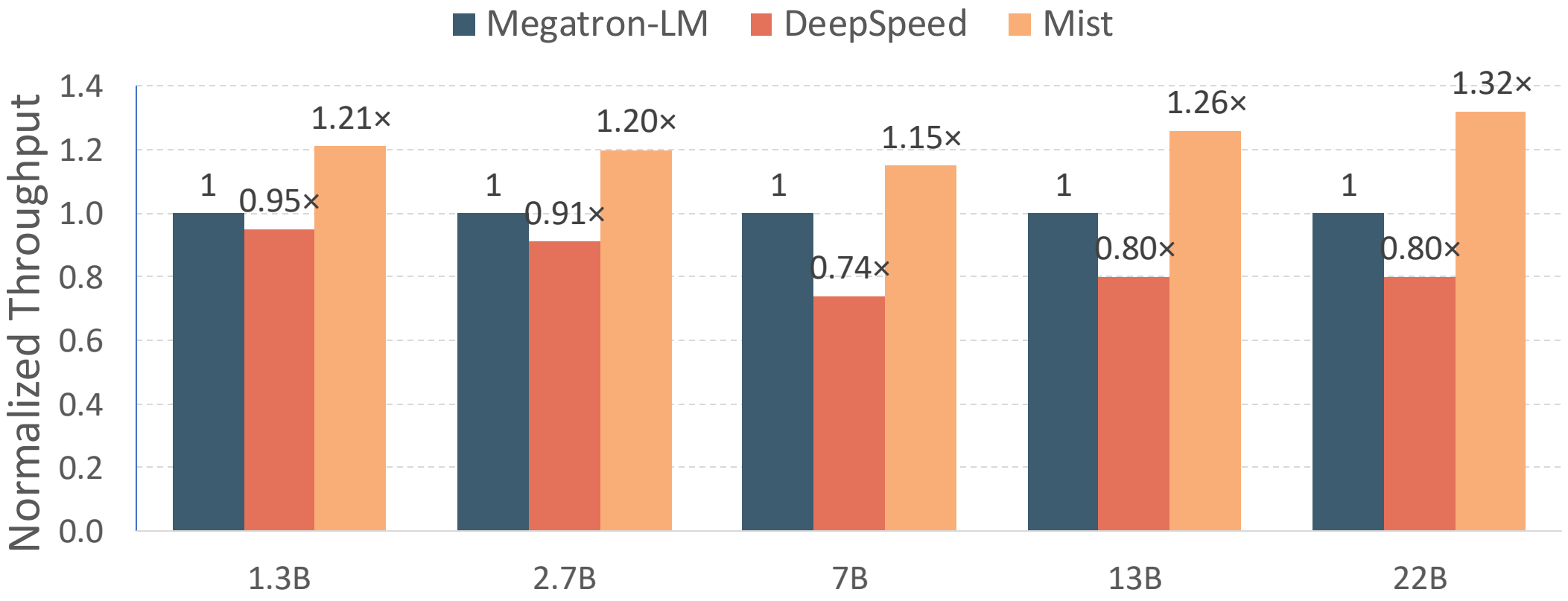
Workloads

GPU	Models	Param# (billion)	Global Batch Size	Seq Len
L4	GPT, Llama, Falcon	[1.3, 2.6, 6.7, 13, 22]	[32, 64, 128, 256, 512]	2048
A100	GPT, Llama, Falcon	[1.3, 2.6, 6.7, 13, 22]	[32, 64, 128, 256, 512]	4096

Baselines

Megatron-LM, DeepSpeed, Aceso

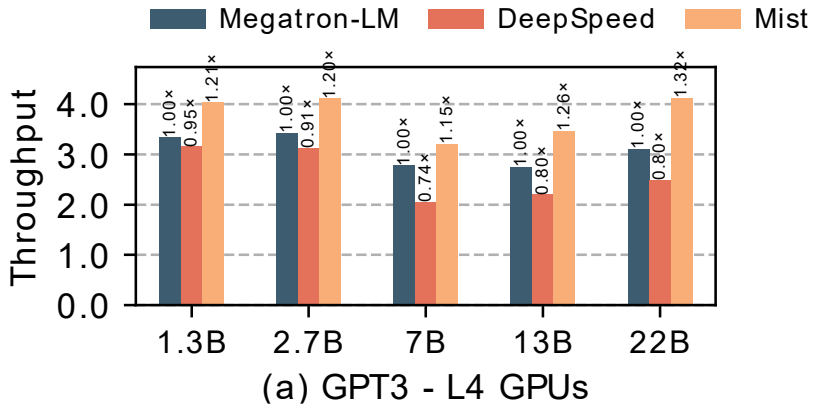
1) Performance Speedup w/ FlashAttn



GPT-3 model on L4 GPUs

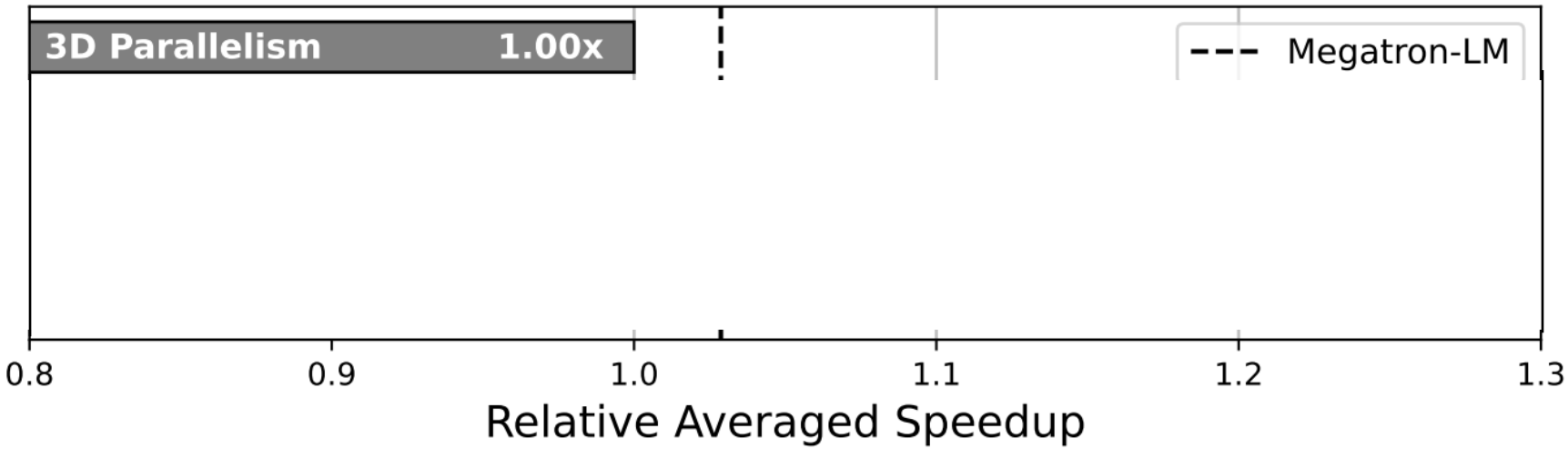
Mist consistently outperform the baselines

1) Performance Speedup w/ FlashAttn



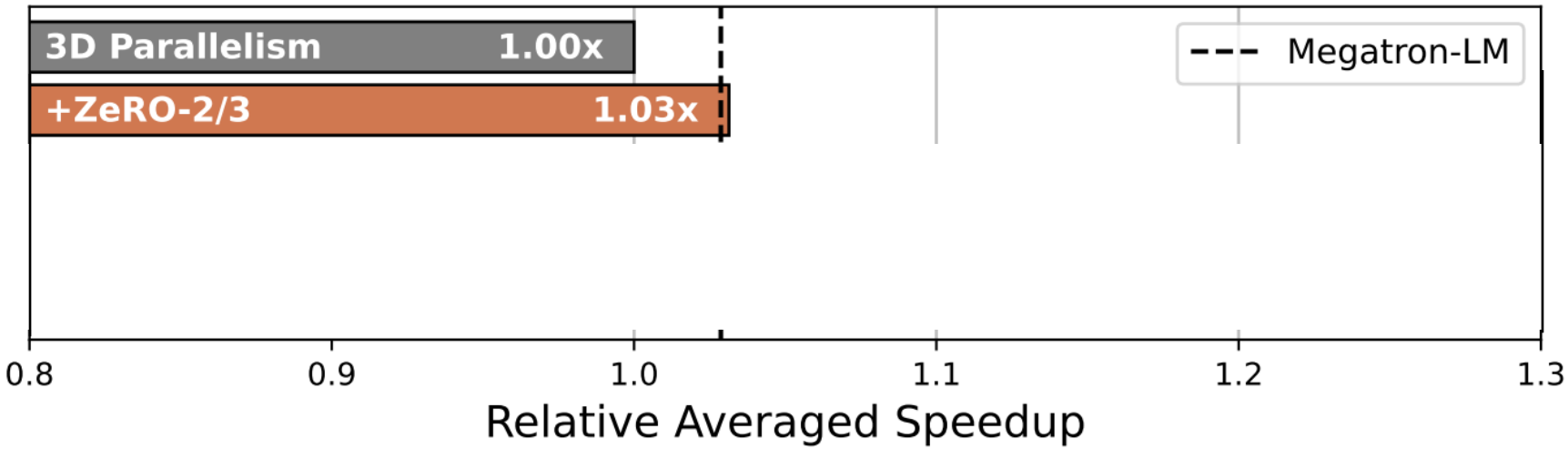
- L4 GPUs: $\mathbf{1.32}\times$ on average (up to $\mathbf{1.59}\times$) over Megatron-LM
- A100 GPUs: $\mathbf{1.34}\times$ on average (up to $\mathbf{1.72}\times$) over Megatron-LM

2) Speedup Breakdown



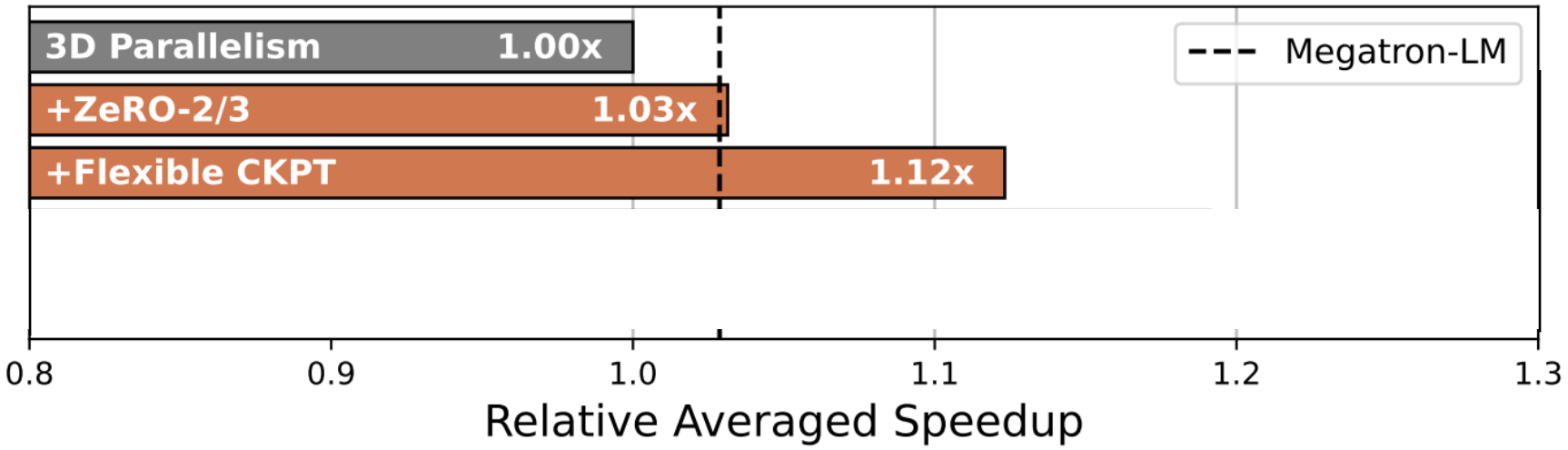
Relative averaged speedup of tuning over different search spaces for GPT model on 8, 16, and 32 L4 GPUs.

2) Speedup Breakdown



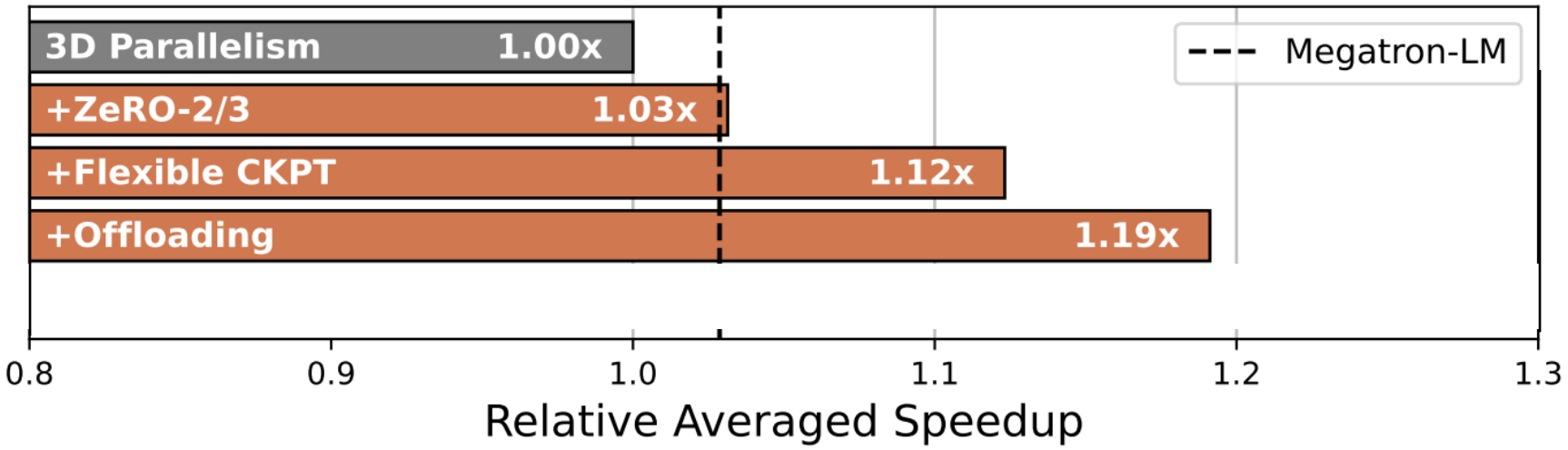
Relative averaged speedup of tuning over different search spaces for GPT model on 8, 16, and 32 L4 GPUs.

2) Speedup Breakdown



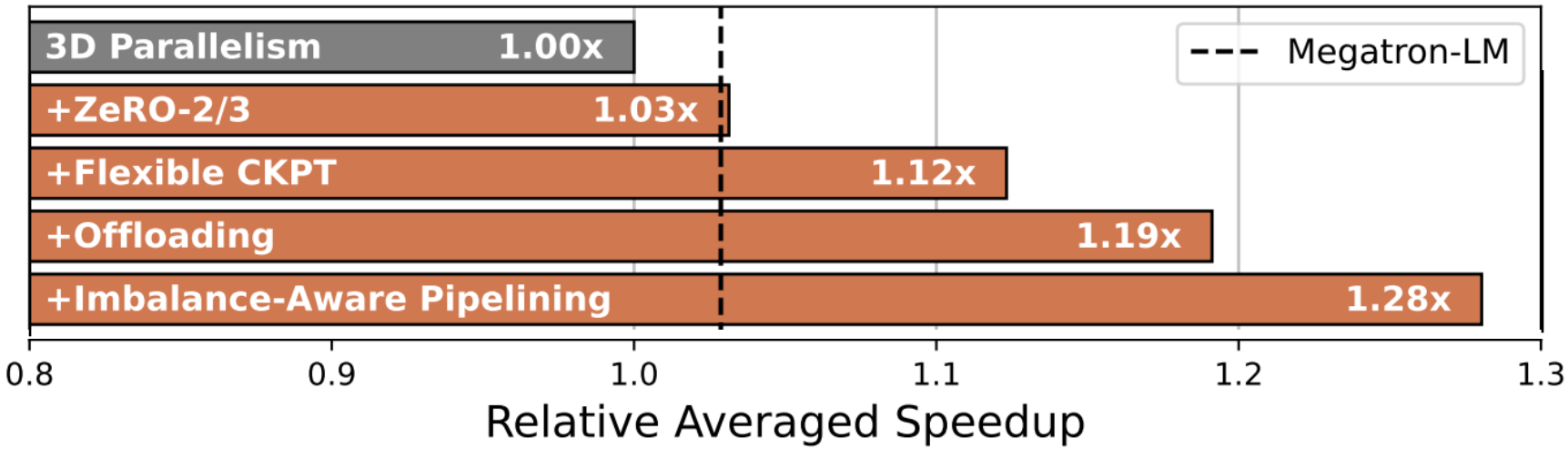
Relative averaged speedup of tuning over different search spaces for GPT model on 8, 16, and 32 L4 GPUs.

2) Speedup Breakdown



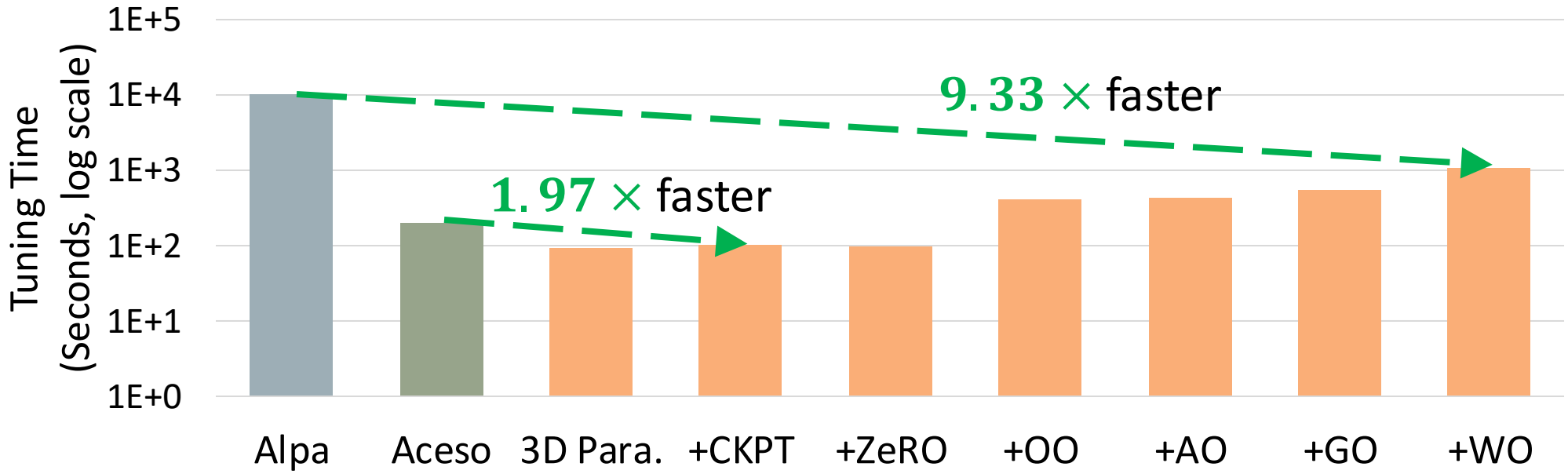
Relative averaged speedup of tuning over different search spaces for GPT model on 8, 16, and 32 L4 GPUs.

2) Speedup Breakdown



Relative averaged speedup of tuning over different search spaces for GPT model on 8, 16, and 32 L4 GPUs.

3) Tuning Time



- Vs. Aceso: with the same search space, Mist is $\mathbf{1.97}\times$ faster.
- Vs. Alpa: even with larger search space, Mist is $\mathbf{9.33}\times$ faster.

Tuning a 22B model on 32 GPUs.

Executive Summary

- Mist accelerates distributed training by co-optimizing *parallelism* and *memory optimizations*.
- Key Challenges
  - **Exploded** and complex configuration search space.
  - **Inaccurate** performance prediction: due to missing overlap and ignoring inter-microbatch imbalance.
- Mist addresses the challenges with
  ① Symbolic-based Performance Analysis
  ② Overlap-Centric Scheduling & Imbalance-aware hierarchical tuning
- Key Result: Up to $\mathbf{1.73}\times$ better vs. Megatron-LM and $\mathbf{2.04}\times$ vs. the automatic method Aceso, respectively.