USENIX
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

OSDI 26

SUN YAT-SEN UNIVERSITY 中山大學

EcoServe：Efficient LLM Serving on Commodity GPU Clusters with Data-Reduced Cross-Instance Orchestration

Jiangsu Du*, Hongbin Zhang*, Taosheng Wei, Zhenyi Zheng, Jiazhi Jiang, Kaiyi Wu, Zhiguang Chen, Yutong Lu

Sun Yat-sen University, Computer Science and Engineering

* Equal Contribution

LLMs Empowering Diverse Fields



Programming



Literary



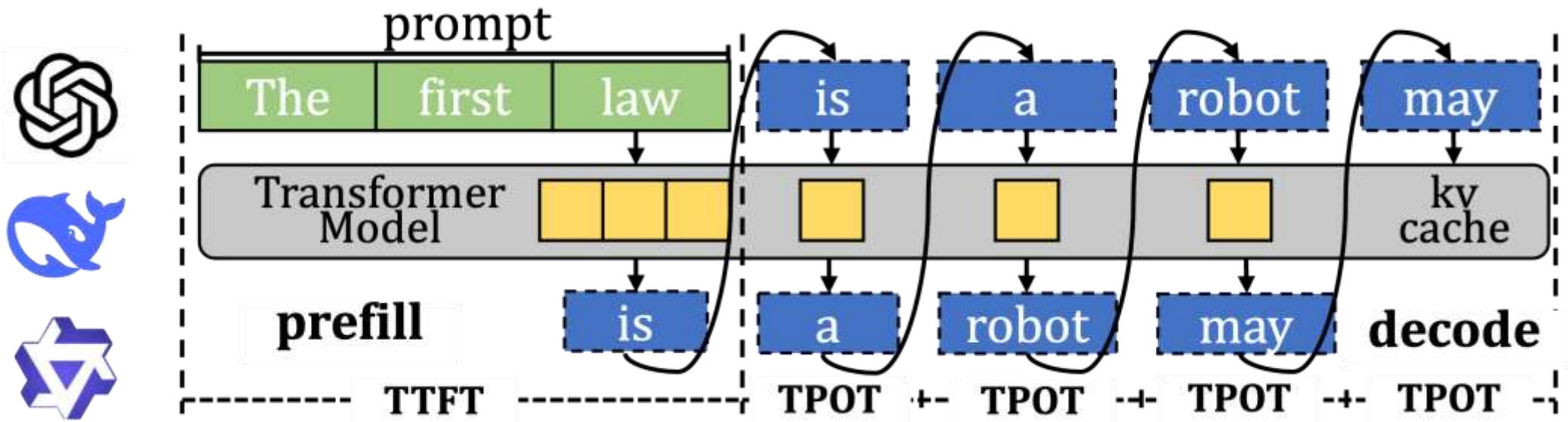
Healthcare



Scientific Research

How do we serve such large models efficiently and affordably?

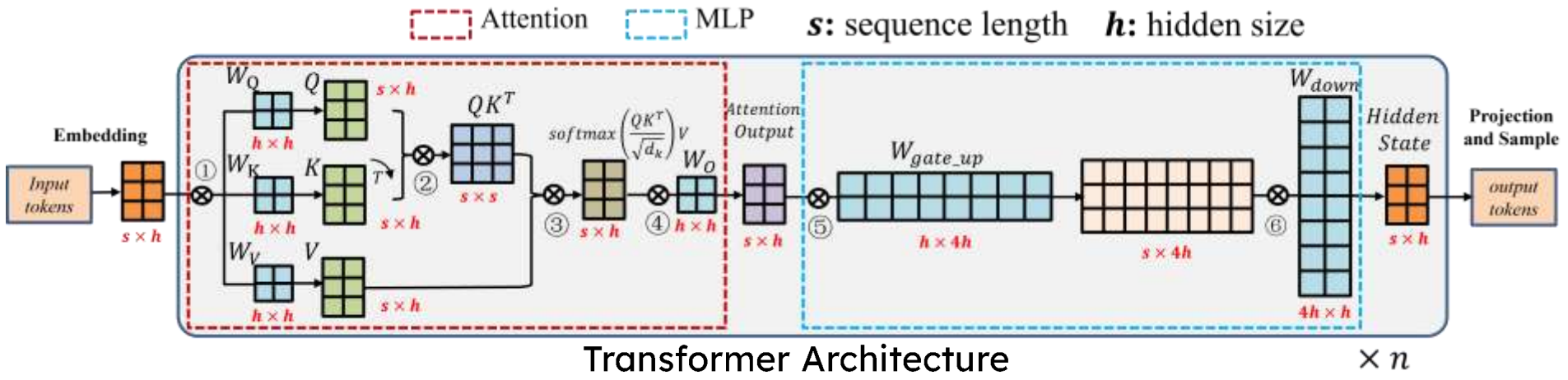
LLM Serving Tasks



Two Latency Metrics:

TTFT - Time to first token for prefill phase
TPOT - Time per output token for decode phase

Computational Intensity in Prefill/Decode

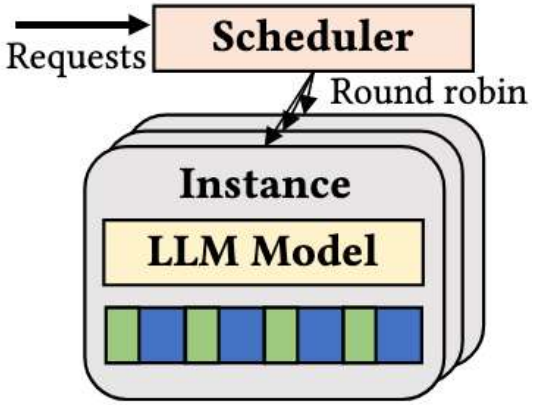


Notation	Description
S	The length of prompt/prefix
B	The number of batched requests
H	Input dimension of the hidden layer

	operation	Prefill			Decode		
		FLOPS	Memory Access	Approx. Intensity	FLOPS	Memory Access	Approx. Intensity
①	QKV Projection	$6BSH^2$	$6SH + 3SH^2$	**BS**	$6BH^2$	$6BH + 3H^2$	**B**
②	Attention QK	$2BS^2H$	$2BSH + BS^2H$	**S**	$2BS^2H$	$2BSH+BH(S+1)$	**1**
③	Attention (QK)V	$2BS^2H$	$2BSH + BS^2H$	**S**	$2BS^2H$	$2BSH+BH(S+1)$	**1**
④	Output Projection	$2BSH^2$	$2BSH + H^2$	**BS**	$8BH^2$	$2BH + H^2$	**B**
⑤	Dim Expansion	$8BSH^2$	$2BSH + 4H^2$	**BS**	$8BH^2$	$2BH + 4H^2$	**B**
⑥	Dim Reduction	$8BSH^2$	$2BSH + 4H^2$	**BS**	$8BH^2$	$2BH + 4H^2$	**B**

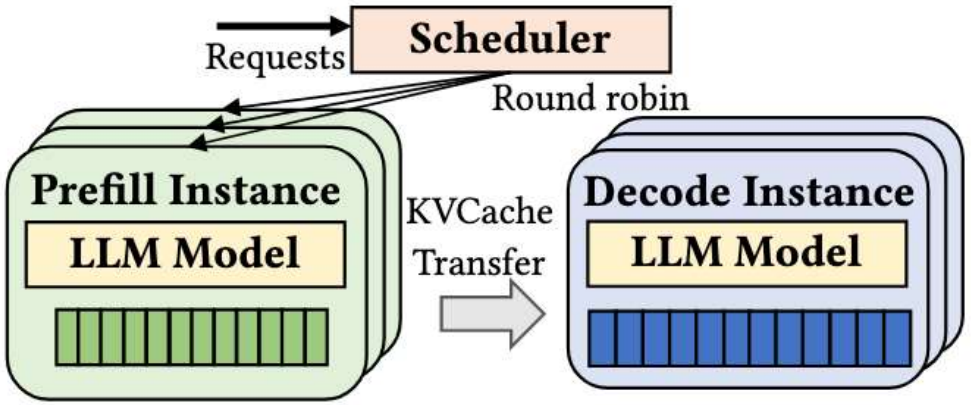
Prefill is compute-bound and decode is memory-bound.
Decode stage requires a large batch size to achieve high arithmetic intensity.

Existing Cluster-level LLM Serving System



(a) NoDG Strategy

- Non-Disaggregated (NoDG) Strategy
- Prefill and decode share the same instance.

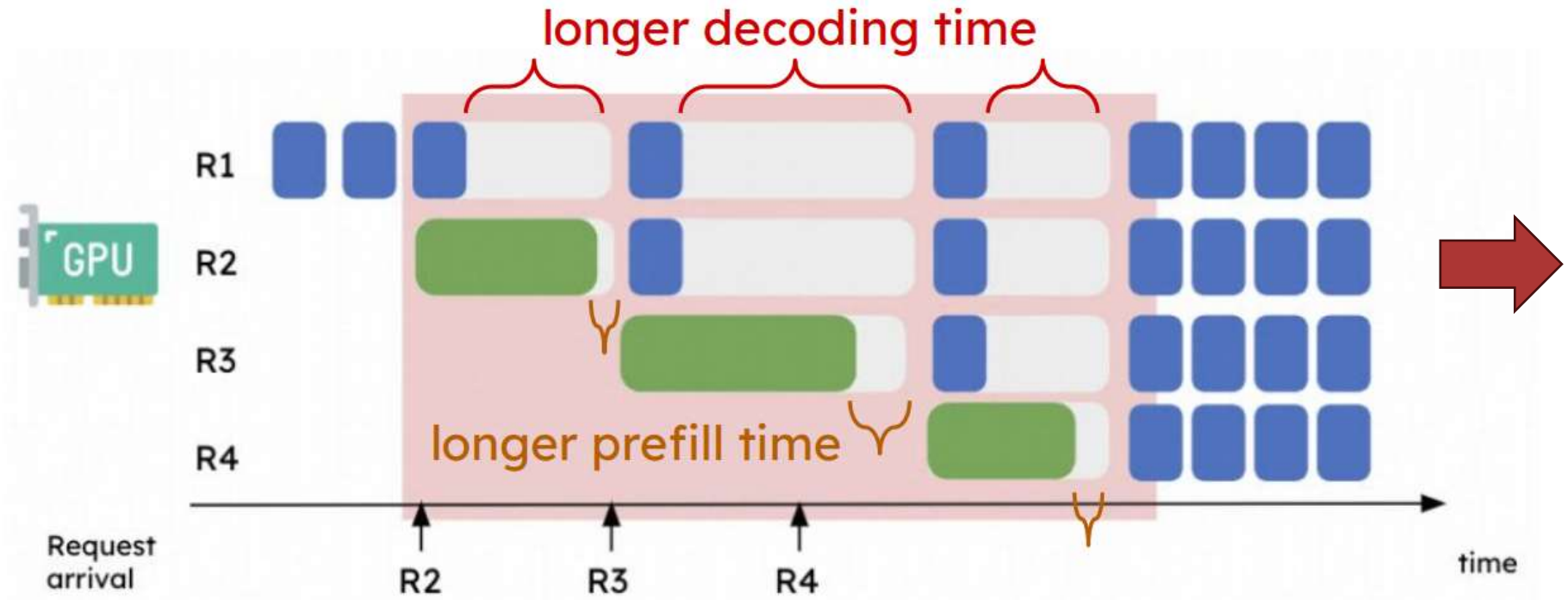


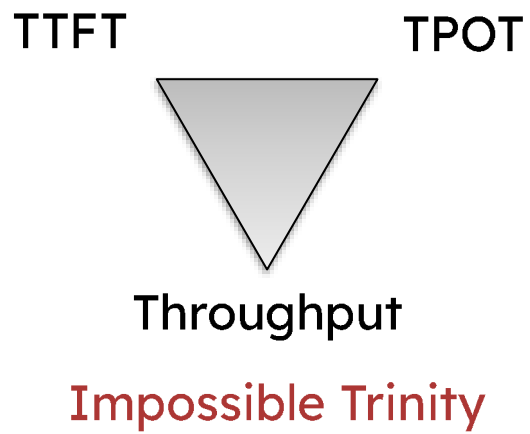
(b) FuDG Strategy

- Fully-Disaggregated (FuDG) Strategy
- Prefill and decode are executed on separate instances, and the KV cache is transferred between them.

NoDG: Prefill-Decoding Interference

Batching (for Throughput) prefill and decoding phases together hurt both TTFT and TPOT

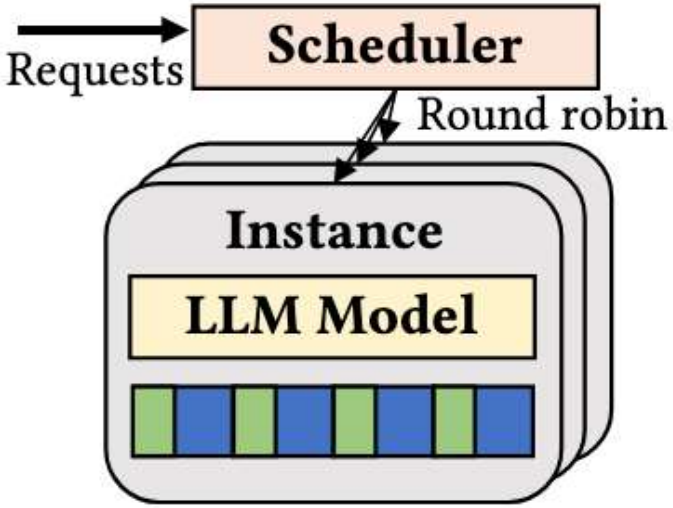




https://www.usenix.org/system/files/osdi24_slides-zhong-yinmin.pdf

NoDG Strategy

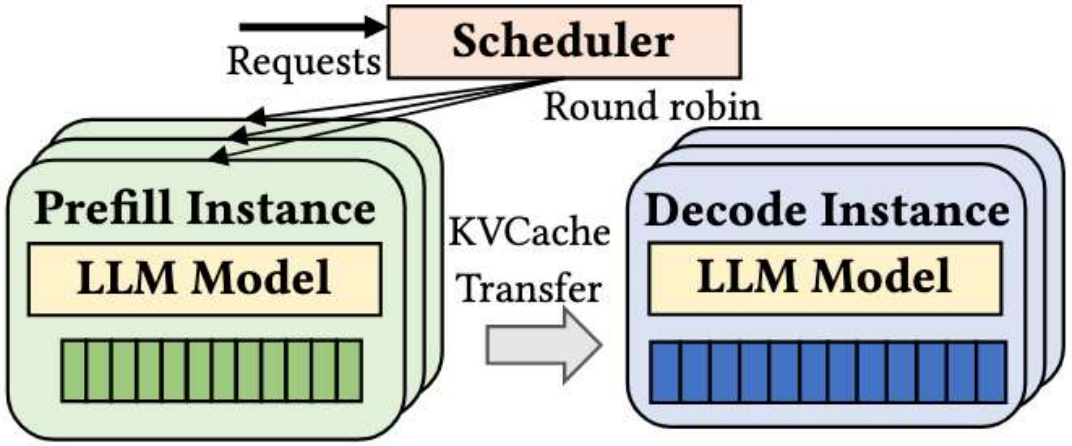
- Prefill and decode run in the same instance.
- Advantages:
  - Simple design, no KV-cache transfer.
  - Easy scaling, by replication.
- Disadvantages:
  - Prefill and decode interfere with each other, leading to low goodput.



(a) NoDG Strategy

FuDG Strategy

- Prefill and decode are executed on separate instances, and the KV cache is transferred between them.
- Advantages:
  - Flexible. Completely eliminates prefill–decode interference.
- Disadvantages:
  - Requires massive KV-cache transfer between instances. High-end interconnects.
  - Prefill and Decode Load imbalance.
  - Expensive and Complex in both hardware and engineering.

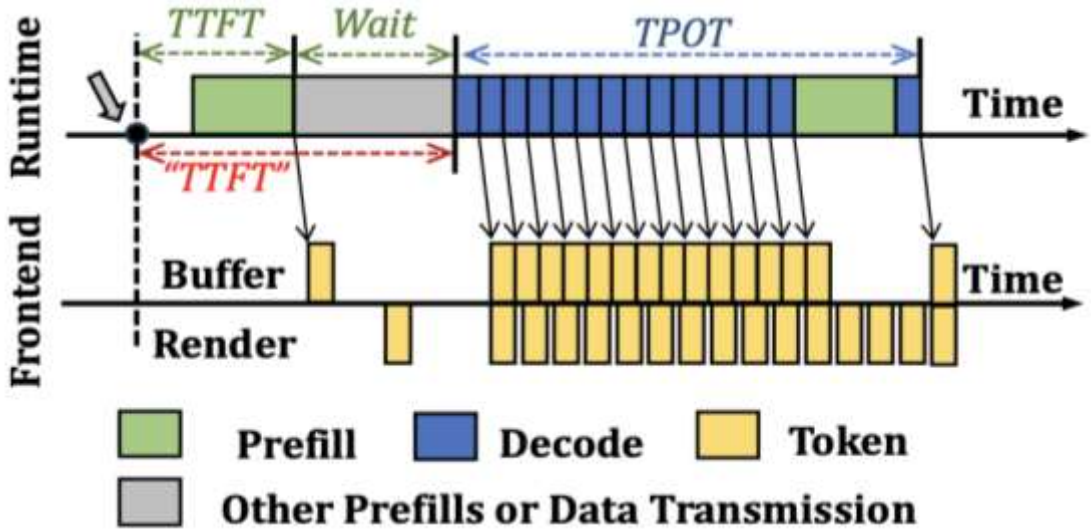


Model	Device	Tokens/s	Data volume
Llama-30B	L20	6584.6	9.796 GB/s
Llama-30B	A800	26189.2	38.96 GB/s
CodeLlama-34B	L20	6838.92	1.25 GB/s
CodeLlama-34B	A800	25978.88	4.76 GB/s

Can we achieve higher goodput and keep the engineering simple?
At the same time, better to fit commodity hardware.

Key Obversion

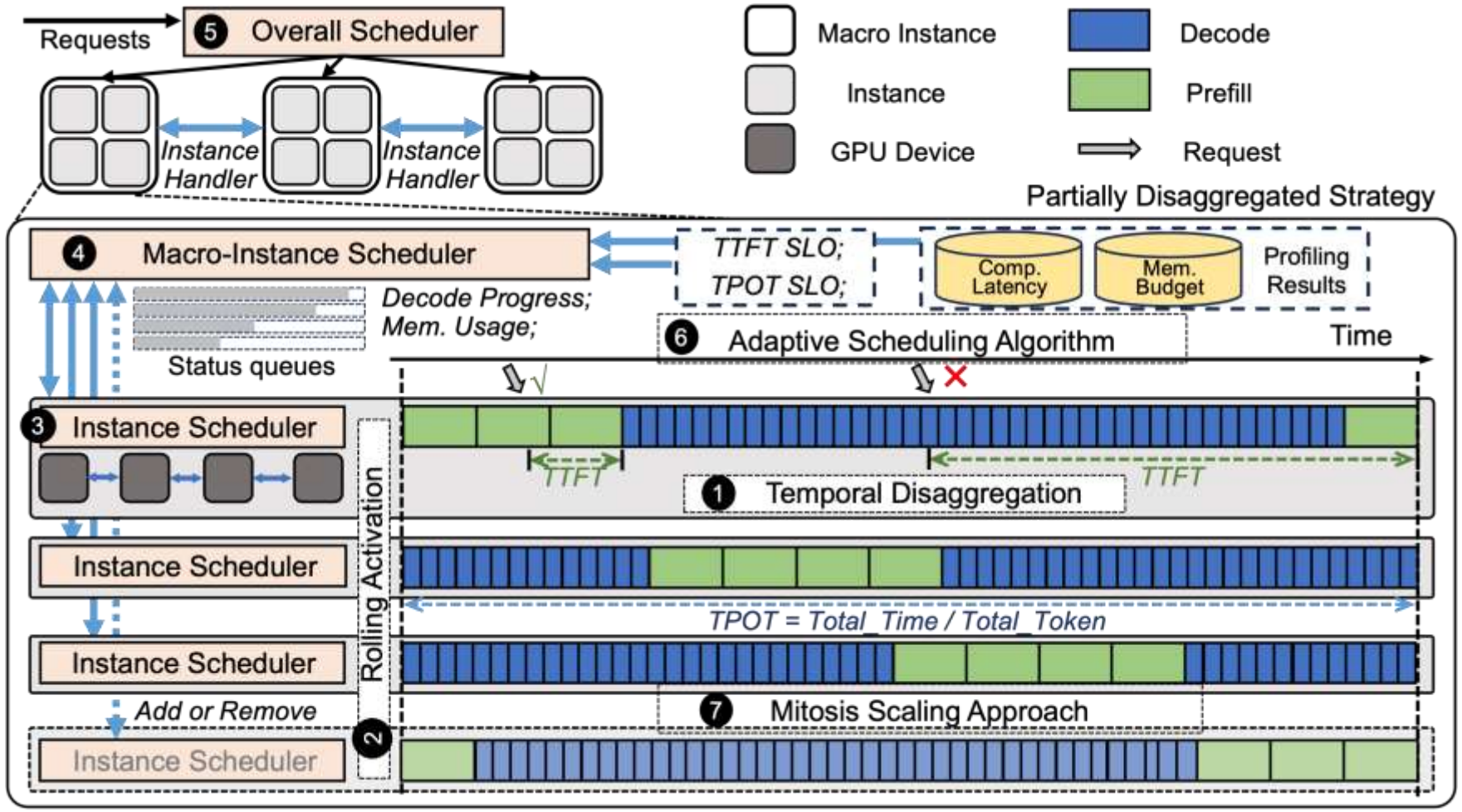
- Inter-Instance Cooperation is Essential to Mitigate Interference.



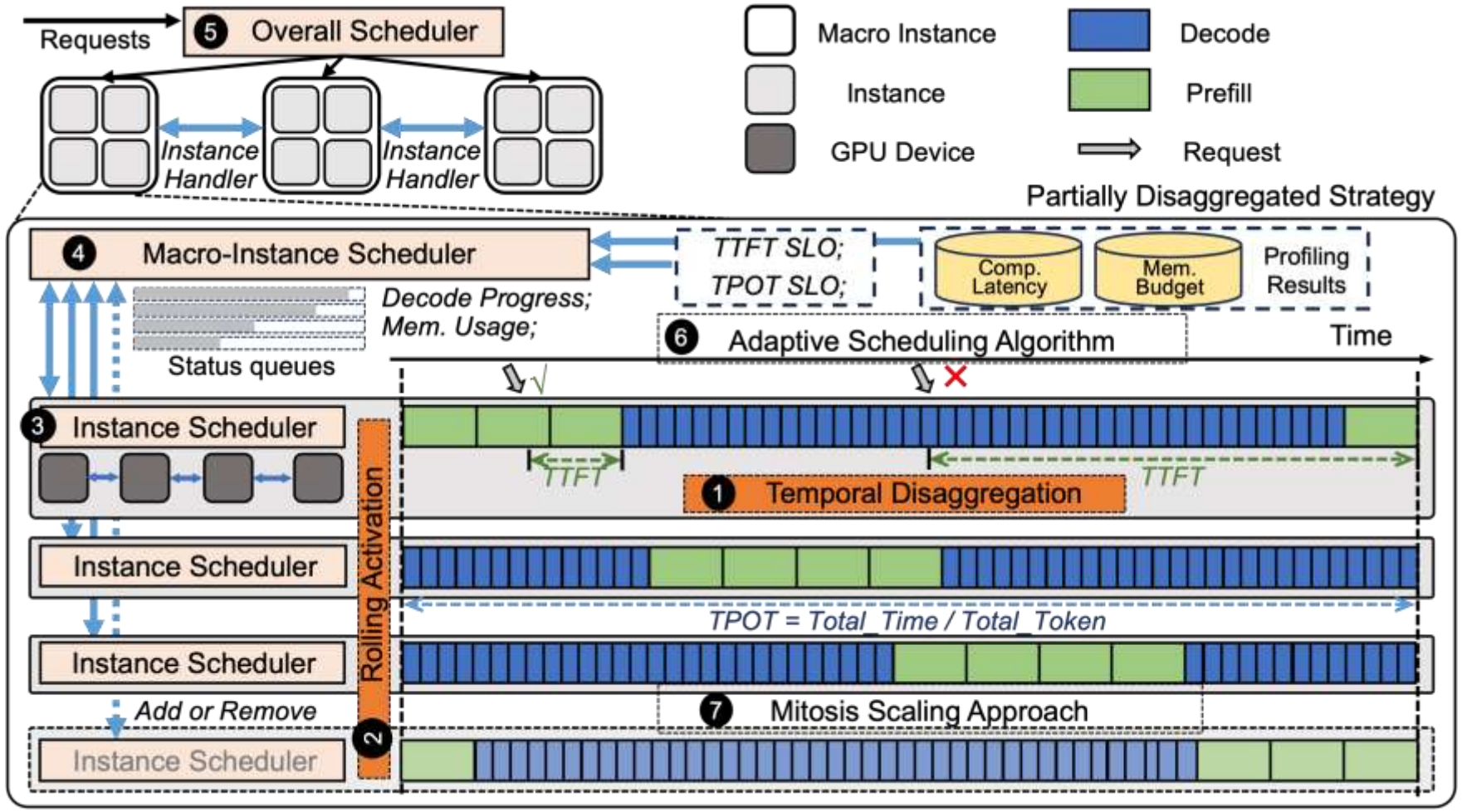
- Runtime Execution can be Flexible
  - Each token's rendering time is important for user experience.
  - Render tokens within TTFT and TPOT SLOs
    - User experience is guaranteed.

- Each instance can:
  - Keep performing prefills until the TPOT limit is reached.
  - Or
  - Keep performing decodes until there are enough tokens wait to be rendered

EcoServe System

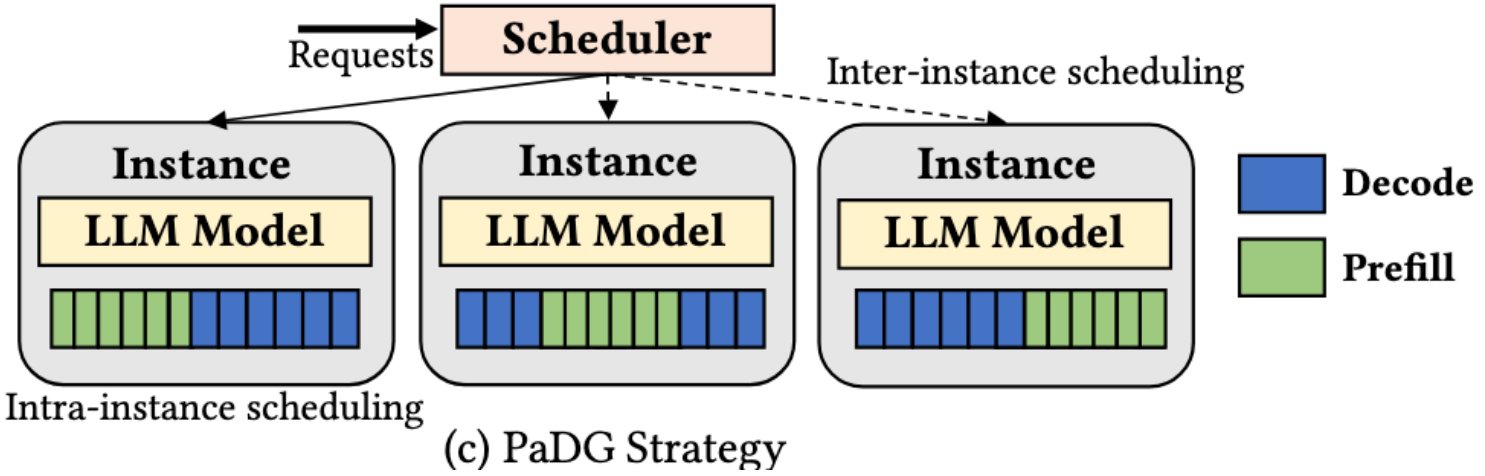


EcoServe: Partially Disaggregated Strategy



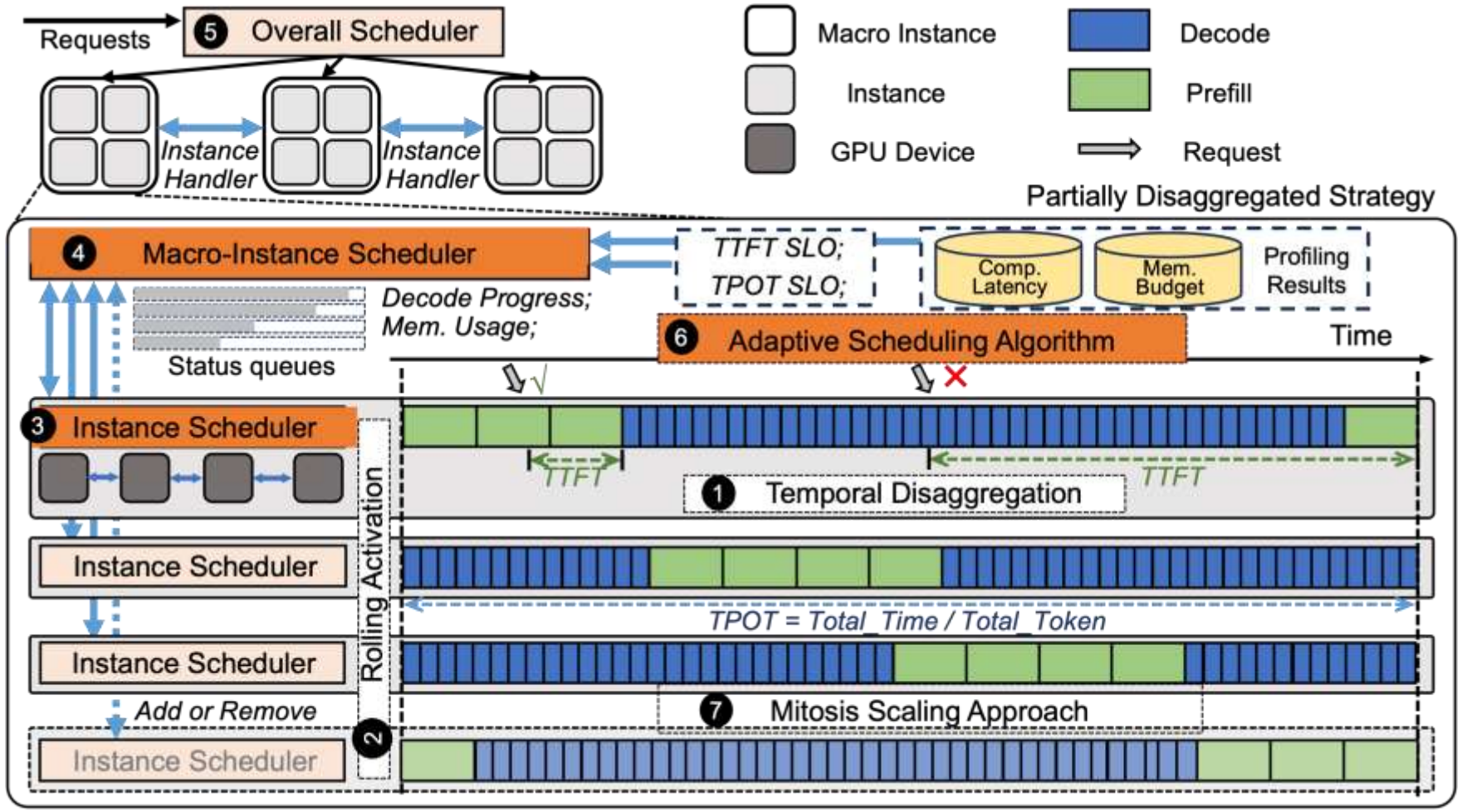
EcoServe: Partially Disaggregated Strategy

- Temporal Disaggregation
  - Disaggregate Prefill and Decode along the Temporal Dimension.
  - Form larger batch size to improve intensity.
    - Perform more Prefill within SLO limits to reach larger batches.
  - Achieve higher throughput without TTFT/TPOT interference.
- Rolling Activation
  - Route new requests to one instance that is in prefill phase.
  - Other instances continue Decode execution.
  - Always one instance ready for new requests.



(c) PaDG Strategy

Macro-Instance and Instance Cooperation



Macro-Instance and Instance Cooperation

- Instance
  - A full-model execution unit.
  - Conceptually similar to an independent vLLM/SGLang instance.
  - Adopts a Prefill-first strategy.
- Macro-Instance
  - A group of instances working together, managed by a macro-instance scheduler.
  - Routes new Prefill requests to the instance in prefill phase.
  - Employs a constraint checking algorithm to ensure SLO compliance.
- Cooperation
  - Macro-Instance receives new requests, check constraints, update active instance and route requests.
  - Instance continues Decode execution, and performs Prefill when new requests are inserted.

Adaptive Scheduling Algorithm

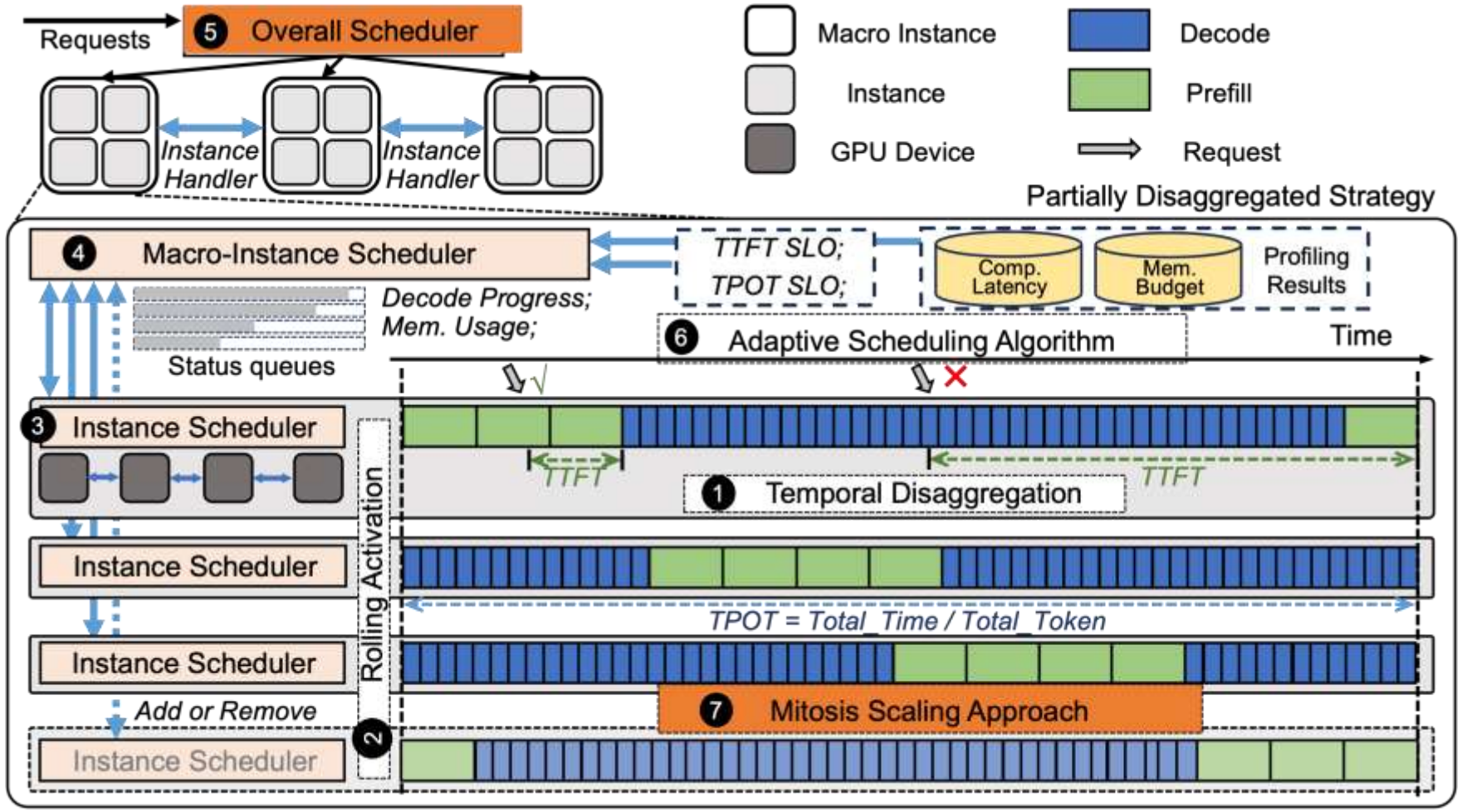
- Profiling
- Check TTFT constraint
  - Ensure that there are enough prefill time for the new request.
- Check TPOT constraint
  - Verify that Decode progress remains within the per-token latency (TPOT) requirement.
- Check memory capacity
  - Confirm sufficient GPU memory is available before admitting new Prefill requests.
- If all constraints are satisfied,
  - route the request to the current active instance;
- Otherwise,
  - activate the next instance.

Algorithm 1: Constraint Checking Algorithm

Data: Latency constraints: S_{TTFT}, S_{TPOT}
Input: I: Statuses of all instances in a macro instance;
r: A new request

Function CheckConstraints(I, r):
  // 1. Check TTFT constraint
  $t_{switch} \leftarrow$ last phase switching timestamp;
  $pending_prefills \leftarrow \{ r \in instance.reqs \mid r.arrival_time \geq t_{switch} \} \cup \{ req \}$;
  $t_{total} \leftarrow \sum(pending_prefills$ durations);
  if $t_{total} > S_{TTFT}$ **then**
    return NotSatisfied;
  // 2. Check TPOT constraint
  $existed_decodes \leftarrow \{ r \in instance.reqs \mid r.arrival_time < t_{switch} \}$;
  $saved_tpots \leftarrow [\,]$;
  $current_time \leftarrow$ current timestamp;
  foreach $r \in existed_decodes$ **do**
    $L \leftarrow r.output_length$;
    $saved_tpot \leftarrow L \times S_{TPOT} - (current_time - r.first_token_time)$;
    append $saved_tpot$ to $saved_tpots$;
  $mean_saved_tpot \leftarrow mean(saved_tpots)$;
  if $mean_saved_tpot < t_{total}$ **then**
    return NotSatisfied;
  // 3. Check memory capacity
  if $req_kvcache_size > remain_memsize$ **then**
    return NotSatisfied;
  return Satisfied;

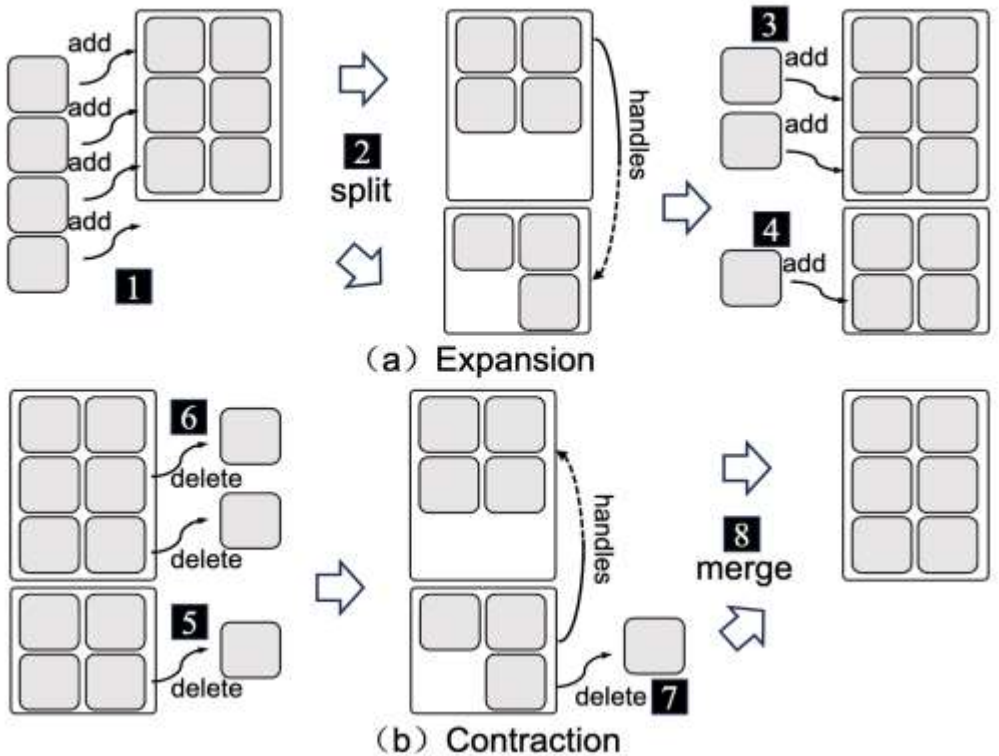
Scaling Approach



Scaling Approach

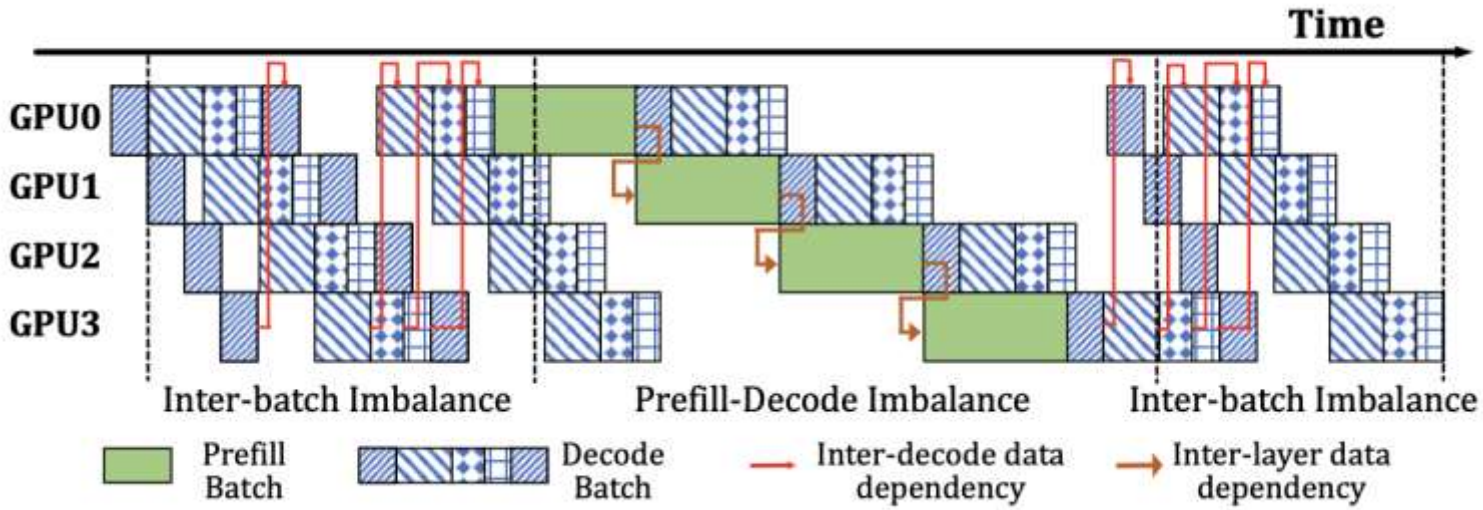
- Mitosis Scaling Approach
  - Enabling elastic and fine-grained scaling.
  - First, add/delete instance to/from a macro instance.
  - Second, divide/merge macro instances
  - Avoids service interruption during scaling by transferring instance handles between macro instances.

- Overall Scheduler
  - Globally coordinates macro-instances to balance load and meet SLOs.
  - Manage instance handlers between macro instances, avoid instance initialization and service interruption.



Better parallelism compatibility

- Pipeline Stage Load imbalance in NoDG
  - Execution gap: Prefill and Decode differ greatly in duration.
  - Frequent phase switching create bubbles and idle time.
- EcoServe: Temporally disaggregates Prefill and Decode.
  - Significantly reduces switching frequency.
  - Alleviates load imbalance.



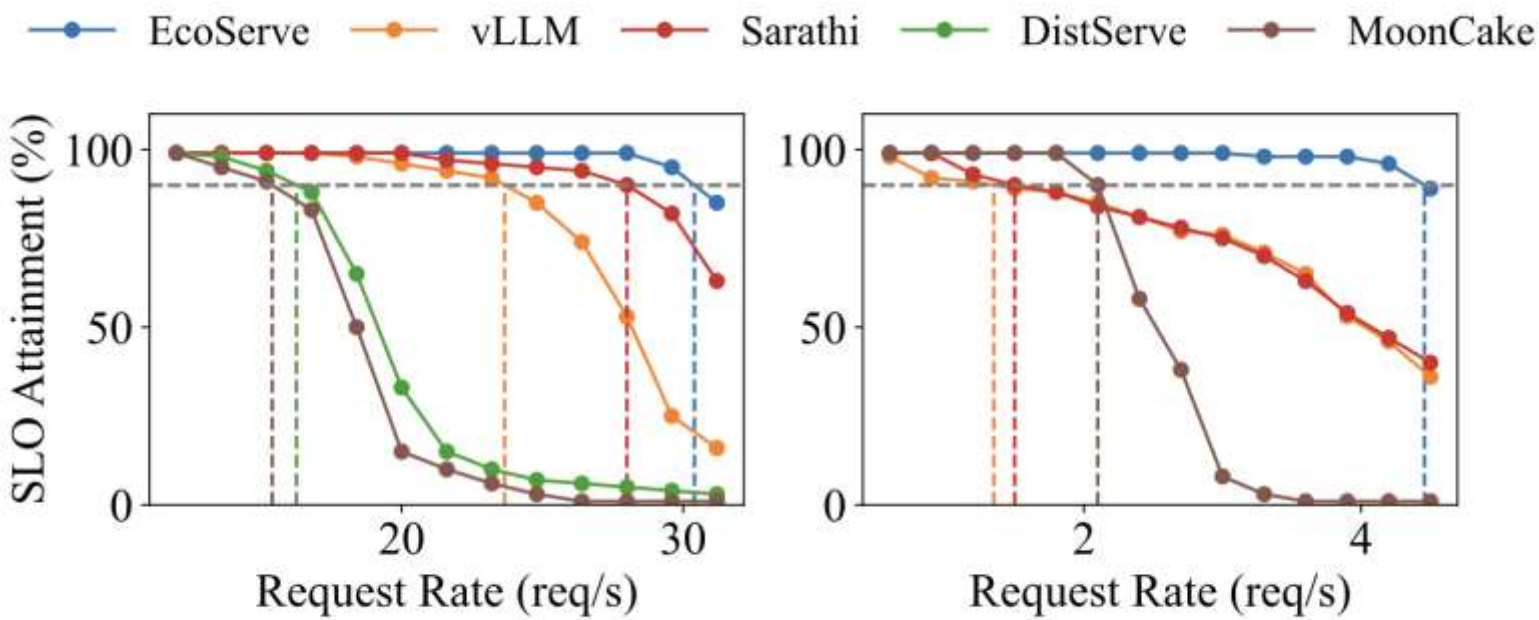
Evaluations

- **Setup**
  - L20: 8 nodes × 8 NVIDIA L20-48GB (PCIe), 64 GPUs; 10 Gbps Ethernet.
  - A800: 2 nodes × 8 NVIDIA A800-80GB (PCIe), 16 GPUs; 25 Gbps RoCE.
  - H100: 2 nodes × 8 NVIDIA H100-80GB, 16 GPUs; NVLink intra-node and 8 × 400 Gbps IB inter-node.
- **Datasets and SLO Settings**

DataSet	In_{Avg}	In_{Med}	Out_{Avg}	Out_{Med}	SLO_{TTFT}	SLO_{TPOT}
Alpaca-gpt4	20.63	17.00	163.80	119.00	1s	100ms
ShareGPT	343.76	148.00	237.20	152	5s	100ms
LongBench	2686.89	2736.50	101.78	19	15s	100ms

- **Baseline:**
  - **NoDG:** vLLM – 0.7.3, Sarathi (Chunked Prefill),
  - **FuDG:** DistServe, MoonCake - 0.3.2

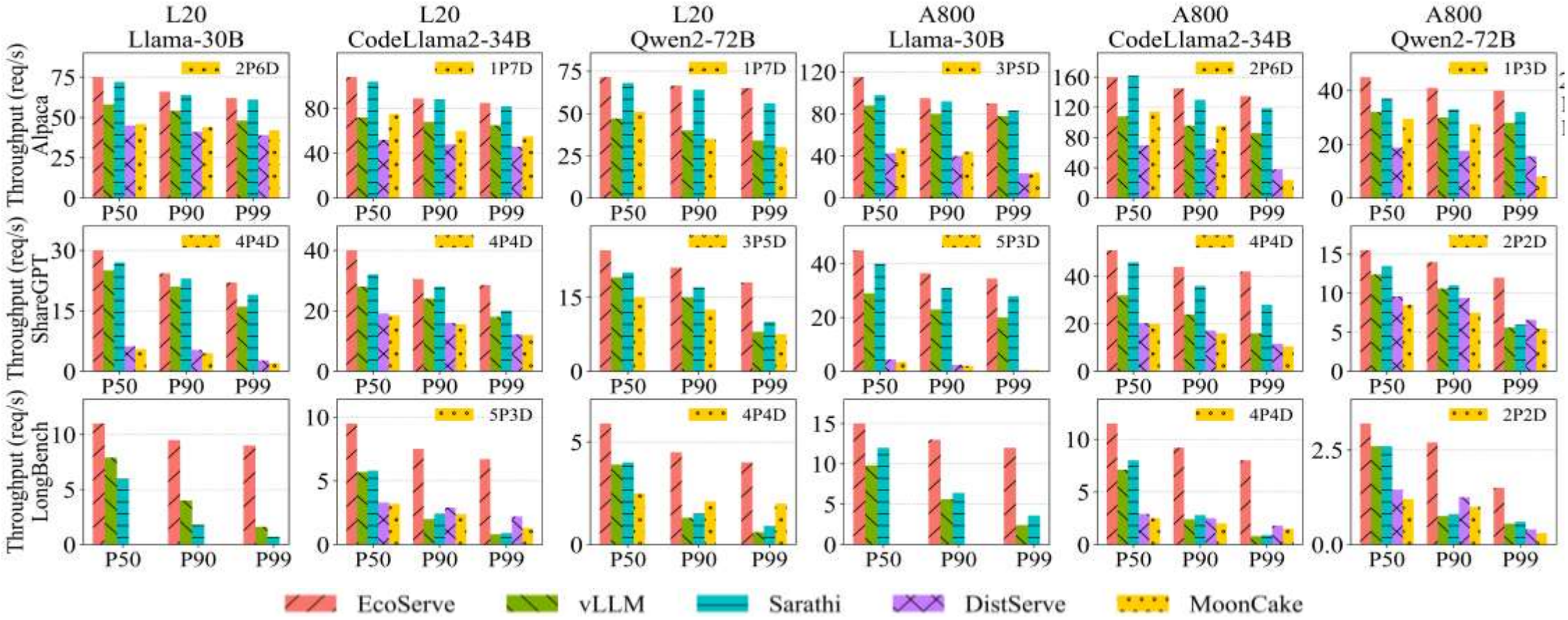
SLO Attainment



L20 Node

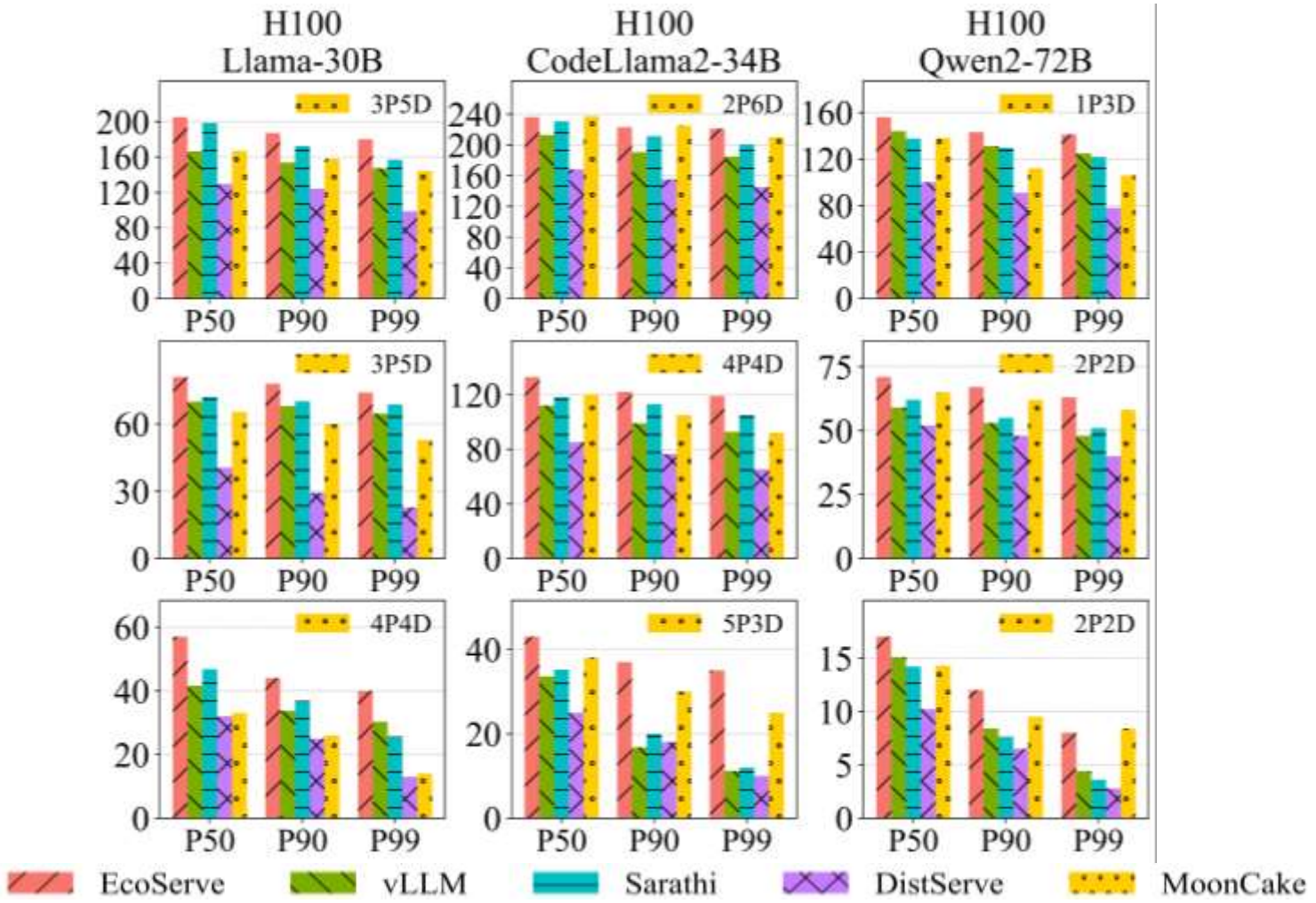
Same request rate, higher SLO attainment.

Goodput



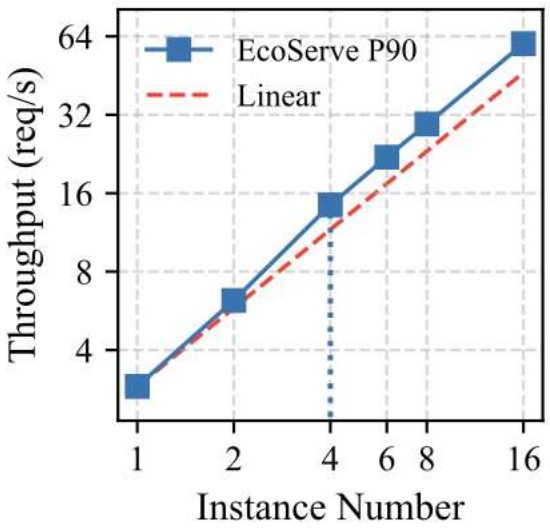
Same SLO attainment, 2x to 2.5x higher goodput across L20 and A800

Goodput

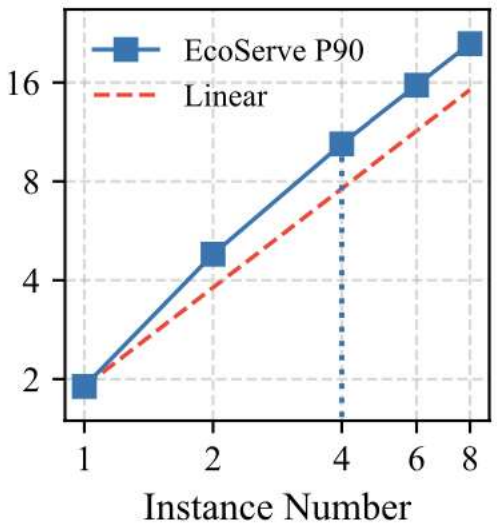


Same SLO attainment, higher goodput on H100.

Static coarse-grained scaling

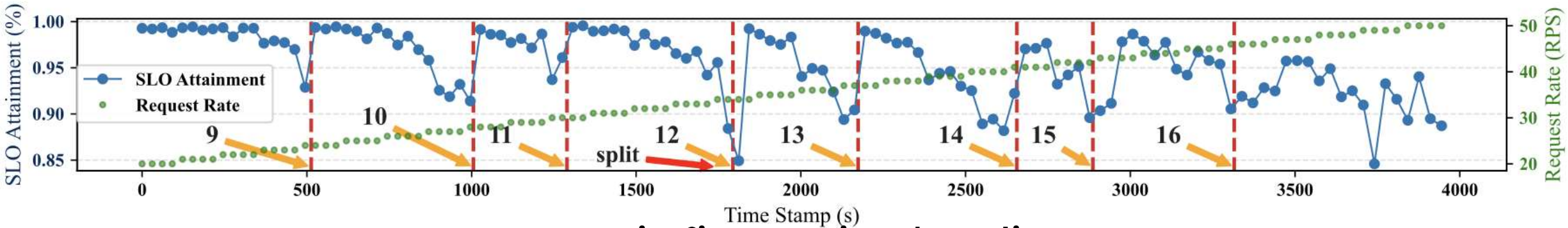


(a)CodeLlama2-34B L20 ShareGPT



(b)Qwen2-72B L20 ShareGPT

Static coarse-grained scaling
Superlinear scaling achieved.



Dynamic fine-grained scaling

Parallel Compatibility.

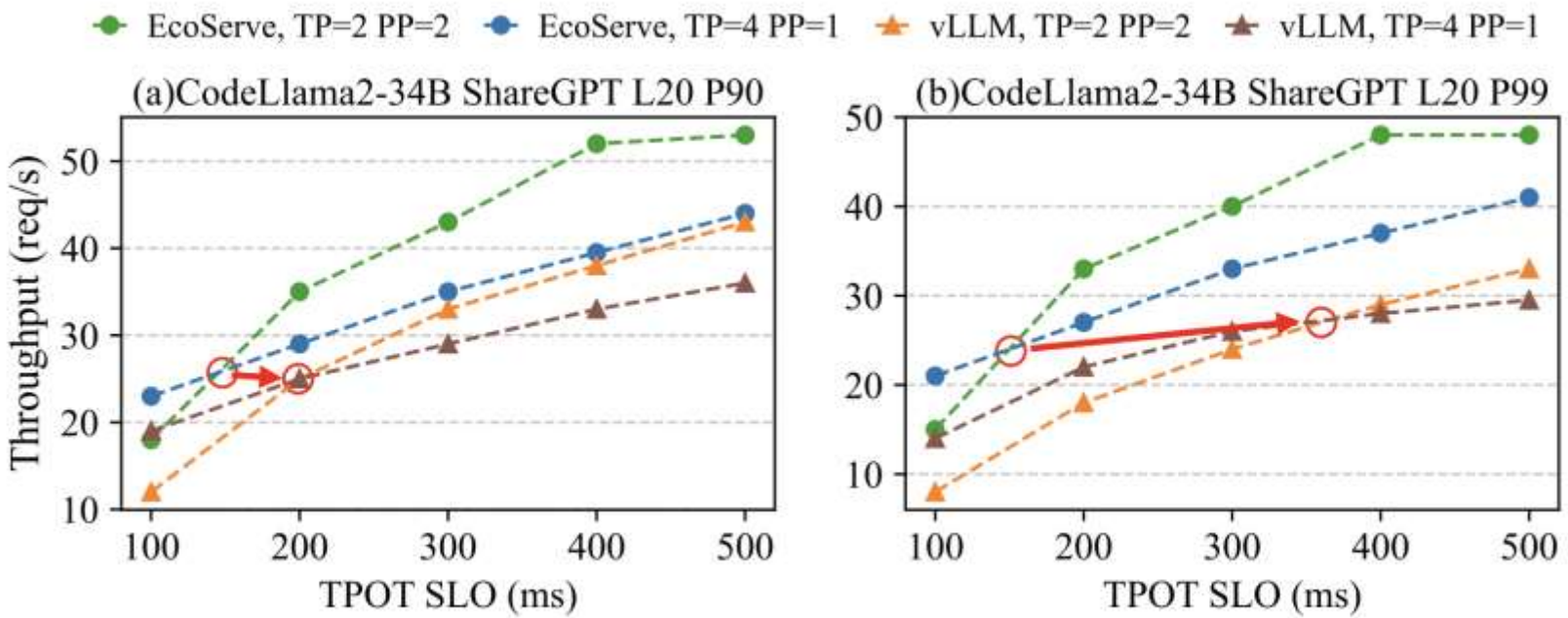
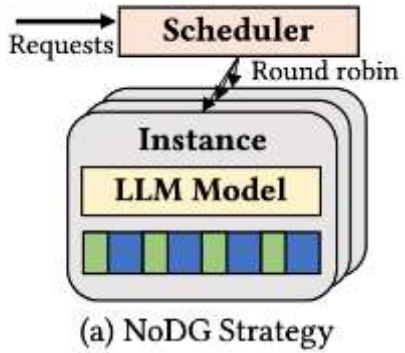
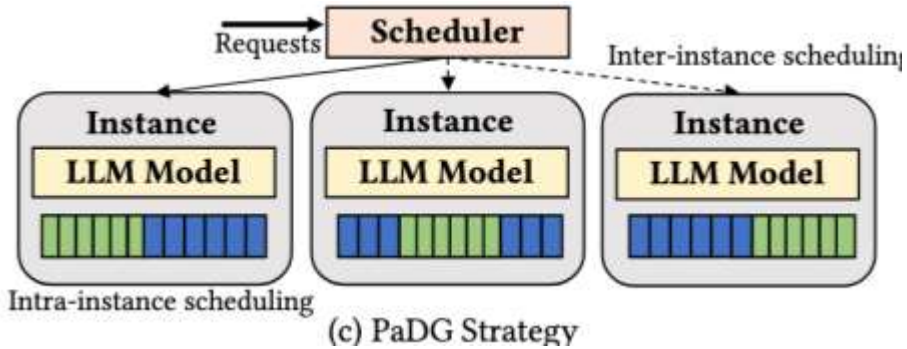


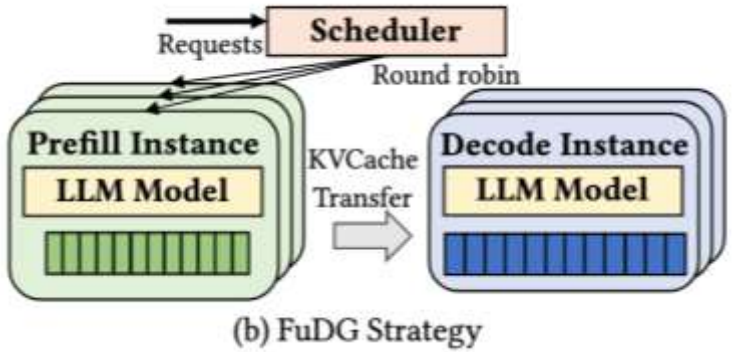
Figure 14: Pipeline parallel compatibility.

When the TPOT constraint is relaxed, compared to vLLM, EcoServe's pure pipeline parallelism throughput exceeds that of hybrid parallelism setup earlier.

No one-size-fits-all solution







Small model
Relaxed SLOs
Commodity Hardware

Medium model
Relaxed SLOs
Commodity Hardware

Large model
Tight SLOs
Advanced Hardware

Open Source

https://github.com/MLSysU/EcoServe

Thanks for listening