LoongServe: Efficiently **Serving Long**-Context Large Language Models with **Elastic Sequence Parallelism**

Bingyang Wu[1], Shengyu Liu[1], Yinmin Zhong[1],

Peng Sun[2], Xuanzhe Liu[1], Xin Jin[1]

[1]Peking University, [2]Shanghai AI Lab

Presenter: Zewen Jin, Hongrui Zhan, Shen Fu @ USTC

Outline

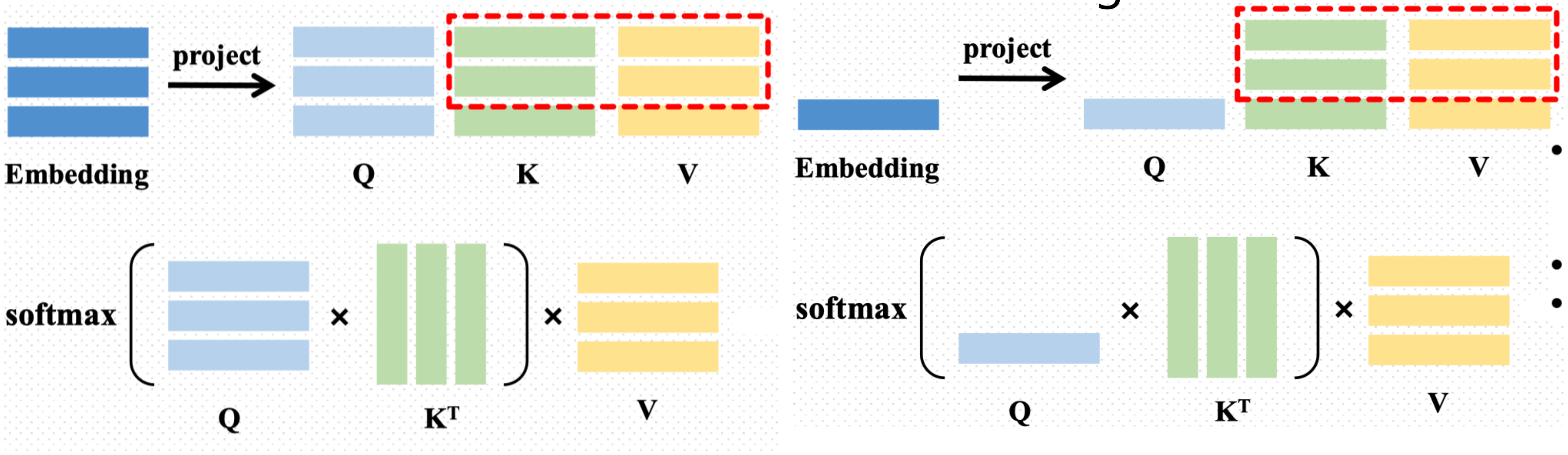
- **Background & Motivation**
  - Prefill & Decoding
  - Sequence Parallelism
- **Design**
  - Elastic Sequence Parallelism
  - Scheduling w/ Cost Model
- **Implementation**
- **Evaluation**
- **Discussion**

Background – Prefill & Decoding

❑KV Cache

❖Storing the KV tensors to avoid recomputation in decoding

Prefill: Generate KV Cache Decoding: Use KV Cache



October 15

- 💡 [OSDI'24] InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management
- 👨 Ping Gong, Jiawei Yi, Juncheng Zhang
- 📕 slides, 📃 Q&A summary, 📺 video

❖Huge KV cache footprint for a sentence of 1M tokens for Llama2-7B

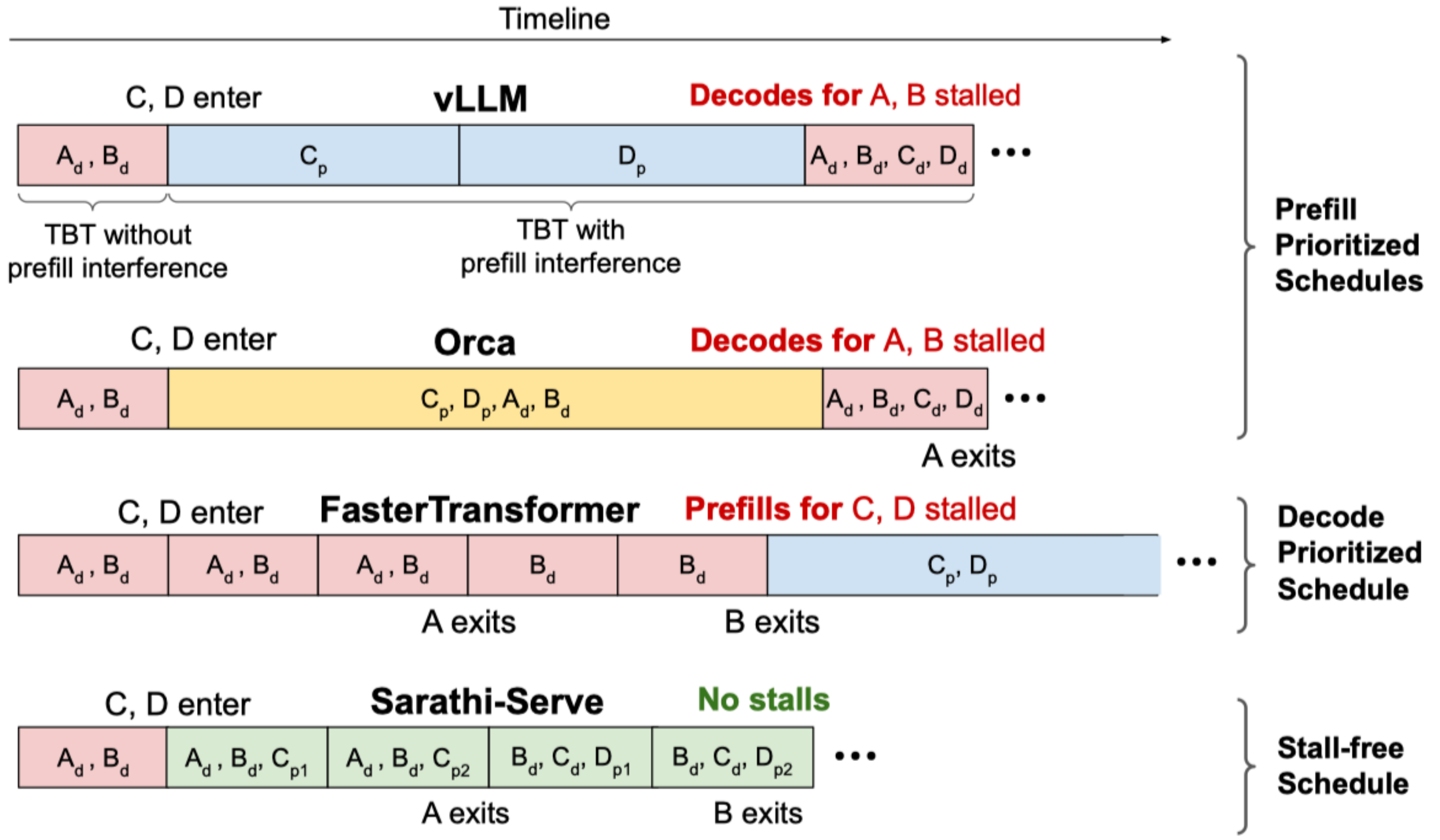
➢$1000000(seqlen)*4096(hidden\_size)*32(layer)*2(k+v)*2(fp16)/1024^3$

$= 488.3GiB$

Background – Prefill & Decoding

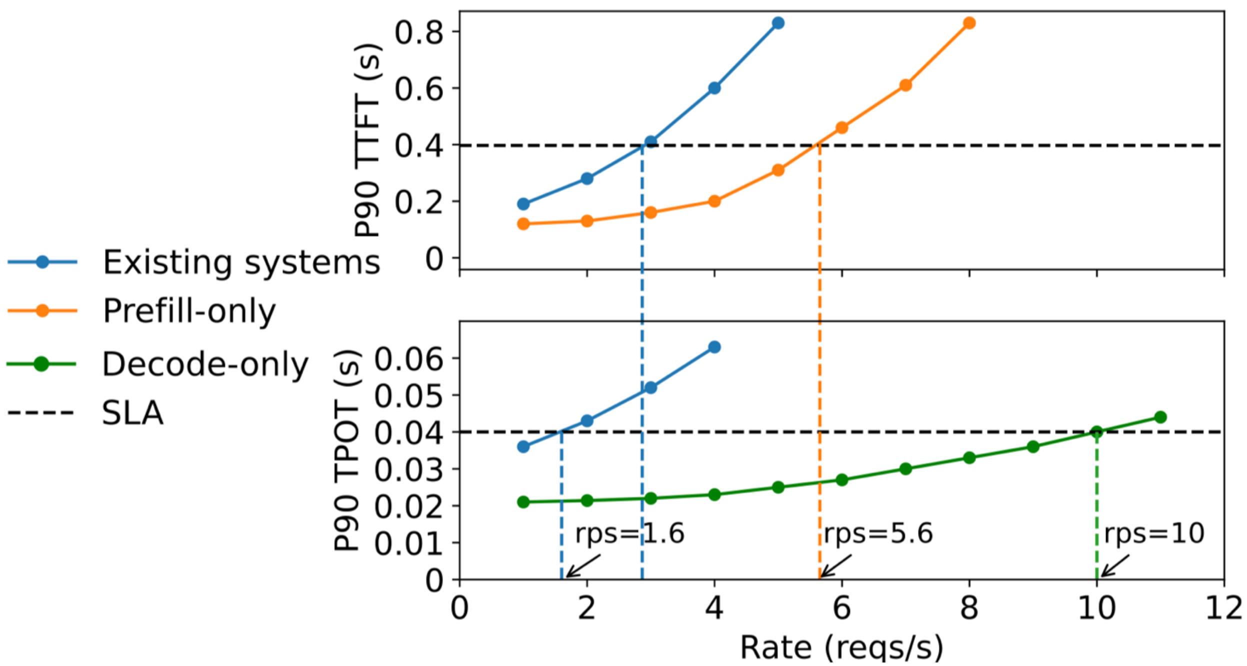
❑Chunked prefill: fine-grained scheduling

- ❖to avoid prefill or decoding delayed for too long



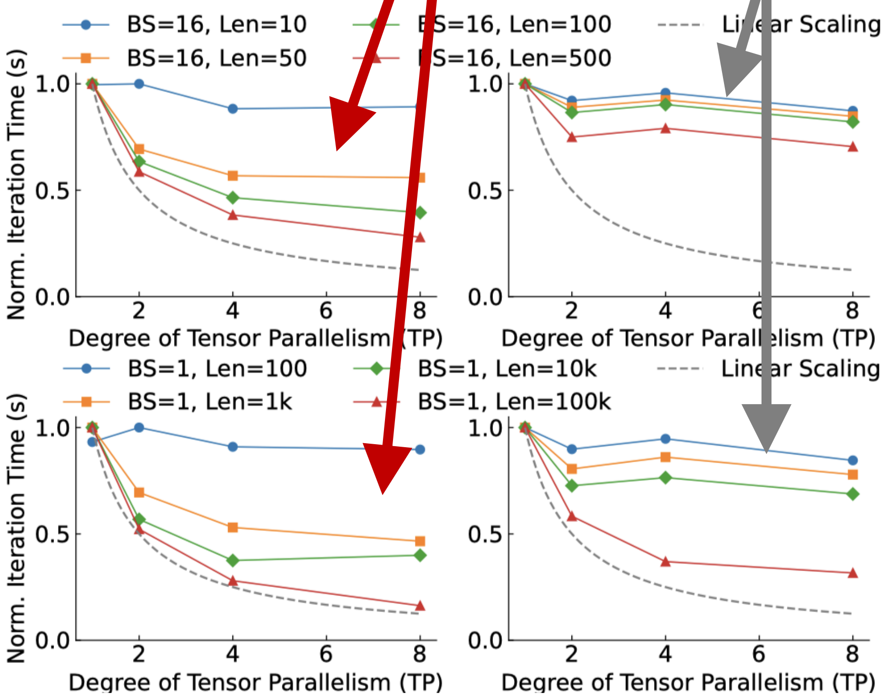
Background – Prefill & Decoding

- P/D disaggregation: dedicated GPUs for P/D
  - to avoid interference between P/D
  - to allocate dedicated parallel GPUs to P/D
    - Prefill: compute-bound; Decoding: memory-bound



Significant gain of Prefill from adding TP degree

Less gain of Decoding from adding TP degree

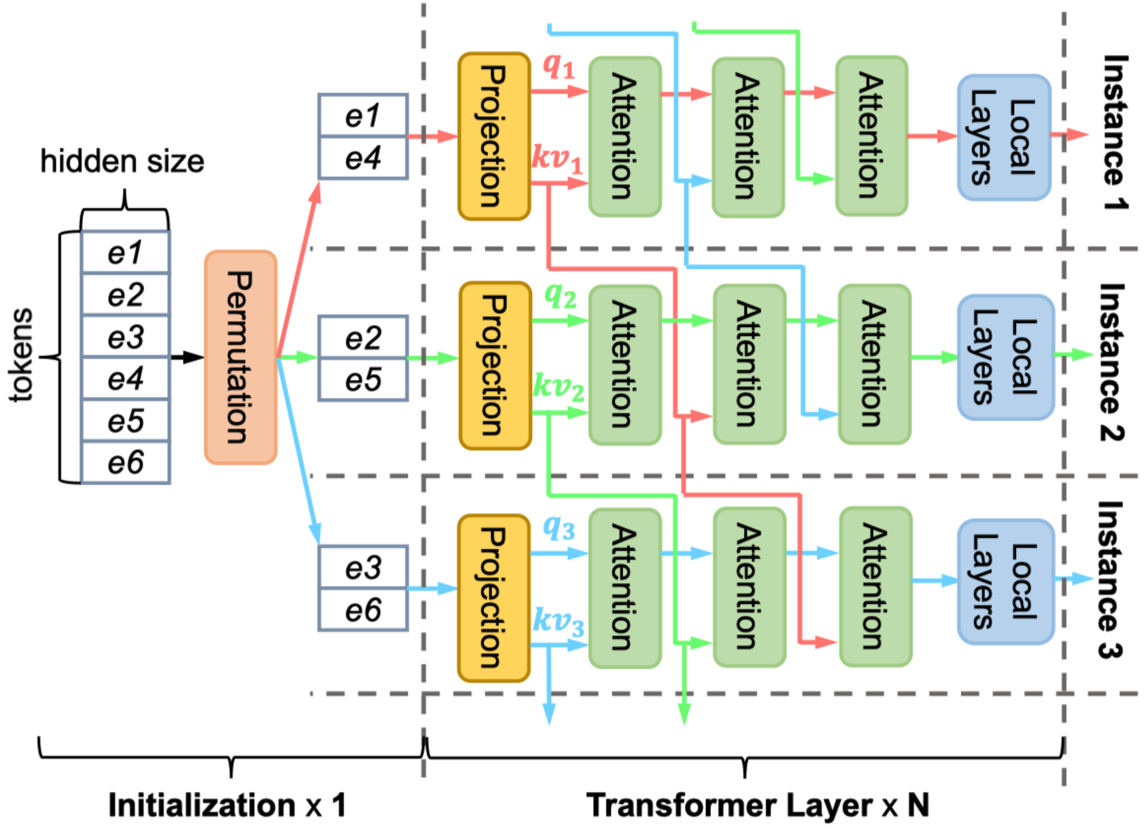


Prefill Decoding

Background – Sequence Parallelism

❑Ring Attention

❖Circulating kv_i among GPUs to get complete qkv results



Motivation & Challenge

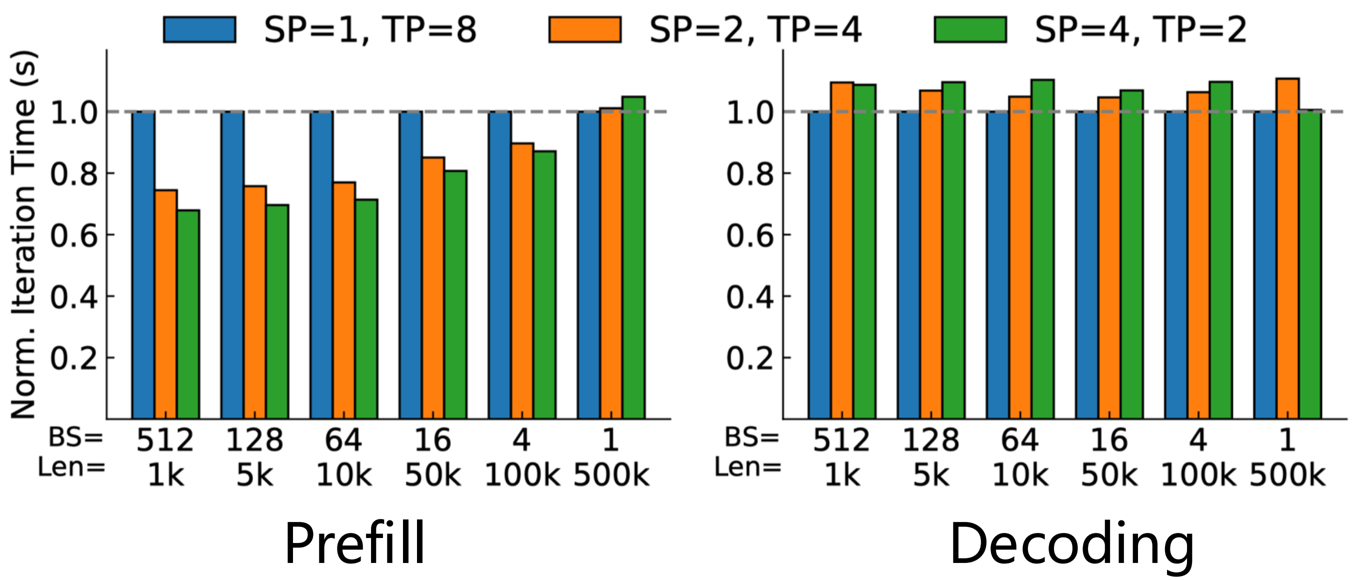
❑Motivation:

- ❖Static setup vs dynamic requirement
  - ➢prefill: **different lengths** across requests
  - ➢decoding: **increasing lengths** for each request
  - ➢prefill→decoding: different requirements for **different phases**

Motivation & Challenge

❑Motivation:

- ❖Static setup vs dynamic requirement
- ❖Sequence Parallelism with KV cache
  - ➢SP natively supports KV cache replication
  - ➢SP is comparable to TP



Motivation & Challenge

❑Motivation:

- ❖Static setup vs dynamic requirement
- ❖Elastic Sequence Parallelism

❑Challenge:

- ❖Migration overhead of KV Cache for scaling
- ❖Fast scheduling in large searching space
  - ➢DoP of each batch, GPU grouping, request batching, KV cache placement

Design: Overview

❑Elastic sequence parallelism

①❖Scale-down: prefill→decoding

②❖Scale-up: decoding (length↗)

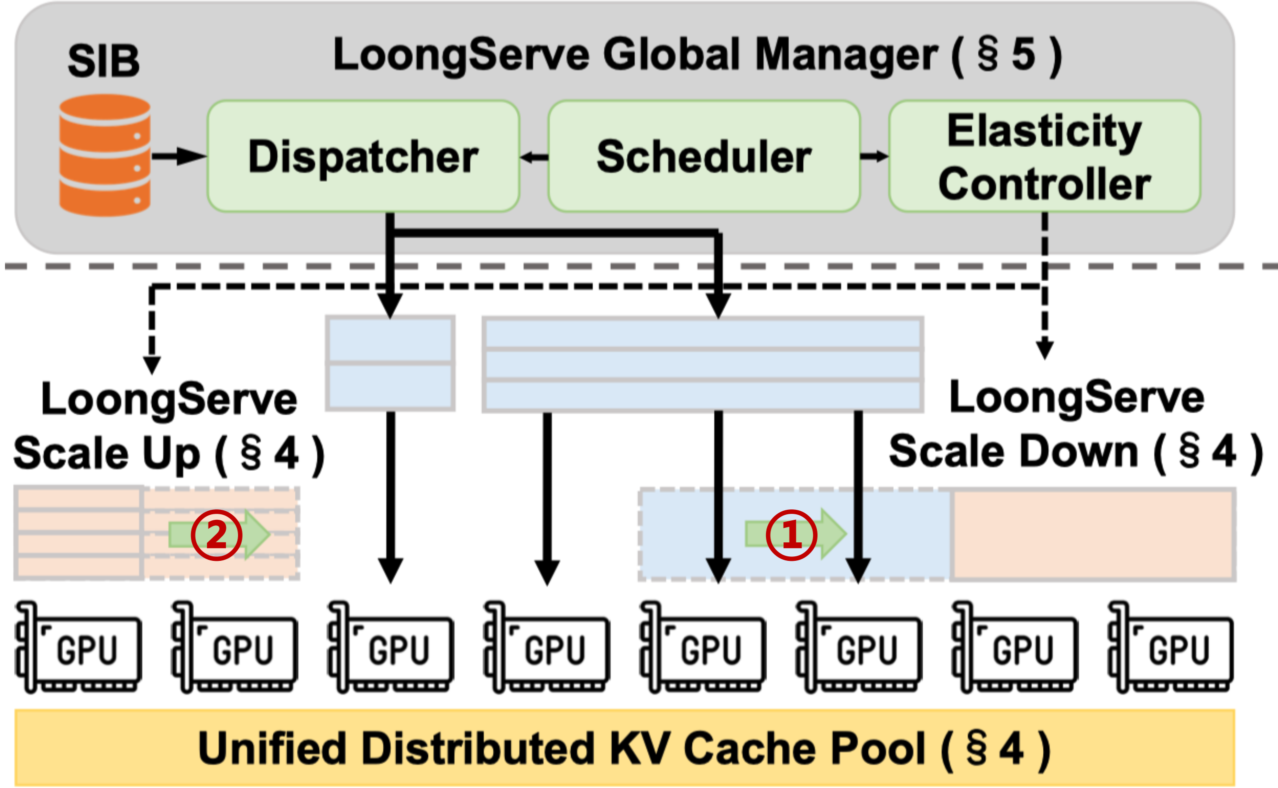
❑Scheduling w/ cost models

❖Request batching

❖GPU grouping

❖DoP scheduling

❖KV cache placement

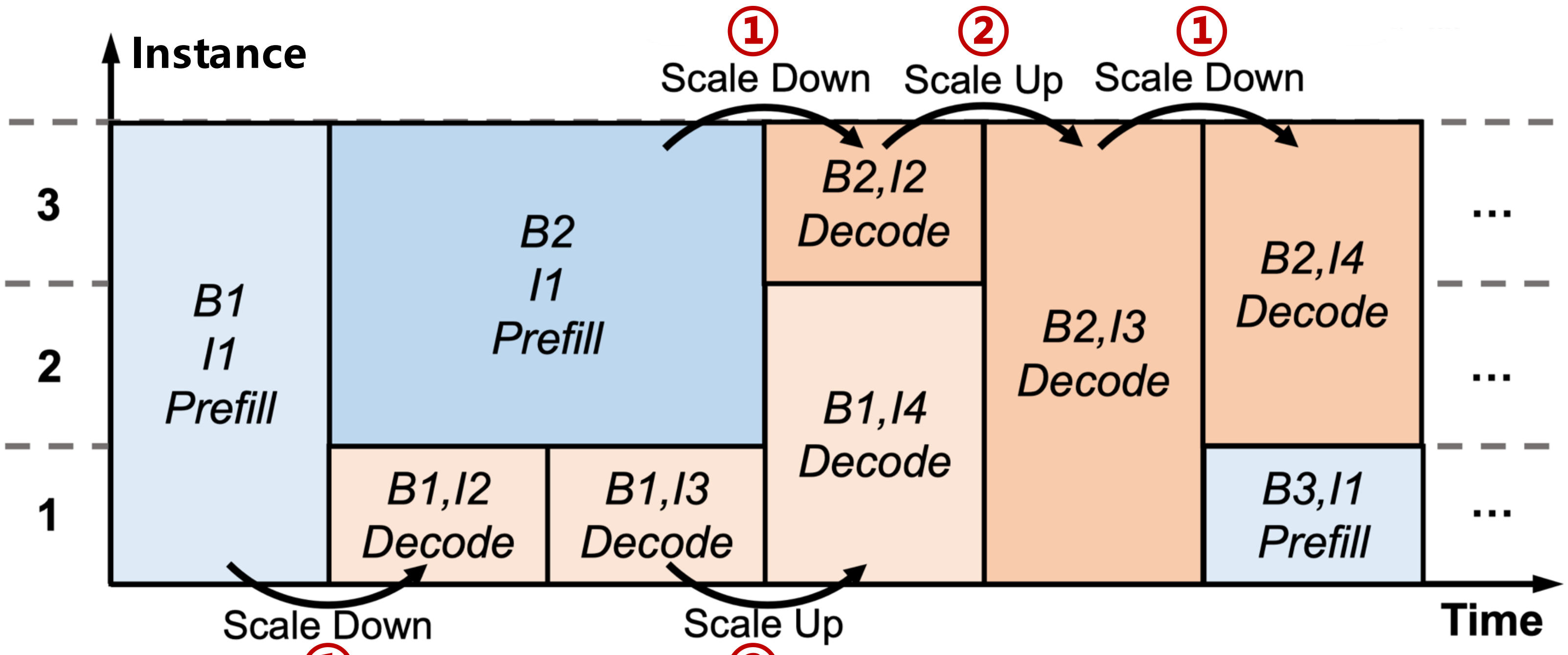


Design: Overview

❑Elastic sequence parallelism

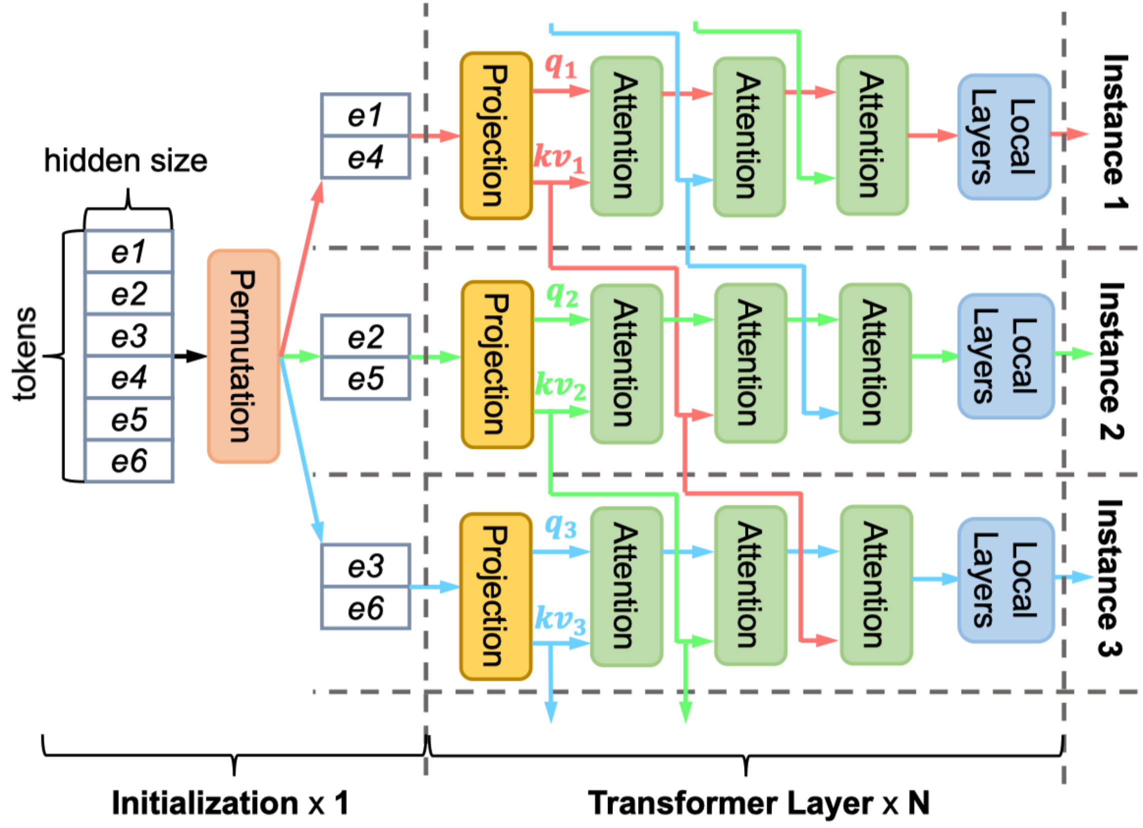
①❖Scale-down: prefill→decoding

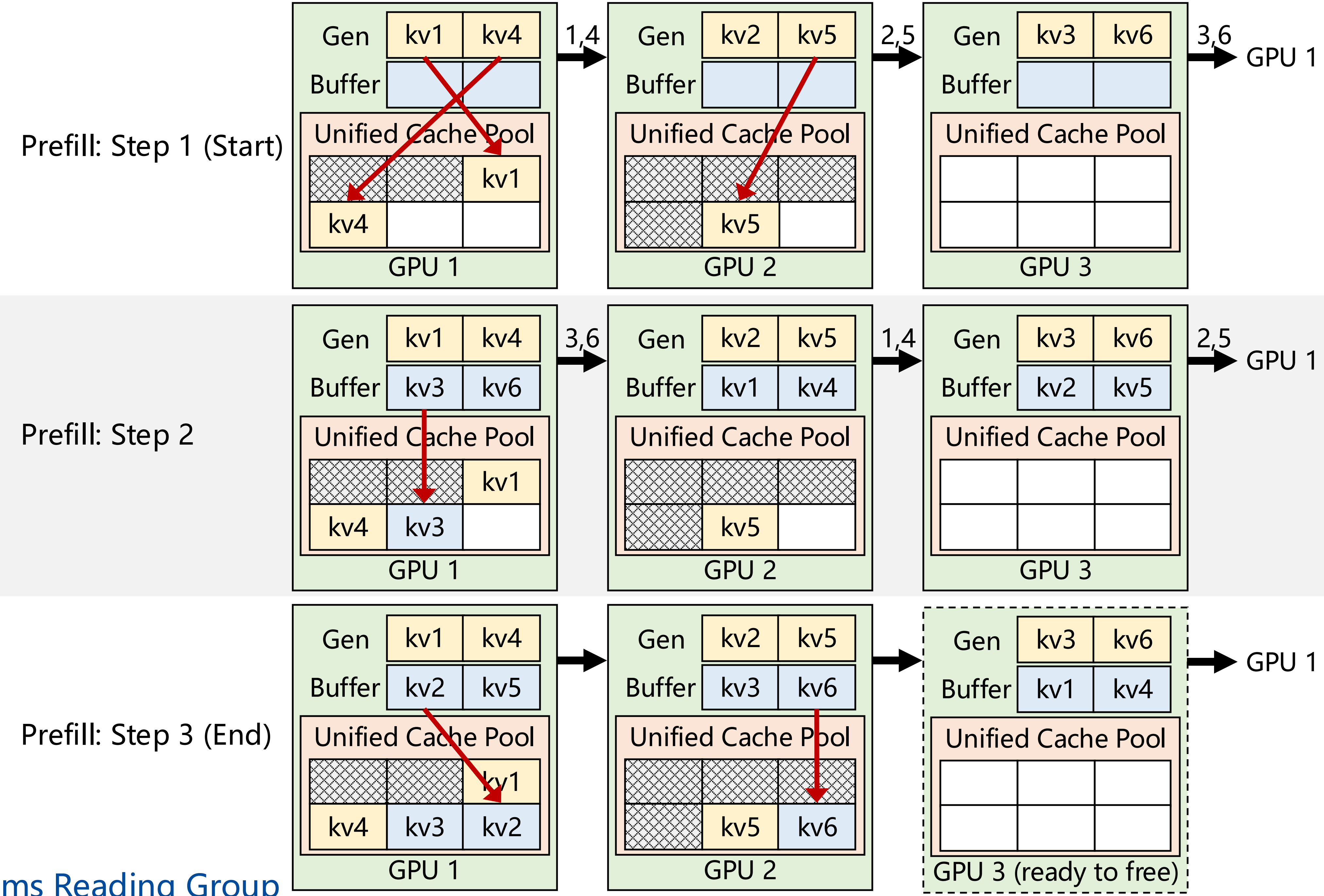
②❖Scale-up: decoding (length↗)



ESP: Group Scale-down (Prefill→Decoding)

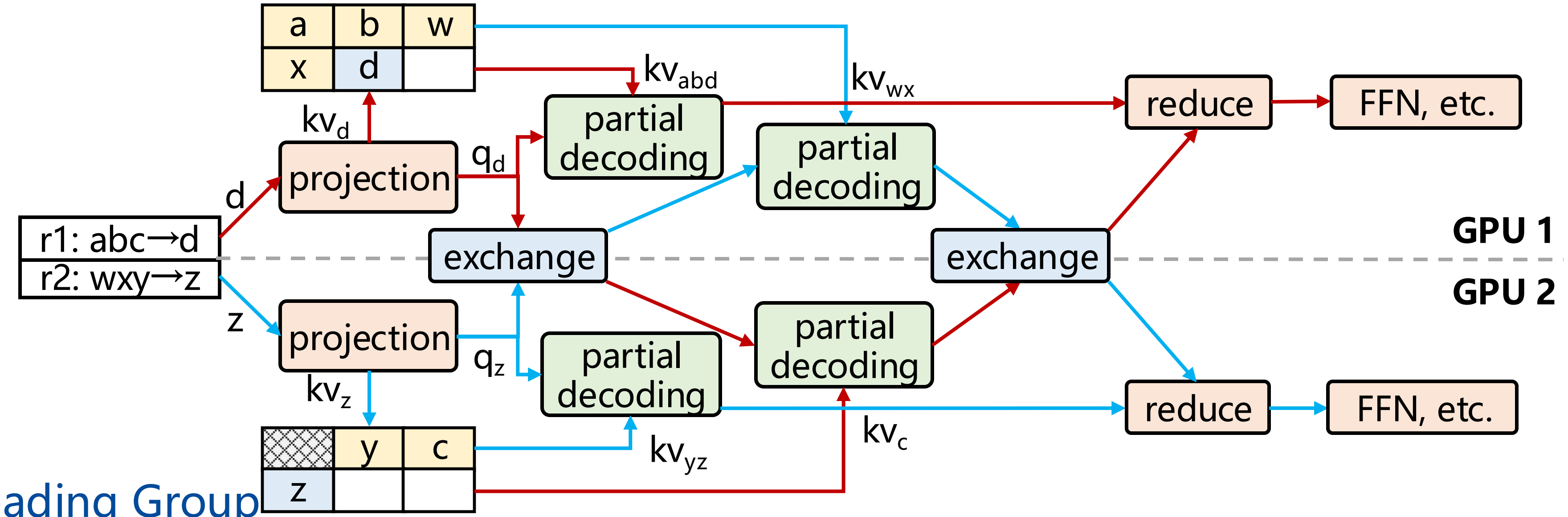
- ❑ **ESP: proactive migration w/ SP**
  - ❖ In SP, the KV cache chunks are **inherently circulated** among GPUs
    - ➢ no additional overhead of KV cache migration




Prefill: Step 1 (Start)
Prefill: Step 2
Prefill: Step 3 (End)
Gen
Buffer
Unified Cache Pool
kv1
kv2
kv3
kv4
kv5
kv6
1,4
2,5
3,6
GPU 1
GPU 2
GPU 3
GPU 3 (ready to free)

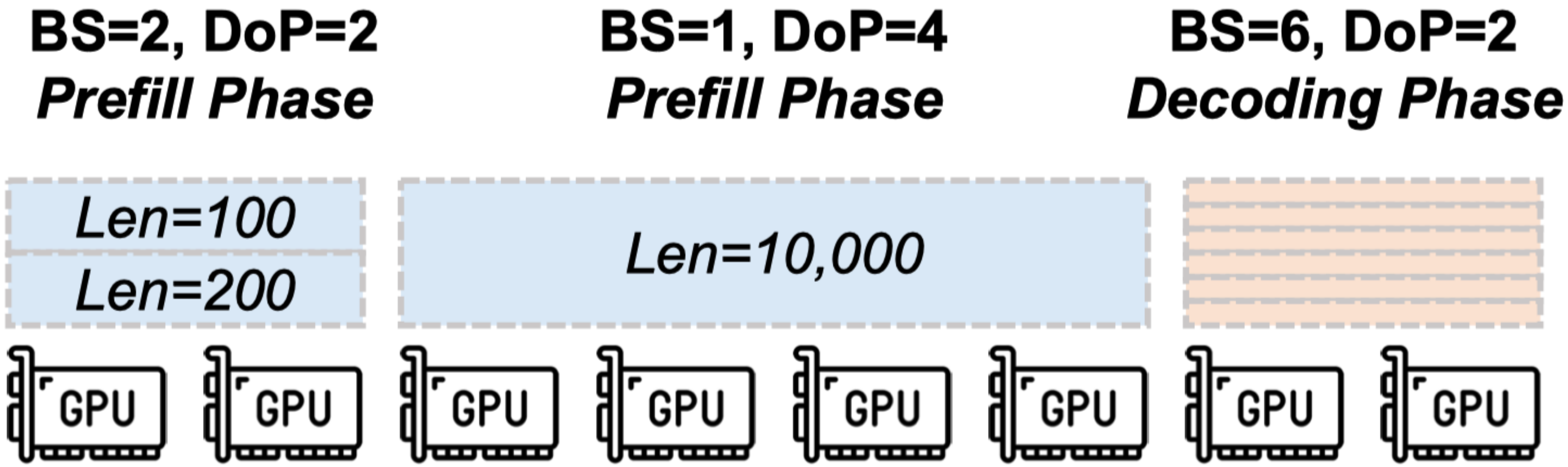
ESP: Group Scale-up (Decoding Length↗)

- ❑ **Existing: Moving requests to new GPUs w/o huge KV cache**
  - ❖ Problem: huge overhead of cache migration
- ❑ **ESP: Multi-master distributed decoding**
  - ❖ ①Compute q_i, kv_i; ②store kv_i in local cache pool; ③circulate q_i
  - ❖ ④Reduce the results; ⑤continue with layer_norm and FFN



Design: Scheduling with Cost Models

- **Scheduling space:**
  - dispatch which requests from a pending queue
  - how to batch requests
  - DoP
  - allocate which GPUs



Design: Scheduling with Cost Models

❑Dispatching

- ❖FCFS (First-Come-First-Serve)
- ❖Adding requests until
  - ➢memory is not enough
    - to avoid eviction and recomputation
  - ➢becoming compute-bound
    - to avoid slowdown
  - ➢preemption cost is too high for preempted batches
    - to avoid slowdown decoding batches too much

Design: Scheduling with Cost Models

- **Elastic instance allocation**
  - Adding idle instances to a batch
  - If idle instances are not enough
    - try to scale-down decoding instances and allocate for decoding
      - until too many KVs to migrate in decoding instances
  - If still not enough
    - try to preempt decoding instances with most unused KV slots

Design: Scheduling with Cost Models

- **Batching**
  - Group requests with the similar lengths
  - Use dynamic programming to minimize average TTFT
  - Avoid FFN to be compute-bound
- **Elastic scaling plan generation**
  - Scale-down: minimum DoP s.t. memory is enough
  - Scale-up: adding DoP until memory is enough

Implementation

- ❑ **15k lines of code:**
  - ❖ Language: C++, CUDA, Python, and Triton
  - ❖ Extended from Striped Attention
  - ❖ Communication: Ray RPC, NCCL
  - ❖ Reusing some components from vLLM and LightLLM
  - ❖ Front end: similar to OpenAI API
  - ❖ GitHub repo: https://github.com/LoongServe/LoongServe.

Evaluation: Setup

❑Workload: Poison

- ❖ShareGPT: 4-2.3k (short input, long output)
- ❖L-Eval: 2.7k-210.k (used in Qwen 1.5)
- ❖LV-Eval: 15.1k-497.3k
- ❖Mixed: ⅓ ShareGPT + ⅓ L-Eval + ⅓ LV-Eval

❑Baseline:

- ❖vLLM[OSDI23]: fine-grained KV cache management
- ❖DeepSpeed-MII[arXiv24]: SplitFuse (seq_len ≤ 32k)
- ❖LightLLM w/ SplitFuse: open-source for long seq_len
- ❖DistServe[OSDI24]: P/D Disaggregation

Evaluation: Setup

❑Metrics

- ❖per-token latency (end-to-end latency / number of tokens)
- ❖SLO attainment (requiring P90 latency <= SLO)

❑Hardware:

- ❖A800 (80 GB) x 16
- ❖200 Gbps IB NiC x 4
- ❖400 GBps NVLink (full connectivity between each pair of GPUs)

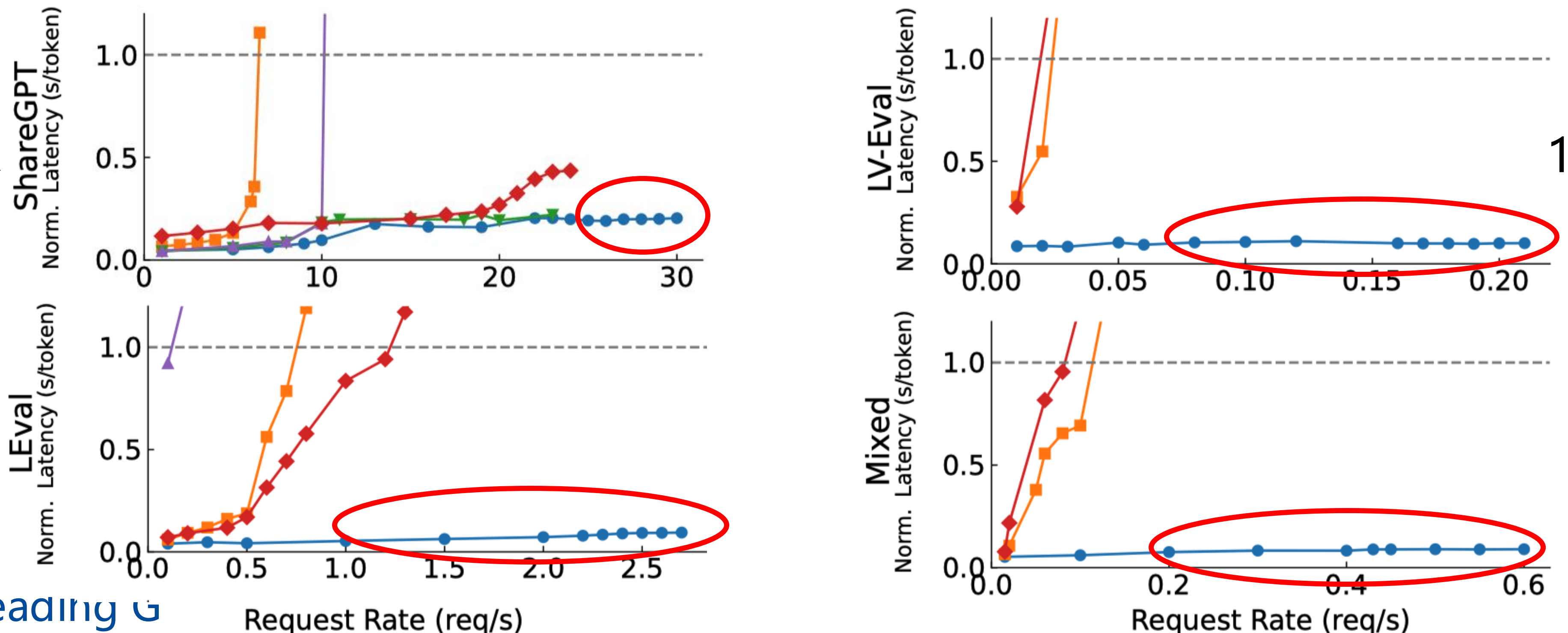
❑Model

- ❖LWM-1M-Text (Llama-2-7B + 1M seq_len)

Evaluation: Single-Node End-to-End (Decoding)

- **Metrics**: end-to-end latency per token
- **LoongServe outperforms a lot**

System	Paralleism
vLLM	TP=8
DeepSpeed-MII	TP=8
LightLLM	TP=8
DistServe	P(TP=4) D(TP=4)
LoongServe	TP=2; ESP



Evaluation: Single-Node End-to-End (Prefill)

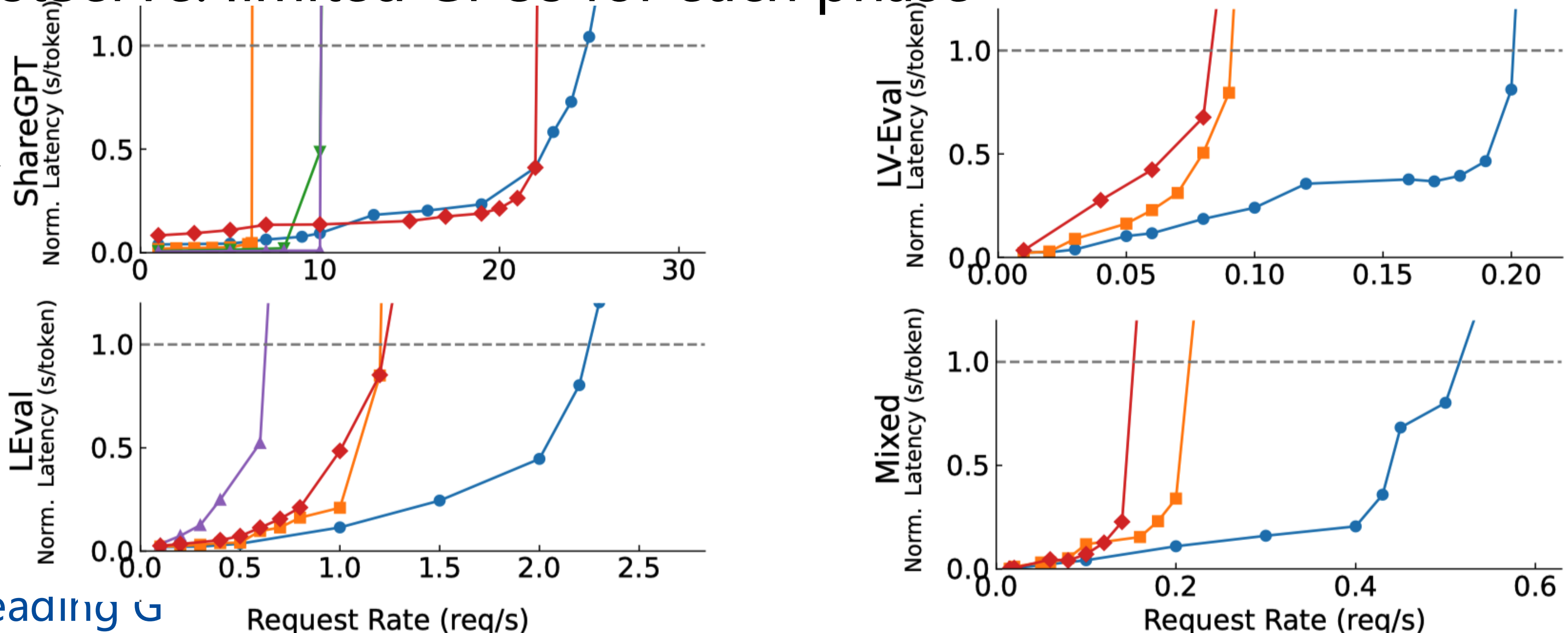
- **LoongServe still outperforms**
  - vLLM: interference of P/D
  - DeepSpeed-MII, LightLLM (SplitFuse)
    - inefficient prefill + still interference of P/D
  - DistServe: limited GPUs for each phase

System	Paralleism
vLLM	TP=8
DeepSpeed-MII	TP=8
LightLLM	TP=8
DistServe	P(TP=4) D(TP=4)
LoongServe	TP=2; ESP≤4

4-2.3k

15.1k-497.3k

2.7k-210.k



Evaluation: Single-Node End-to-End (Both)

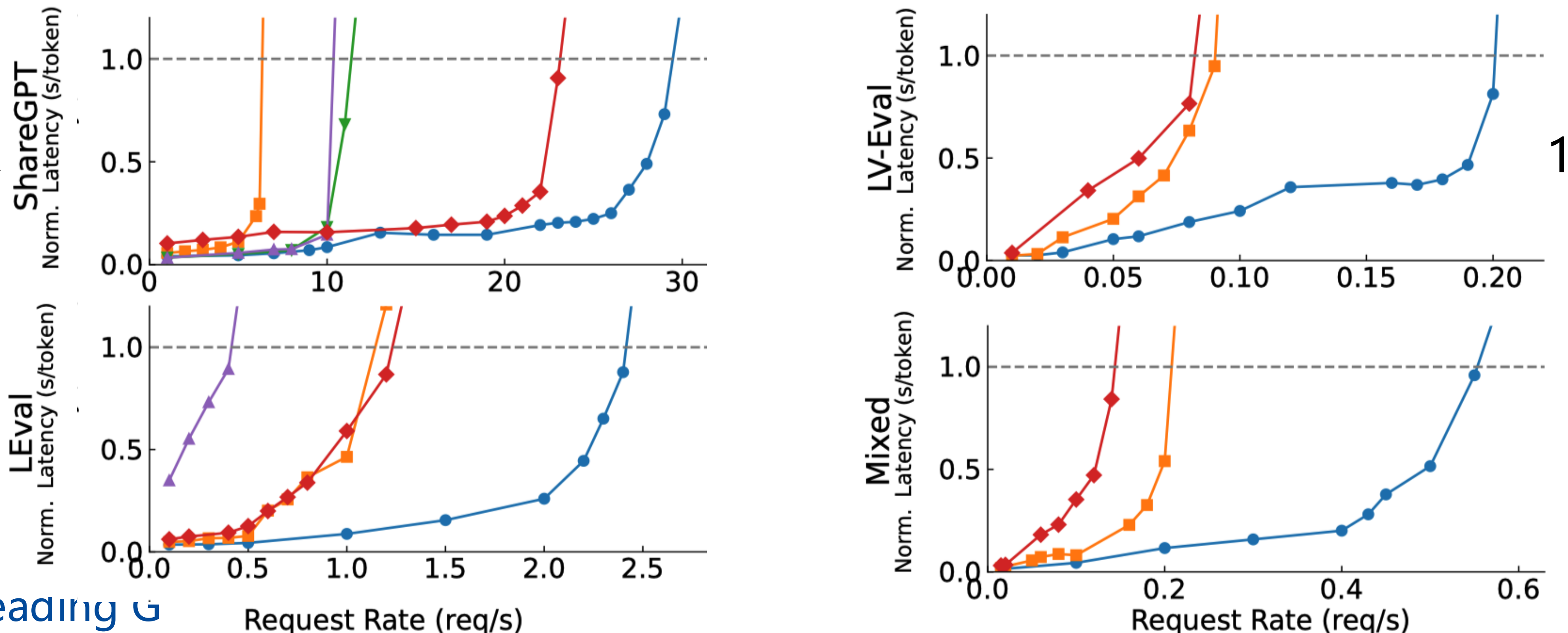
- **LoongServe throughput speedup**
  - vLLM: up to 4.64x
  - DeepSpeed-MII, LightLLM: up to 3.85x
  - DistServe: up to 5.81x

System	Parallelism
vLLM	TP=8
DeepSpeed-MII	TP=8
LightLLM	TP=8
DistServe	P(TP=4) D(TP=4)
LoongServe	TP=2; ESP≤4

4-2.3k

15.1k-497.3k

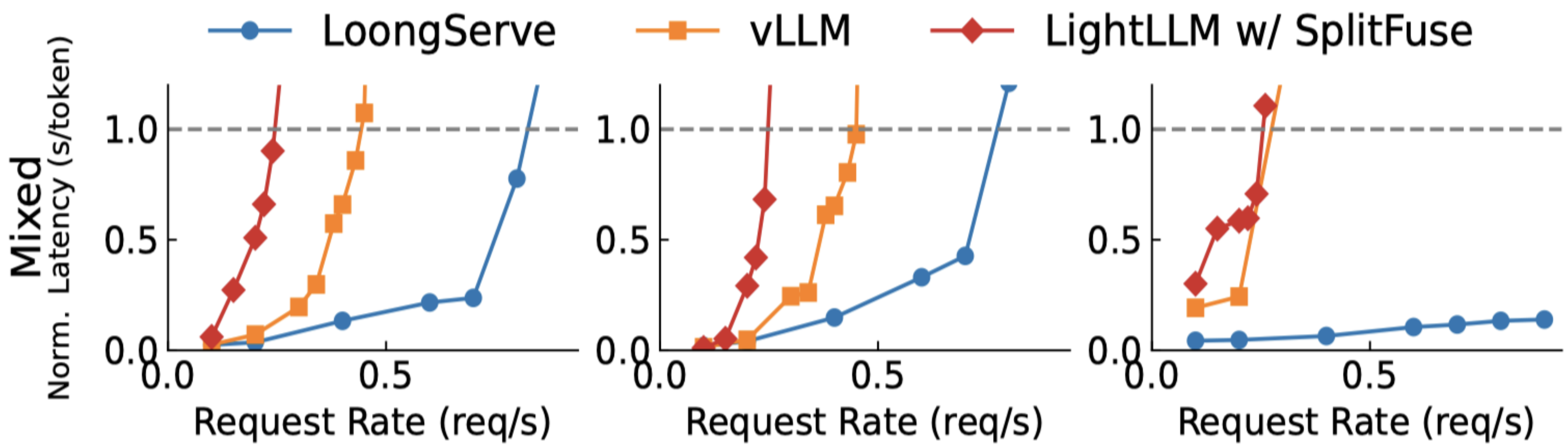
2.7k-210.k



Evaluation: Two-Node End-to-End

- **LoongServe also outperforms**
  - vLLM: up to 1.86x
  - LightLLM: up to 3.37x
    - lower than one-node speedups (significant inter-node comm. overhead of SP)

System	Paralleism
vLLM	TP=8, 2 nodes
LightLLM	TP=8, 2 nodes
LoongServe	TP=2; ESP≤8



avg latency

avg latency per input token

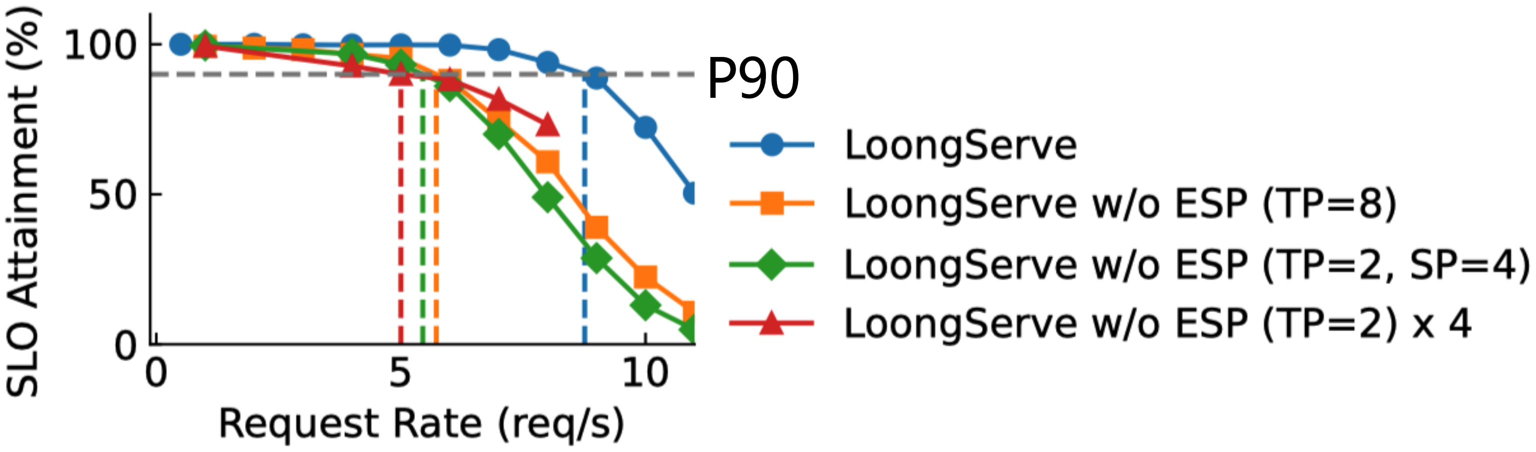
avg latency per output token

Evaluation: Ablation Study

- ❑Benefit of ESP vs static parallelism
- ❑Scale-up: benefit and frequency
- ❑Overhead of scale-down and scale-up
- ❑Accuracy of LoongServe analytical model

Evaluation: Ablation Study

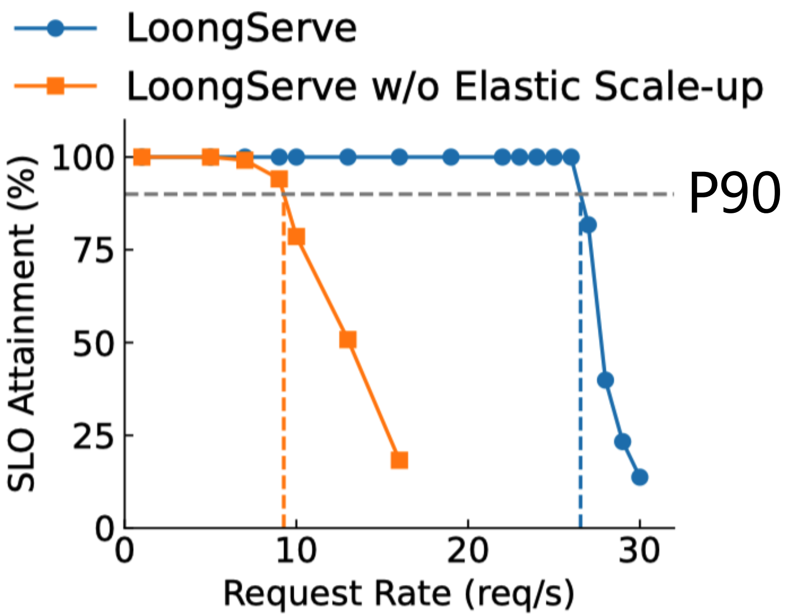
- **Benefit of ESP vs static parallelism**
  - P90 goodput: maximum throughput where P90 latency ≤ SLO
  - ESP outperforms static SP + TP schemes
    - speedup: 2.33x, 1.98x, 1.53x



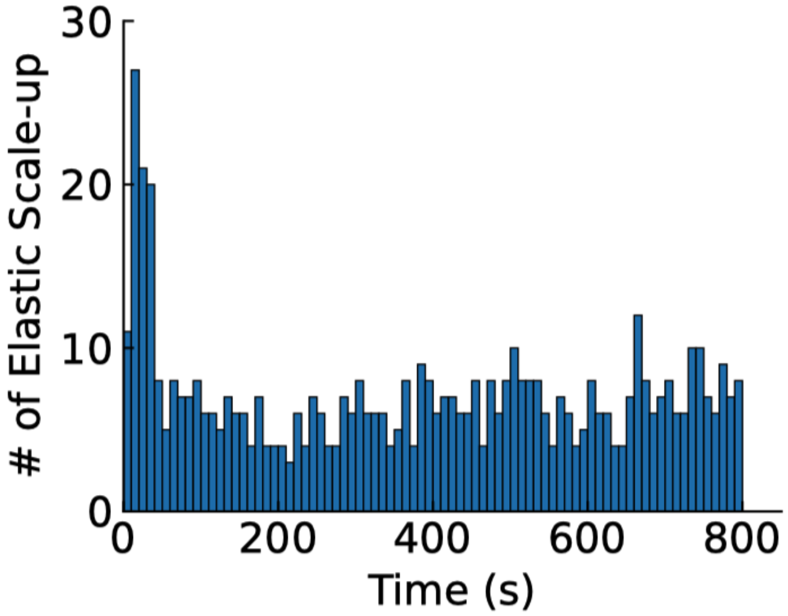
P90 goodput

Evaluation: Ablation Study

- **Scale-up: benefit and frequency**
  - ShareGPT: short input length, long output length
    - P90 goodput: 2.87x vs LoongServe w/o Scale-up
    - necessary scale-up to handle dynamic workloads



P90 goodput for ShareGPT

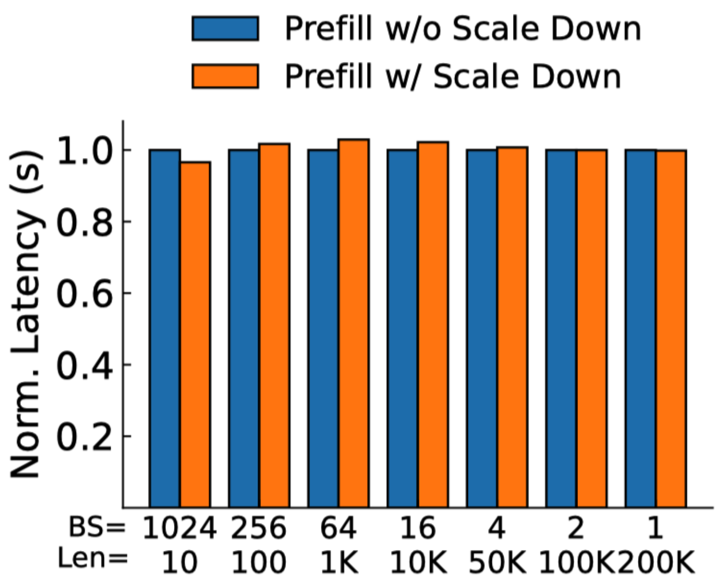


frequency of scale-up for ShareGPT

Evaluation: Ablation Study

❑Overhead of scale-down: ≤2%

- ❖additional KV cache copy operation to the cache pool

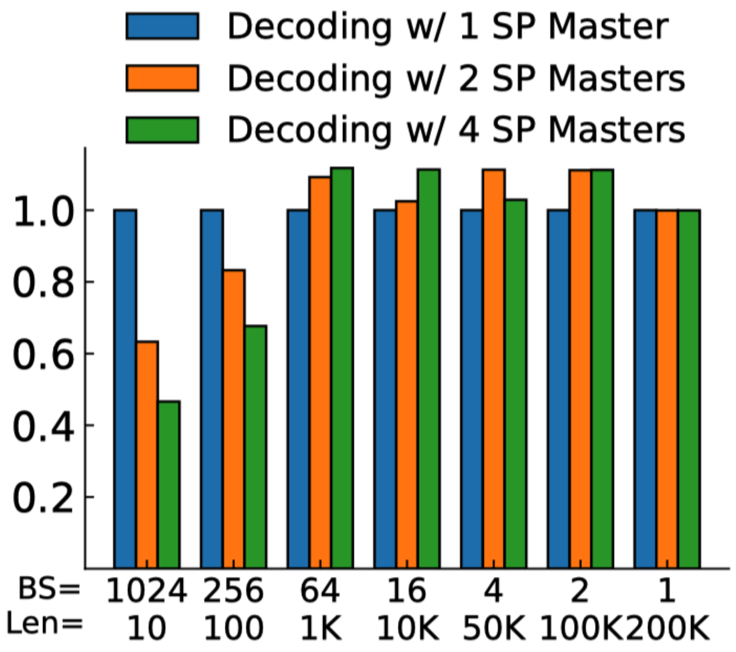


Scale-down overhead

Evaluation: Ablation Study

❑Different number of SP masters:

- ❖Setup: 4 instances with 1/2/4 masters
- ❖FFN*Projection executed in a single master
- ❖Higher BS: lower latency
  - ➢more tasks are parallelized
  - ➢4-master outperforms 1-master
- ❖Lower BS:
  - ➢overhead of comm. and sync.
  - ➢in worst cases,
    - 4-master is slower than 1-master by ≤10%

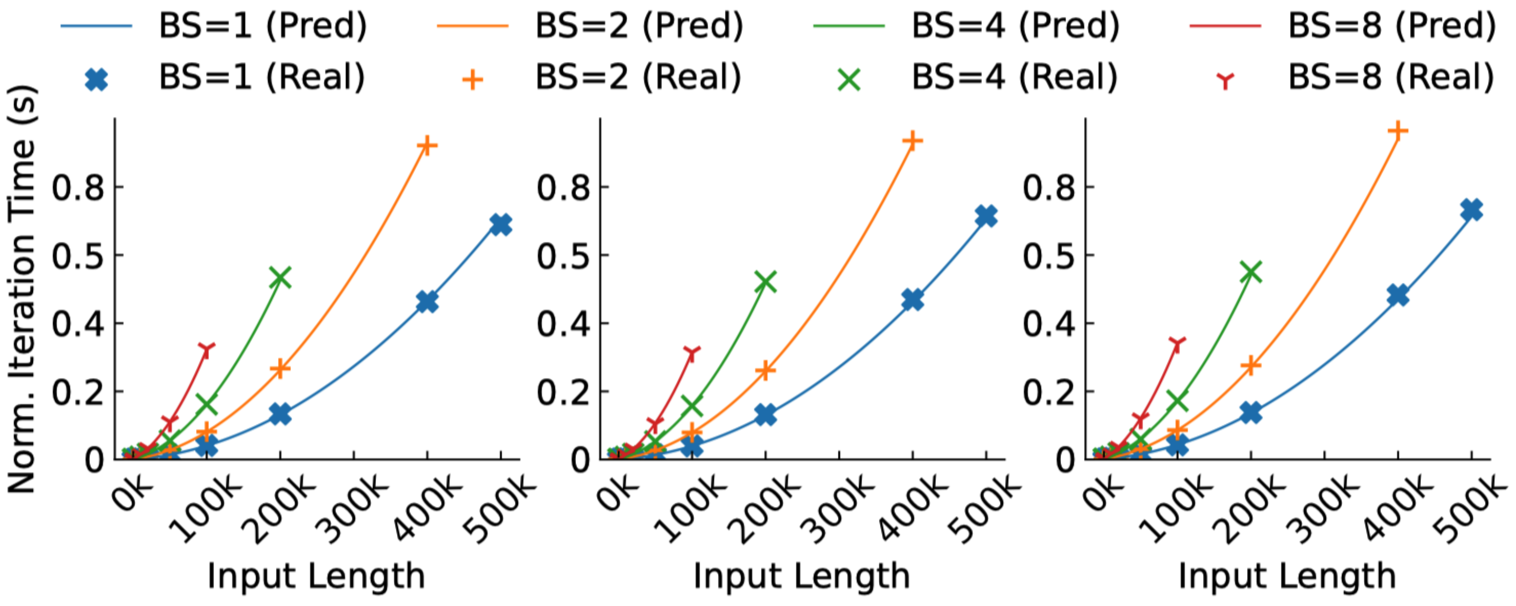


Scale-up overhead

Evaluation: Ablation Study

❑Accuracy of LoongServe analytical model

- ❖≤10% deviation
  - ➢under different parallelism schemes and input lengths



(a) SP2TP4. (b) SP4TP2. (c) SP8TP1.

Discussion

- **Is two-node evaluation enough for LoongServe?**
  - More nodes enlarge the cache pool size
  - More nodes involve higher inter-node comm. overhead of SP
  - Maybe two-node setup is enough to serve a 7B model
- **Doubts:**
  - Zero-overhead scaling down?
    - No when scaling down decoding instances to boost prefill instances
  - Writing:
    - clarity: e.g., unclear KV cache migration scheduling when scaling
    - mismatch with caption, code, etc.