Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve

Amey Agrawal, *Georgia Institute of Technology;* Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, and Bhargav Gulavani, *Microsoft Research India;* Alexey Tumanov, *Georgia Institute of Technology;* Ramachandran Ramjee, *Microsoft Research India*

OSDI' 24

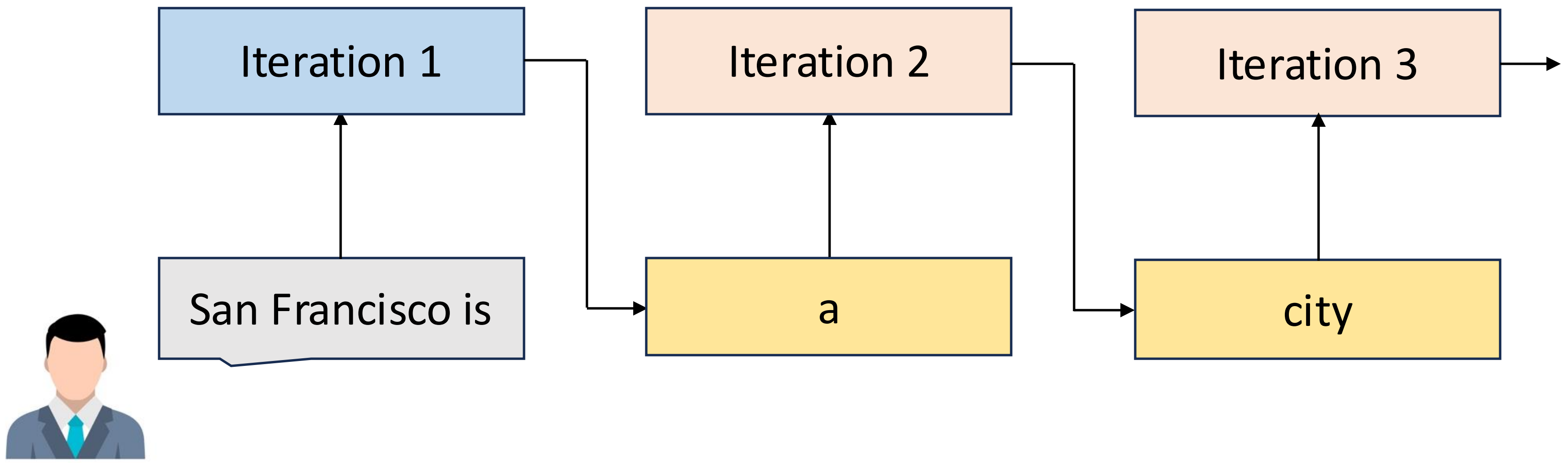
Presented by Yinhe Chen, Dongqi Tian

USTC, CHINA
ADSLAB
先进数据系统实验室

Outline

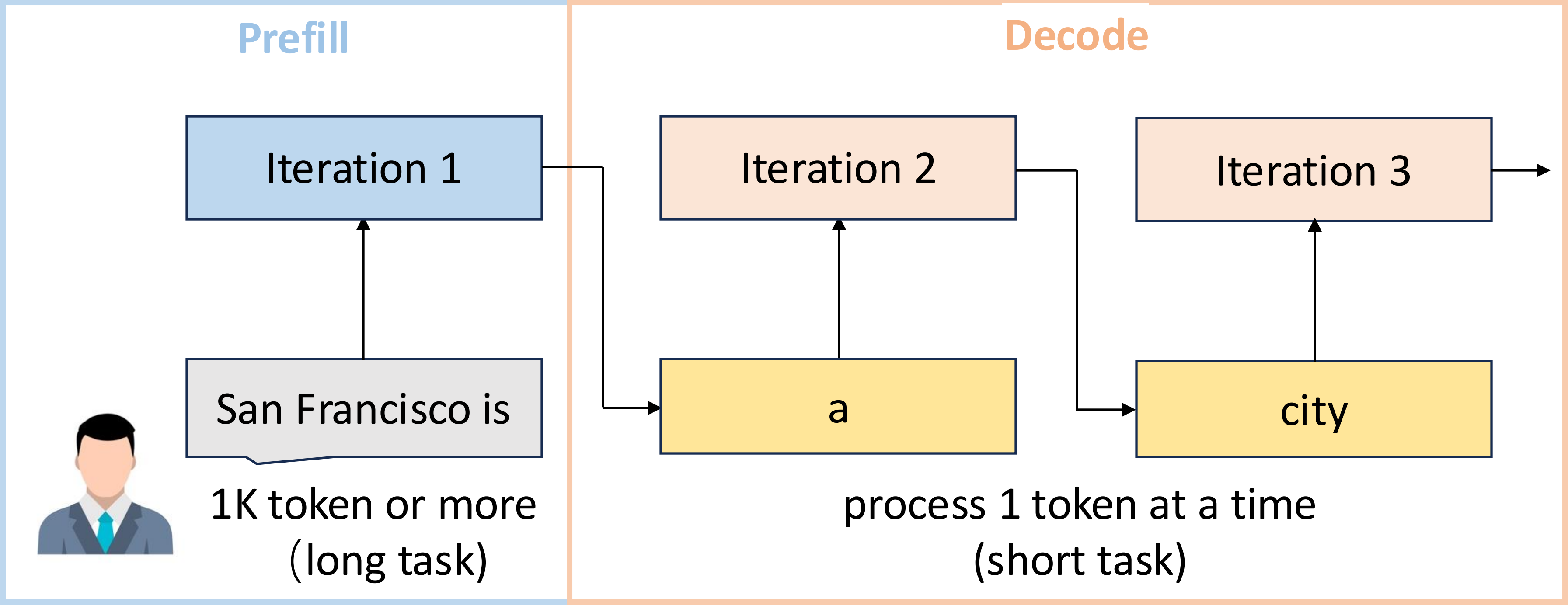
- Background and Existing Solutions
- Design and Implementation
- Evaluation
- Discussion

Auto-regressive Nature of LLMs



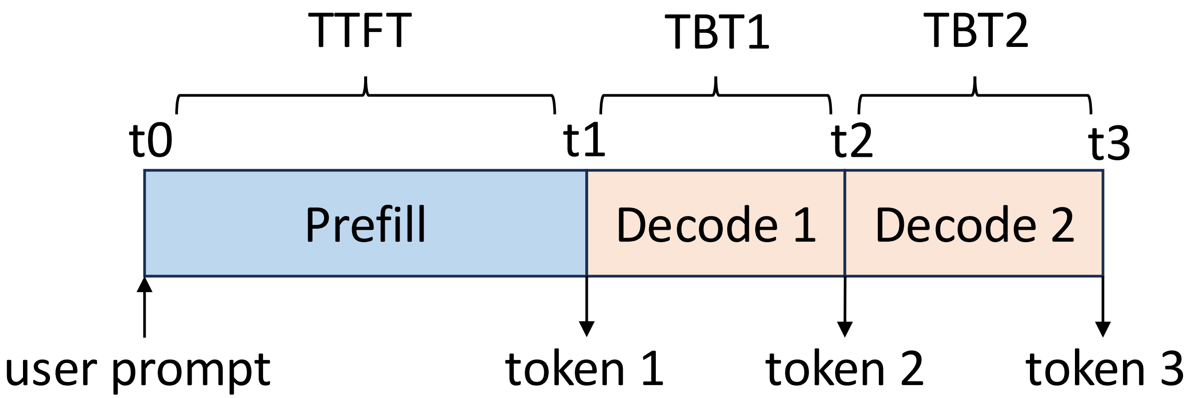
Two Phases of LLM Inference

- LLM inference serving request goes through two phases



Performance Metrics

- Latency
  - Time-to-first-token (TTFT)
    - Time processing the prompts
  - Time-between-token (TBT)
    - Time interval between each generated token

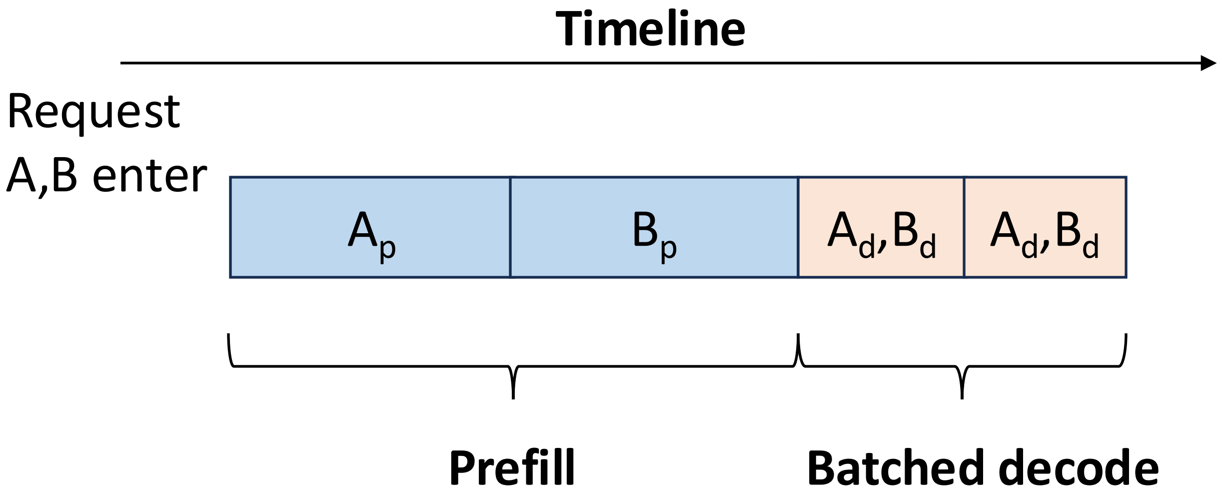


- Throughput
  - Maximum RPS the system can serve

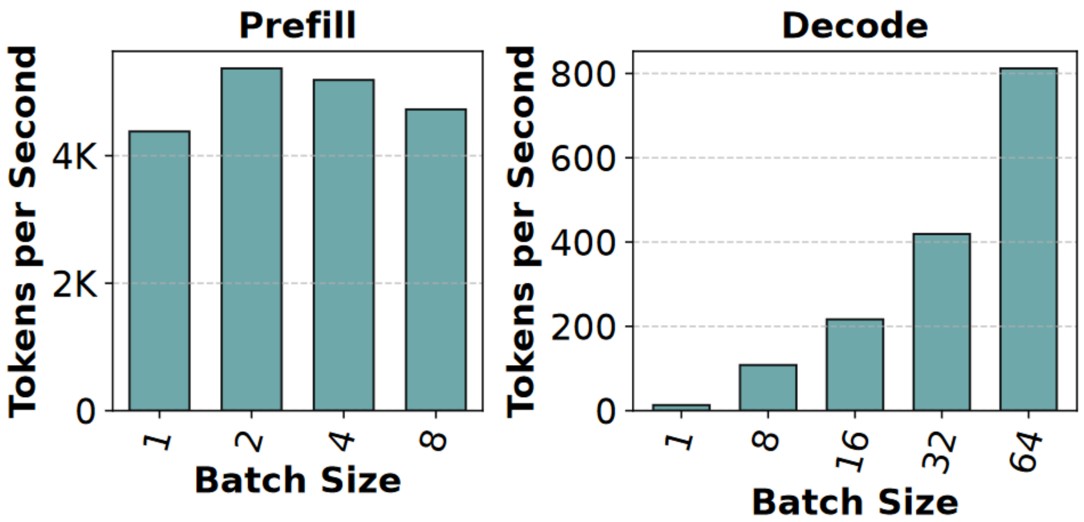
How to optimizing both throughput and latency?

Batching LLM Inference

- Batching: process tokens from different requests concurrently
- Batching enhances the decoding throughput



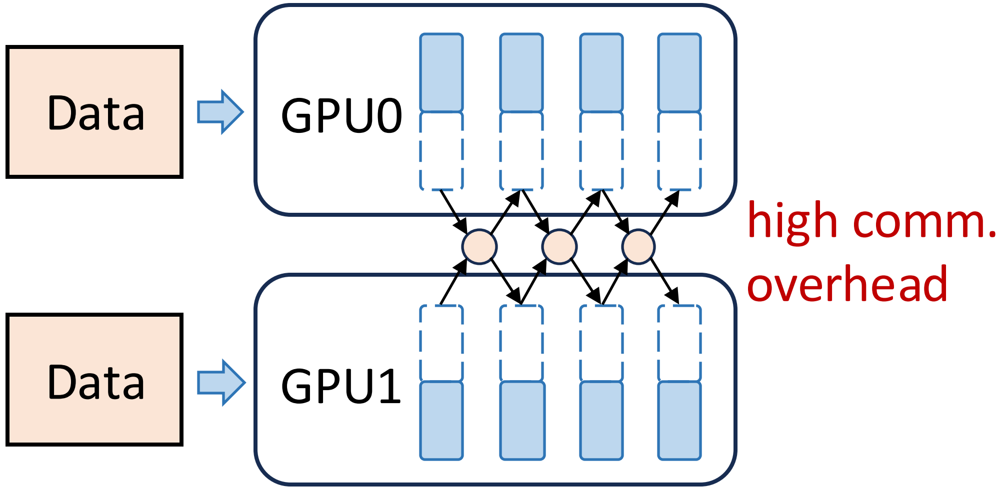
Example of batch decode



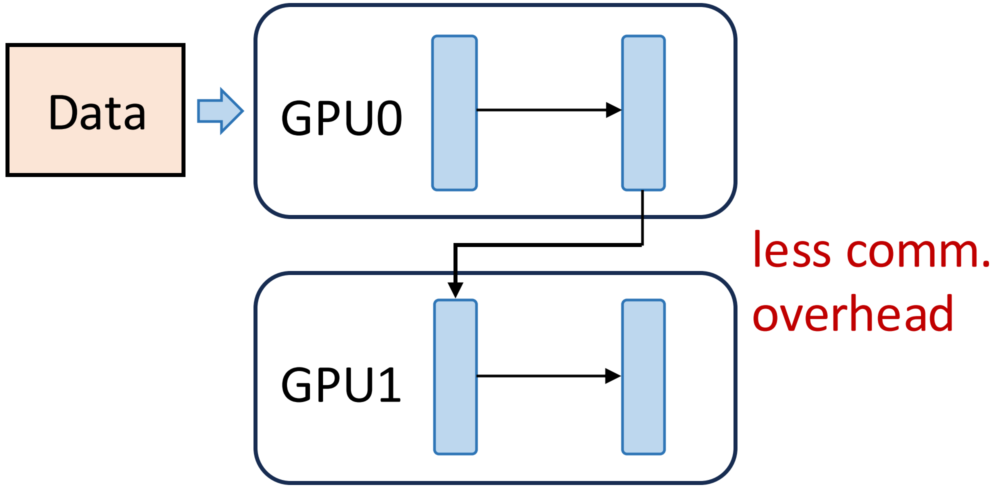
Mistral-7B on 1 A100, prompt length = 1024

Multi-GPU LLM Inference

- Single GPU HBM is limited
  - OPT-175B needs 9 A100-40G
  - Caching KV further accelerates
- Splitting model and KV to multi-GPU
  - Tensor parallelism (TP)
    - TP involves high communication cost
    - Inefficient for cross node network
  - Pipeline parallelism (PP)
    - PP only needs to communicate by layer
    - Suitable for cross node network



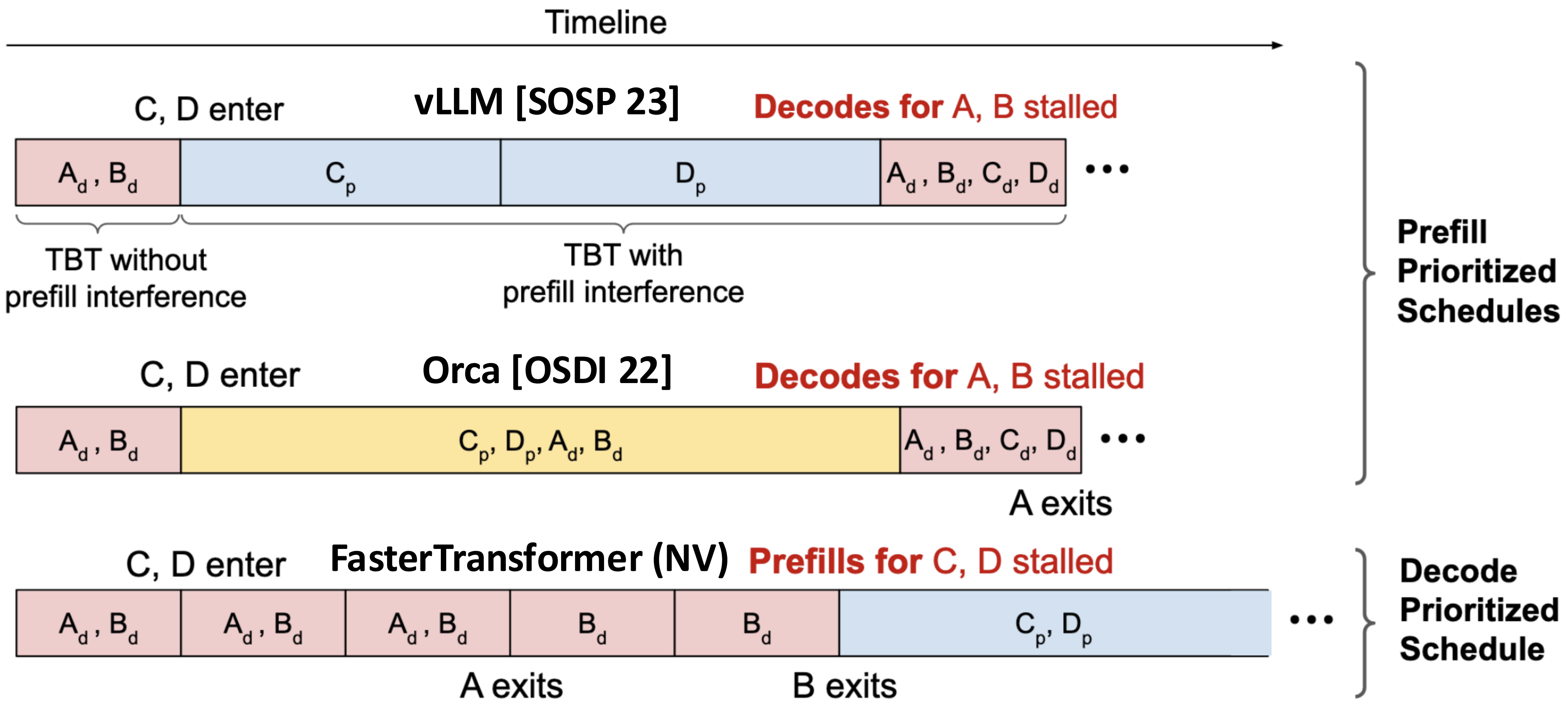
Tensor parallelism



Pipeline parallelism

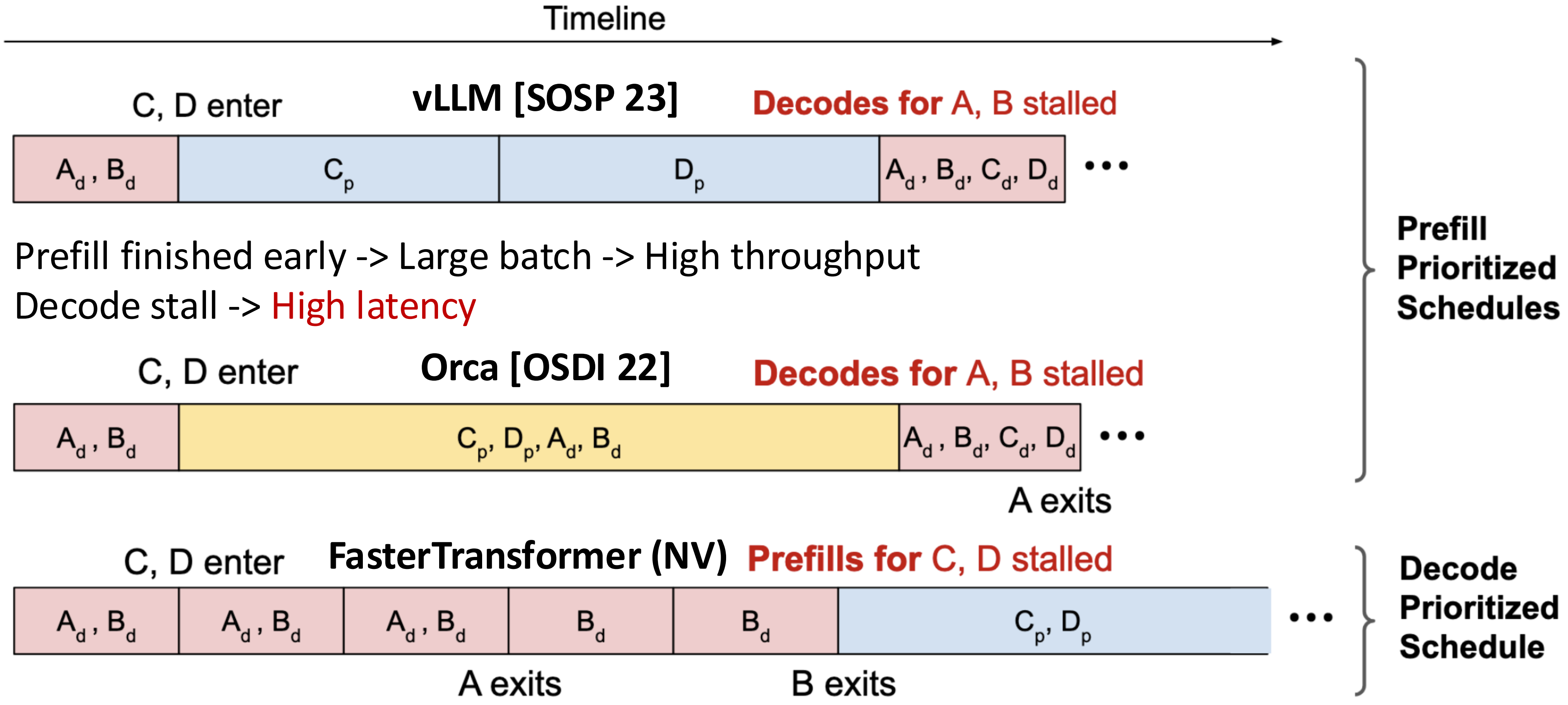
Existing Systems

- Existing works either prioritize prefill or decode



Throughput-latency tradeoff

- Either throughput or latency is sacrificed

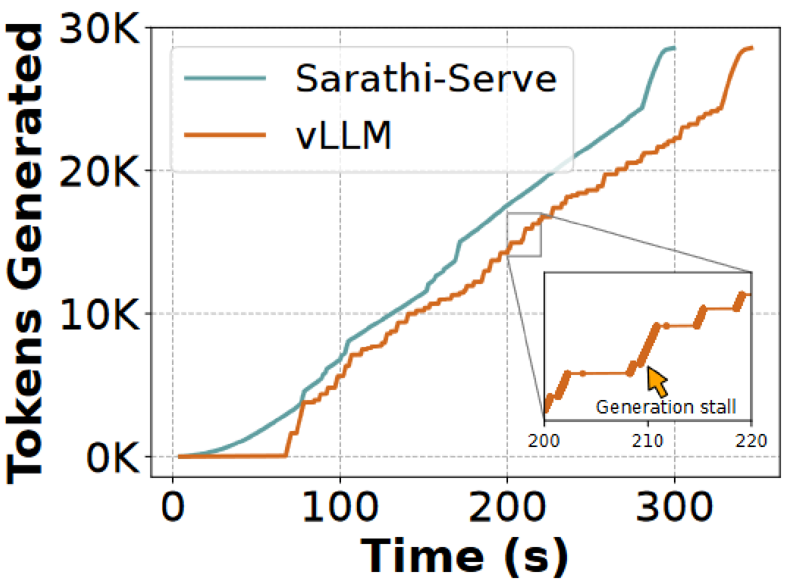


Prefill finished early -> Large batch -> High throughput
Decode stall -> High latency

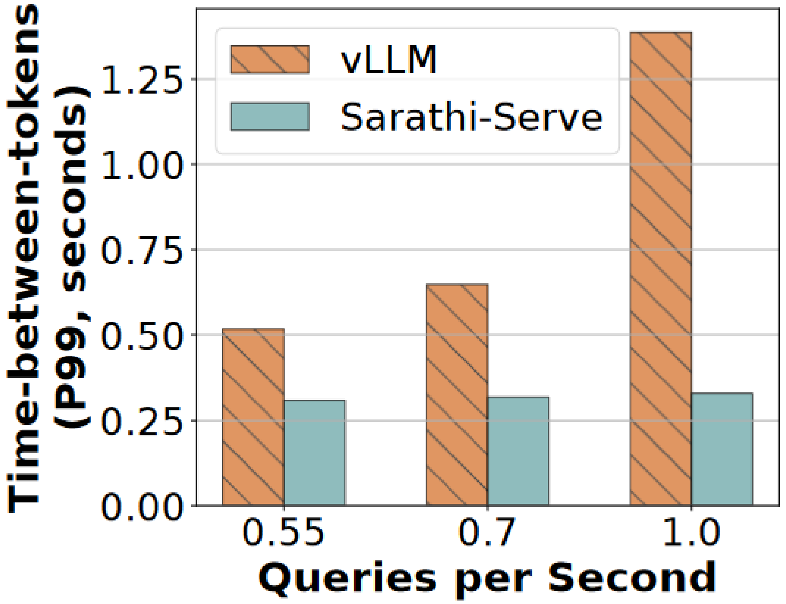
Decode without interference -> Low latency
Prefill stall -> Insufficient decode to batch -> Poor throughput

Throughput-latency tradeoff

- Generation stall can last over seconds
- Increasing load can significantly increase tail latency



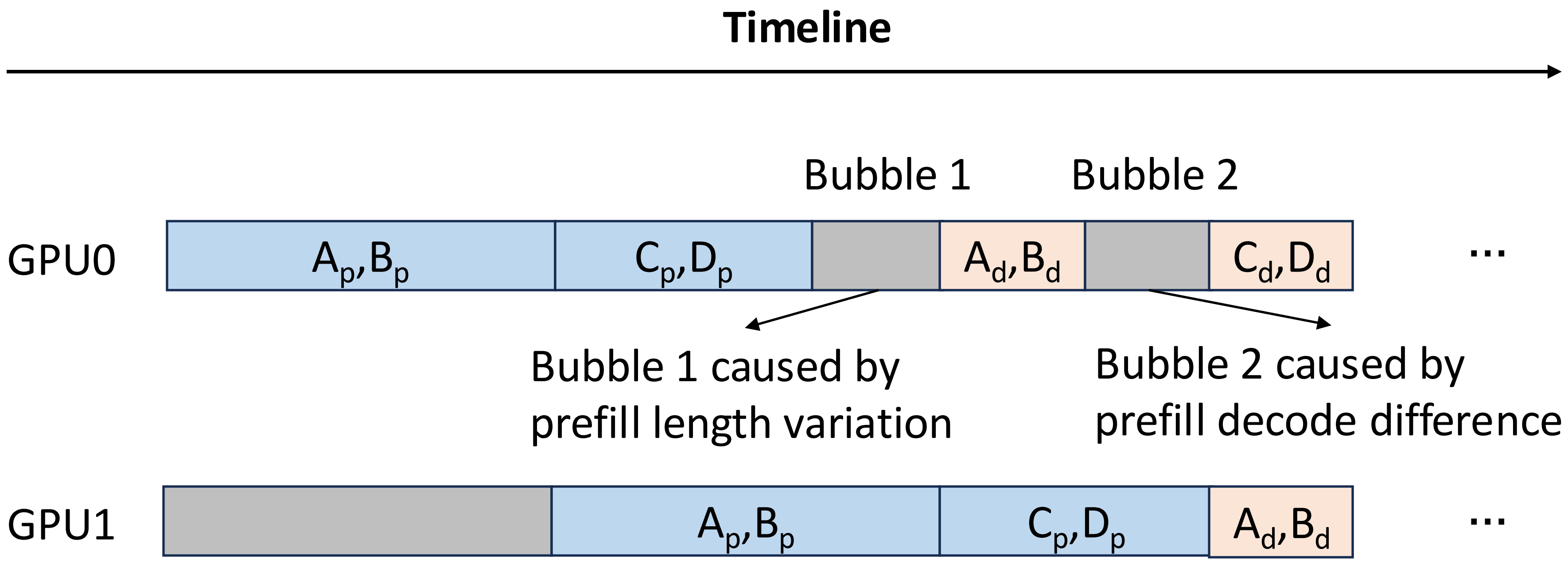
Generation stall



High tail latency

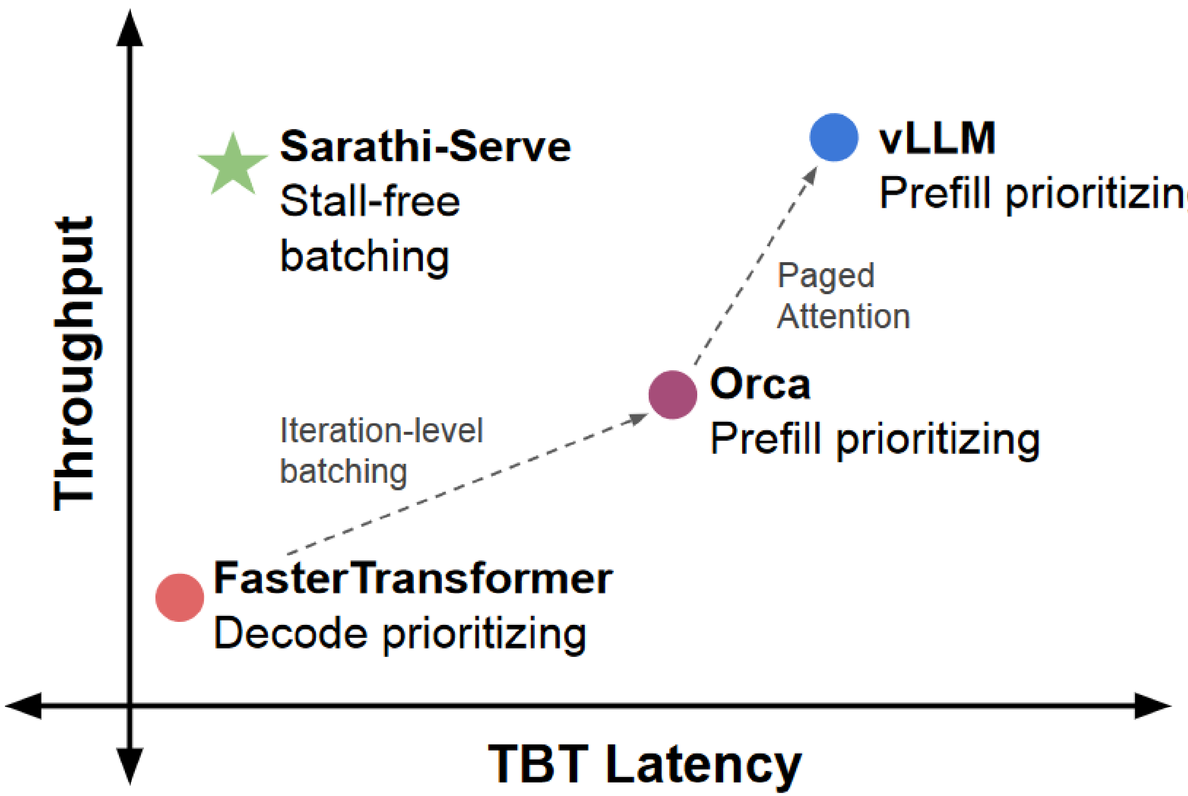
Pipeline Bubbles

- Pipeline bubbles can waste GPU cycles



Current LLM serving systems

- FasterTransformer
  - Decode-prioritizing
  - Poor throughput
- vLLM
  - Prefill-prioritizing
  - High latency
- Orca
  - Prefill-prioritizing
  - Prefill interferes decode
- All involve pipeline bubbles

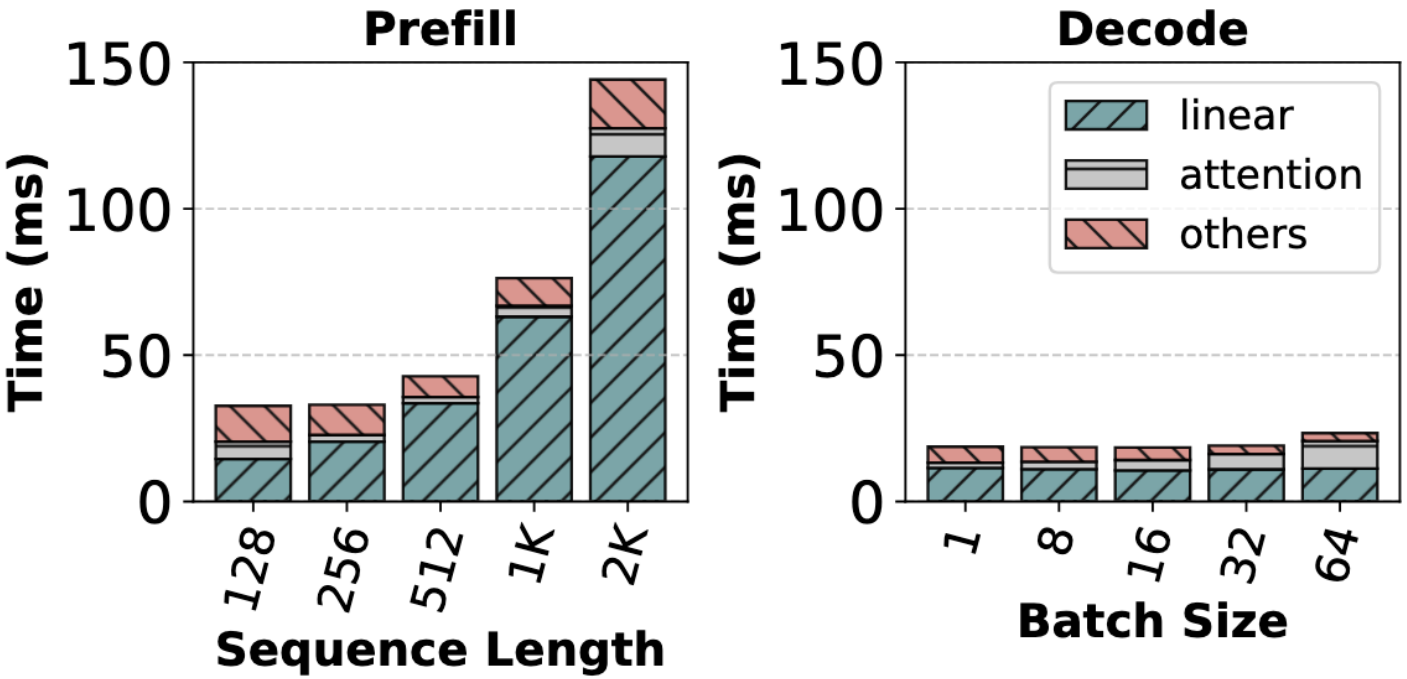


Outline

- Background and Existing Solutions
- Design and Implementation
- Evaluation
- Discussion

Cost Analysis of Prefill & Decode

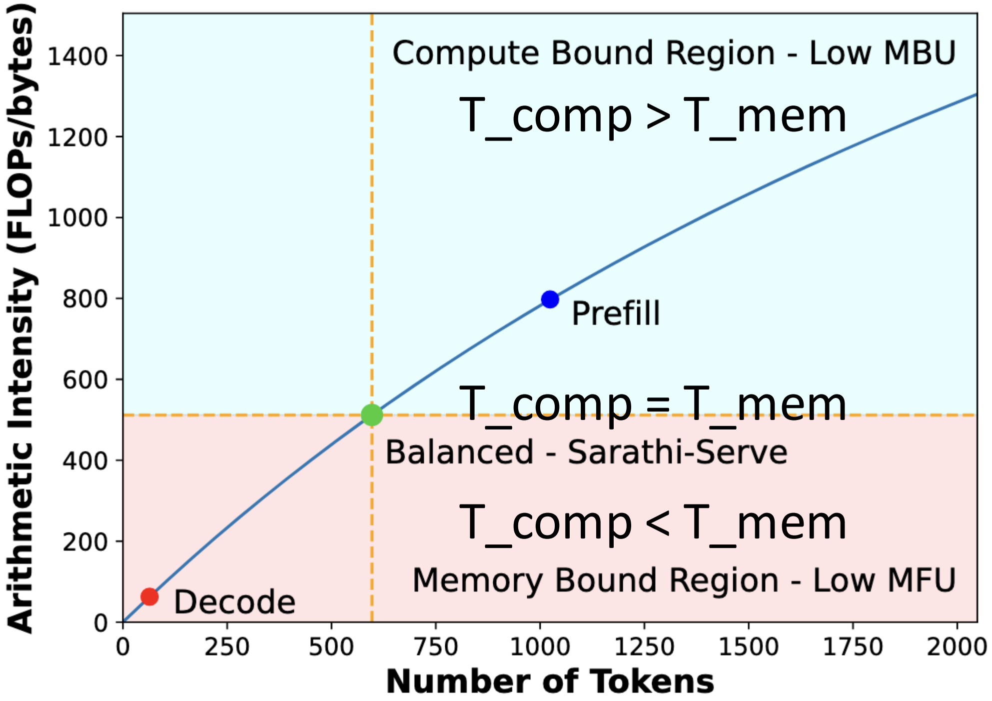
- Linear layer dominates in both prefill & decode
  - Therefore, we focus on cost of linear layer



Prefill and decode time with different input sizes for Mistral-7B running on single A100 GPU.

Cost Analysis of Prefill & Decode

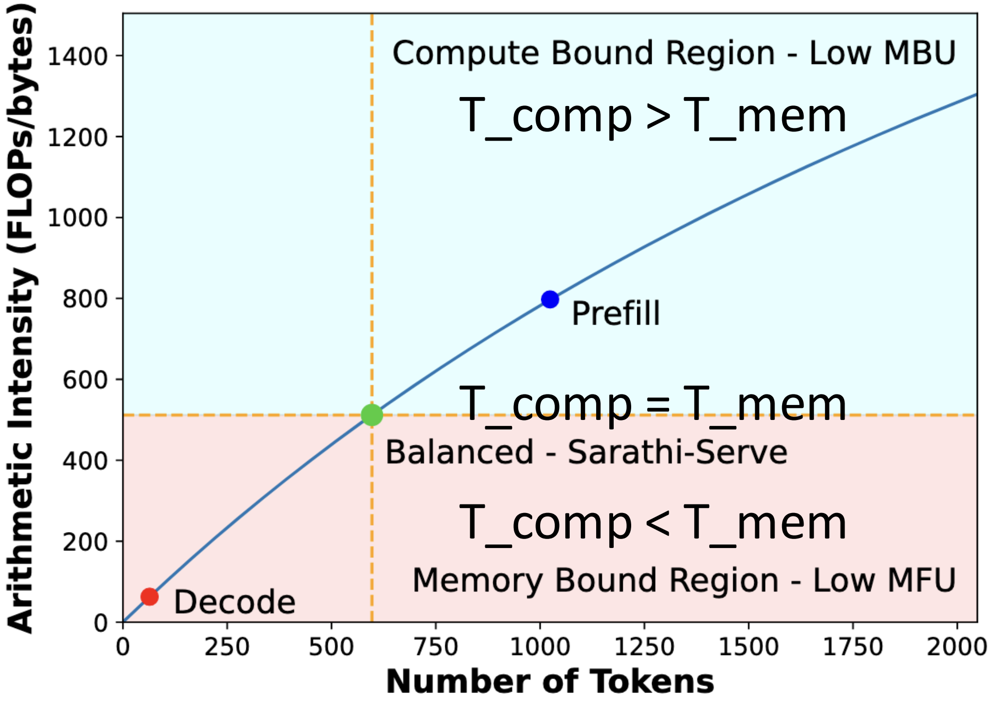
- Linear layer arithmetic intensity varies with number of tokens
  - Prefill: full prompt -> high arithmetic intensity (compute bound)
  - Decode: generated token -> low arithmetic intensity (memory bound)



Arithmetic intensity trend for LLaMA2-70B linear operations
with different number of token running on four A100s.

Cost Analysis of Prefill & Decode

- Linear layer arithmetic intensity varies with number of tokens
  - Prefill: full prompt -> high arithmetic intensity (compute bound)
  - Decode: generated token -> low arithmetic intensity (memory bound)

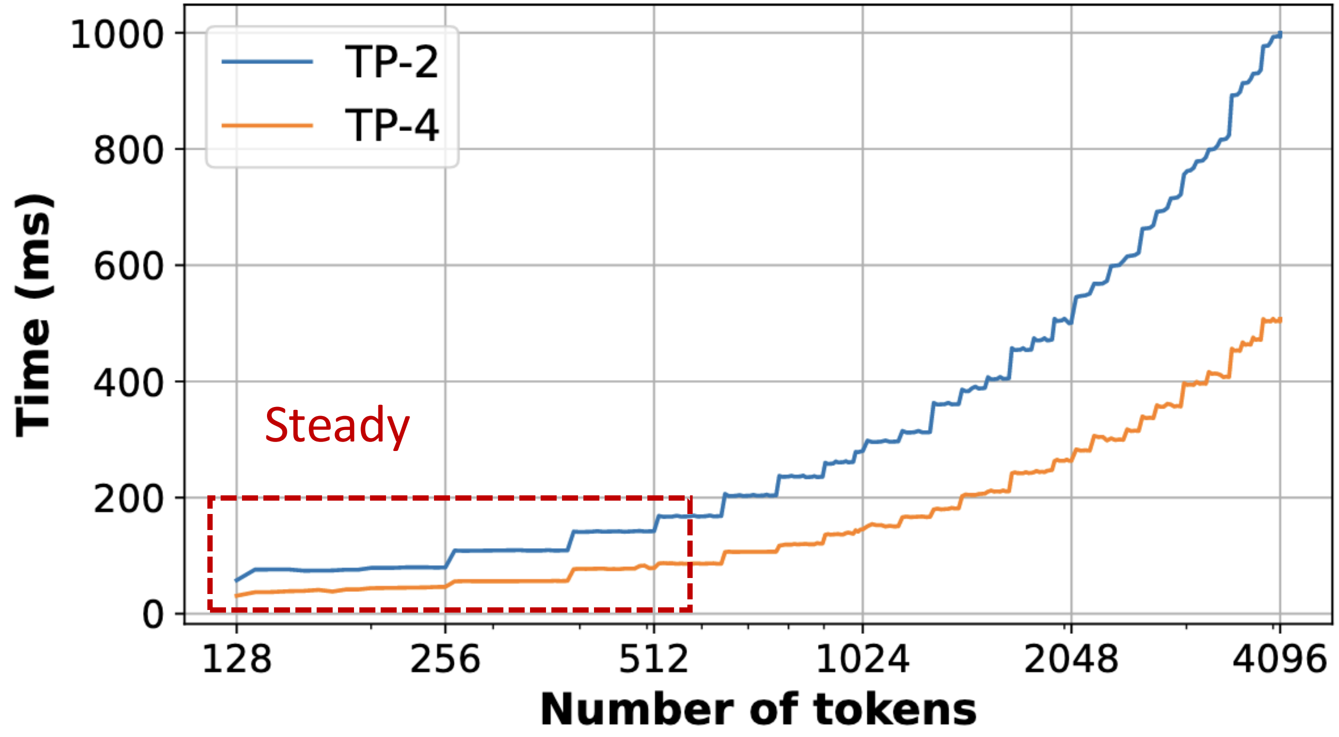


Batch prefill & decode together?

Arithmetic intensity trend for LLaMA2-70B linear operations with different number of token running on four A100s.

Cost Analysis of Prefill & Decode

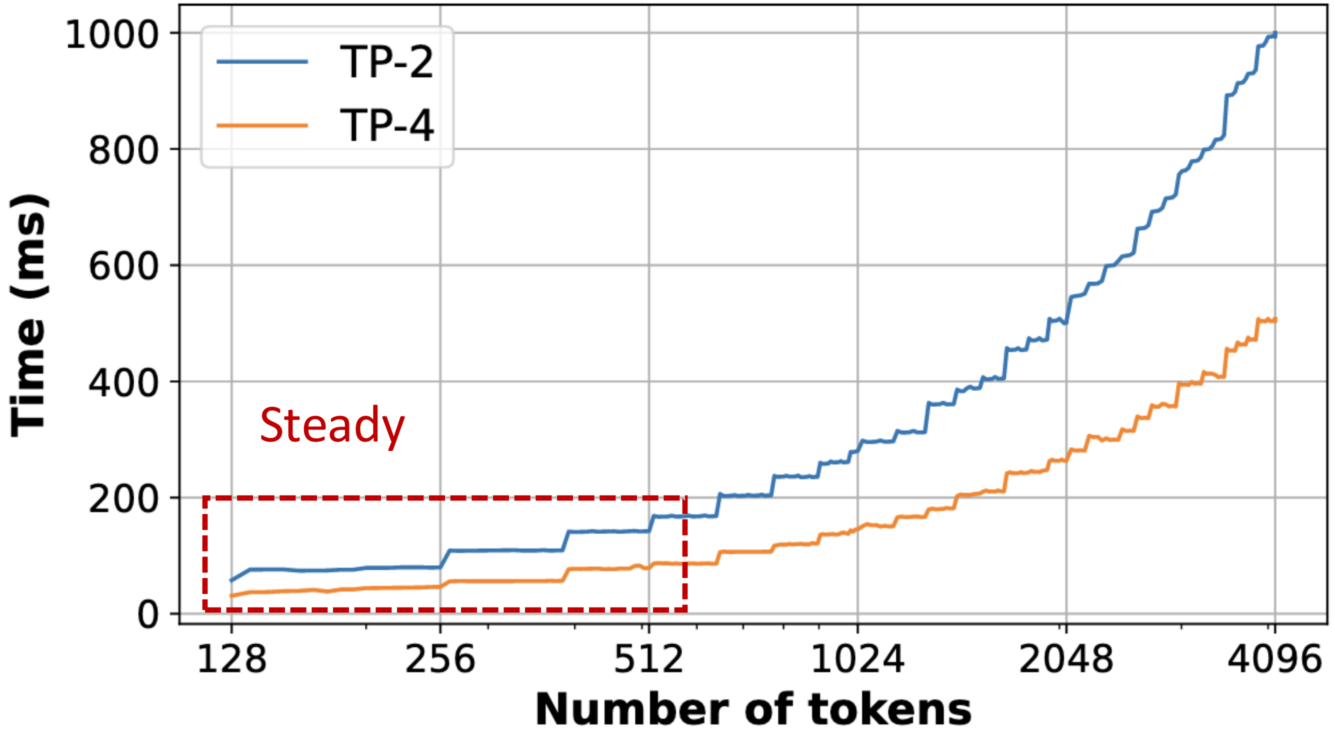
- Linear layer execution time as function of input length
  - Marginal increase for input length < 512 (memory bound)
  - Linear increase for input length > 512 (compute bound)



Linear layer execution time as function of number of tokens in a batch for LLaMA2-70B on A100(s) with different tensor parallel degrees.

Cost Analysis of Prefill & Decode

- Linear layer execution time as function of input length
  - Marginal increase for input length < 512 (memory bound)
  - Linear increase for input length > 512 (compute bound)



Linear layer execution time as function of number of tokens in a batch for LLaMA2-70B on A100(s) with different tensor parallel degrees.

Dataset	Prompt length	
	Median	P90
openchat_sharegpt4	1730	5696
arxiv_summarization	7059	12985

In practice, prefill often > 1024 tokens.

Brief Summary

- Arithmetic intensity
  - Prefill: high intensity
  - Decode: low intensity

Batching prefill & decode seems great!

- Execution time is decided by token count
  - Marginal increase initially -> marginal batching overhead
  - Then linear growth -> increasing batching overhead

Limiting token count ensures low latency and efficiency.

Design

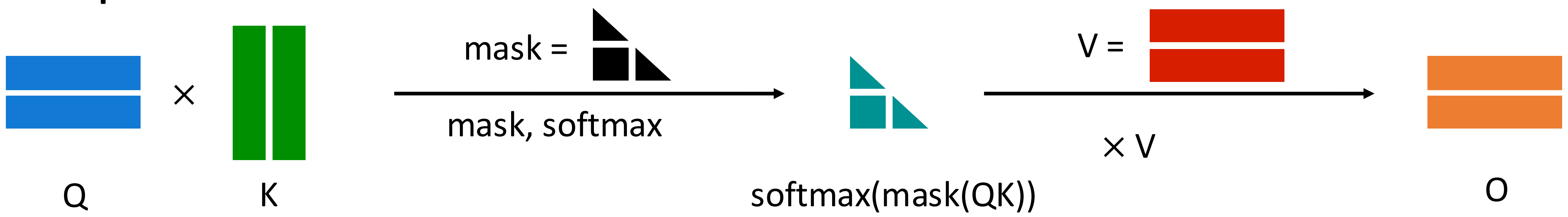
How can we batch prefill and decode while limiting token count?

[Chunked Prefill] Split long prefill into several short chunks

[Stall-free batching] Batch prefill and decode together w.o. stall

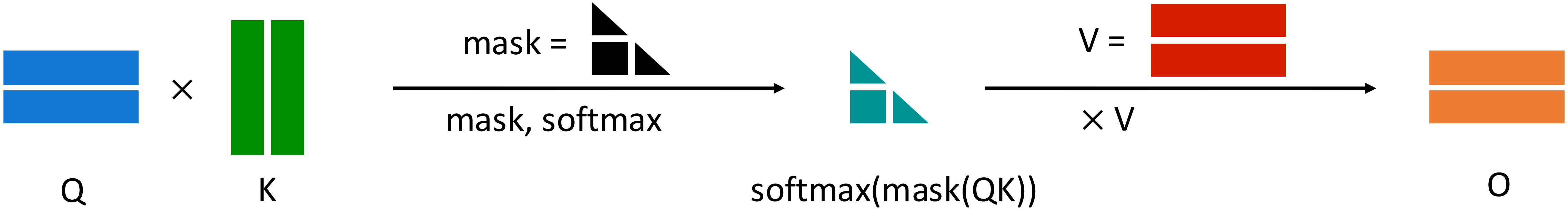
Design: Chunked Prefill

- Full prefill attention

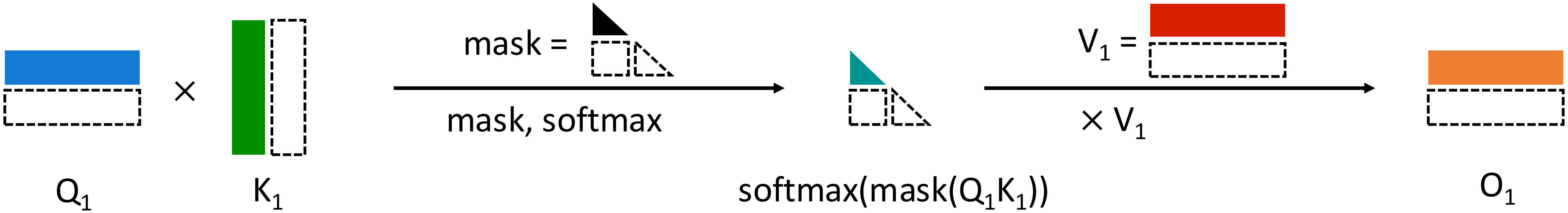


Design: Chunked Prefill

- Full prefill attention

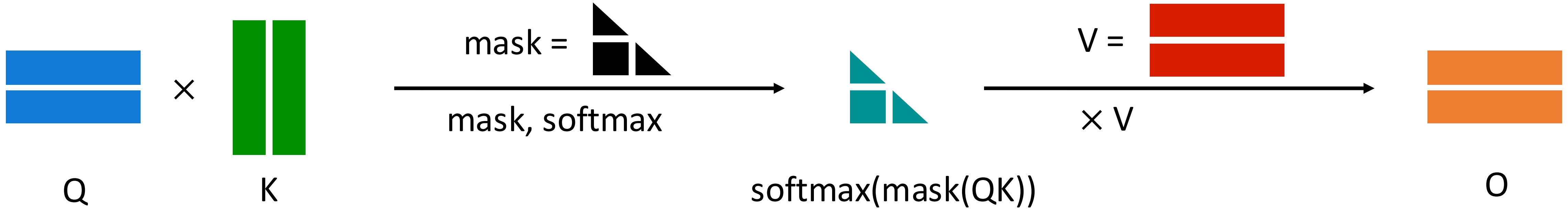


- Chunked prefill attention: chunk 1

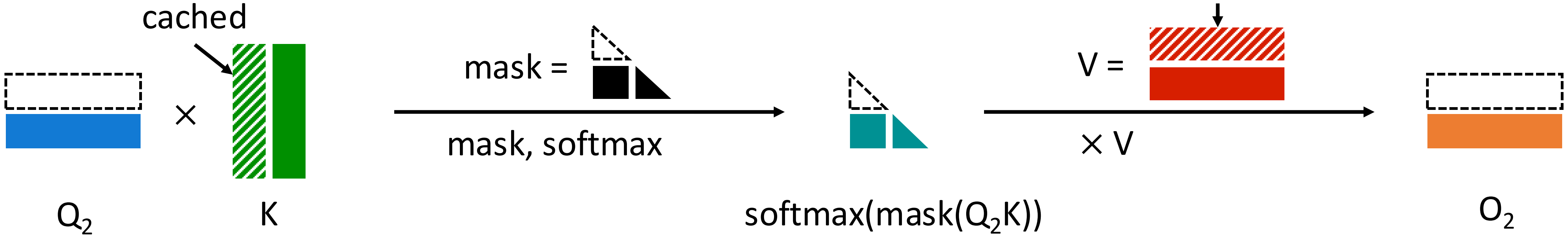


Design: Chunked Prefill

- Full prefill attention



- Chunked prefill attention: chunk 2



Full prefill = Conjunct (chunk 1, chunk 2)

Design: Chunked Prefill

- Chunked prefill attention masks

	k0	k1	k2	k3
q0	1	-	-	-
q1	1	1	-	-
q2	1	1	1	-
q3	1	1	1	1

1st chunked prefill

	k0	k1	k2	k3	k4	k5	k6	k7
q4	1	1	1	1	1	-	-	-
q5	1	1	1	1	1	1	-	-
q6	1	1	1	1	1	1	1	-
q7	1	1	1	1	1	1	1	1

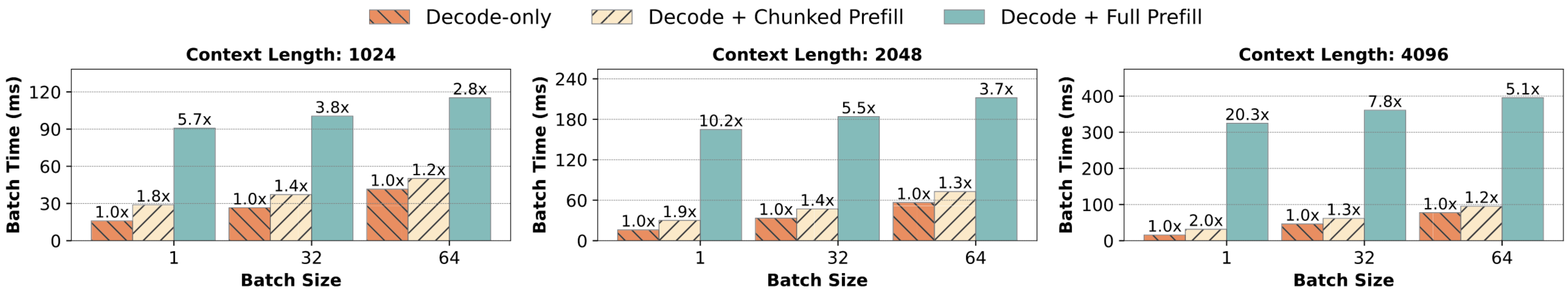
2nd chunked prefill

	k0	k1	k2	k3	k4	k5	k6	k7	k8	k9	k10	k11
q8	1	1	1	1	1	1	1	1	1	-	-	-
q9	1	1	1	1	1	1	1	1	1	1	-	-
q10	1	1	1	1	1	1	1	1	1	1	1	-
q11	1	1	1	1	1	1	1	1	1	1	1	1

3rd chunked prefill

Design: Stall-Free Batching

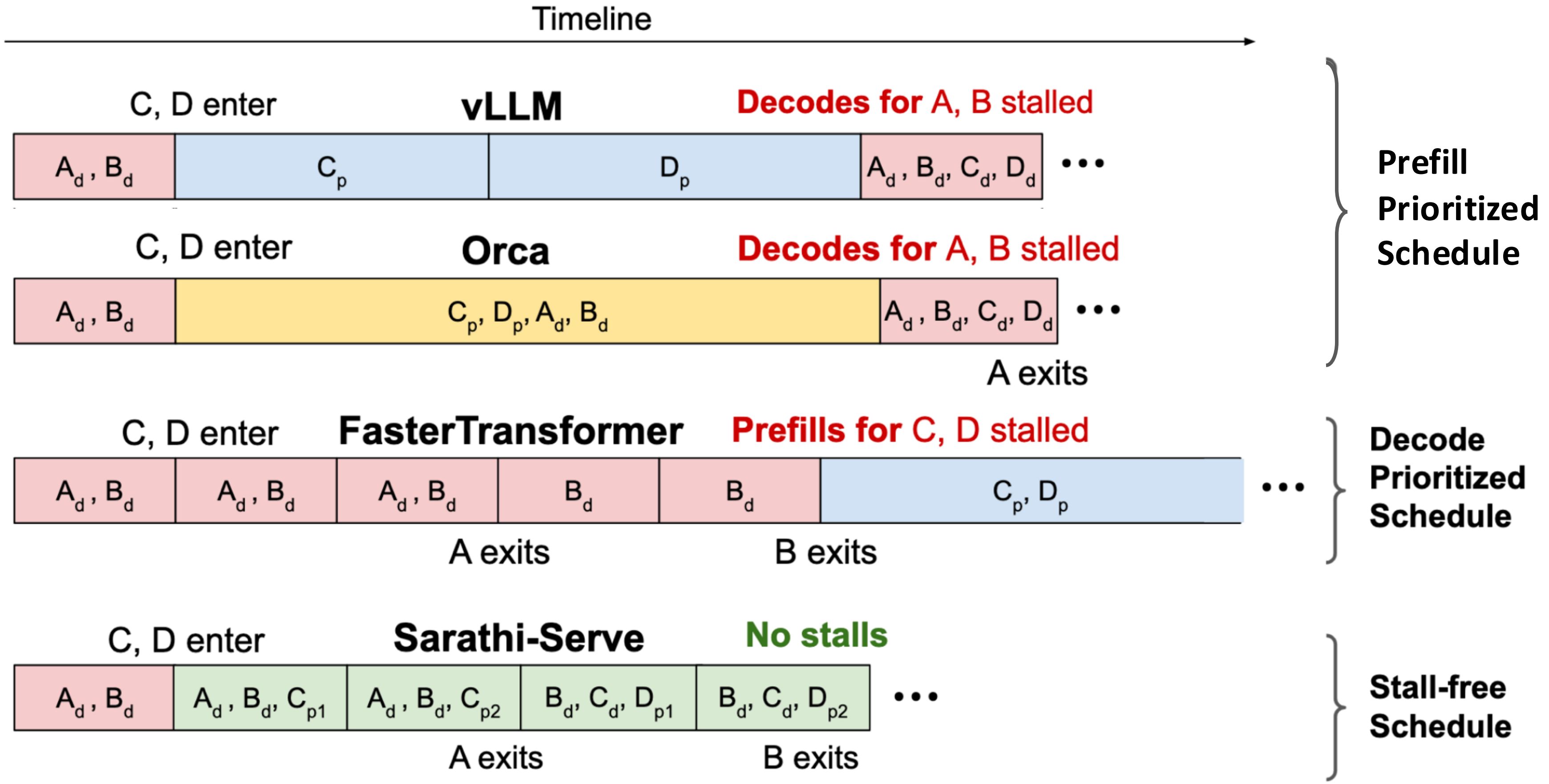
- Add prefill task to decode batch
  - Decode + full prefill
  - Decode + chunked prefill



Mistral-7B on one A100, with token count limitation for chunked prefill set to 256.

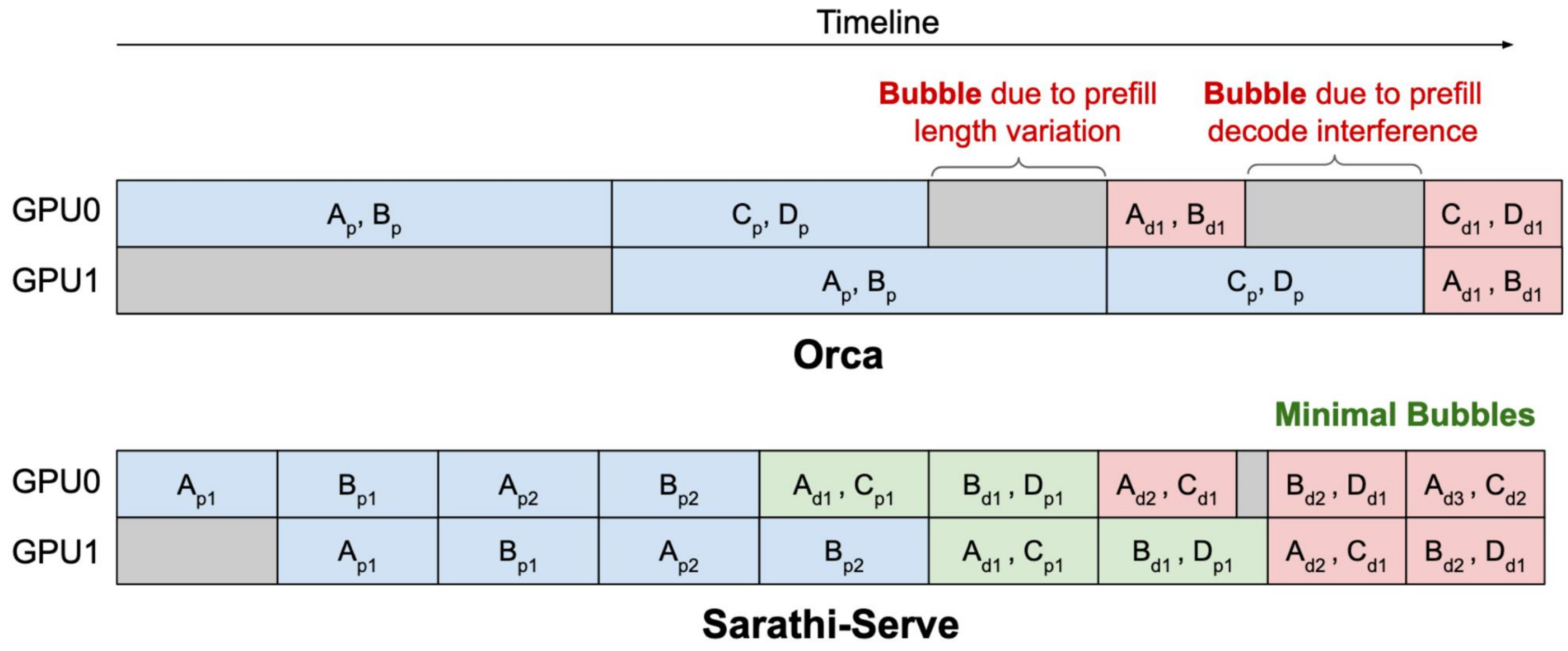
Design: Stall-Free Batching

- Stall-free batching coalesced with chunked prefill
  - Limit token count per batch to a certain value (*token budget*)
  - Goodness 1: efficiency for both prefill and decode



Design: Stall-Free Batching

- Stall-free batching coalesced with chunked prefill
  - Limit token count per batch to a certain value (*token budget*)
  - Goodness 2: pipeline bubble reduction



Practical Details

- Factors to Consider When Determining Token Budget
  1. TBT reduction -> smaller token budget
  2. Chunked prefill overhead -> larger token budget
     - Lower GPU utilization
     - Repeated KV cache access
  3. Tile-quantization -> token budget divided by tile size
  4. Pipeline bubble -> smaller token budget
- Implementation
  - Based on vLLM
  - Paged chunk prefill kernel: FlashAttention v2 & FlashInfer
  - Communication in TP & PP: NCCL

Outline

- Background and Existing Solutions
- Design and Implementation
- Evaluation
- Discussion

Evaluation: Setup

- Models, GPUs and SLOs

Model	Attention Mechanism	GPU Configuration	Memory Total (per-GPU)	relaxed SLO P99 TBT (s)	strict SLO P99 TBT (s)
Mistral-7B	GQA-SW	1 A100	80GB (80GB)	0.5	0.1
Yi-34B	GQA	2 A100 (TP2)	160GB (80GB)	1	0.2
LLaMA2-70B	GQA	8 A40 (TP4-PP2)	384GB (48GB)	5	1
Falcon-180B	GQA	4 A100 x 2 nodes (TP4-PP2)	640GB (80GB)	5	1

- KV reduction
  - GQA (Grouped-Query Attention): share KV across different heads of Q
  - SW (sliding window): limit attention context to fix length
- SLO
  - SLO for P99 TBT is set to 5x (strict) and 25x (relaxed) decode execution time without interference
  - TTFT is not included in SLO

Evaluation: Setup

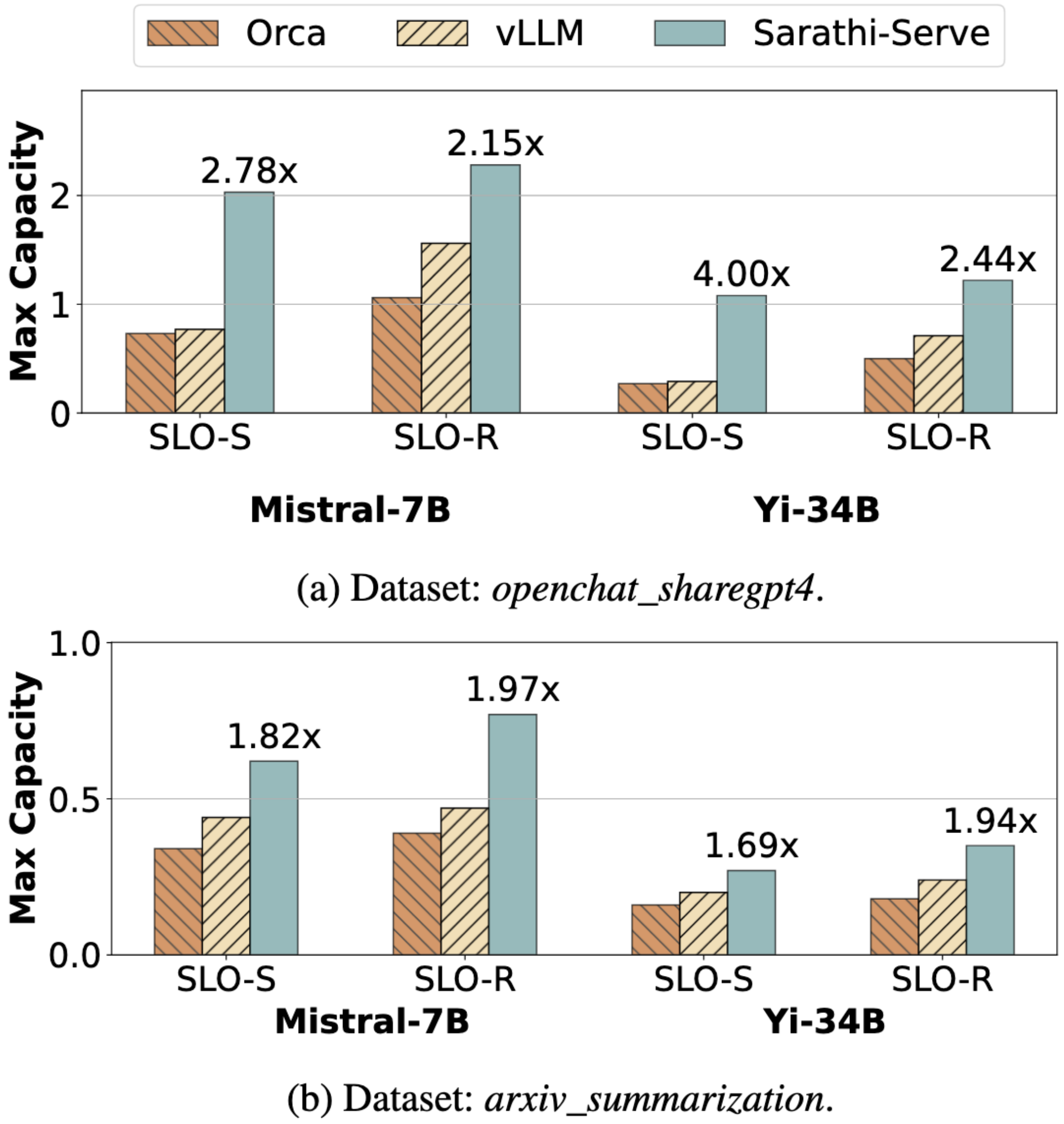
- Workload
  - Sampled from following datasets
  - Generated by a Poisson Process

Dataset	Prompt Tokens			Output Tokens		
	Median	P90	Std.	Median	P90	Std.
openchat_sharegpt4	1730	5696	2088	415	834	101
arxiv_summarization	7059	12985	3638	208	371	265

- Baseline
  - vLLM and Orca
  - *Where is FasterTransformer?*

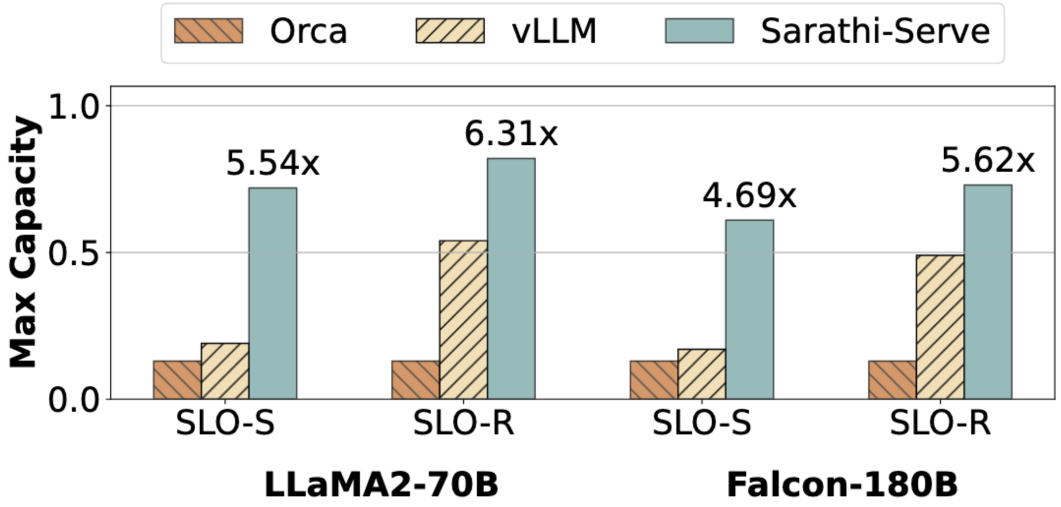
Evaluation: Capacity (RPS)

- Capacity improvement
  - up to 4.0x (compared to Orca)
  - up to 3.7x (compared to vLLM)
- SLO-S vs SLO-R
  - Orca and vLLM improves a lot
    - Decode stall hurts P99 TBT
  - Sarathi performs similarly
    - Various token budget
    - 2048 for relaxed SLO , 512 for strict SLO
    - Tight token budget helps tail latency

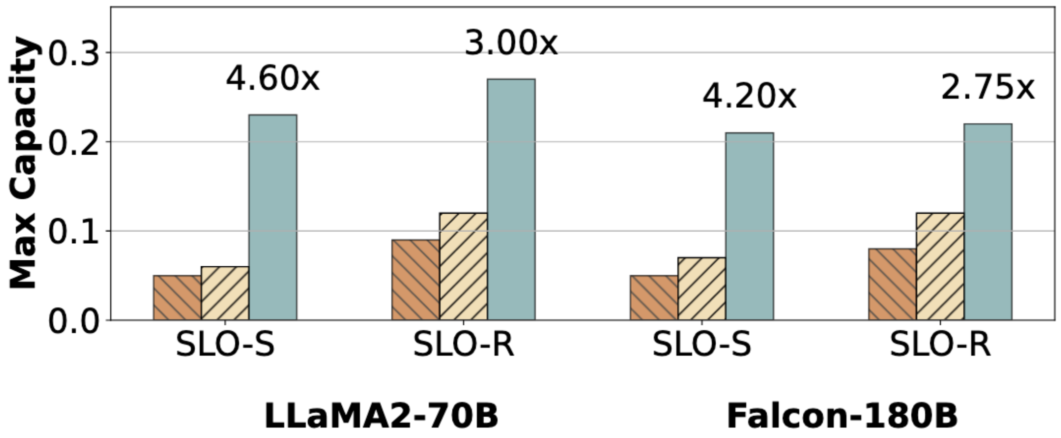


Evaluation: Capacity (RPS)

- PP = 2
- Capacity gain
  - up to 6.3x (compared to Orca)
  - up to 4.3x (compared to vLLM)
- Increment in capacity gain
  - Pipeline bubble reduction
- vLLM always outperforms Orca
  - Orca batch policy increases tail latency
  - Orca is not equipped with Paged-Attention



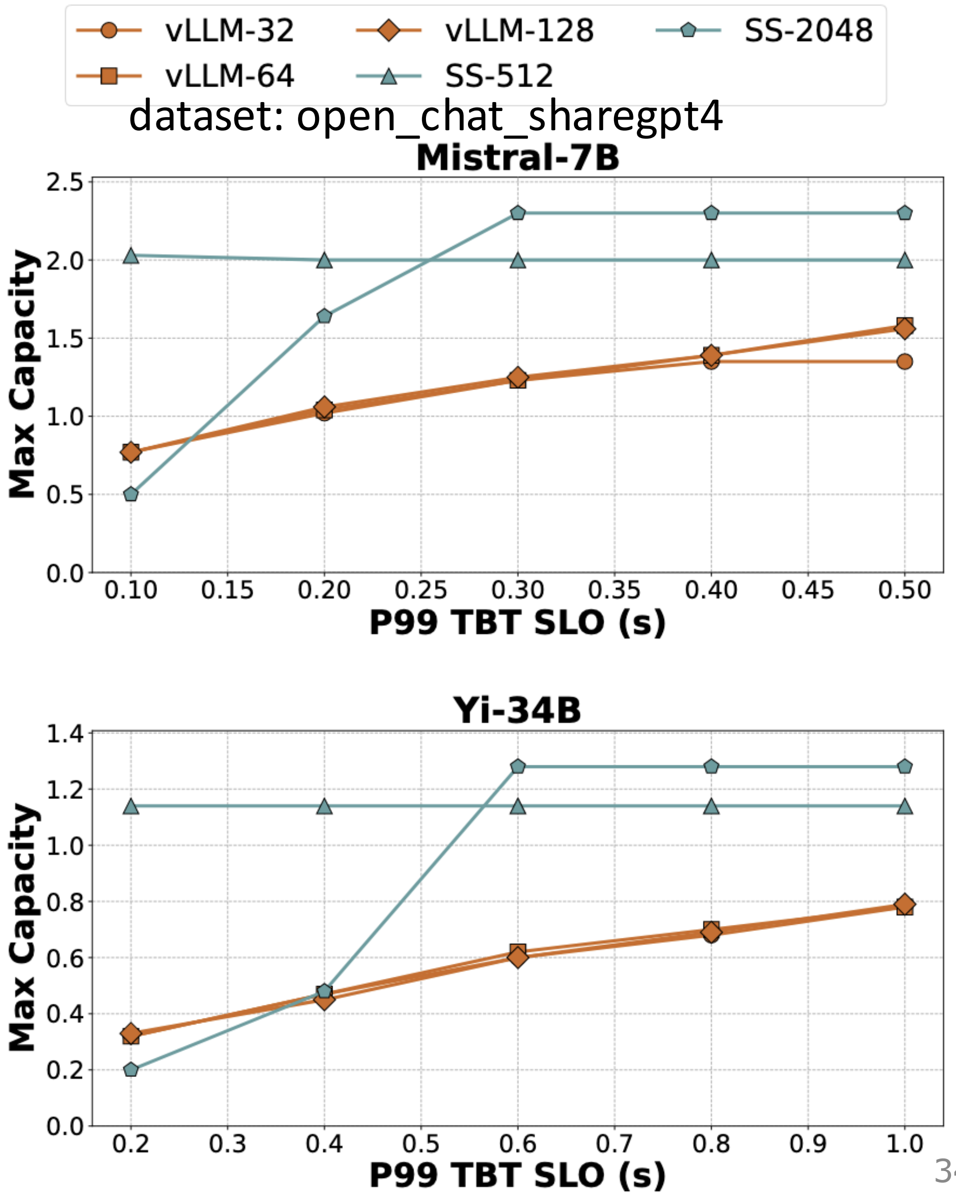
(a) Dataset: *openchat_sharegpt4*.



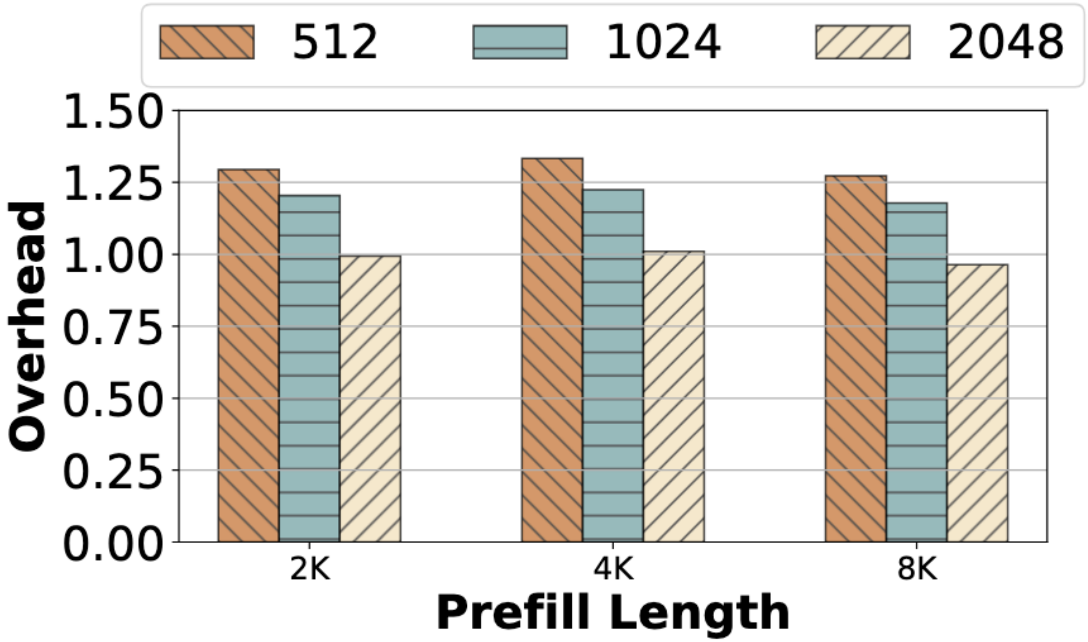
(b) Dataset: *arxiv_summarization*.

Evaluation: Throughput-latency Tradeoff

- Setup
  - vLLM with different batch size
  - Sarathi with different token budget
- vLLM
  - Capacity drops under strict SLO
  - Generation stall
- Sarathi-Serve
  - SS-512 performs consistently well
  - SS-2048 performs better under relaxed SLO
  - Choose optimal token budget for various SLO



Evaluation: Chunked Prefill Overhead



Overhead of chunked-prefills in prefill computation for Yi-34B (TP-2) normalized to the cost of no-chunking.

- Overhead is almost unchanged across different prefill length
- Smaller chunk -> higher overhead (from repeated KV access)
- token budget 512: about 25% overhead
- token budget 2048: negligible overhead

Evaluation: Ablation Study

Scheduler	openchat_sharegpt4		arxiv_summarization	
	P50 TTFT	P99 TBT	P50 TTFT	P99 TBT
hybrid-batching-only	0.53	0.68	3.78	1.38
chunked-prefill-only	1.04	0.17	5.38	0.20
sarathi-serve (combined)	0.76	0.14	3.90	0.17

TTFT and TBT latency measured in seconds for Yi-34B TP2 with a token budget of 1024.

- Impact of individual techniques
  - Hybrid-batching: prefill prioritizing, bad P99 TBT
  - Chunked-prefill: decode prioritizing, bad P50 TTFT
  - Sarathi-serve: optimal P99 TBT as well as P50 TTFT

Outline

- Background and Existing Solutions
- Design and Implementation
- Evaluation
- Discussion

Discussion

- Pros
  - ◆ Identify the throughput-latency trade-off in LLM serving
  - ◆ Comprehensive analysis of cost of prefill and decode
  - ◆ Chunked prefill reduces batch latency with marginal overhead
  - ◆ Stall-free batching unifies different kinds of batch (P, D, and P+D)
- Cons
  - ◆ Insufficient comparison to FasterTransformer
  - ◆ Capacity does not consider SLO for TTFT, only TBT