gLLM: Global Balanced Pipeline Parallelism Systems for Distributed LLMs Serving with Token Throttling

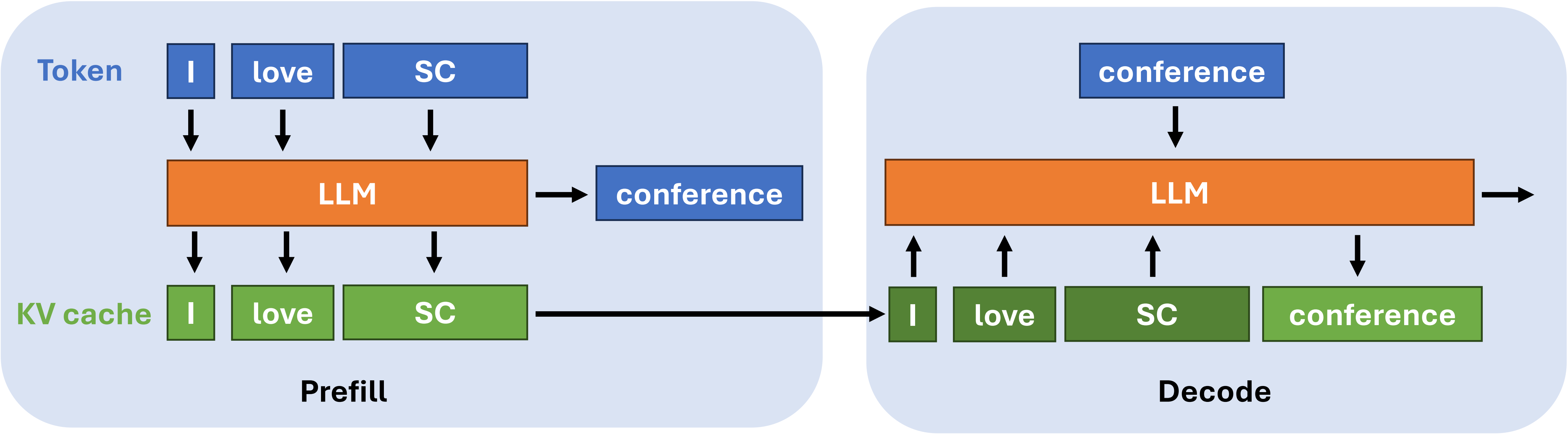
Tianyu Guo, Xianwei Zhang, Jiangsu Du, Zhiguang Chen,
Nong Xiao, Yutong Lu

Email: guoty9@mail2.sysu.edu.cn

Time: Thursday, 20 November 2025
11:37am-12:00pm CST

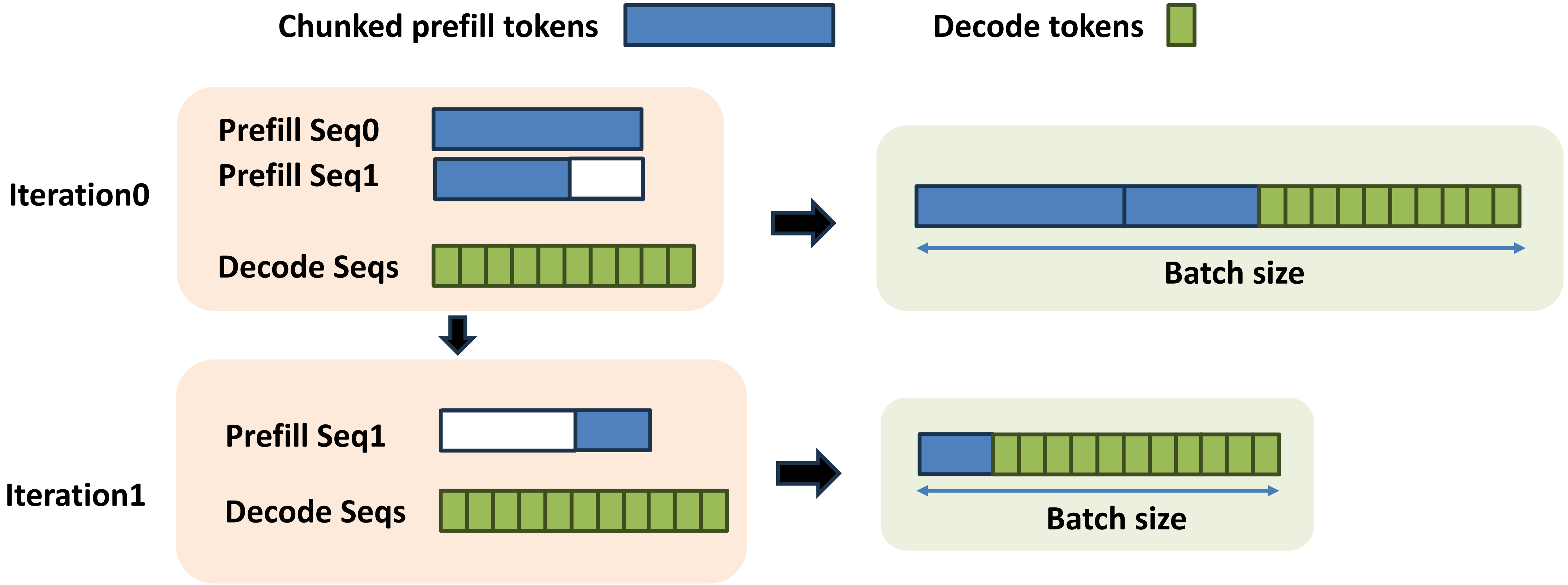
Background: LLM Inference and KV Cache

- LLM Inference: Autoregressive Decoding with KV cache
  - Decoding: Next token prediction based on previous tokens
  - Autoregressive: Generate token one by one
  - KV cache: Intermediate data kept for decoding



Background: Scheduling Policies

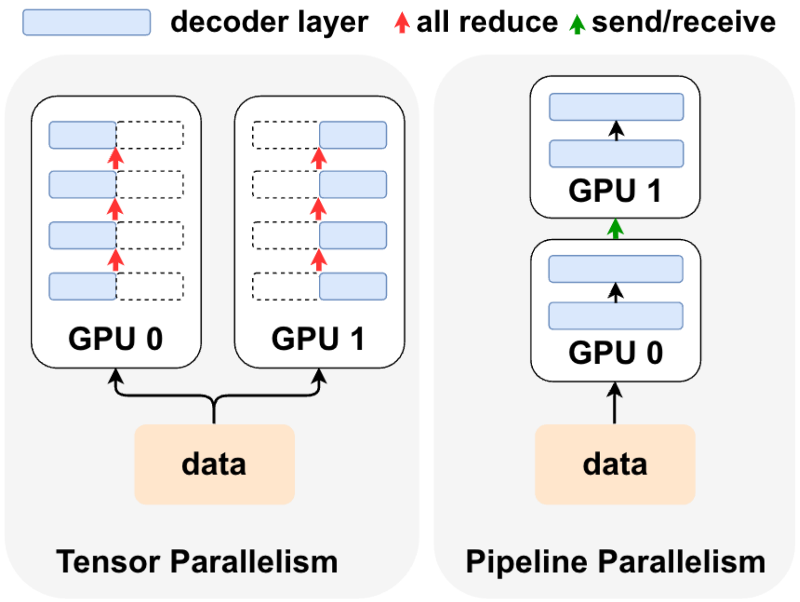
- Continuous batching[1]: Iteration-level request scheduling
- Sarathi-Serve[2] : Batch prefill tokens at **chunked** granularity with decoding tokens

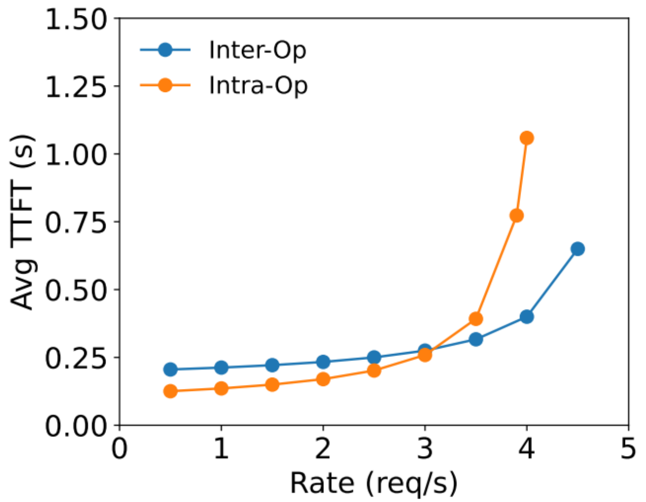


[1] Orca: A Distributed Serving System for Transformer-Based Generative Models
[2] Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve

Background: Parallelism Strategies and Distributed Serving

- Pipeline Parallelism: **Inter-layer** Model Partition
  - Lightweight communication demand (send/receive)
  - Throughput-oriented
- Tensor Parallelism: **Intra-layer** Model Partition
  - Heavy communication demand (all reduce)
  - Latency-oriented
- Distributed serving requires high inter-node bandwidth demands
  - Cross-node setups often adopt a pipeline parallel deployment

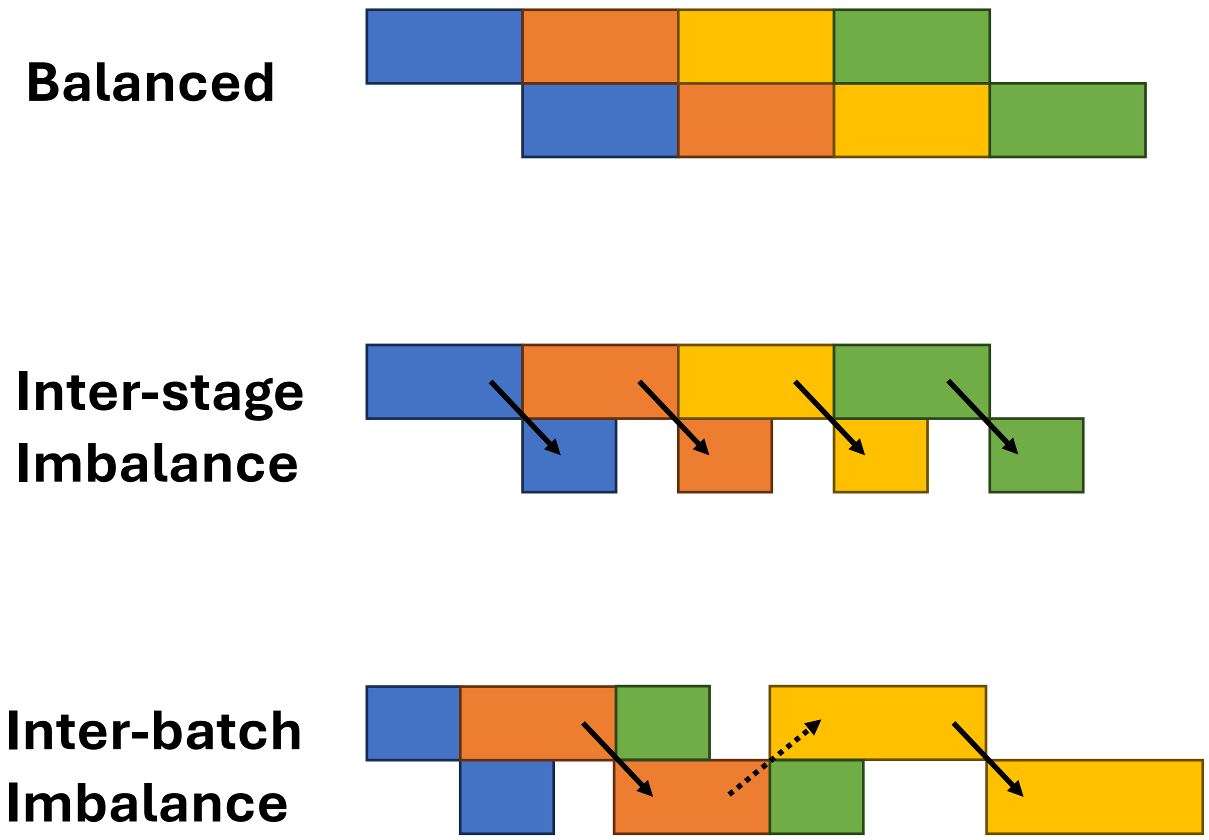




[1] DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving

Motivation: Pipeline Bubbles

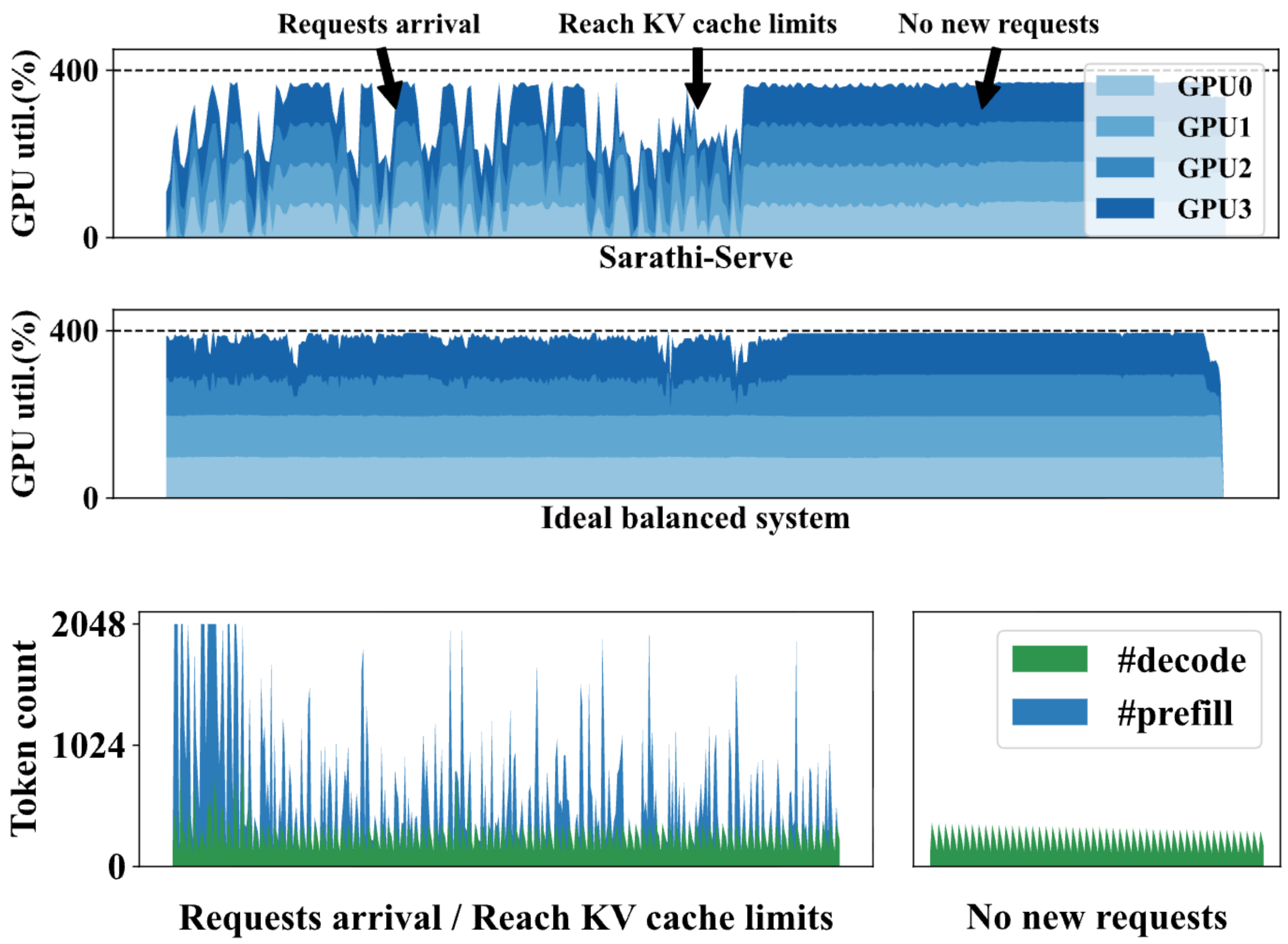
- Pipeline Bubbles
  - **Inter-stage** imbalance: Uneven computation distribution across pipeline stages
  - **Inter-batch** imbalance: Variation in computation requirements across different micro-batches
- Dependency
  - Inter-stage dependency
  - Inter-batch dependency



Balanced computation between micro-batches is required

Motivation: Fluctuation in Scheduled Token Count

- GPU is under-utilized caused by unbalanced scheduling
  - **Requests arrival / Reach KV cache limits**: Performance drops due to large fluctuations in prefill and decode tokens
  - **No new requests**: Minor performance degradation due to fluctuations in decode token count



Smooth token scheduling ensures balanced computation

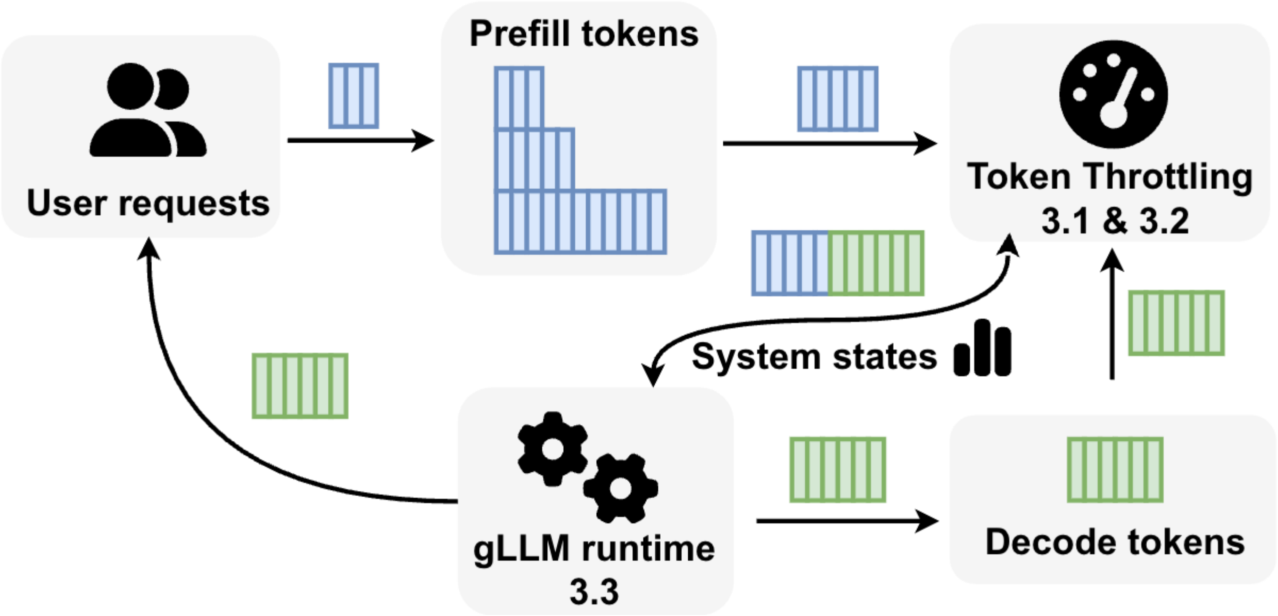
Motivation: Scheduling Demands

- Pipeline bubbles caused by fluctuation in scheduled token count
  - **Balanced** scheduling: the runtime of adjacent micro batch is similar
- Prefill and decode stages have distinct characteristics
  - **Decoupled** scheduling (run together): the numbers of scheduled prefill tokens and decode tokens do not interfere with each other
- Scheduling demands change over time
  - **Dynamic** scheduling: adjust prefill rate according to system state

An intelligent scheduling strategy is needed

Design: gLLM——Global Balanced Pipeline Parallelism System

- **Token Throttling** to achieve balanced computation
  - Decoupled scheduling for prefill and decode phase
  - Dynamically adjust scheduled token number
- **gLLM runtime**
  - Runtime tailored for pipeline parallelism

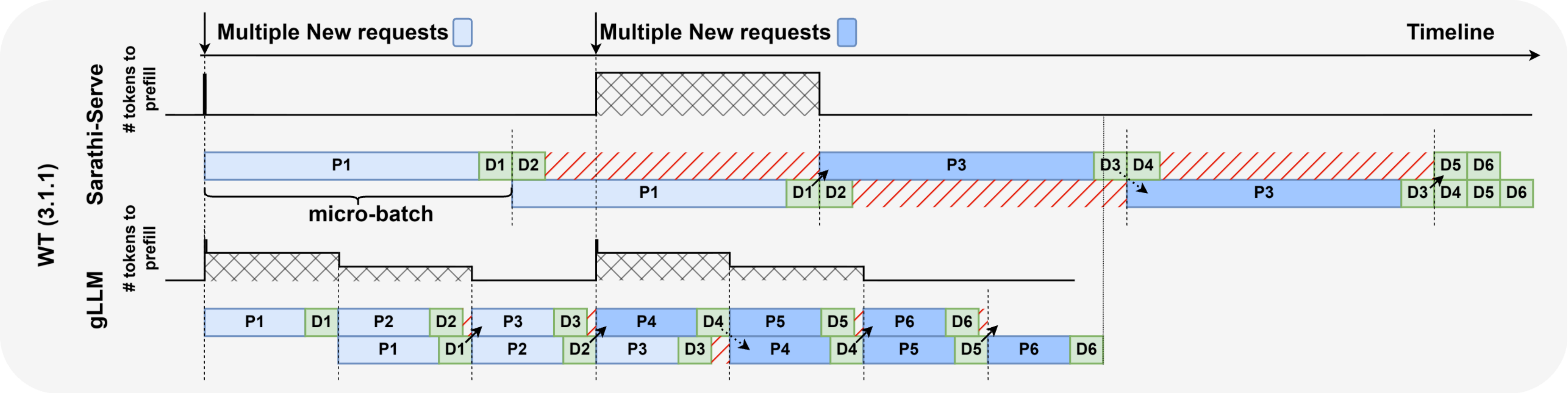


Design: Prefill Token Throttling——WT

- Throttling by **Tokens Count Awaiting Prefill**
  - Scheduled prefill tokens may fluctuate due to insufficient pending tokens
  - We compute scheduled prefill tokens ($\#P$) from waiting tokens ($\#WP$) and the number of iterations ($\#T$) to process all tokens waiting for prefill

$$\#P = \min(\max\left(\frac{\#WP}{\#T}, \#MinP\right), \#MaxP)$$

$\#MinP$/ $\#MaxP$: Minimum/Maximum scheduled prefill token count

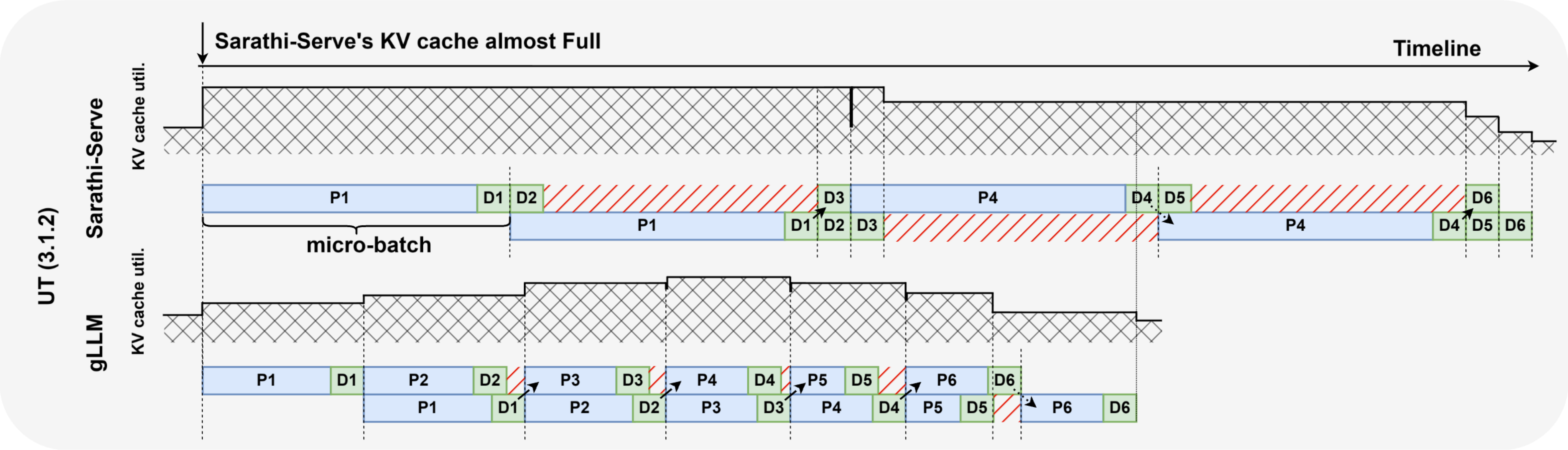


Design: Prefill Token Throttling——UT

- Throttling by **KV Cache Utilization Rate**
  - Scheduled prefill tokens may also fluctuate due to insufficient KV cache capacity
  - We compute $\#P$ from KV cache free rate ($\#KV_{free} \in [0,1]$)

$$\#P = \max(\#MaxP \times KV_{free}, \#MinP)$$

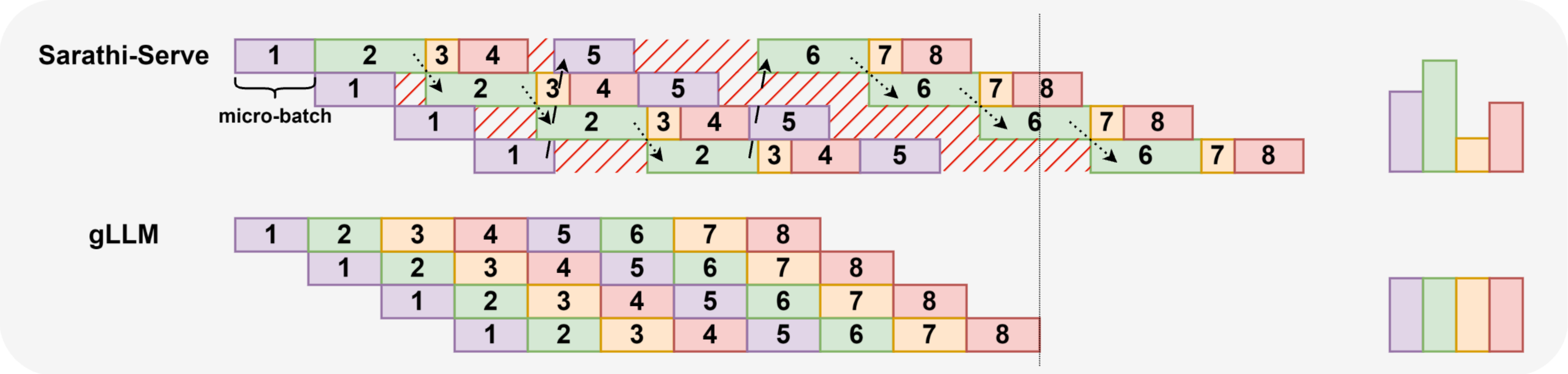
$\#MinP$/ $\#MaxP$: Minimum/Maximum scheduled prefill token count



Design: Decode Token Throttling

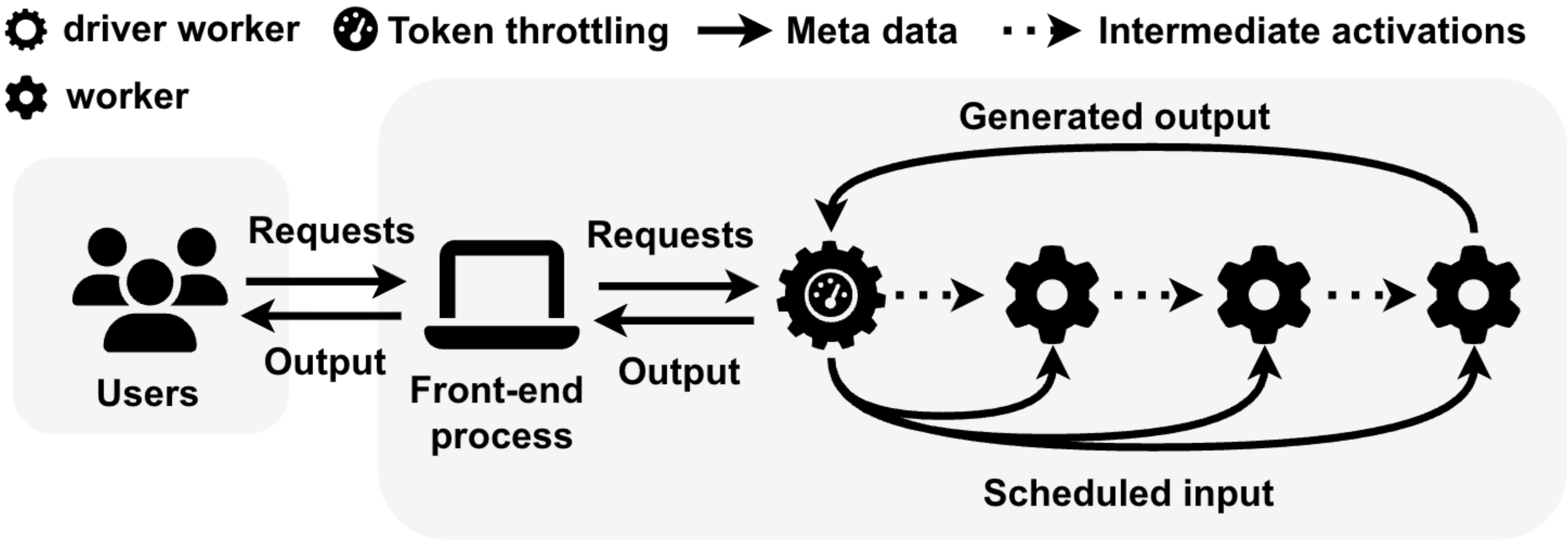
- Throttling by **Tokens Count Under Decode**
  - Scheduled decode token count ($\#D$) depends on the number of active decode requests ($\#RD$) and the pipeline stages ($\#PP_{depth}$)

$$\#D = \frac{\#RD}{\#PP_{depth}}$$



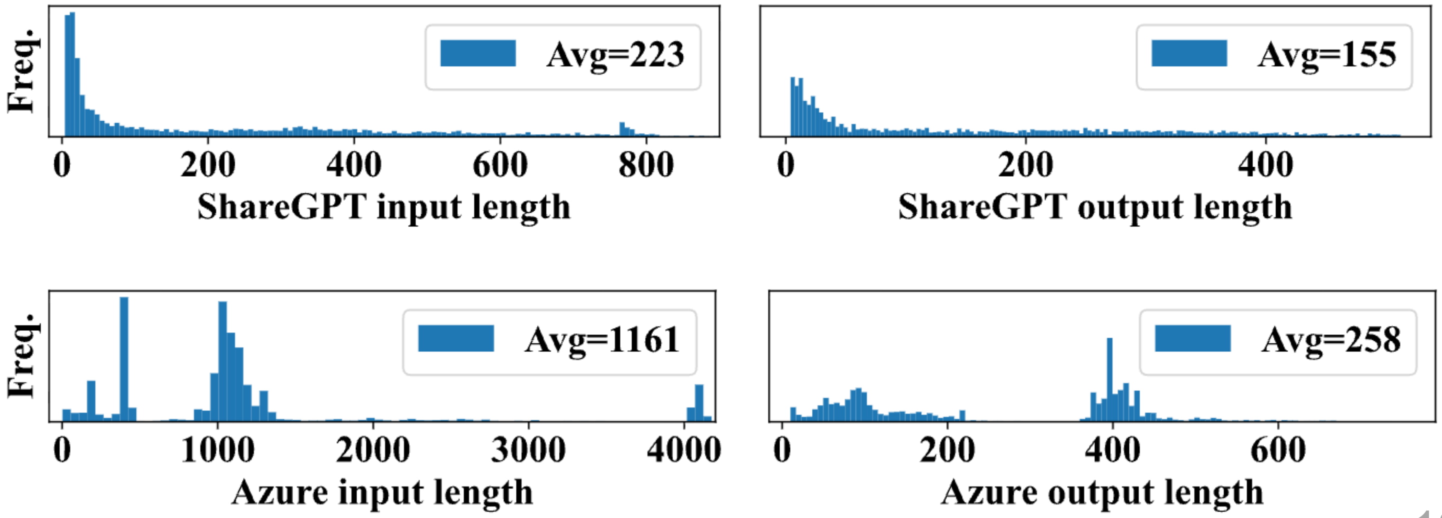
Design: gLLM Runtime

- gLLM runtime: An **asynchronous** runtime designed for pipeline parallelism
  - Non-blocking pipeline operations
  - Decoupled frontend-backend processing
  - Preemptive metadata scheduling



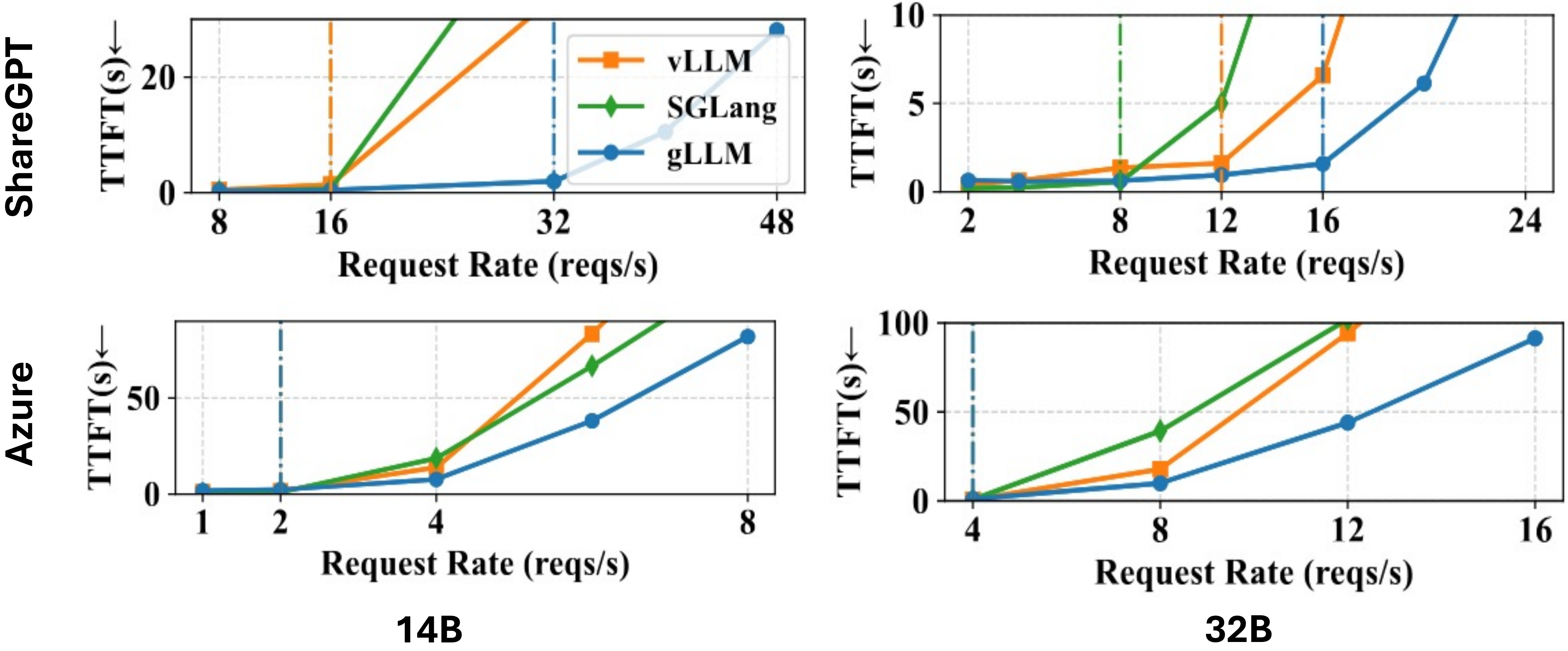
Evaluation: Experimental Setup

- Models: Qwen2.5 (14/32B) and Llama3.1(100B)
- Schemes:
  - vLLM (v0.8.1 V1) with pipeline parallelism
  - SGLang (v0.4.3.post2) with tensor parallelism
  - gLLM with pipeline parallelism
- Metrics
  - Time to first token (TTFT)
  - Time per output token (TPOT)
  - End to end latency (E2EL)
  - Throughput
- Workloads: ShareGPT and Azure



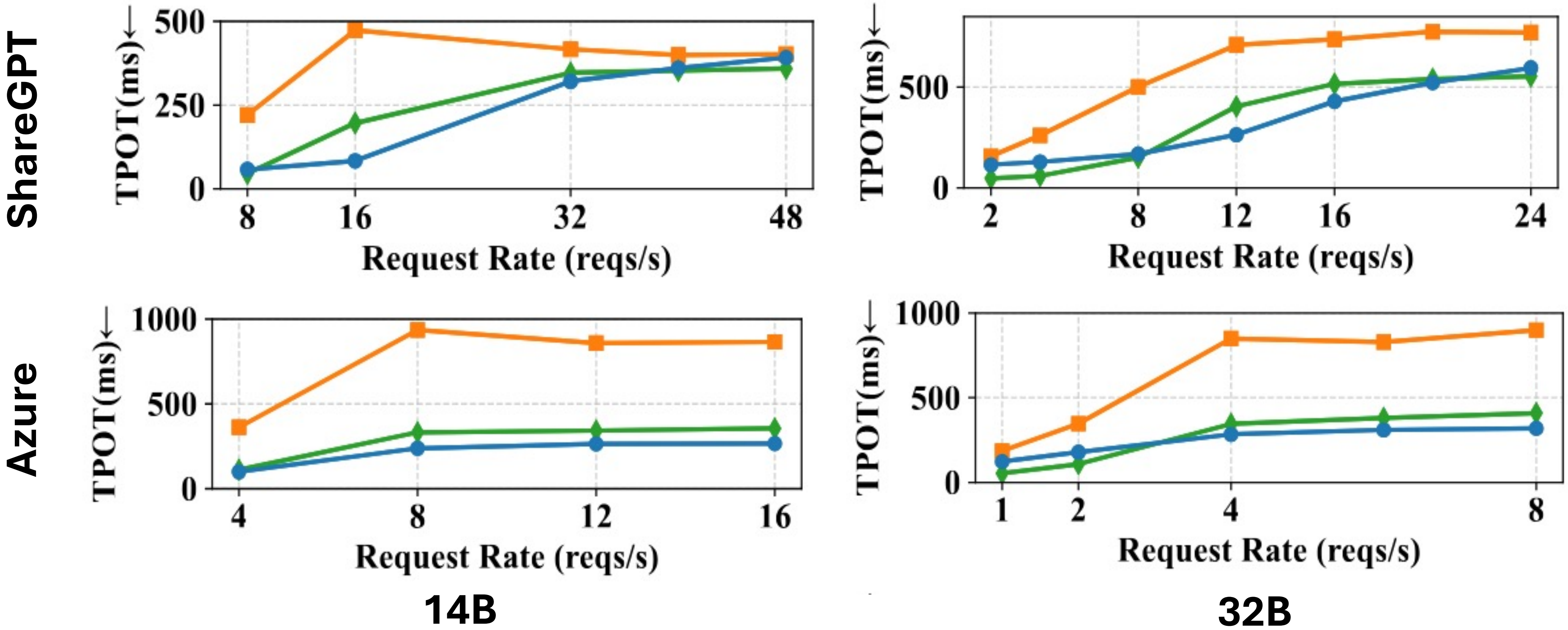
Evaluation: Latency and Throughput —— TTFT

- TTFT will rise significantly at some point due to requests queuing
  - gLLM reaches its turning point at 1-2× higher request rates



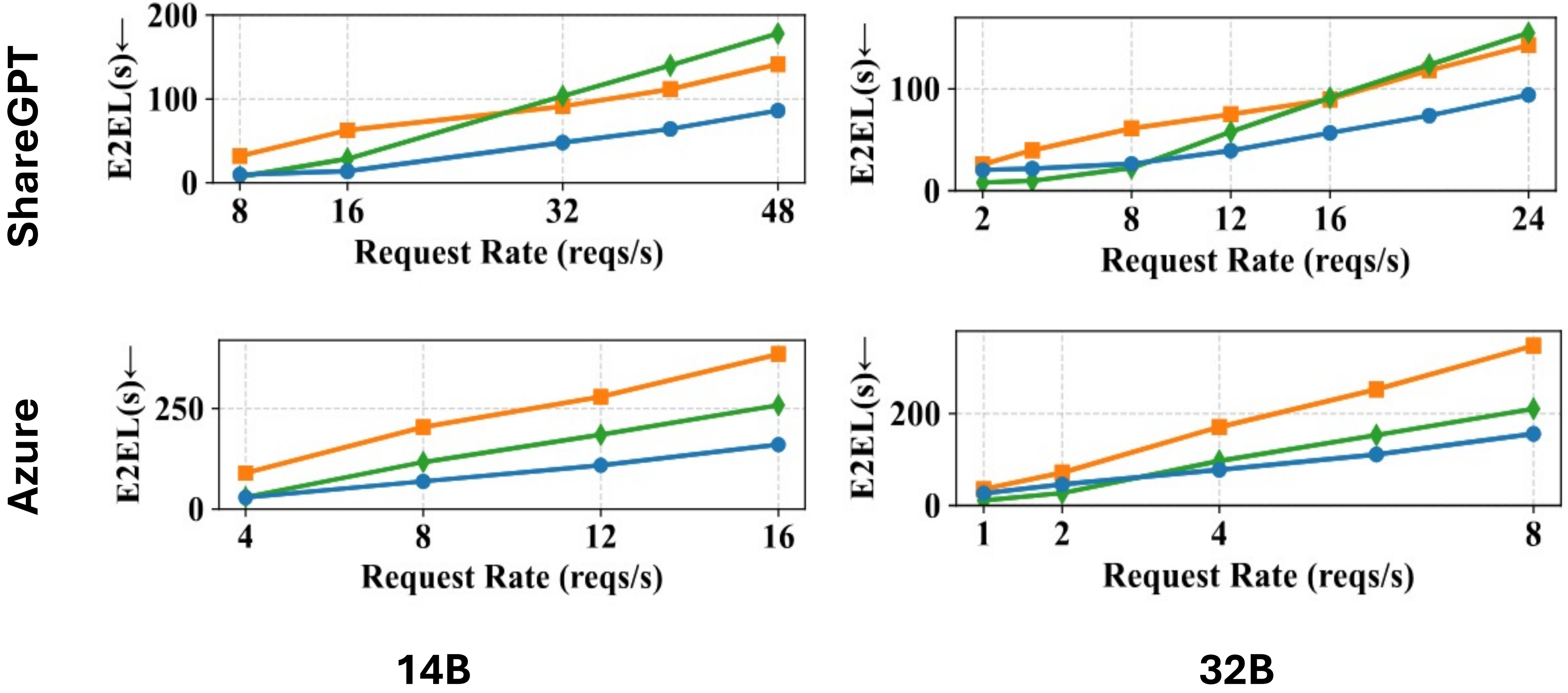
Evaluation: Latency and Throughput —— TPOT

- TPOT first increases with the request rate and then stabilizes as the batch size reaches its maximum
  - gLLM maintains an average 2%-10% lower TPOT in most cases



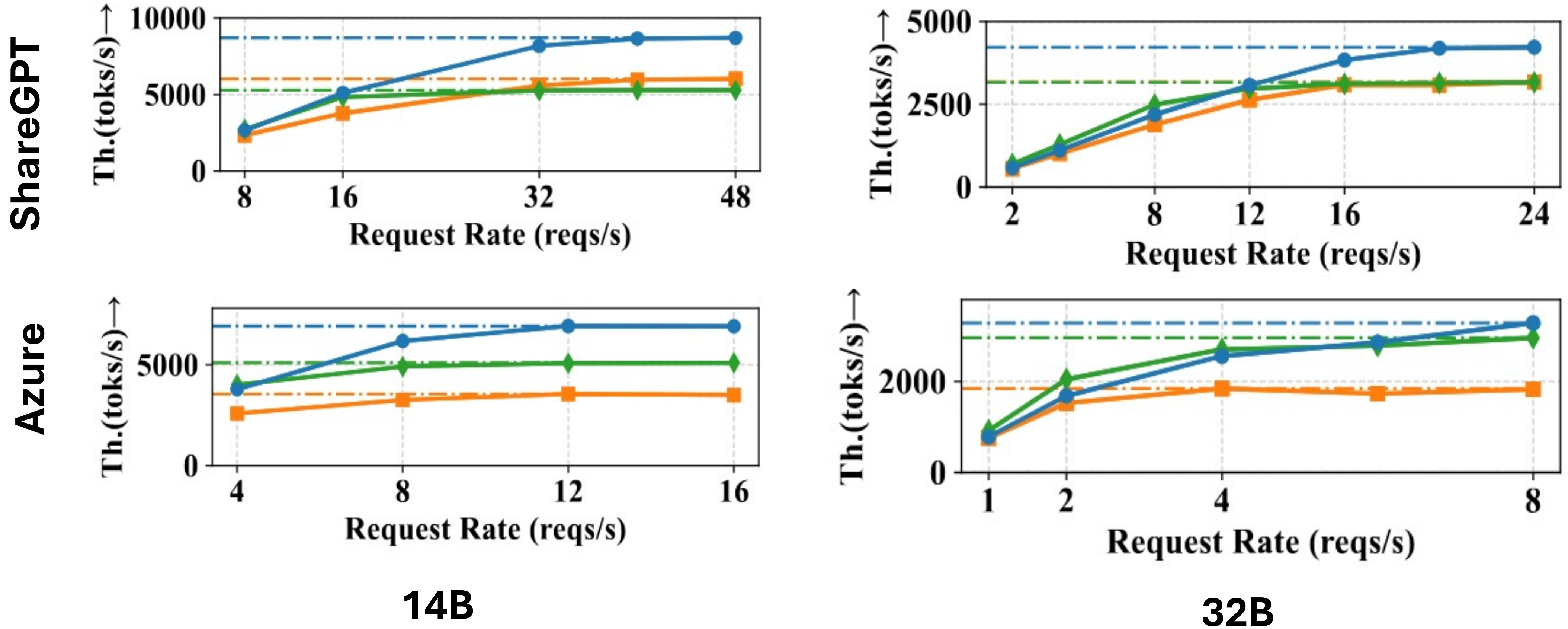
Evaluation: Latency and Throughput —— E2EL

- E2EL shows an approximately linear increase trend
  - gLLM achieves 0.53-0.92× lower slope



Evaluation: Latency and Throughput —— Throughput

- Throughput gradually plateaus as request rates increase. This plateau represents the system's maximum processing capacity
  - gLLM improves processing capacity 29%-150%



Conclusion

- In LLM serving, **pipeline bubbles** arise from computation imbalance caused by insufficient prefill tokens or limited KV cache
- We propose **Token Throttling**, which dynamically adjusts prefill and decode batch size based on real-time feedback
- We present **gLLM**, a distributed serving system that employs Token Throttling
- On representative LLM workloads, gLLM boosts throughput by 11%–398% over state-of-the-art systems with lower latency

中山大學
SUN YAT-SEN UNIVERSITY

GitHub https://github.com/gty111/gLLM

ARSYSU

gLLM: Global Balanced Pipeline Parallelism Systems for Distributed LLMs Serving with Token Throttling

Tianyu Guo, Xianwei Zhang, Jiangsu Du, Zhiguang Chen,
Nong Xiao, Yutong Lu

Email: guoty9@mail2.sysu.edu.cn

Tianyu's Homepage

gLLM

Thank You!

SC25 St.Louis, MO hpc ignites.