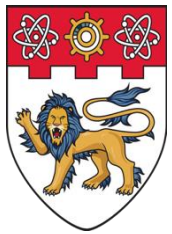
High Throughput and Low Latency LLM Serving via Adaptive KV Caching

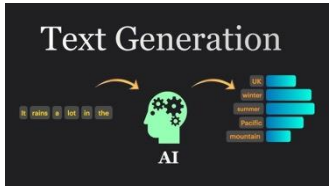
Wenyan Chen, Chengzhi Lu, Huanle Xu, Kejiang Ye and Chengzhong Xu



EuroSys 2026

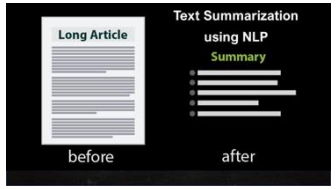
Rise of LLMs

- **LLM serving permeates our daily work and life**



Text Generation

Translation



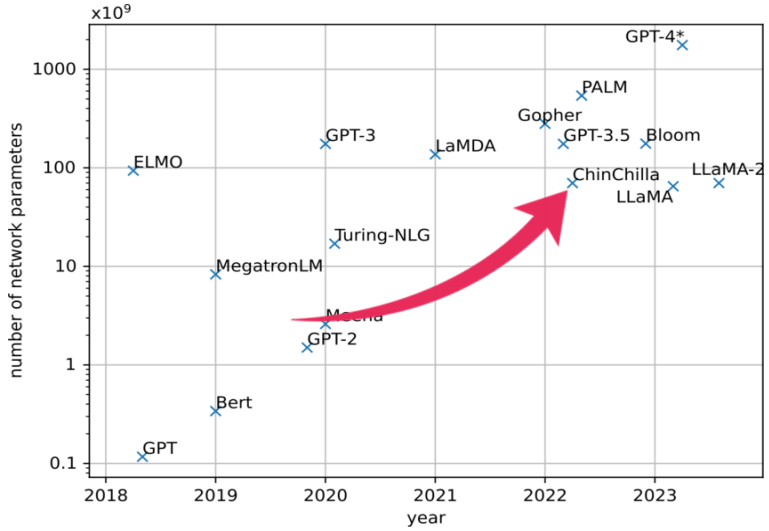
Summarization

Question Answering

Code Generation

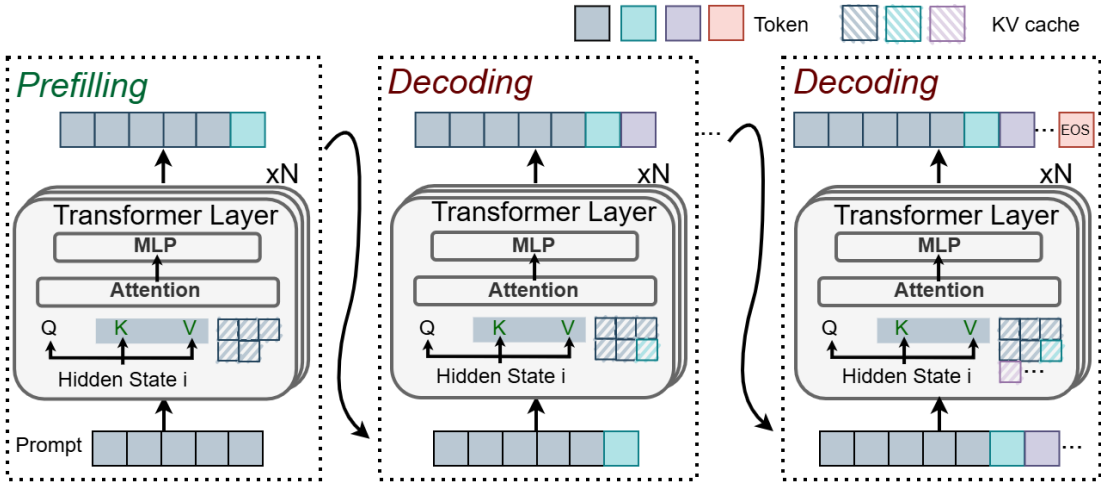
...

- **The scale of LLMs are increasing rapidly**



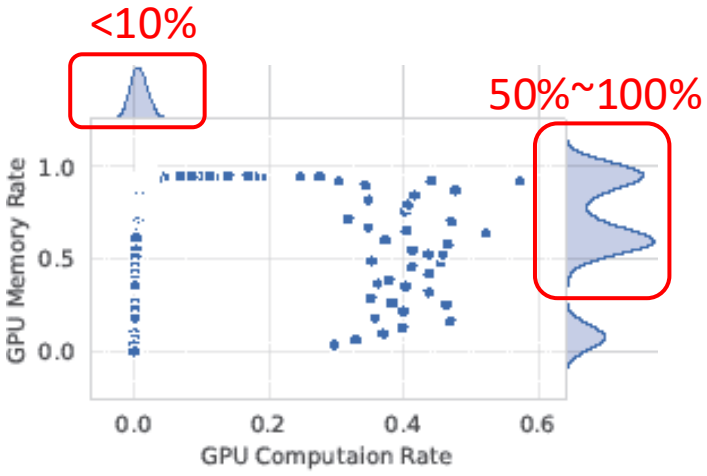
Transformer-based LLMs

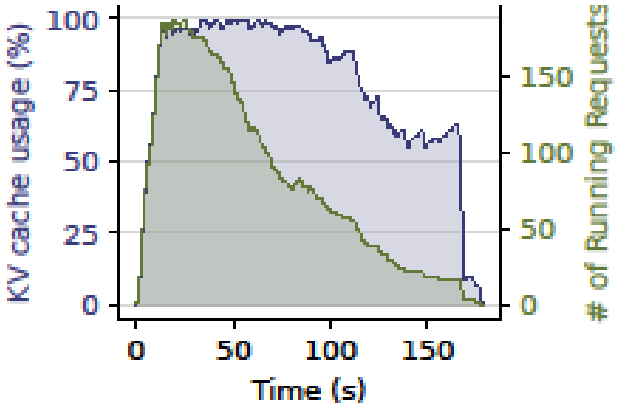
- **KV Caches Mechanism**
  - Reuse KV caches to avoid redundant recomputations
  - Memory footprint is increasing due to the generated output tokens growing
- **Prefiling & Decoding**
  - *Prefilling*: generate KV caches of the input prompt
  - *Decoding*: generate new tokens sequentially



Memory & Computation Imbalance

- **GPU Memory becomes a critical bottleneck for LLM serving**
  - GPU memory rate is high (50%~100%) while GPU computation utilization is relatively low (<10%)
  - Running requests decreases while the KV caches usage remains high





Settings
Model: Llama2-13B
QPS: 50 req/s
Hardware: A100-80G

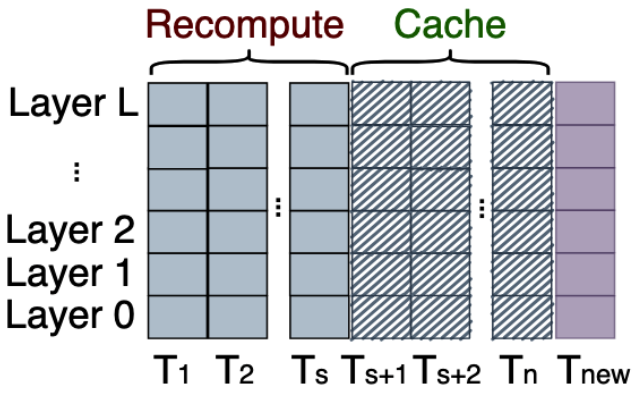
Existing Solutions

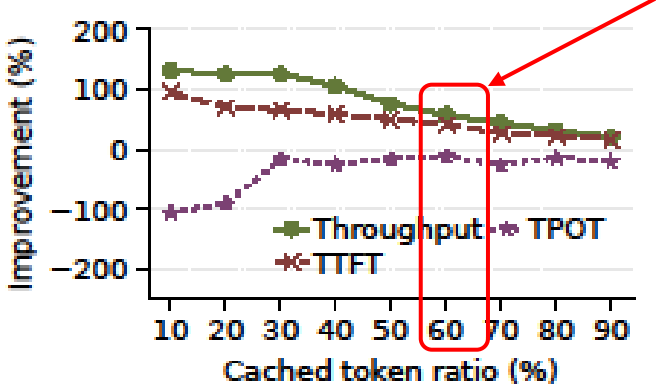
- **Swapping**
  - Swap out **all KV caches** of preempted requests to the host memory and swap in when GPU memory is available
- **Recomputing**
  - **Recompute all KV caches** of the preempted requests (Concatenate all generated tokens with prior prompt as prefilling)

Both of them need to restore all KV caches on GPU at the decoding stage

Benefits of Partial Token-wise Caching

- **Partial token-wise caching**
  - Recomputing a subset of old tokens and caching the remaining tokens with full KV caches (reduce memory wastage)
  - Improve the concurrency of requests processed in batches and reduce TTFT without degradation with TPOT (improve computation utilization)

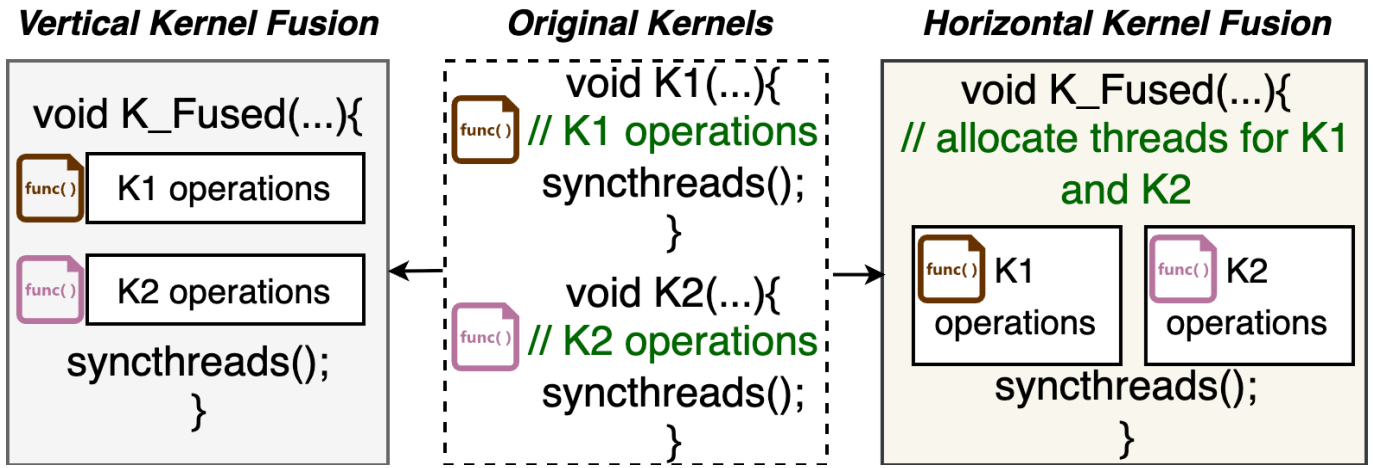




With a suitable cached token ratio, **throughput improves 57% without TPOT perf. degradation**

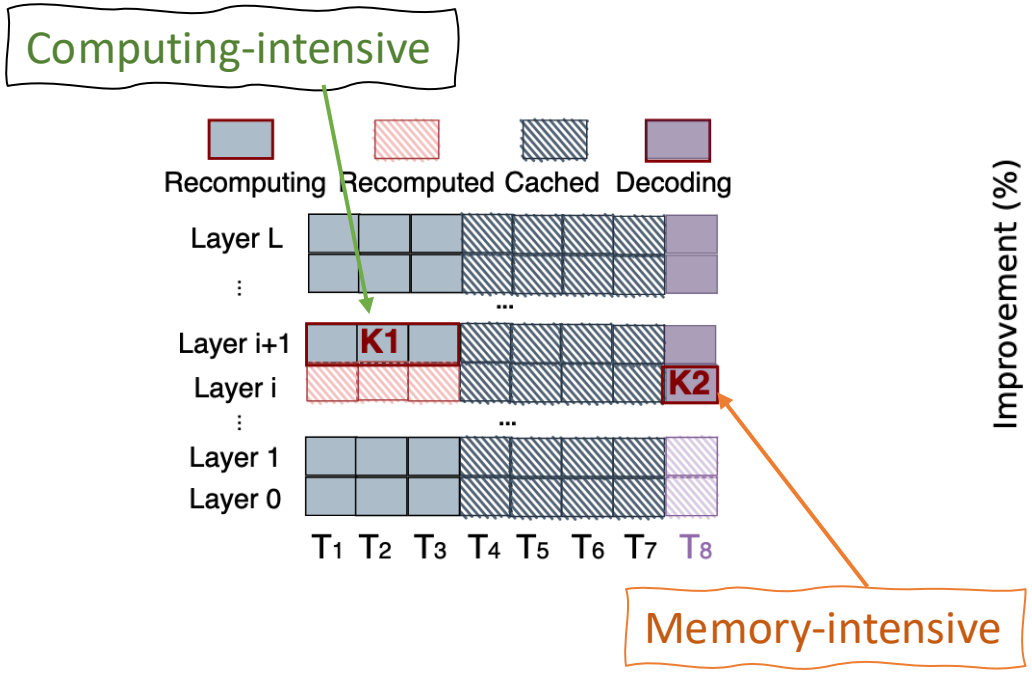
Benefits of Layer-wise Kernel Fusion

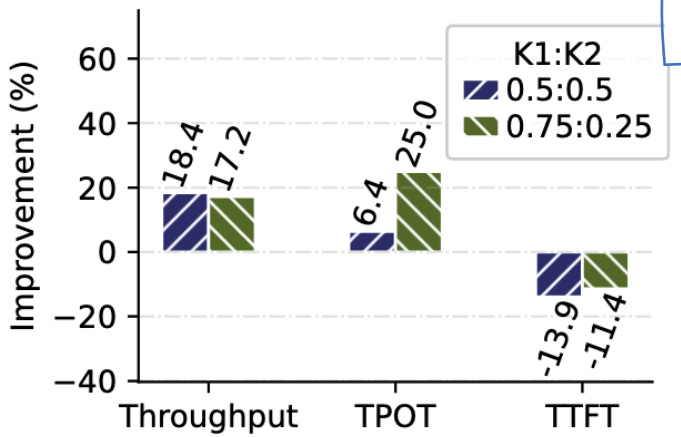
- **Layer-wise Kernel Fusion**
  - ***Vertical*** kernel fusion & ***horizontal*** kernel fusion
  - Reduce kernel launch overhead
  - ***Vertical:*** eliminate redundant global memory access between various kernels
  - ***Horizontal:*** can fully utilize the GPU SM cores spatially



Benefits of Layer-wise Kernel Fusion

- **Layer-wise Kernel Fusion**
  - Recomputing kernel (K1) and decoding kernel (K2) fused into a Kernel
  - Improve throughput and TPOT 👍
  - Degrade TTFT due to memory requirement 👎





With a suitable thread allocation, **throughput and TPOT perf. improves with slight TTFT perf. degradation**

Challenges and Design Ideas

Maximize output token throughput while the TPOT SLOs are met

Challenges

1. Dynamic request arrivals and sequence generation lengths

2. Interdependence between token-wise caching and kernel fusion

Dual-level Optimizations

Design

1. Request-level adaptive batch size and token-wise caching

2. Layer-level kernel fusion and comm-com overlap; Iteratively refine the values in 1

System Design of eLLM

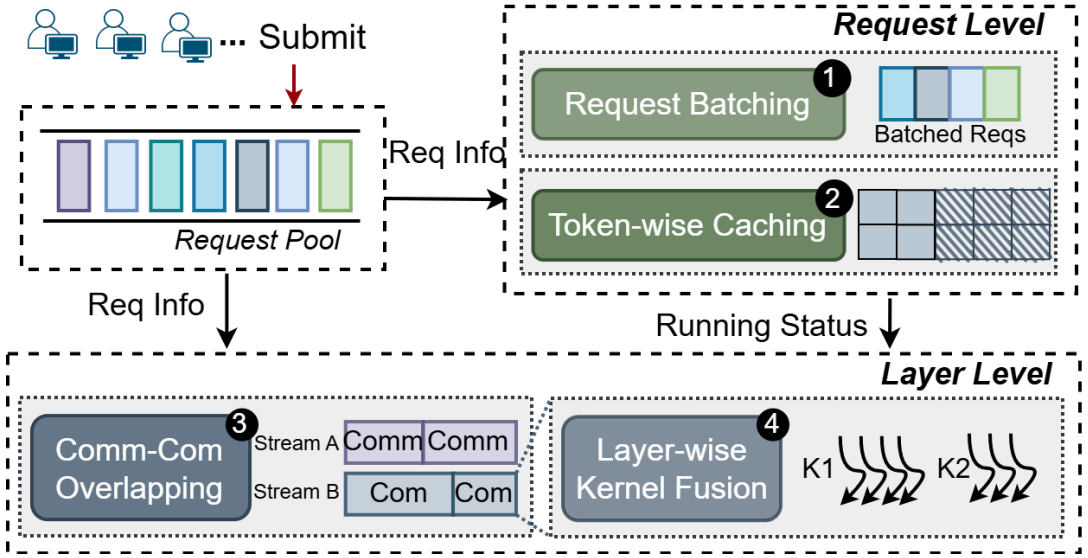
- **eLLM components**

① **Request Batching**: Select max request number in waiting queue

② **Token-wise Caching**: Determine suitable non-cached token ratio

③ **Comm-Com Overlapping**: Fine-tune which layers to overlap for swapping and recomputing

④ **Layer-wise Kernel Fusion**: Identify which operations to be fused and how to allocate thread resources



Optimization Problem

- **Target** Maximize output token throughput
- **Constraints** Meet TPOT SLOs & Memory footprint does not exceed memory capacity

Batch size | Non-cached token ratio | Threads Config.

$$\max_{(b,r,\delta)} \frac{b}{T(b,r,\delta)}$$

$$s.t. \quad T(b,r,\delta) + WT_{max} \leq \text{SLO},$$

$$\sum_{i=1}^{b} 4Lhs_i(1-r) + M_W \leq M_G \cdot N - M_o,$$

$$0 \leq r \leq 1,$$

$$1 \leq b \leq B.$$

WT_{max}: the longest waiting time
N: GPU number
M_G: memory capacity
M_o: Kernel fusion required memory
B: requests in the waiting queue

Request-level Optimization

- **Inference latency estimation**
  - Recomputing volume

$$V_{b,r}^{\text{rec}} = \sum_{i=1}^{b} \left(24h^2 L s_i r + 4hL s_i^2 r^2 + 2hV s_i r + \epsilon_0\right)$$

  - Decoding volume

$$V_{b,r}^{\text{dec}} = \sum_{i=1}^{b} \left(24h^2 L + 4hL(s_i + 1) + 2hV s_i + \epsilon_1\right)$$

$$T(b,r) = \frac{1}{FLOPS}\left(V_{b,r}^{\text{rec}} + V_{b,r}^{\text{dec}}\right)$$

- **Batching and token-wise caching**

$$\max_{(b,r)} \frac{b}{T(b,r)} \Rightarrow \min_{(b,r)} \frac{T(b,r)}{b} \Rightarrow \min_{(b,r)} \frac{\sum_{i=1}^{b}\left[24h^2 L(s_i r + 1) + 4hL\left(s_i^2 r^2 + s_i + 1\right) + 2hV s_i (1 + r)\right]}{FLOPS \cdot b}$$

b and r can be solved with SciPy

Layer-level Optimization

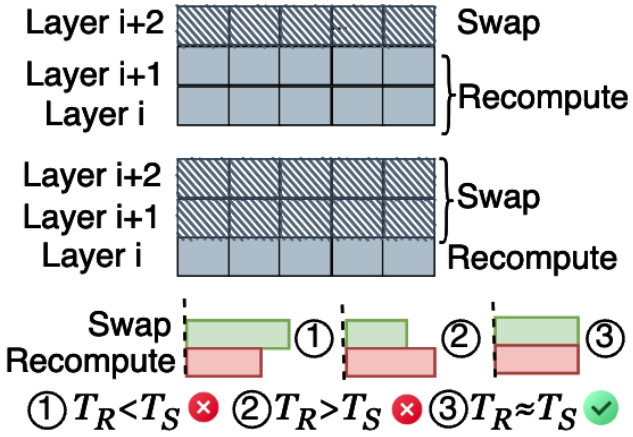
- **Layer-wise Overlapping**
  - Swapping latency

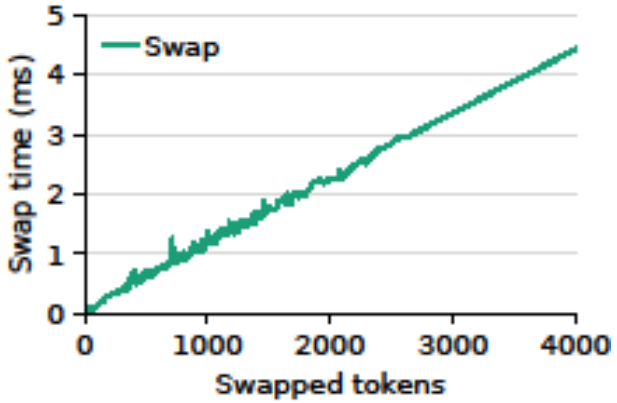
$$T_{k,l_1} = (\alpha k + \beta) \cdot l_1 / L$$

  - Recomputing latency

$$T_{b,r,l_2} = \frac{1}{FLOPS} \sum_{i=1}^{b} \left[\left(24h^2 s_i r + 4h s_i^2 r^2\right) \frac{l_2}{L} + 2 s_i r h V \right]$$

Determine layers for swapping and recomputing based on the latency estimations





Layer-level Optimization

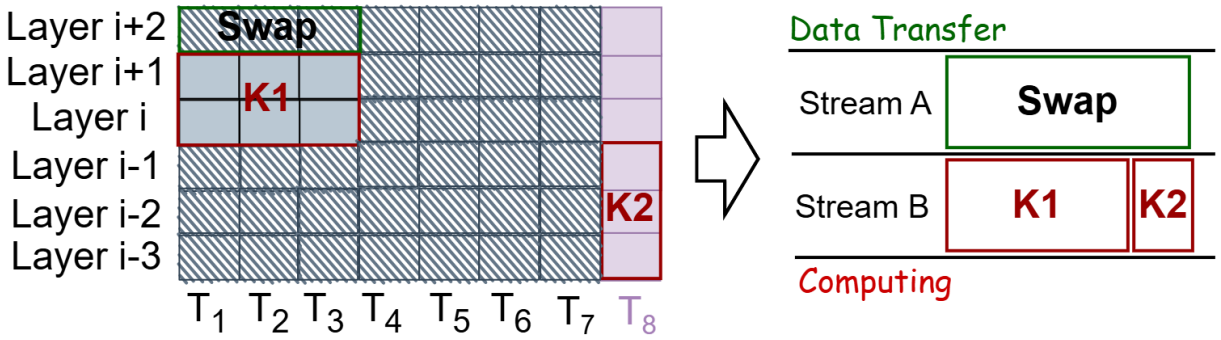
- **Layer-wise Kernel Fusion**
  - K1 computation volume

$$V_{K1} = \sum_{i=1}^{b} \left(24h^2 l_2 s_i r + 4h l_2 s_i^2 r^2 + 2hV s_i r\right)$$

  - K2 computation volume

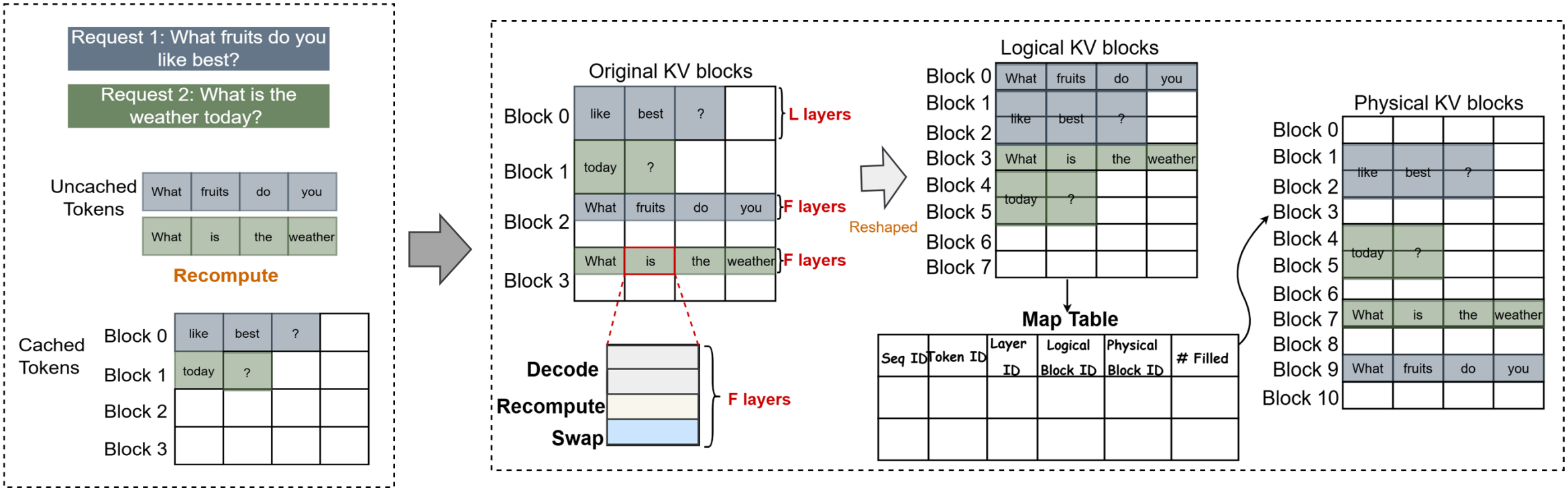
$$V_{K2} = \sum_{i=1}^{b} (24h^2(l_1 + l_2) + 4h(l_1 + l_2)(s_i + 1) + 2hV s_i)$$

δ can be solved with V_{k1}/V_{k2}



Token-wise and Layer-wise KV Caching

- Each token's KV cache needs to store all layers within a block unit address
- Design a **fine-grained, layer-wise** GPU block table
- A new **map table** to index the physical address to the fine-grained block unit



Evaluations

- **Models and Datasets**

Table 1. Models and datasets used in the experiments.

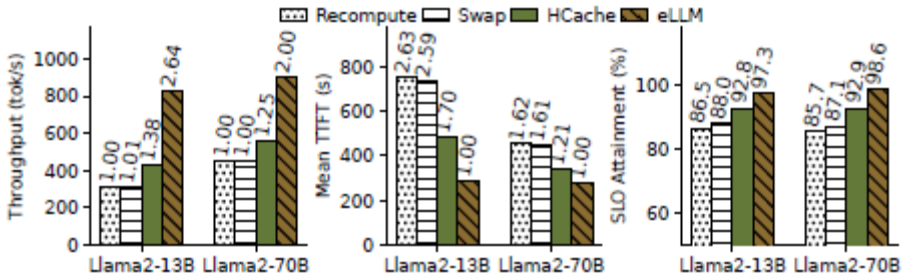
Dataset	Model	Atten.	# Layers	# GPUs	SLO (ms)
ShareGPT [36]	Llama2-13B	MHA	40	1 A100	50
	Llama2-70B	GQA	80	4 A100	500
L-Eval [5]	Llama2-13B	MHA	40	1 A100	60
	Llama2-70B	GQA	80	4 A100	200

- **Baselines**
  - *vLLM-Swap*[1] : Uses host memory to store the overloaded KV caches and reload them
  - *vLLM-Recompute*[1]: Evicts all tokens of overloaded requests and recomputing them
  - *HCache*[2]: Uses hidden states to store in GPU memory of certain layers to fast recovery
- **Request Generation**
  - LLM invocation trace in Azure[3]
  - Avg request load is 25 req/s

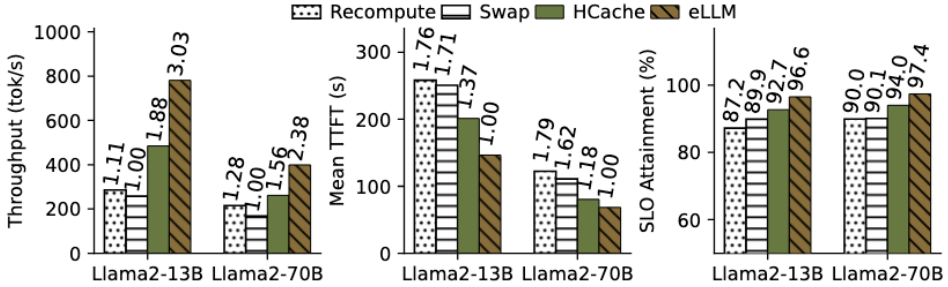
[1] Kwon W, Li Z, Zhuang S, et al. Efficient memory management for large language model serving with pagedattention. *In proceedings of SOSP 2023*
[2] Gao S, Chen Y, Shu J. Fast state restoration in llm serving with HCache. *In proceedings of EuroSys 2025*
[3] Stojkovic J, Zhang C, Goiri Í, et al. Dynamollm: Designing llm inference clusters for performance and energy efficiency. *In proceedings of HPCA 2025*

End-to-End Performance

- **End-to-end Performance**
  - **Throughput** improves by upto 3.03x
  - **TTFT** improves by upto 2.63x
  - **SLO attainment** achieves upto 98.6%

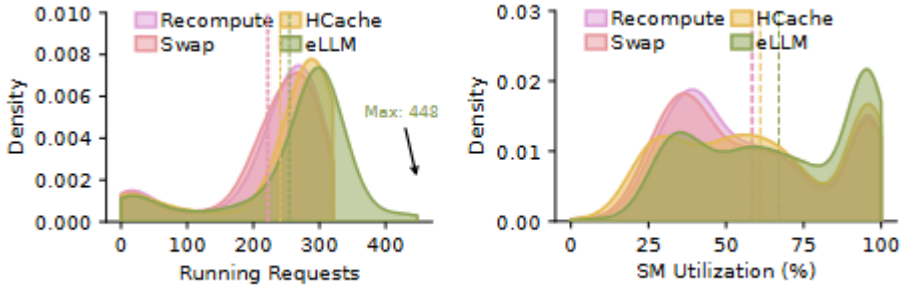


(a) ShareGPT



(b) L-Eval (Long prompts)

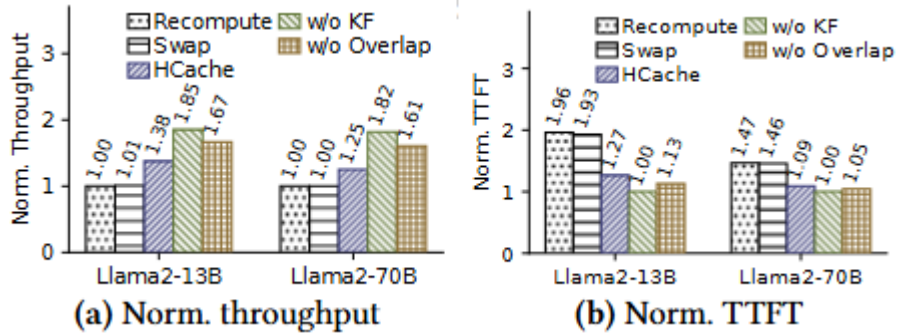
- **Runtime Behaviors**
  - Running requests is higher (448)
  - SM utilization 15% improvement (67%)



(a) Running requests (b) SM utilization

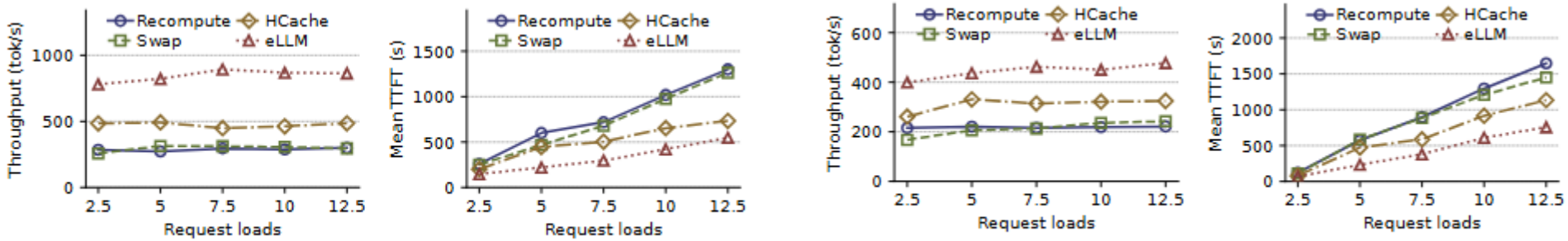
End-to-End Performance

- **Evaluation of individual components**
  - Disable ***Comm-Com Overlapping***
    - Throughput improves by upto 1.85x
    - TTFT reduces by upto 1.96x
  - Disable ***Kernel Fusion***
    - Throughput improves by upto 1.67x
    - TTFT reduces by upto 1.73x



(a) Norm. throughput (b) Norm. TTFT

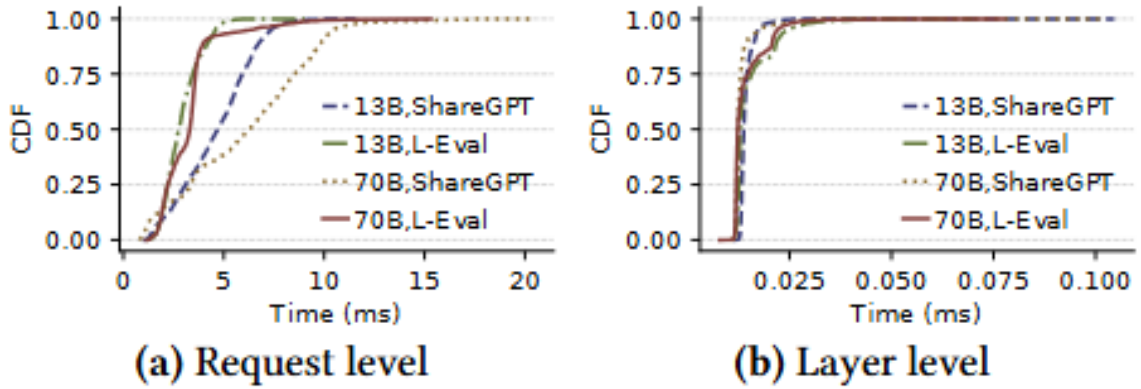
- **Sensitivity to various loads**
  - eLLM remains better performance for various loads



(a) Llama2-13B, Throughput (b) Llama2-13B, TTFT (c) Llama2-70B, Throughput (d) Llama2-70B, TTFT

End-to-End Performance

- **System Overhead**
  - ***Request-level***
    - Finding *b* and *r* overhead: average under 6ms
  - ***Layer-level***
    - Indexing a layer of a token address: average 0.015ms
    - Determining the optimal thread allocation: average 0.03ms



Conclusion

😭 **Problem –** How to achieve high throughput and low latency for LLM serving by addressing the imbalance between memory and computation?

🤔 **Key Insight –** There are benefits from caching part of tokens and fusing resource-complementary kernels

 **Key Ideas**

- Adaptive batching and token-wise caching at the request level
- Comm-com overlapping and kernel fusion at the layer level

🎉 **Results –** eLLM improves throughput by **3.03x** and reduces TTFT by **2.63x** with **SLO compliance** for TPOT

✉️ wenyan.chen@ntu.edu.sg

Thanks & QA