CaR: An Efficient KV Cache Reuse System for Large Language Model Inference

Kexin Chu, Tzechinh Liu, Yunding Li, Pengchao Yuan, Wei Zhang
University of Connecticut

CONTENTS

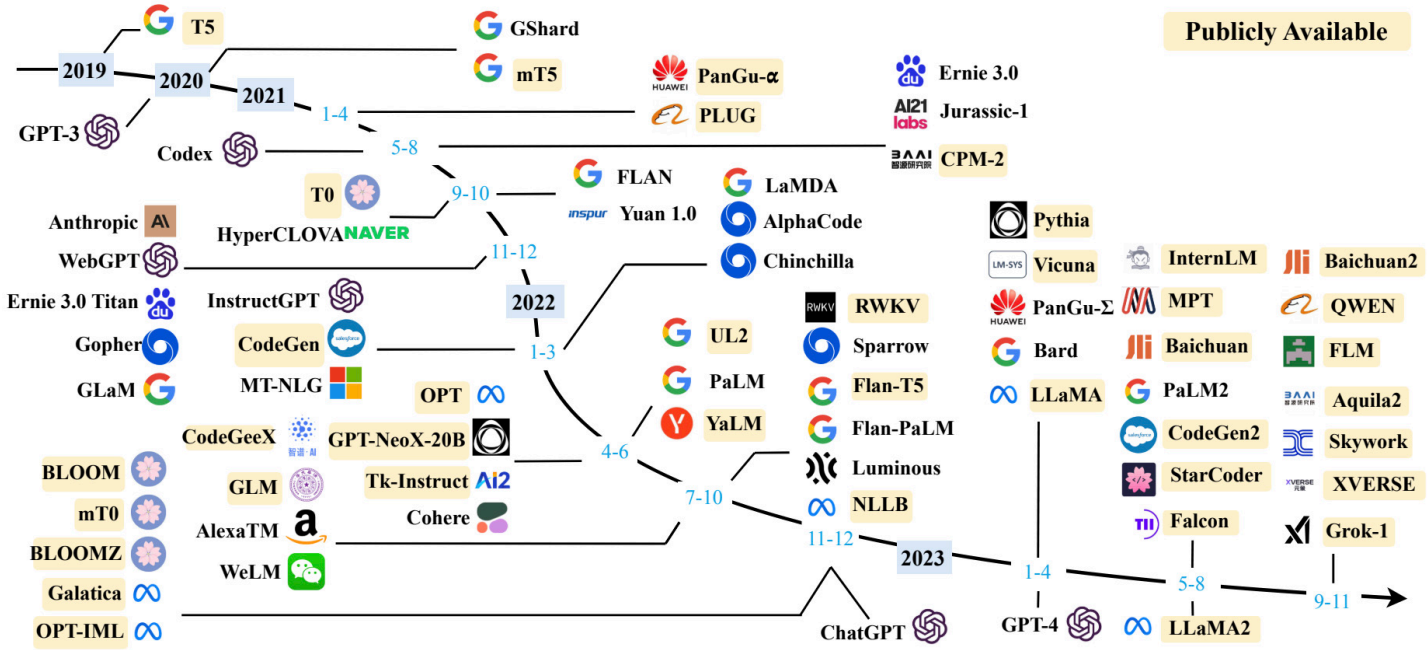
- Intruduction to LLMs and KV cache
- Motivation and Challenges of CaR
- Design Details of CaR system
- Our Experiment and Conclusion

01

Intruduction to LLMs and KV cache

The era of LLMs

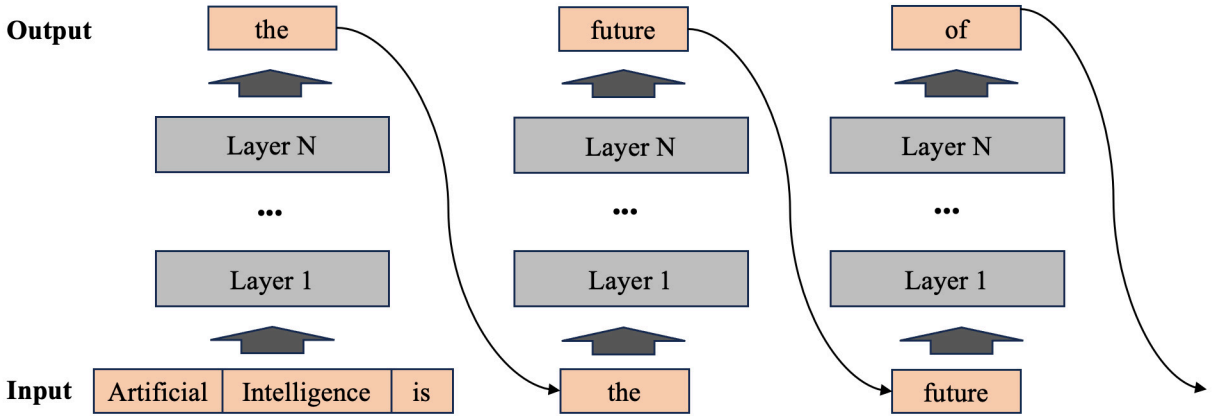
- More companies are launching LLM models, expanding the AI landscape. [1]



[1] A Survey of Large Language Models (https://arxiv.org/pdf/2303.18223)

LLM Inference Process

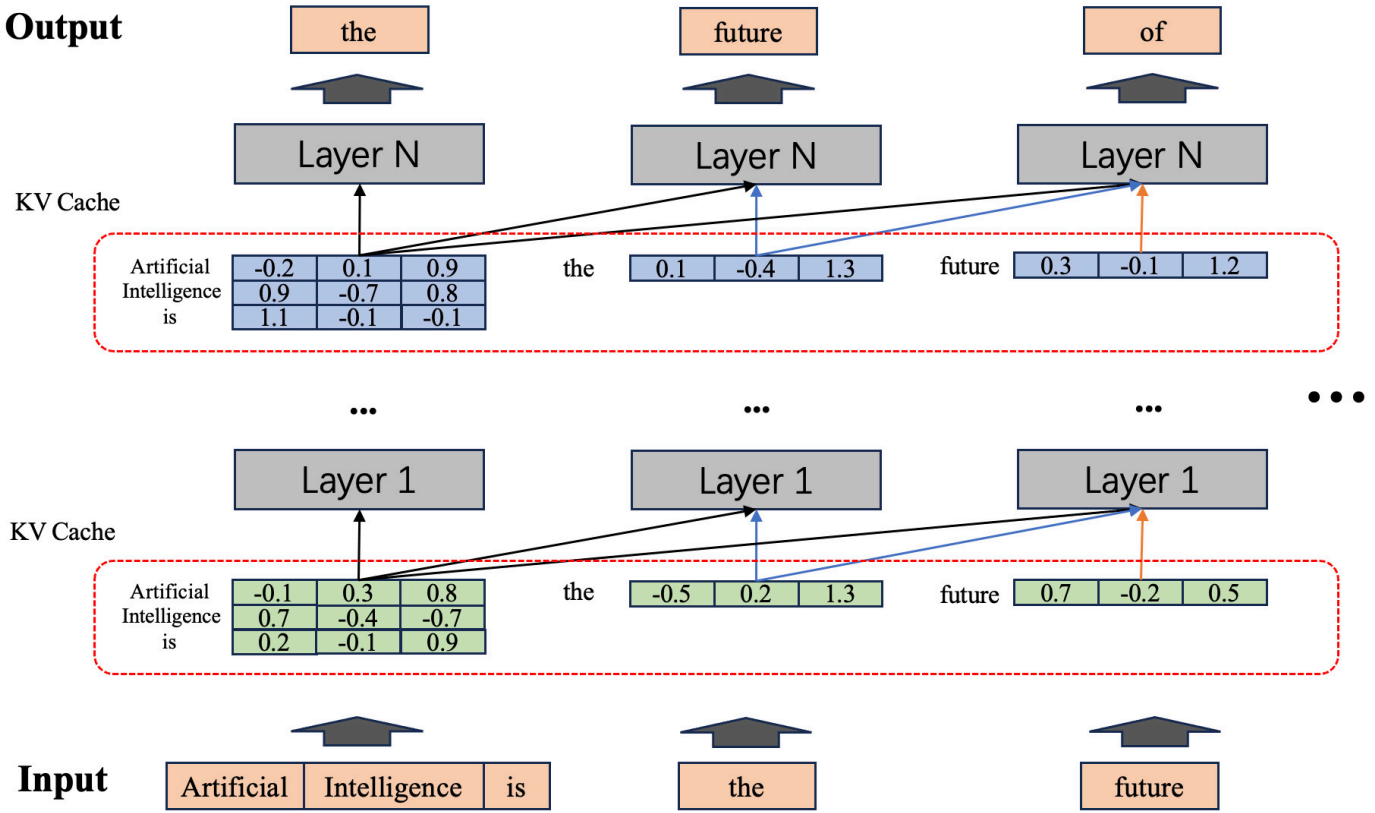
- Two stages:
  - Prefill: first iteration, compute all input tokens in a single pass
  - Decode: utilize previous generated token as input
- Regressive generation
- Layer by layer



Self-Attention & KV cache

Self Attention :

$$Q = xW_Q$$
$$K = xW_K$$
$$V = xW_V$$
$$out = softmax(\frac{Q K^T}{\sqrt{h}}) * V$$



Serving LLMs is Expensive

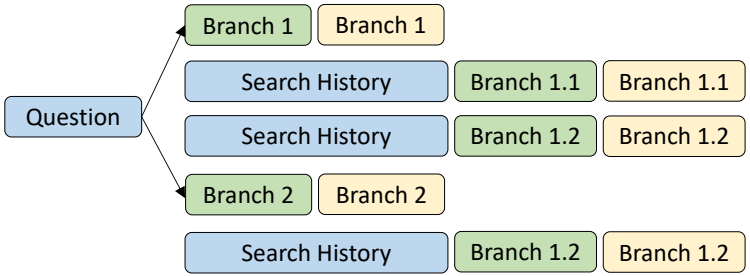
- Compute Sensitive (large model parameters)
  - LLMs run on high-end GPUs such as Nvidia A100
  - Each GPU can only serve a handful of requests per second
    - For LLaMA-13B and moderate size inputs, 1 A100 can process <1 requests per second
  - A ton of GPUs are required for production-scale LLM services.
- Large KV cache size
  - KV cache size: $2 * seq\_len * hidden\_size * sizeof(TYPE)$
  - For LLaMA-13B and A100 GPUs, the generated KV cache takes close 30% of the HBM.

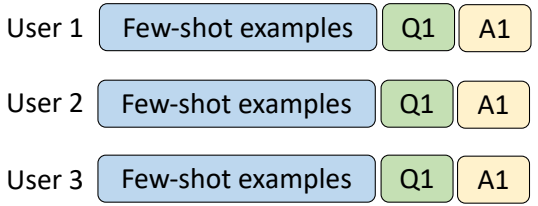
02

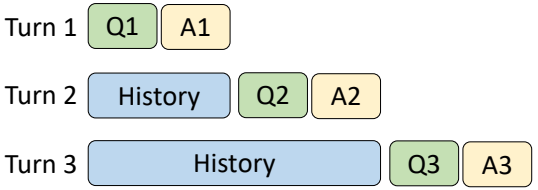
Motivation and Challenges of CaR

Opportunity for KV cache reuse

- The related KV cache is directly discarded once the request is completed.
- Scenarios with similar prefix tokens
  - Few-shot aibot
  - Multi-turn chat
  - Tree of thoughts

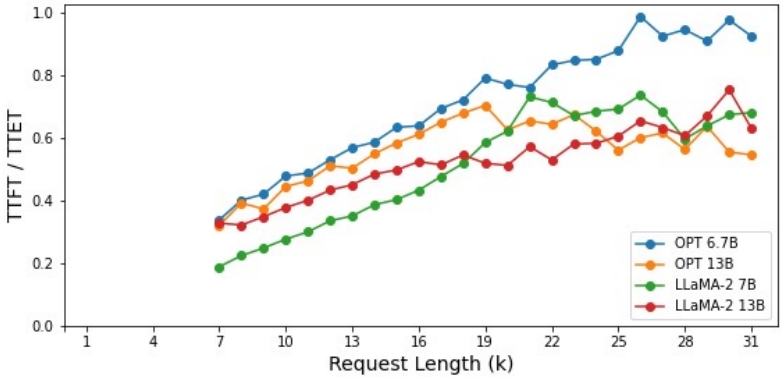


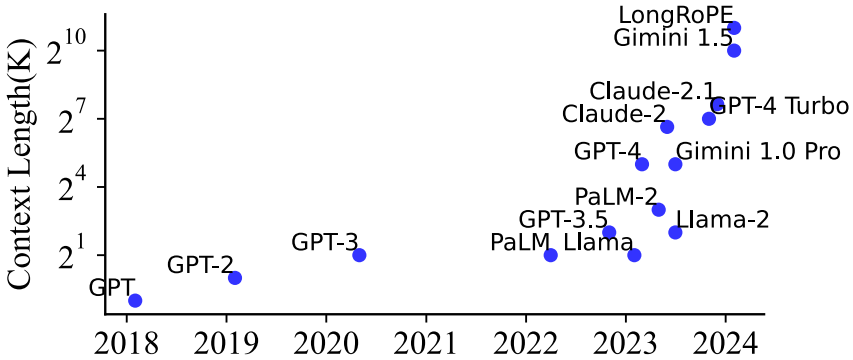




Prefill Re-compute Costs

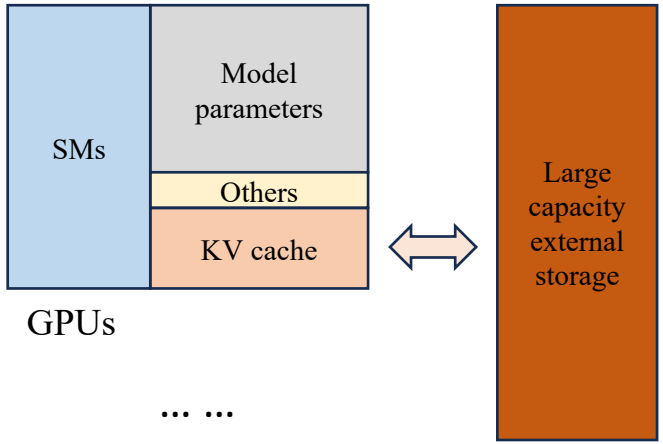
- The computational overhead during the prefill stage increases with the length of the prompt.
- LLM models are developed to support longer contexts.





Motivation

- The expensive computation in the prefill stage can be avoided by reusing the KV cache from the previous requests.
- The large KV cache storage has become a bottleneck in LLM inference.
- Multi-tier Memory System
  - Active blocks are stored in HBM of GPUs.
  - Inactive blocks are stored in external Mem.



Challenges in Multi-tier KV Cache System

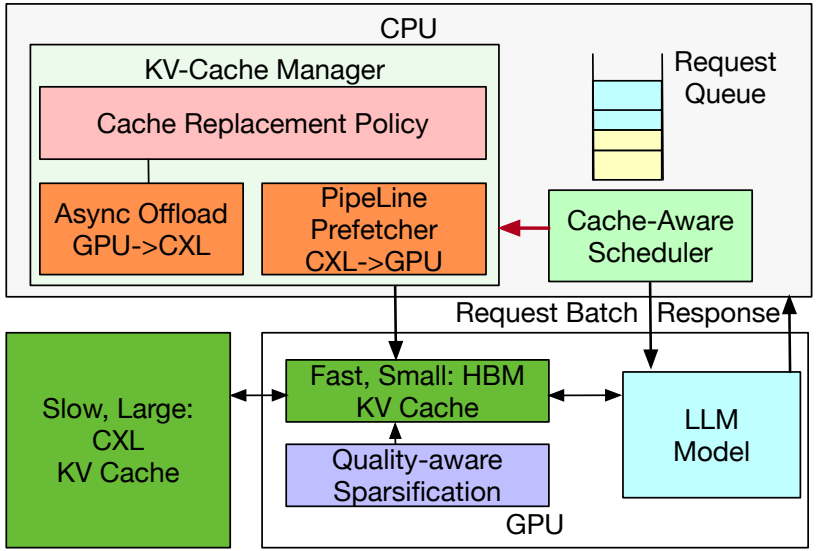
- Where to place the KV cache of previous requests?
  - The HBM on GPUs is small and fast.
  - The CXL-based external memory is large but slow.
- How to limited the data transfer overhead?
  - Maximize the use of KV cache in HBM of GPUs
  - Avoid frequent data transmission between GPUs and CXL
- When to migrate the data across the tiers?
  - Avoiding the GPU stall waiting for the data to be loaded from CXL-based memory.
  - Avoid prefetch too early.

03

Design Details of CaR System

Overview of CaR System

- Co-design of Scheduler and Cache Management.
  - Replacement Policy
  - Cache -Aware Scheduler
  - Prefetch Predictor
- Quality-aware Compression algorithm.
- Pipeline data loading and asynchronous offloading

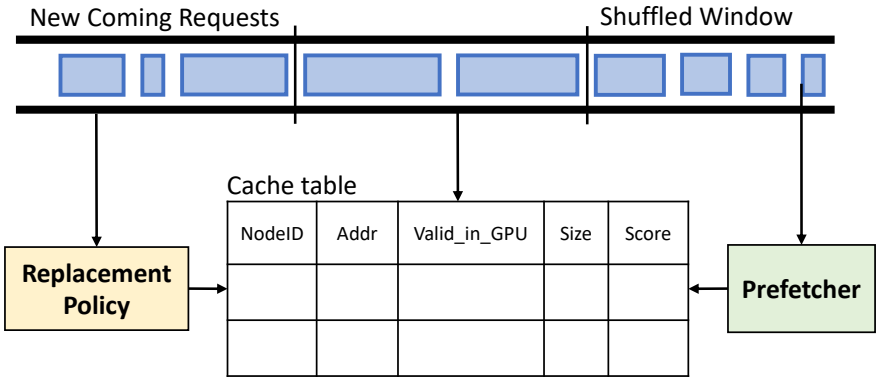


Co-design of Scheduler and Cache Management

- Maximizing KV cache reuse requires avoiding task waiting caused by frequent data offloads and prefetches.

- Scheduler
  - Use shuffle windows to categorize requests into three groups.
    - A fixed size window is used to prevent frequent changes in the order of requests.
  - Standard：
    - The size of KV cache to be transmitted from CXL to GPUs

  - Highest: the needed KV cache already in the HBM.
  - Second: do not involve any historical KV caches.
  - Lowest: The size of KV cache to be prefetched

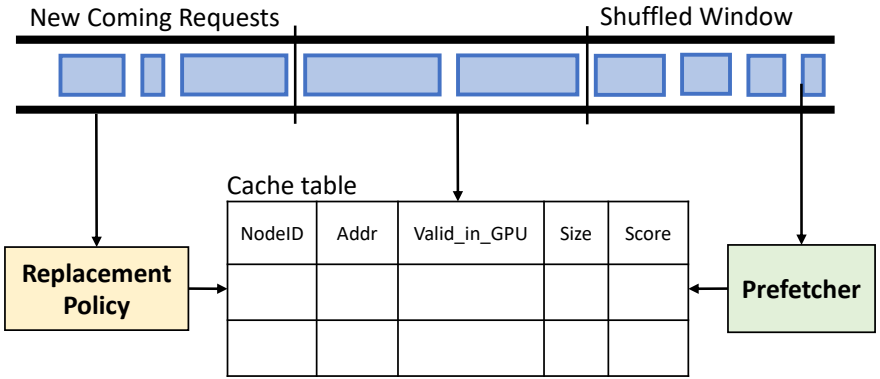


Co-design of Scheduler and Cache Management

- Replacement Policy
  - Determine which block of KV cache need to be offload when HBM is full.
  - Avoid the deletion of the KV cache that is relied by active requests.

  - Use Score to target the KV cache blocks.
  - Standard: evict the memory block with the smallest score

  - When a request enter to shuffled windows, the score of related blocks adds 1
  - When a request enter to GPU for inference, the score of related blocks adds 1
  - When a request complete it's inference, the score of related blocks subs 2



Co-design of Scheduler and Cache Management

- Prefetch Predictor
  - Determine when to start prefetch the related KV cache of the first priority requests in shuffled window.

  - Predict when the GPUs need to load a new requests from the queue.
    - [2] utilize the LLM model to predict the output sequence length. (number of decode iterations)
  - N represent the total number of requests currently being handled on GPUs.
  - T donate the current time.
  - len_i is the predicted sequence length for request i.
  - SIZE is the size of the KV cache need to be fetched.

$$fetch(t) = \mathbb{I}_{\min_{0 \le i \le N}\{TPOP \times len_i - (t - timer_i)\} \le \frac{SIZE}{bandwidth}}(t)$$

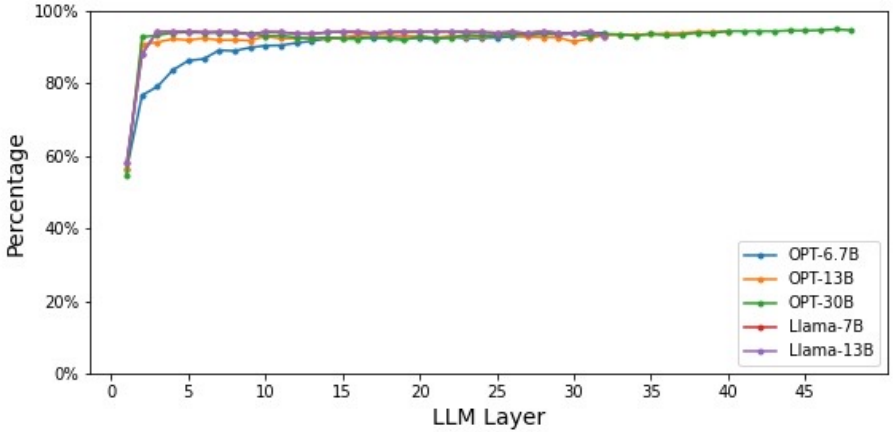
[2] *Response length perception and sequence scheduling* (https://arxiv.org/pdf/2305.13144)

Quality-aware Compression

- The size of the KV cache increases linearly when the token numbers increase.
- The sparsity in the attention score matrix is pretty high, especially in the deeper layers (95%).

$$softmax(\frac{Q\ K^T}{\sqrt{h}})$$

- remove the KV cache with low quality score.
  - $quality = \sum_{col} attention\_score[*]$
- The compression ratio is adapted based on the sparsity of attention score matrix.



Data Loading and Offloading

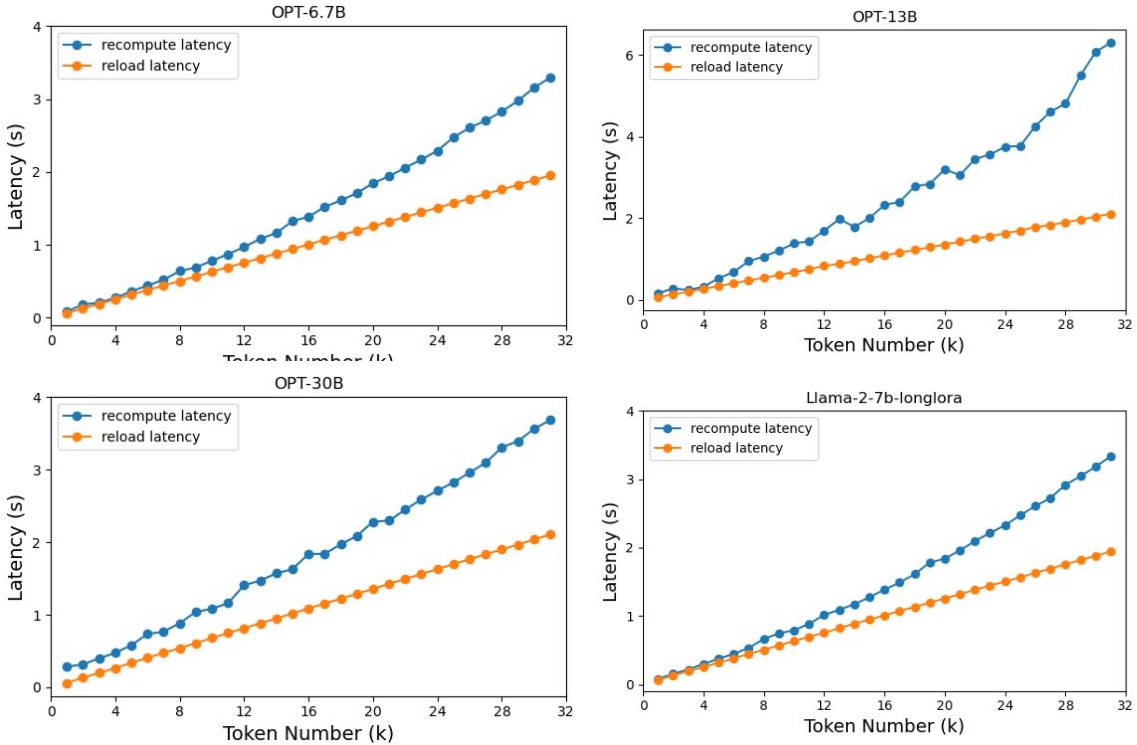
- Layer-wise pipeline data loading
  - In LLM inference, The computation of $layer_i$ depends on the output of $layer_{i-1}$.
  - Pipeline data loading: Overlap the transfer of the KV-cache for $layer_{\mathrm{i}}$ with the computation of $layer_{i-1}$.
- Asynchronous data offloading
  - Minimize the offloading overhead by eliminating it from the request's critical path.
  - Set a threshold: the usage of KV cache blocks on HBM over 90%.
  - HBM is not full -> the requests do not need preempt while data offloading.

04

Our Experiment and Conclusions

Experiment and Conclusion

- Recomputation vs KV cache reused
  - Leveraging stored KV cache for reuse yields considerable improvements over recomputation.
  - The KV cache reuse demonstrates the potential to reduce TTFT by 30% in scenarios with long context.
  - With OPT-13B, this reduction even exceeds 60%.



THANKS