CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion

CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving

Jinho Choi

2025.07.04

Contents

- Background
- Problem Definition
- Previous Works
- CacheBlend Details
- Integration with CacheGen
- References

Background

- GPT flow – prefill + decode

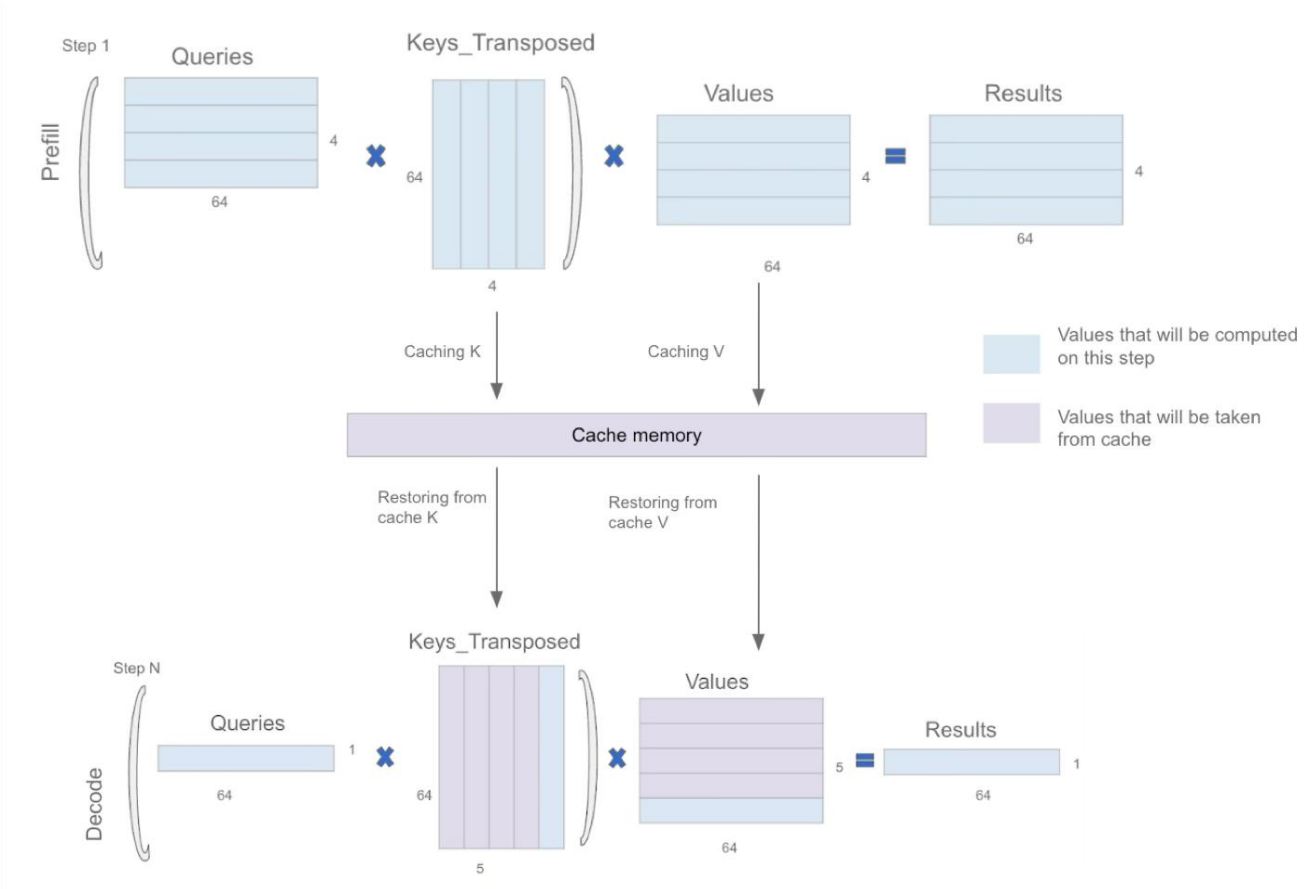


Fig. 1. Representation of prefill stage and decode. User prompt is processed via prefilling, and new tokens are generated during decoding stage.

HYPER ACCEL

Related Works

❑ RAG(Retrieval-Augmented Generation)

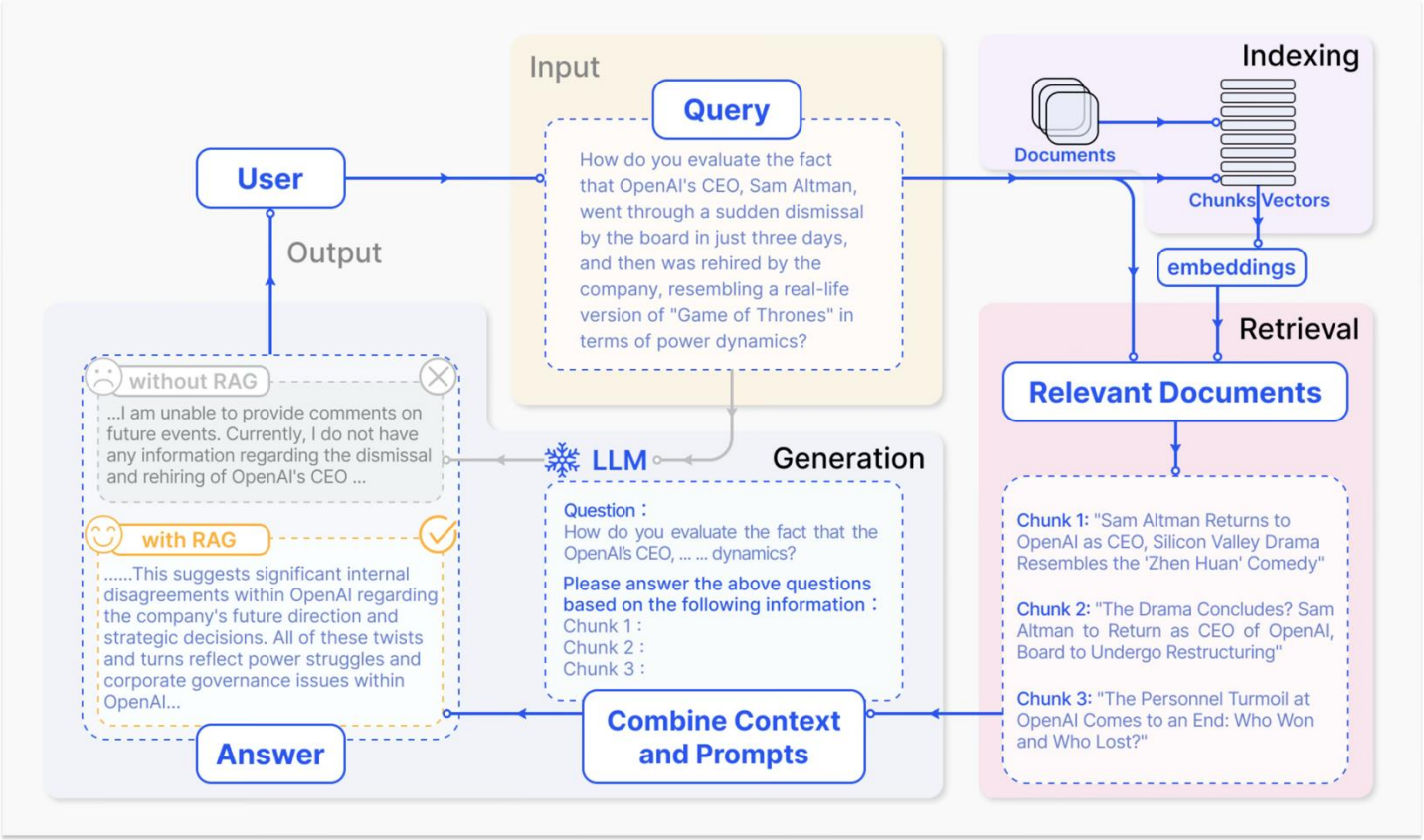


Fig. 2. Overall flow of RAG. If user inputs a query, then RAG engine fetches relevent documents(chunks) from the database. Those hunks are appended to the query, and then processed by LLM.

Problem Definition

- How can we reduce the prefilling time for retrieved chunks?
  - RAG generates a long prompt by cumulatively adding chunks to user query. As retrieved chunks get bigger, prefilling stage will take a long time.

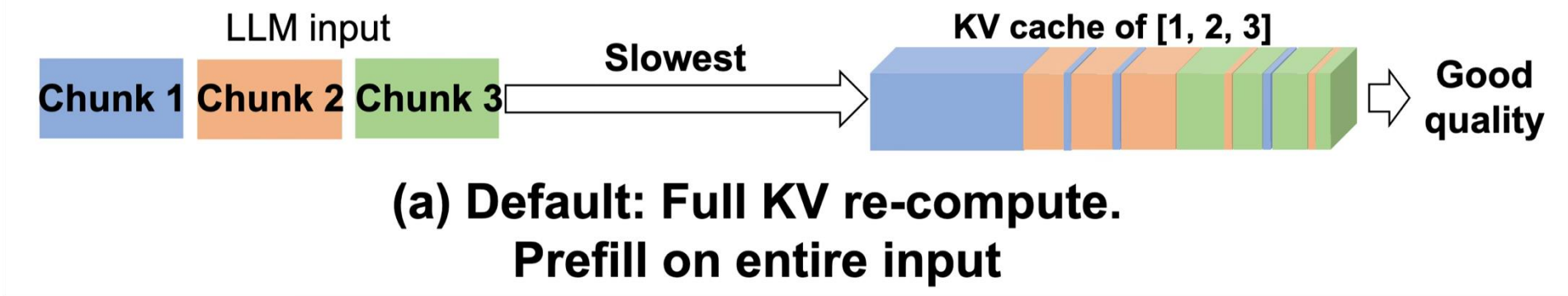


Fig. 3. How retrieved chunks are processed during prefilling stage. Computation time is proportional to the length of the prompt.

Previous Works – Full KV recompute

- Prefill on entire input

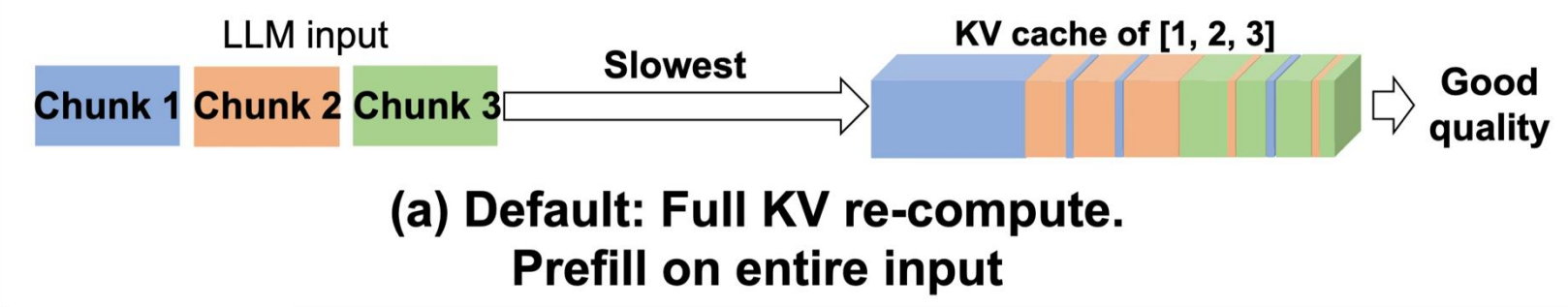


Fig. 4. Prefilling on entire chunks.

Previous Works – Prefix Caching

- Only reusing prefix's KV cache

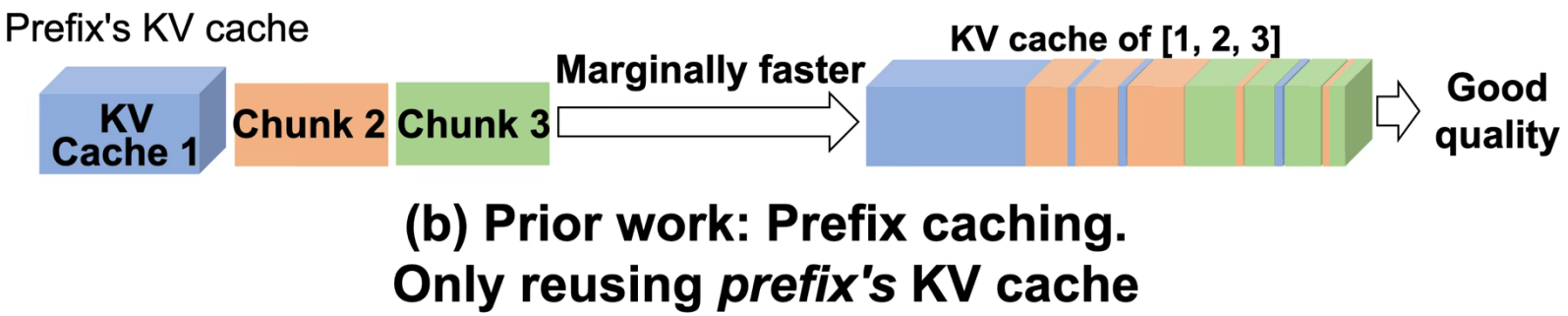


Fig. 5. Fetch KV cache for first chunk. And apply prefilling, starting from second chunk.

Previous Works – Full KV reuse

- Reusing all KV caches, ignoring cross-attention

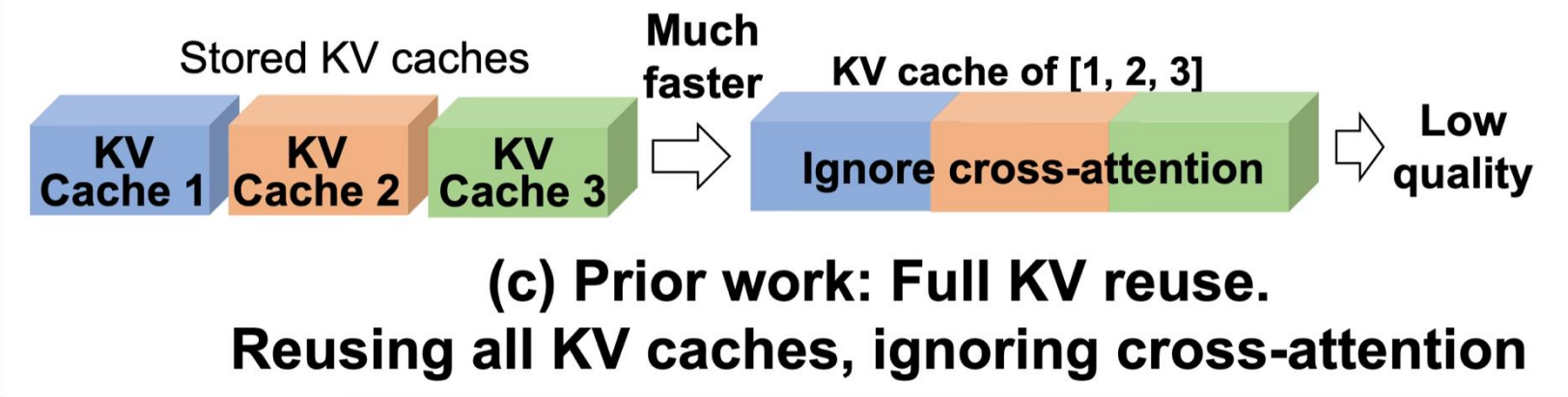


Fig. 6. Chunk's KV caches are stored in the database. It is retrieved, and simply concatenated. Since it ignores cross attention between chunks, it has low quality.

Previous Works – Full KV reuse

- Reusing all KV caches, ignoring cross-attention

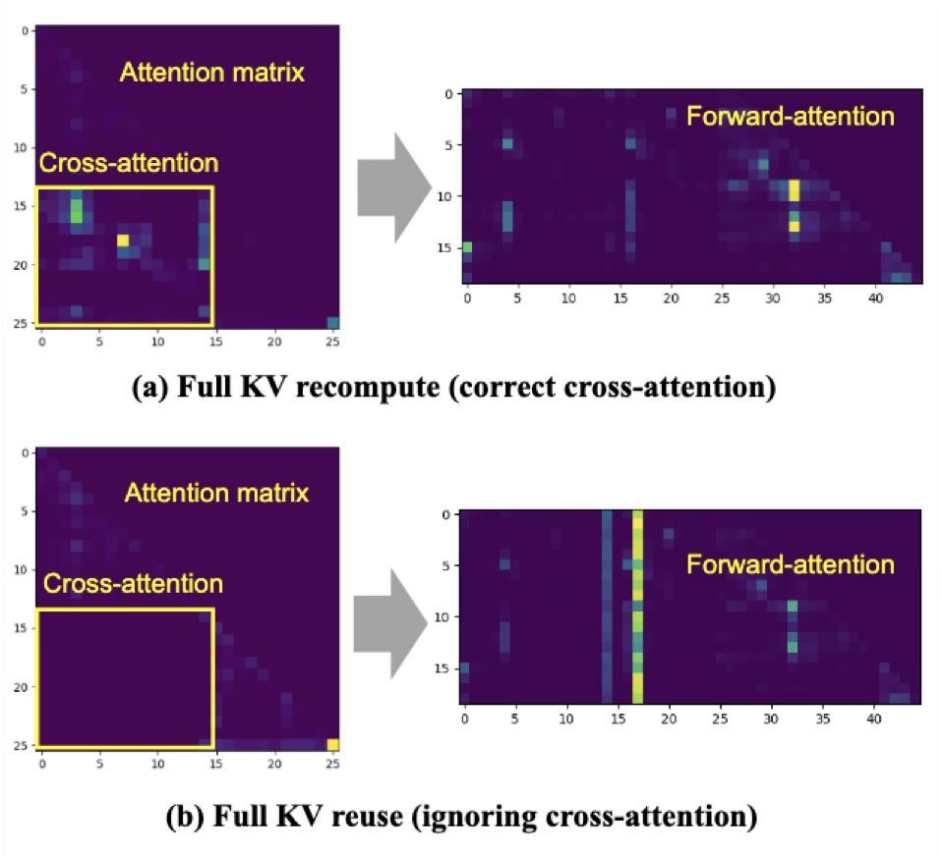


Fig. 7. Comparing Attention Matrix of Full KV recompute and Full KV reuse.

Previous Works – Full KV reuse

- Poor quality due to ignorance of cross-attention

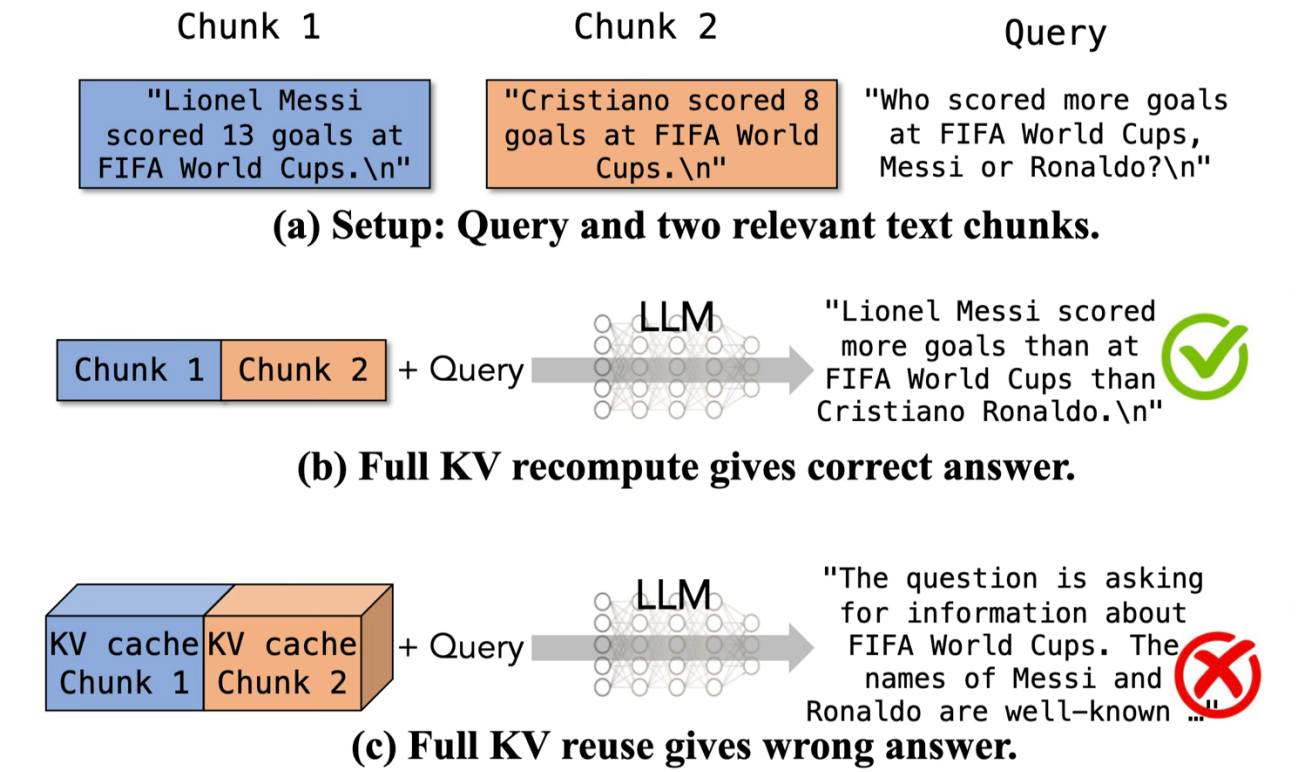


Fig. 8. Full KV reuse ignores cross-attention between chunks. As a result, information between chunks are not propagated to each other. This results in poor quality answers.

CacheBlend Details - Overview

- Selective KV Re-computation
  - Based on Full KV Reuse method, selectively pick important tokens and recompute KV by applying attention.

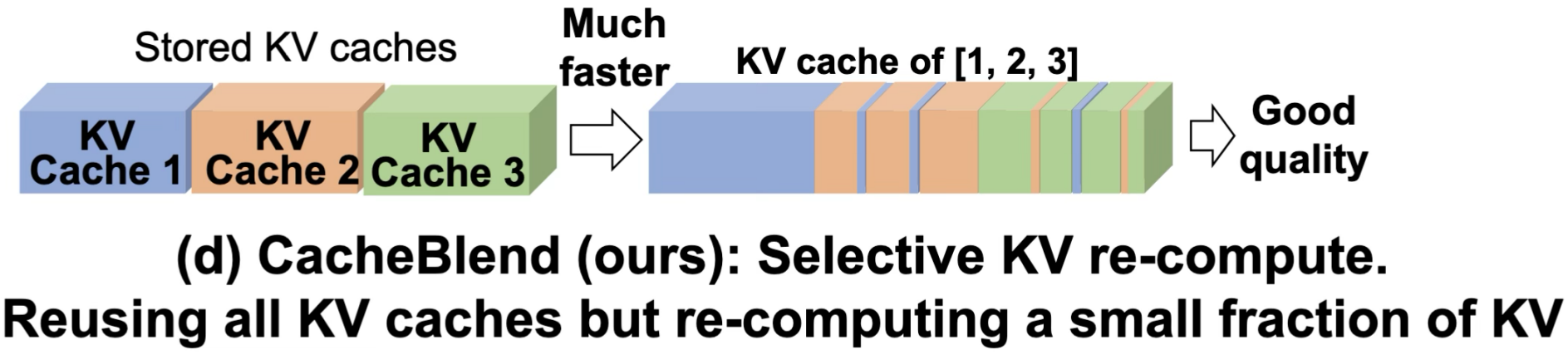


Fig. 9. Overview of CacheBlend.

CacheBlend Details - Overview

❑ Selective KV Re-computation

- Based on Full KV Reuse method, selectively pick important tokens and recompute KV by applying attention.

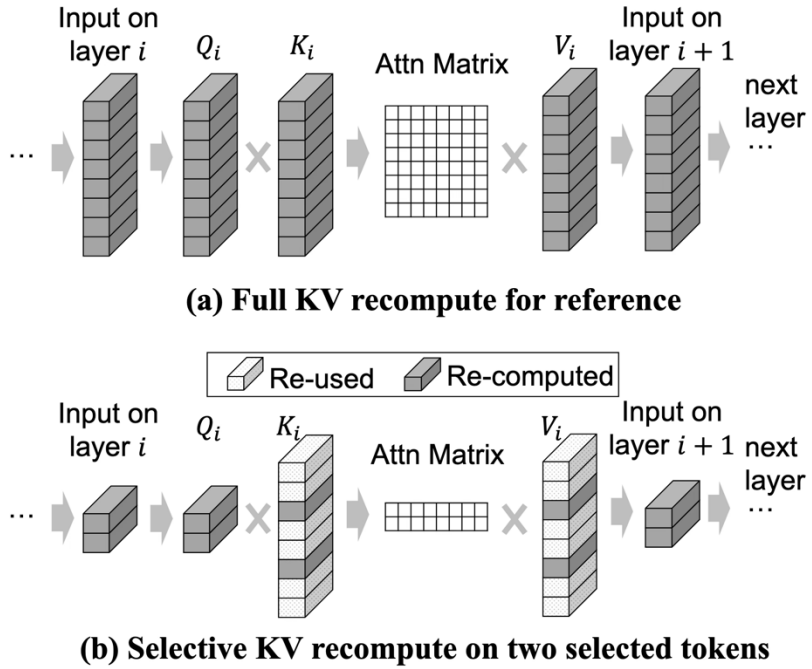
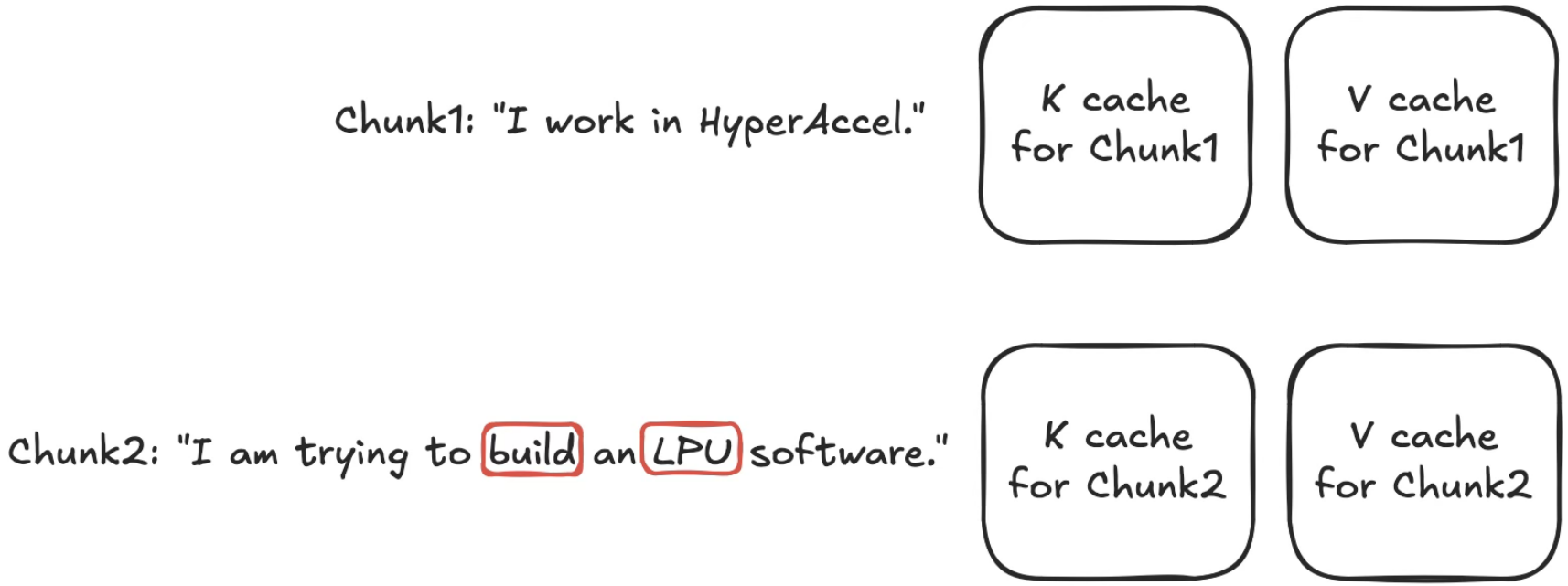


Fig. 10. Computation flow of Selective KV re-computation. In (b), it picks two token and recompute it's KV cache.

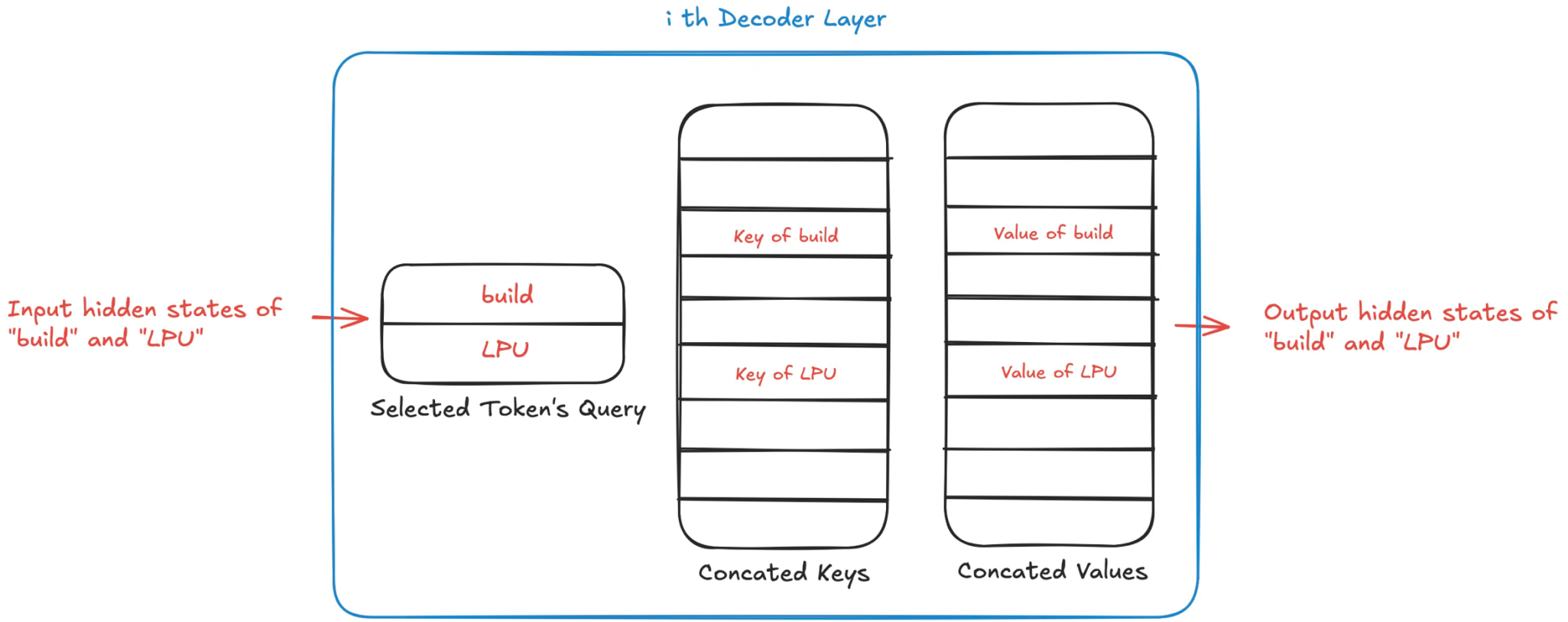
CacheBlend Details – Selective KV recomputation

- Examples
  - There are two chunks and its KV cache. Assume we know that the term “build” and “LPU” is important.



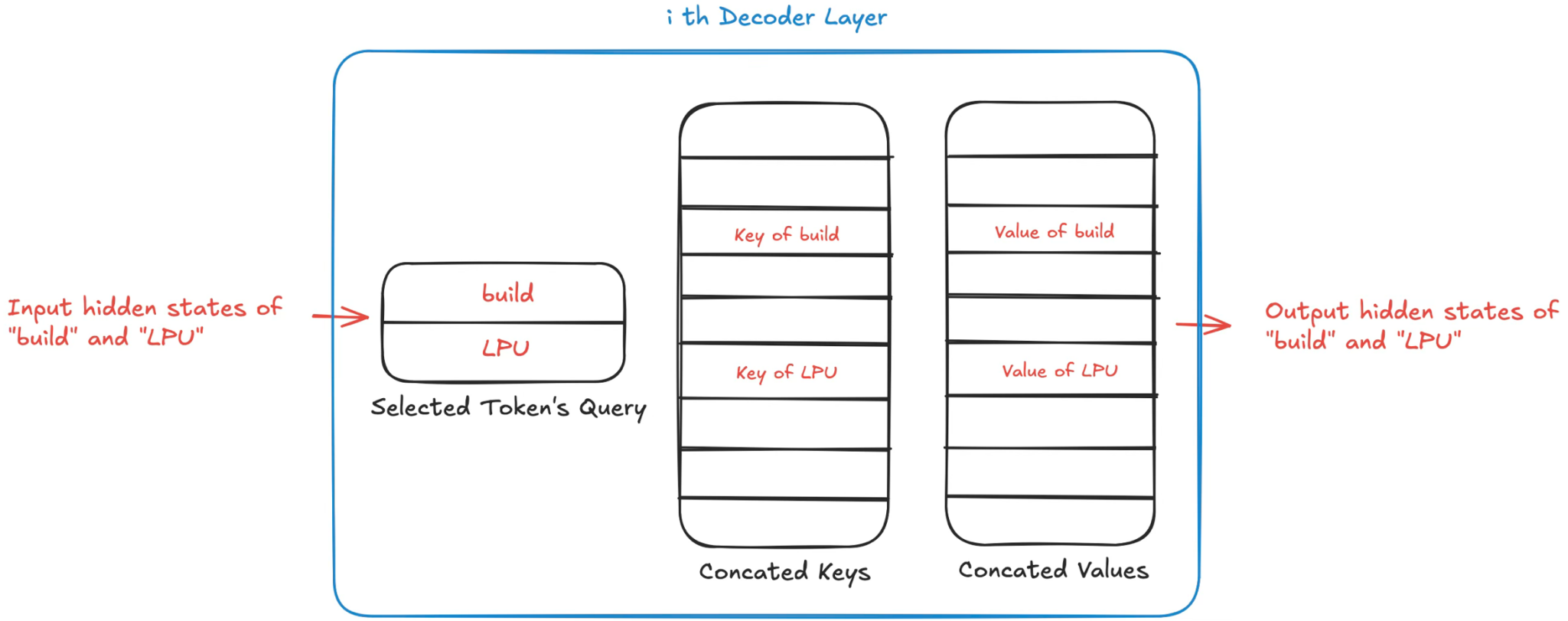
CacheBlend Details – Selective KV recomputation

- Examples
  - Update KV for “build” and “LPU”



CacheBlend Details – Selective KV recomputation

- Examples
  - Context of chunk1 will be aggregated to KV of “build” and “LPU”



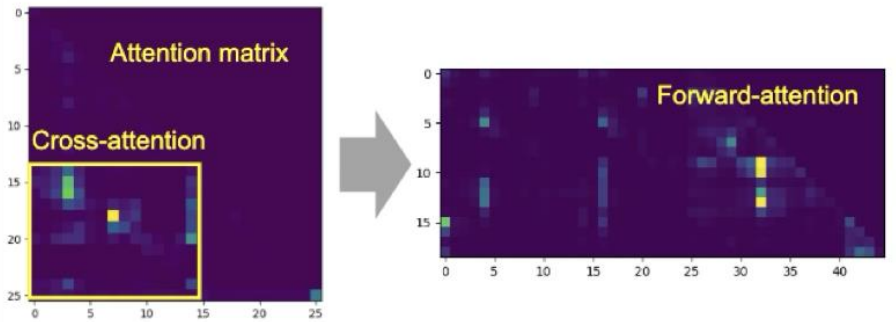
CacheBlend Details – Notations

Notation	Description
i	Layer index
j	Token index
KV	KV cache
KV_i	KV on layer i
$KV_i[j]$	KV on layer i at token j
KV^{full}	Fully recomputed KV cache
KV^{pre}	Pre-computed KV cache
KV^{new}	CacheBlend-updated KV cache
A_i	Forward attention matrix on layer i
A_i^{full}	Forward attention matrix of full KV recompute
A_i^{pre}	Forward attention matrix of full KV reuse
A_i^{new}	Forward attention matrix with CacheBlend
$\Delta_{\text{kv}}(KV_i, KV_i^{\text{full}})[j]$	KV deviation between $KV_i[j]$ and $KV_i^{\text{full}}[j]$
$\Delta_{\text{attn}}(A_i, A_i^{\text{full}})$	Attention deviation between A_i and A_i^{full}

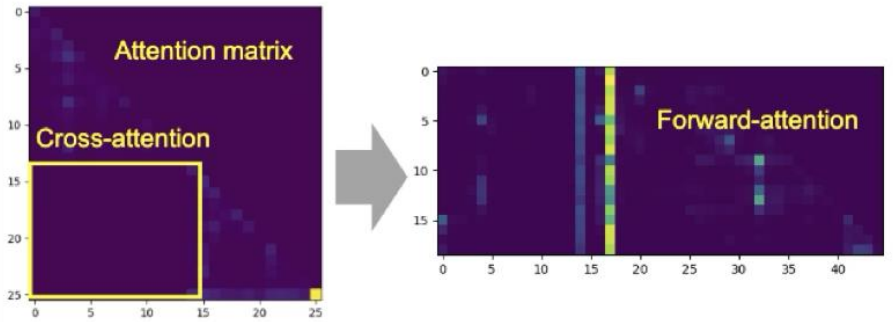
Fig. 11. Important notations.

CacheBlend Details – Selection Algorithm

- How to pick tokens that KV should be recomputed?
  - Objective: minimize Attention deviation



(a) Full KV recompute (correct cross-attention)



(b) Full KV reuse (ignoring cross-attention)

Fig. 12. Comparing attention matrix between "Full KV recompute" and "Full KV reuse". Cross-attention scores are all zero at Full KV reuse.

CacheBlend Details – Selection Algorithm

- How to pick tokens that KV should be recomputed?
  - Objective: minimize KV deviation

    → Update tokens that $KV_i[j]$ has largest gap with $KV_i^{full}[j]$

    i: layer id
    j: token position id

CacheBlend Details – Selection Algorithm

- How to pick tokens that KV should be recomputed?
  - Objective: minimize KV deviation

  → Update tokens that $KV_i[j]$ has largest gap with $KV_i^{full}[j]$

  → Contradiction: We need fully-recompute KV cache…

CacheBlend Details – Selection Algorithm

- How to pick tokens that KV should be recomputed?
  - Objective: minimize KV deviation

    → Update tokens that $KV_i[j]$ has largest gap with $KV_i^{full}[j]$

    → Contradiction: We need fully-recompute KV cache…

    → Intuition: Tokens with highest deviations on one layer are likely to have the highest KV deviations on the next layer.

CacheBlend Details – Selection Algorithm

- How to pick tokens that KV should be recomputed?
  - Objective: minimize KV deviation

    → Update tokens that $KV_i[j]$ has largest gap with $KV_i^{full}[j]$

    → Contradiction: We need fully-recompute KV cache…

    → Intuition: Tokens with highest deviations on one layer are likely to have the highest KV deviations on the next layer.

    → Solution: Fully-recompute the KV of second(layer_id=1) layer, and find HKVD(Highest KV Deviation) tokens. Starting from third layer, we can just use HKVD tokens of first layer.

CacheBlend Details – System Design

- Minor Details – Hiding KV recomputation via pipelining
  - We pick $r\%$ of tokens to recompute. The computation time can be hidden by KV loading operation.

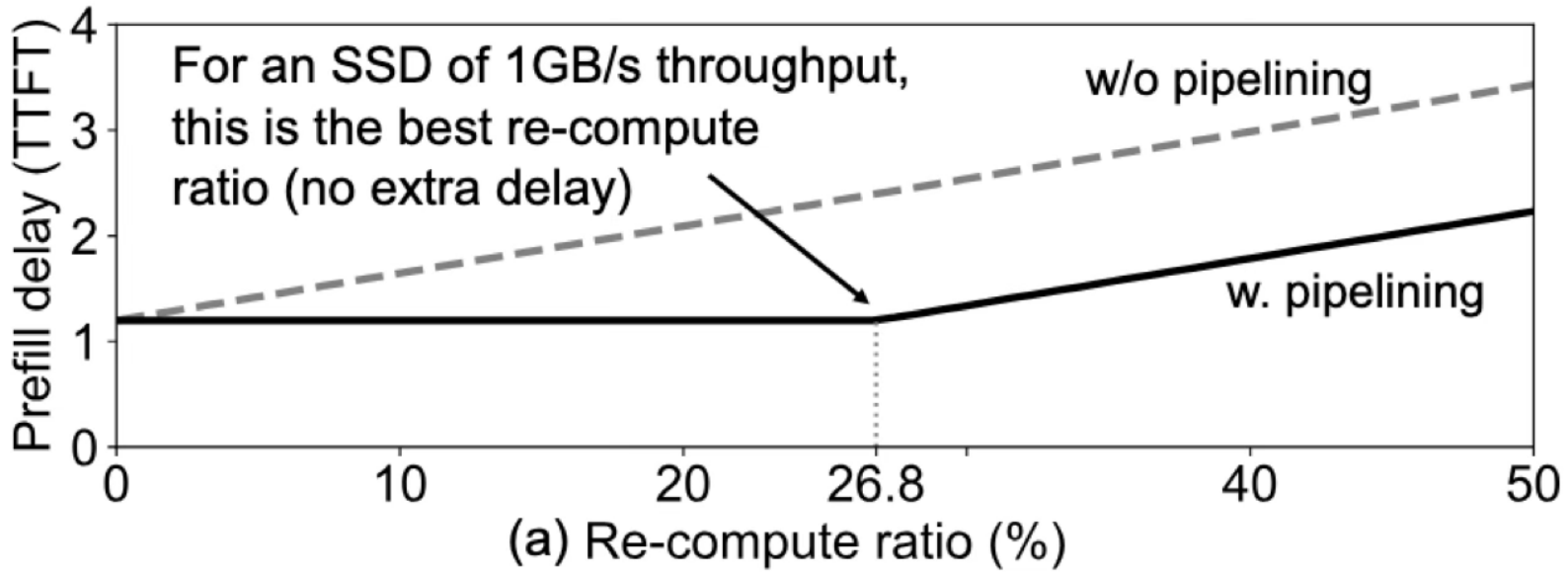


Fig. 13. Relation between re-compute ratio and TTFT. By applying pipelining, re-computation time is hidden by KV loading time.

CacheBlend Details – System Design

❑ Minor Details – Gradual Filtering

- HKVD tokens of first layer is not always the right answer. So we gradually decrease the re-compute ratio.

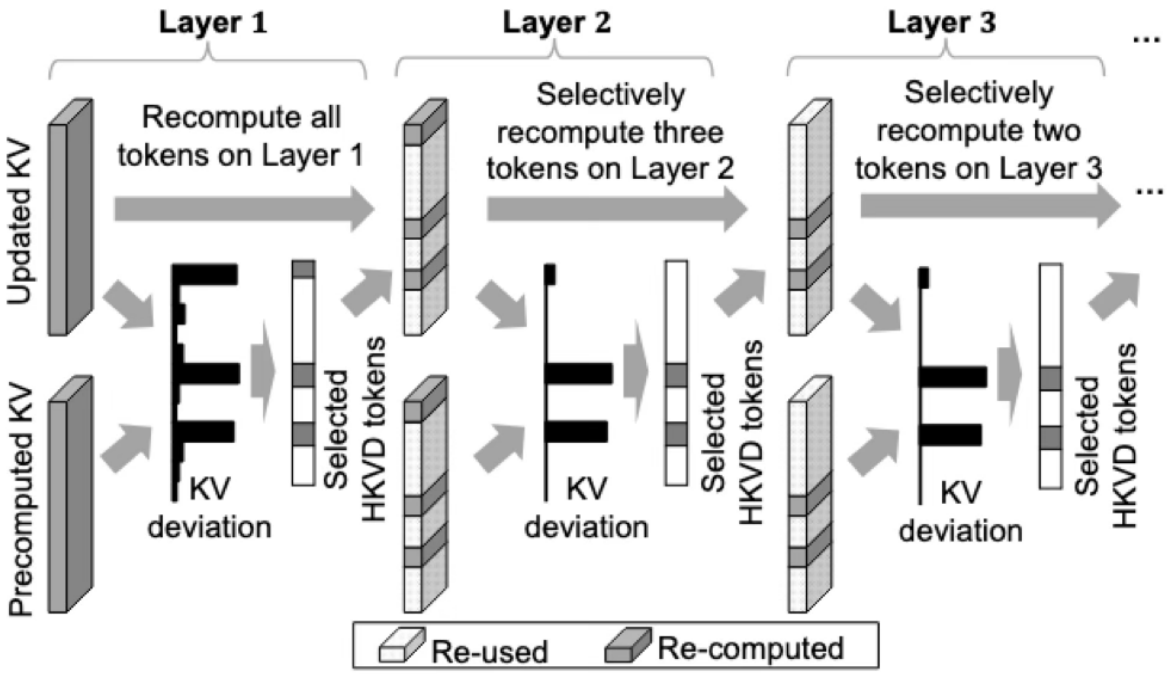


Fig. 14. Gradually lower the re-compute ratio to get reliable HKVD tokens.

HYPER ACCEL

CacheBlend Details – System Design

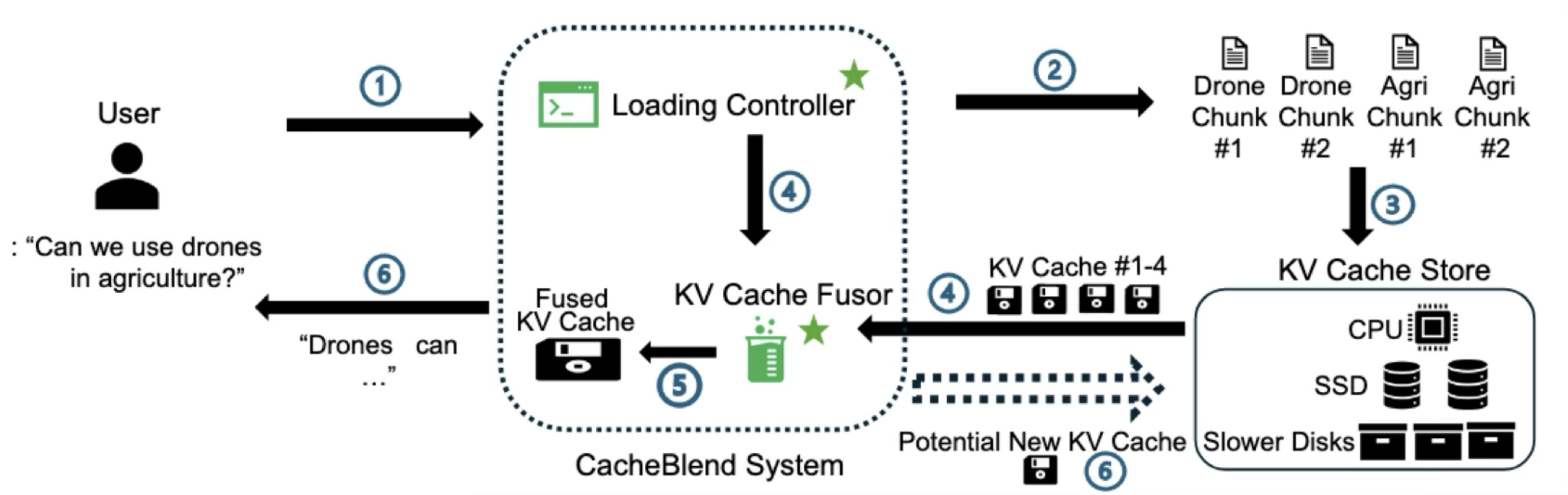


Fig. 15. Overall architecture of LLM serving system with CacheBlend.

CacheBlend Details – Evaluation

- Evaluation
  - Reduced TTFT by 2.2-3.3x
  - Increased the inference throughput by 2.8-5x

Integration with CacheGen - Introduction

- Problem with CacheBlend
  - In CacheBlend, huge KV cache is usually stored in DDR or SSD. It is transferred through network when needed. Network delay is massive.

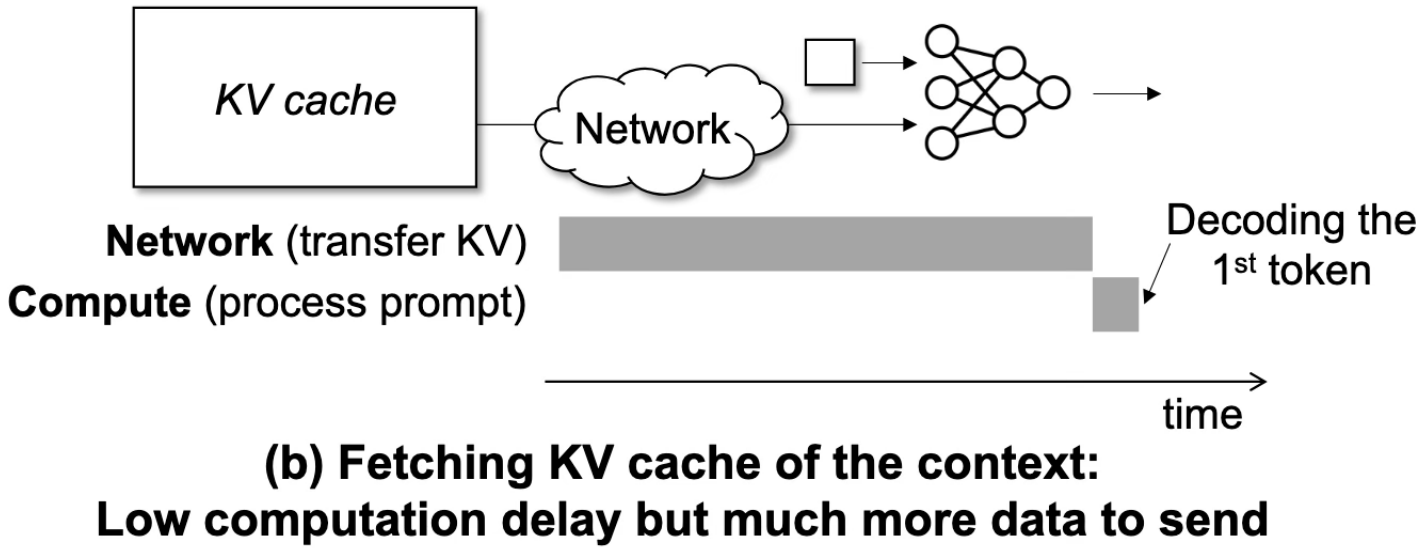


Fig. 16. Analysis of CacheBlend bottleneck. Transferring huge KV cache through network takes too much time.

Integration with CacheGen - Introduction

- What is CacheGen?
  - CacheGen is KV compress-decompress algorithm.

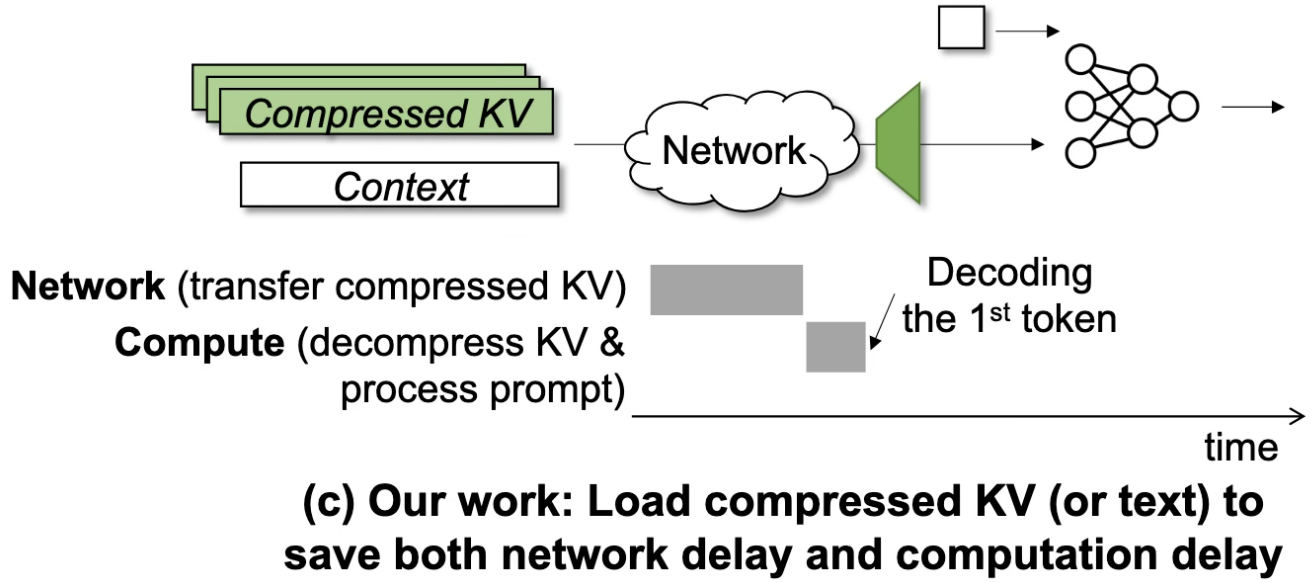


Fig. 17. CacheGen compress the KV cache to shrink the transfer latency. If network congestion is severe, then instead of sending compressed KV cache, it sends the context(corpus).

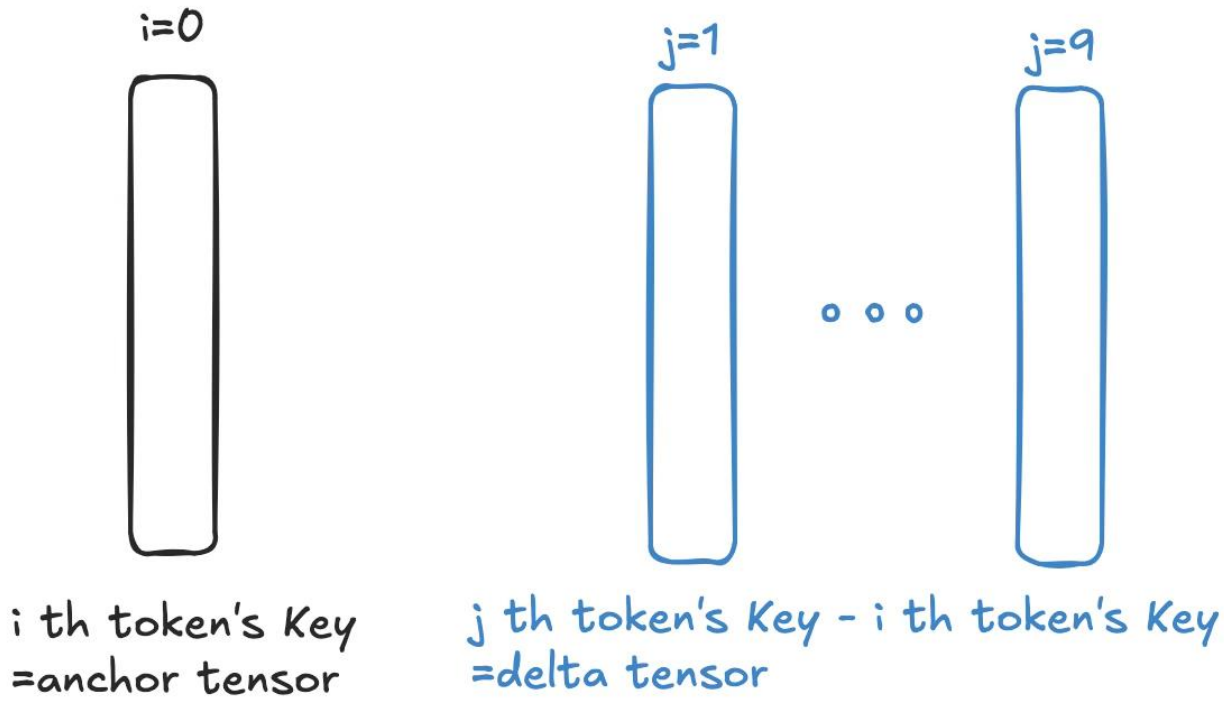
Integration with CacheGen - Details

- CacheGen considers following characteristics of KV cache.
  - Token-wise Locality
  - Output quality sensitivity to layer depth
  - Entropy of KV cache differs by grouping strategy

Integration with CacheGen - Details

- Token-wise Locality
  - KV cache of nearby tokens are similar to each other.

    → Let's use anchor tensor and delta tensor scheme. This will minimize the variance of delta tensors, and benefit high precision in quantization.



Integration with CacheGen - Details

- Output quality sensitivity to layer depth
  - Output quality is more sensitive to KV cache of shallow layer.

  → Apply different compression level by layer depth. Dense quantization(more bits) for shallow layer KV cache.

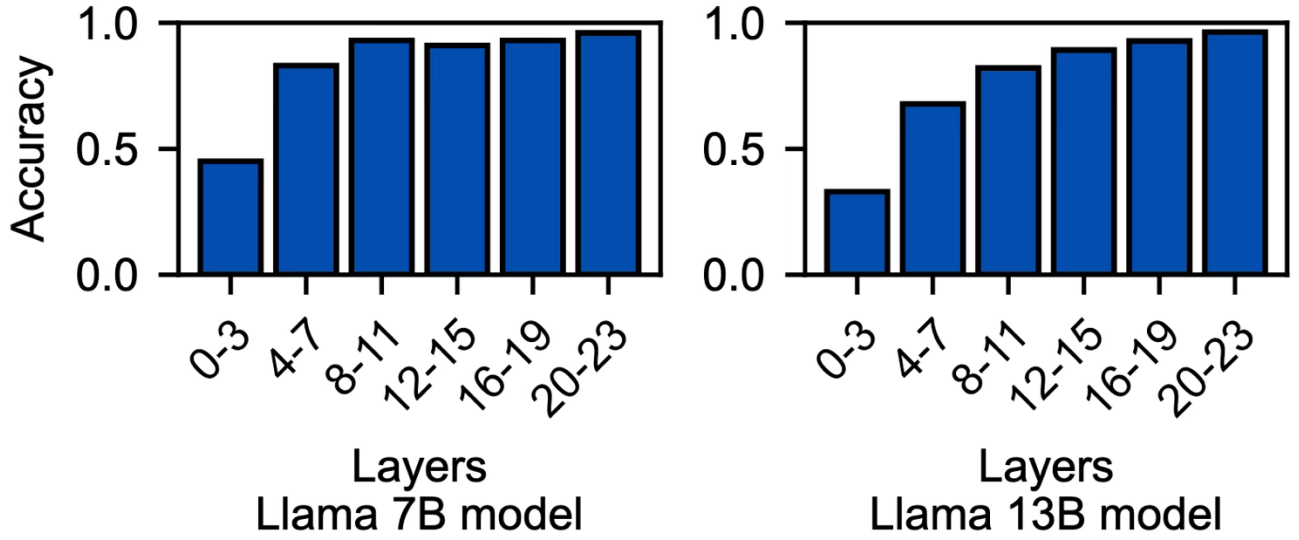


Fig. 18. How much output quality(accuracy) degrades if we round each layer's KV cache. Rounding KV cache of shallow layers severely degraded the quality.

Integration with CacheGen - Details

- Entropy of KV cache differs by grouping strategy
  - Entropy gets smaller as we group by channel or layer.

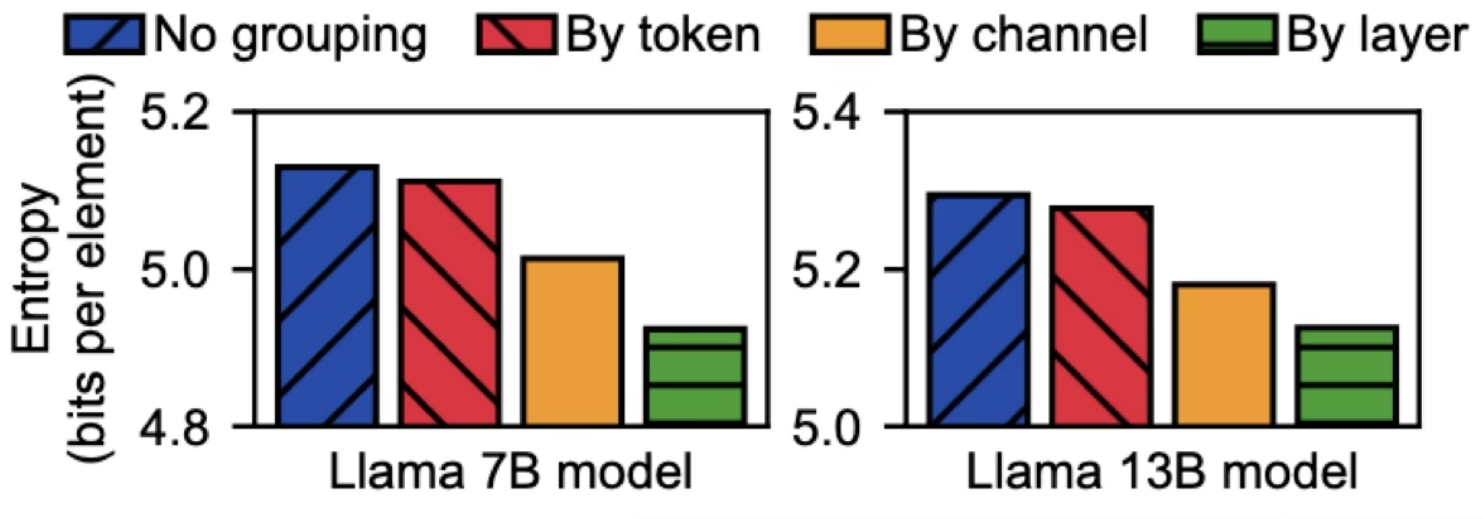
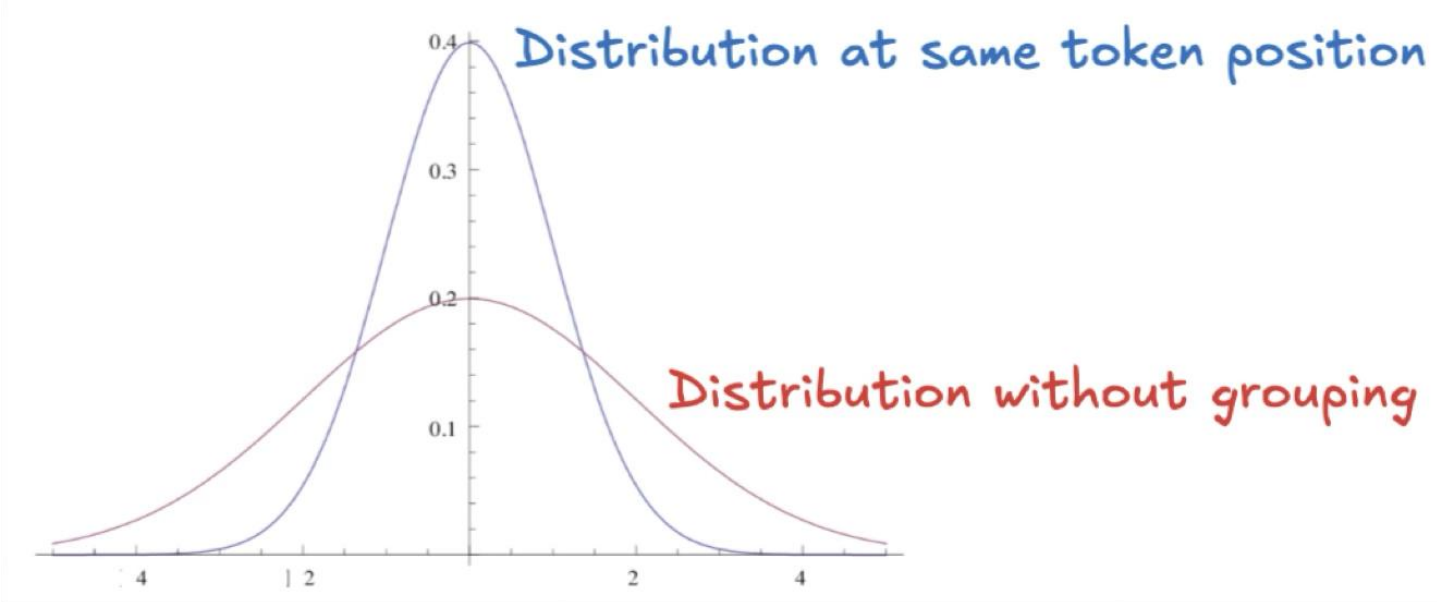


Fig. 19. Comparing entropy if it is grouped by different dimensions. By grouping by layer, we can use less bits to represent KV cache.

Integration with CacheGen - Details

- Entropy of KV cache differs by grouping strategy
  - Entropy gets smaller as we group by channel or layer.

  → We can use lesser bits if we group KV cache by layer or channel.



Integration with CacheGen - Details

- Arithmetic Coding
  - Arithmetic coding is a method of encoding/decoding sequence into floating point range.

Integration with CacheGen - Details

❑ Arithmetic Coding

- Arithmetic coding is a method of encoding/decoding sequence into floating point range.

Probability of "A": 0.6

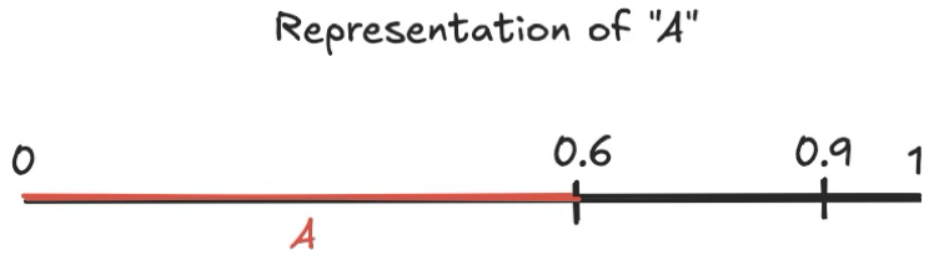
Probability of "B": 0.3

Probability of "C": 0.1

Integration with CacheGen - Details

❑ Arithmetic Coding

- Arithmetic coding is a method of encoding/decoding sequence into floating point range.



Probability of "A": 0.6

Probability of "B": 0.3

Probability of "C": 0.1

Integration with CacheGen - Details

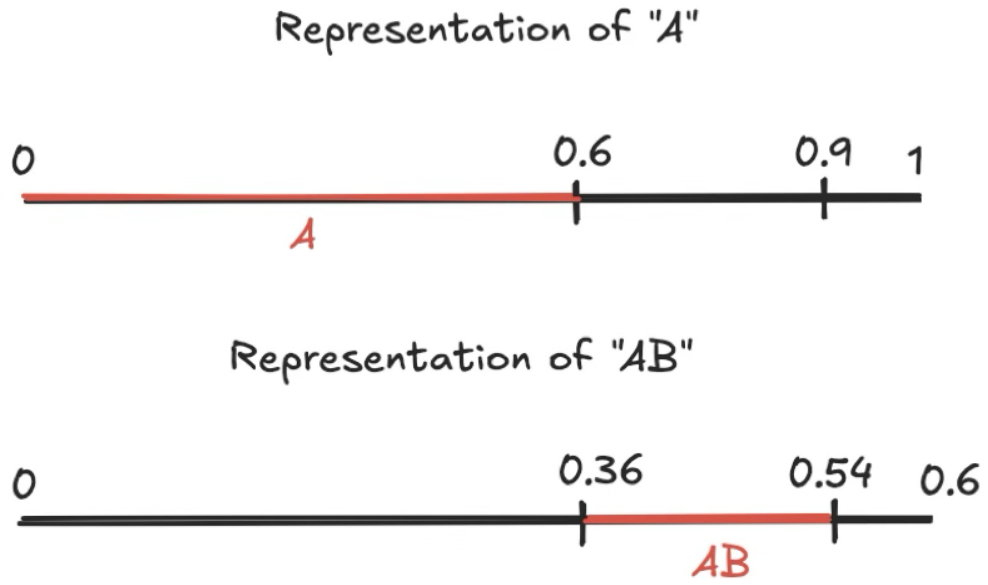
❑ Arithmetic Coding

- Arithmetic coding is a method of encoding/decoding sequence into floating point range.

Probability of "A": 0.6

Probability of "B": 0.3

Probability of "C": 0.1



Integration with CacheGen - Details

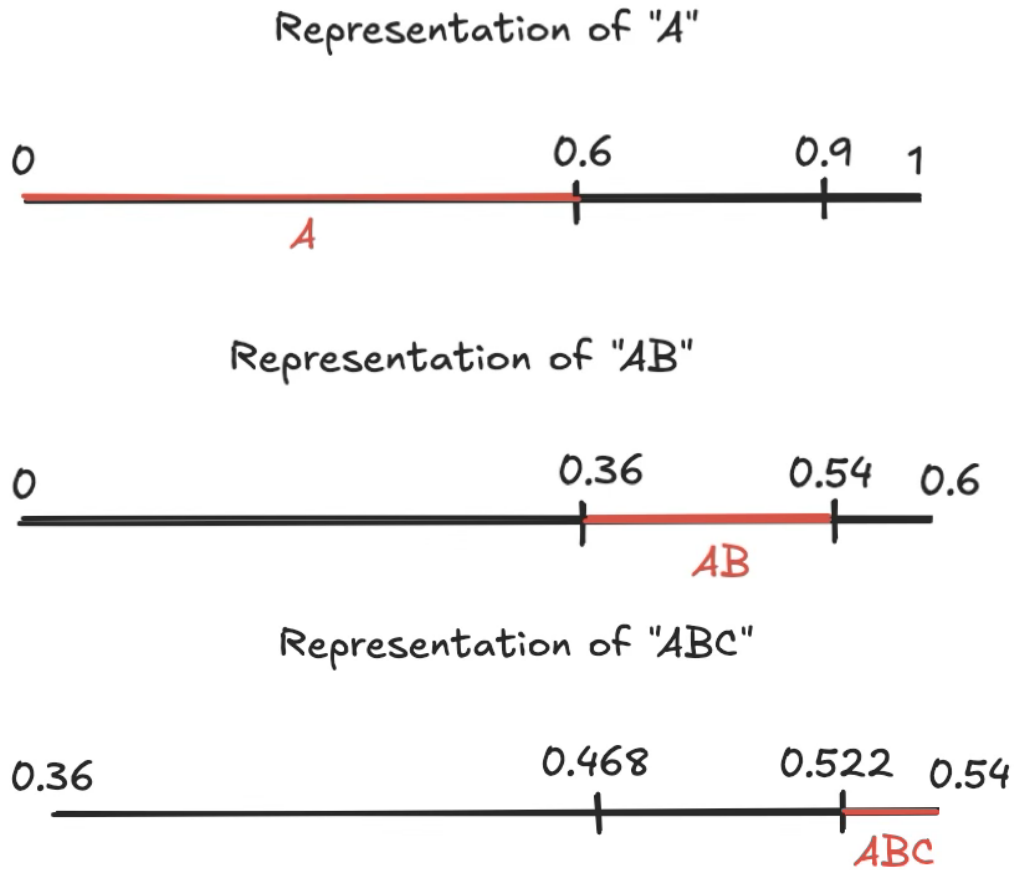
❑ Arithmetic Coding

- Arithmetic coding is a method of encoding/decoding sequence into floating point range.

Probability of "A": 0.6

Probability of "B": 0.3

Probability of "C": 0.1



HYPER ACCEL

Integration with CacheGen - Details

- CacheGen compress KV cache by following steps
  1. Convert KV cache into anchor/delta tensor scheme.
  2. Group anchor/delta tensors by layer.
  3. Apply quantization to each group(ex. INT8). Apply dense quantization(ex. INT16) to shallow layers.
  4. Apply arithmetic encoding to convert sequences into floating point.

Integration with CacheGen - Details

- CacheGen compressed KV transmission size with negligible impact on accuracy.

Technique	KV cache size (in MB, lower the better)	Accuracy (higher the better)
8-bit quantization	622	1.00
CacheGen (this paper)	176	0.98
H2O [153]	282	0.97
CacheGen on H2O	71	0.97
LLMLingua [72]	492	0.94
CacheGen on LLMLingua	183	0.94

Fig. 20. Comparing KV size. CacheGen significantly compressed the KV cache with negligible accuracy drop.

Reference

[1] Jiayi Yao, et al. 2025. CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion. EuroSys (2025), 94-109.

[2] Yuhan Liu, et al. 2024. CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving. ACM SIGCOMM (2024), 38-56.

[3] https://github.com/LMCache/LMCache