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

Author: Jiayi Yao, Junchen Jiang, etc. University of Chicago

Presenter: Yicheng Zhang USTC

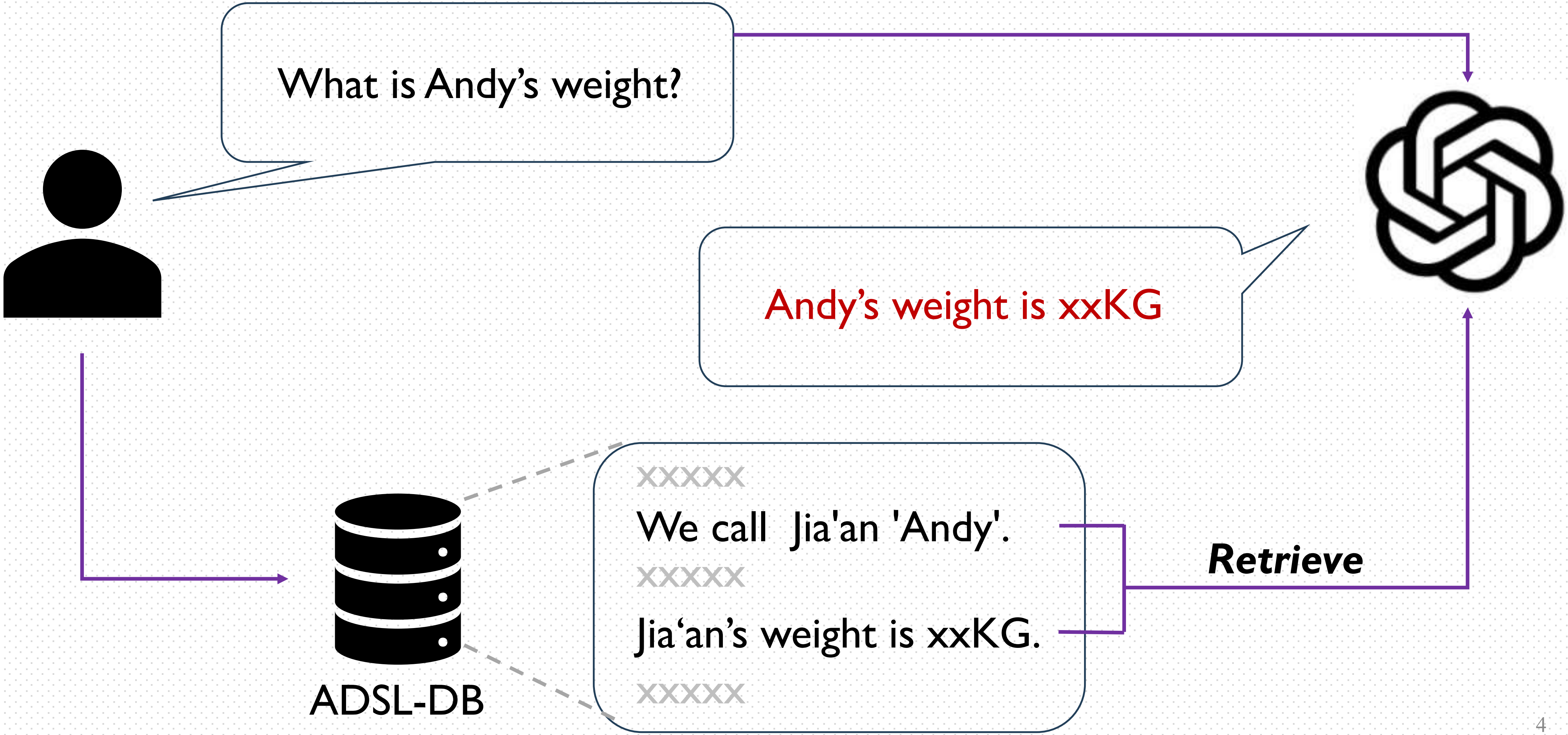
Outline

Retrieval-Augmented Generation

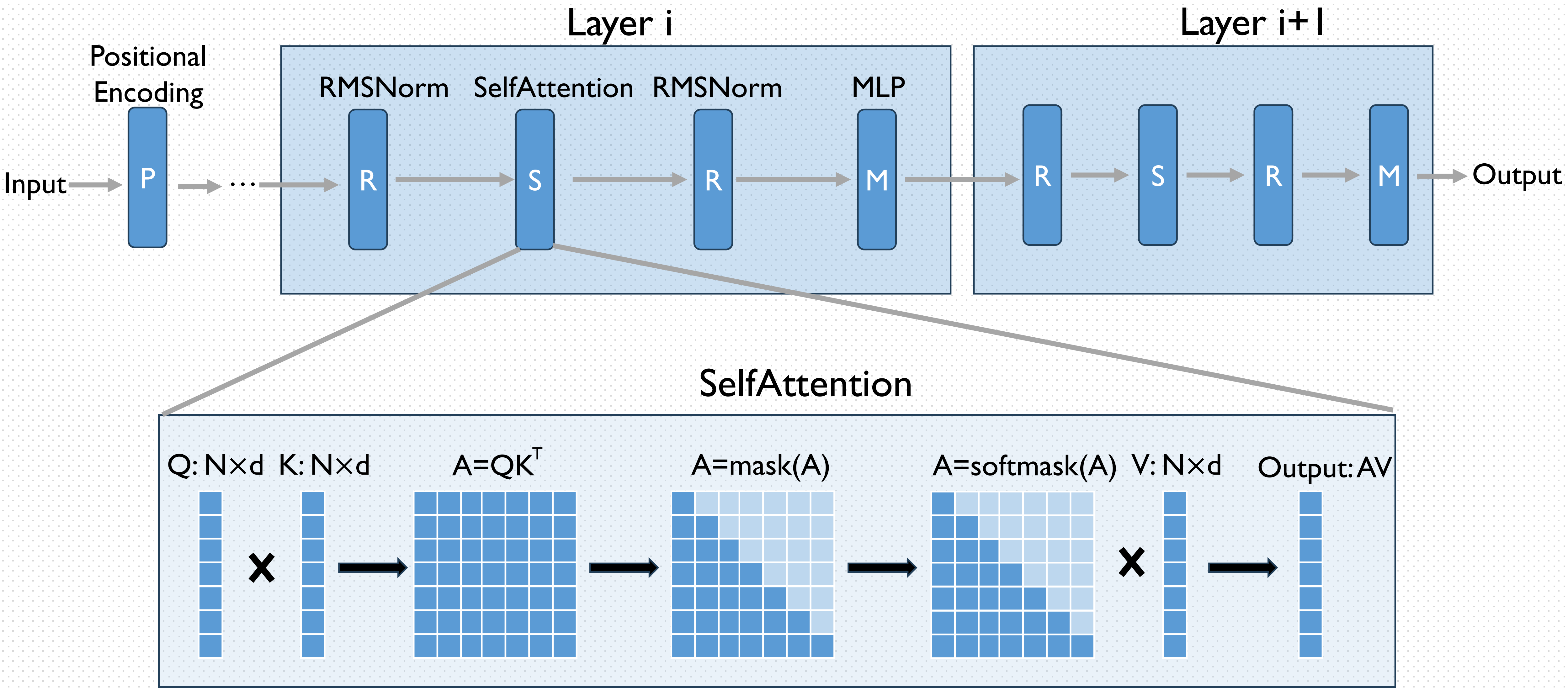
What is Andy’s weight?

Without additional context, I don’t have information about Andy's weight.

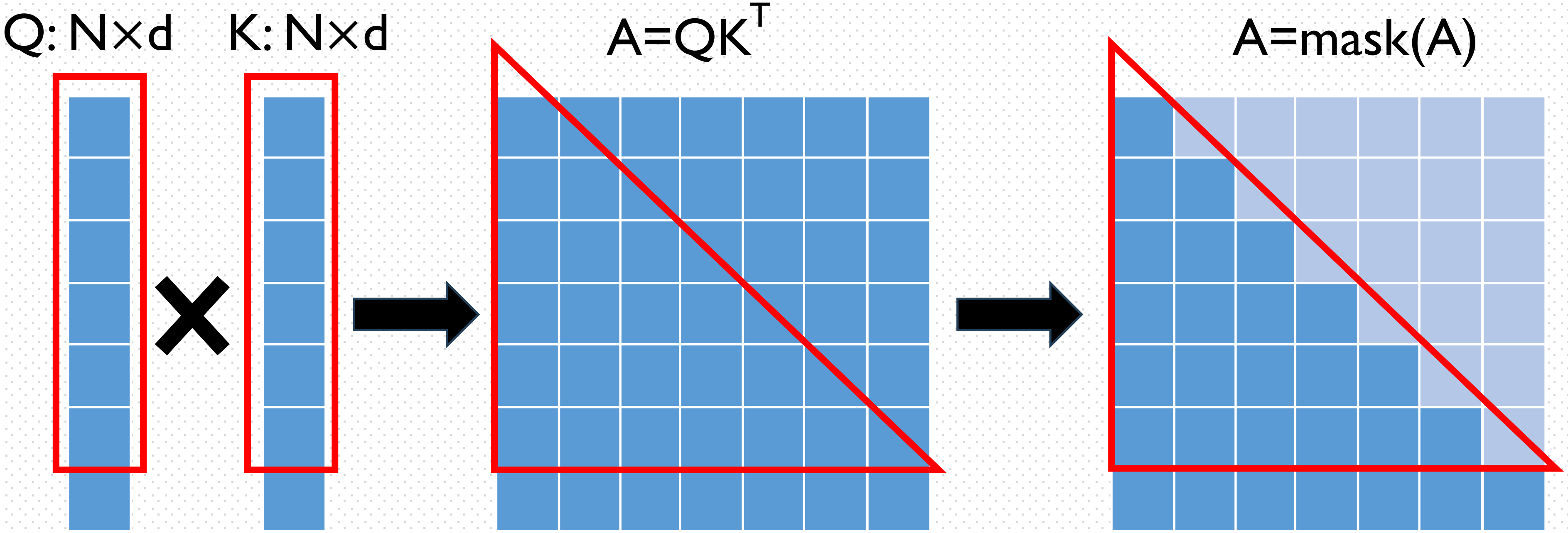
Retrieval-Augmented Generation



KV Cache: in Prefill State



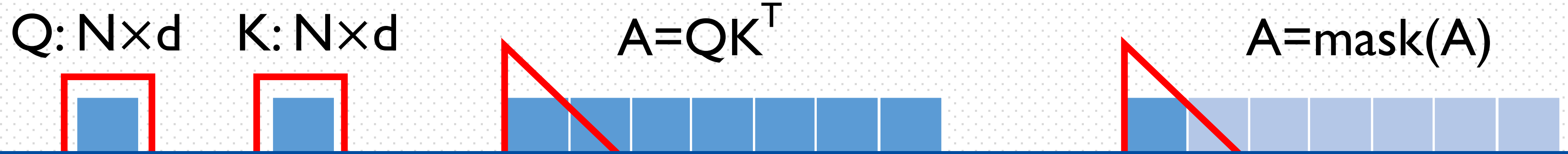
KV Cache: Reuse the KV Cache of same prefix



Each token's attention output depends only on itself and all preceding tokens

Same prefix ⟶ Same attention result

KV Cache: Reuse the KV Cache of same prefix



Can we use KV cache in RAG?

Each token's attention output depends only on itself and all preceding tokens

Same prefix → Same attention result

Reuse KV in RAG: Challenge

Only chunk1's KV can be reused!

Position & Cross-chunk Attention effect

Reuse KV in RAG: Challenge

Chunk1 | Chunk2 | Chunk3

input

Only chunk1's KV can be reused!

Position: Different position means different positional encoding result, KV is different

empty | Chunk2

is not equal to

Chunk2 | empty

Reuse KV in RAG: Challenge

Chunk1 | Chunk2 | Chunk3

input

Only chunk1's KV can be reused!

Position: Different position means different positional encoding result, KV is different

empty | Chunk2

is not equal to

Chunk2 | empty

Can be solved by multiplying the K vector by a rotation matrix

Reuse KV in RAG: Challenge

Chunk1	Chunk2	Chunk3

$\xrightarrow{\text{input}}$

Only chunk1's KV can be reused!

Cross-chunk Attention: Chunks effect attention result of each others

Layer1:

T_1
T_2
T_3

$\Rightarrow$

$Q_1K_1^T$		
$Q_2K_1^T$	$Q_2K_2^T$	
$Q_3K_1^T$	$Q_3K_2^T$	$Q_3K_1^T$

$\Rightarrow$

$Q_1K_1^TV_1$
$Q_2K_1^TV_2+Q_2K_2^TV_2$
$Q_3K_1^TV_3+Q_3K_2^TV_2+Q_3K_3^TV_3$

Referred as

T_1'
T_2'
T_3'

In layer1, we can reuse T_2's KV Cache

Reuse KV in RAG: Challenge

Only chunk1's KV can be reused!

Cross-chunk Attention: Chunks effect attention result of each others

Layer1: $\begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} \Rightarrow \begin{bmatrix} Q_1K_1^T & & \\ Q_2K_1^T & Q_2K_2^T & \\ Q_3K_1^T & Q_3K_2^T & Q_3K_1^T \end{bmatrix} \Rightarrow \begin{bmatrix} Q_1K_1^TV_1 \\ Q_2K_1^TV_2+Q_2K_2^TV_2 \\ Q_3K_1^TV_3+Q_3K_2^TV_2+Q_3K_3^TV_3 \end{bmatrix}$ Referred as $\begin{bmatrix} T_1' \\ T_2' \\ T_3' \end{bmatrix}$

In layer1, we can reuse T_2's KV Cache

Layer2: $\begin{bmatrix} T_1' \\ T_2' \\ T_3' \end{bmatrix} \Rightarrow \begin{bmatrix} Q_1' K_1'^T & & \\ Q_2' K_1'^T & Q_2' K_2'^T & \\ Q_3' K_1'^T & Q_3' K_2'^T & Q_3' K_3'^T \end{bmatrix}$

Reuse KV in RAG: Challenge

Chunk1	Chunk2	Chunk3

input →

Only chunk1's KV can be reused!

Cross-chunk Attention: Chunks effect attention result of each others

Layer1:

T_1	$Q_1K_1^T$			$Q_1K_1^TV_1$	T_1'
T_2	$Q_2K_1^T$	$Q_2K_2^T$		$Q_2K_1^TV_2+Q_2K_2^TV_2$	T_2'
T_3	$Q_3K_1^T$	$Q_3K_2^T$	$Q_3K_1^T$	$Q_3K_1^TV_3+Q_3K_2^TV_2+Q_3K_3^TV_3$	T_3'

Referred as

In layer1, we can reuse T_2's KV Cache

Layer2:

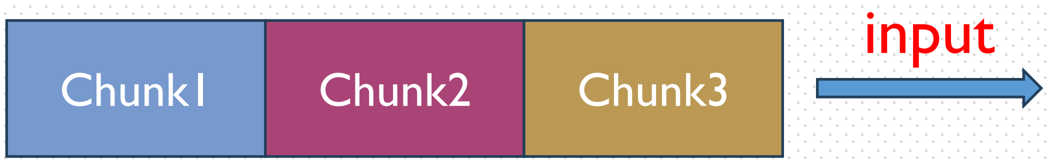
T_1'	$Q_1'\ K_1'^T$		
T_2'	$\boxed{Q_2'\ K_1'^T}$	$Q_2'\ K_2'^T$	
T_3'	$Q_3'\ K_1'^T$	$Q_3'\ K_2'^T$	$Q_3'\ K_3'^T$

$$Q_2'\ K_1'^T = W_Q(Q_2K_1^TV_2+Q_2K_2^TV_2)(W_KQ_1K_1^TV_1)^T$$

However, In layer2, we cannot reuse T_2''s KV Cache

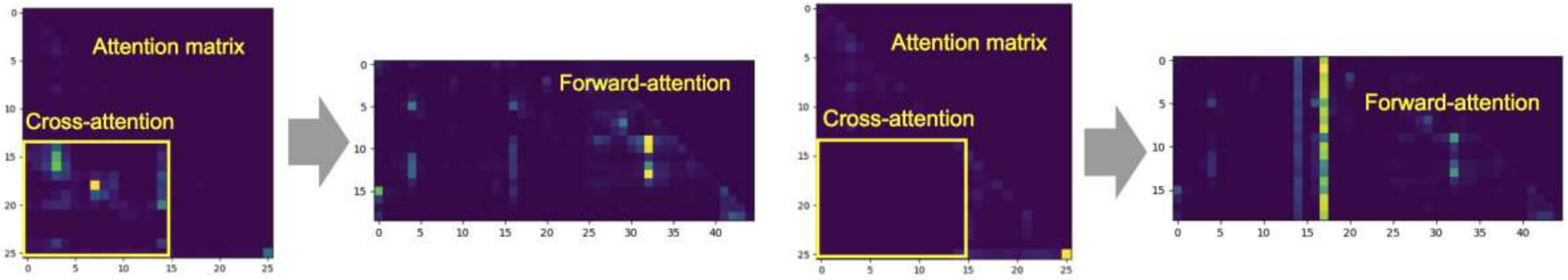
For T_2' cannot be precomputed

Reuse KV in RAG: Challenge



Only chunk1's KV can be reused!

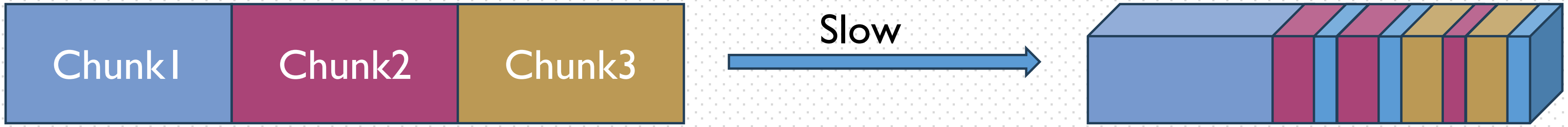
Cross-chunk Attention: Chunks effect attention result of each others



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

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

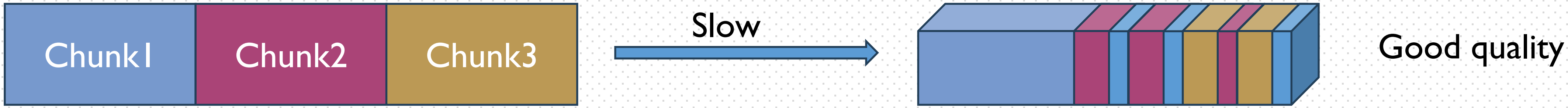
Reuse KV in RAG : Existing Solutions



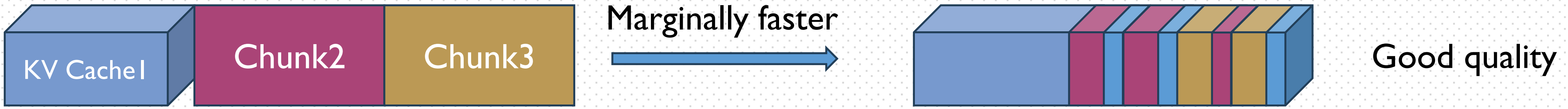
Good quality

Full KV re-compute

Reuse KV in RAG : Existing Solutions

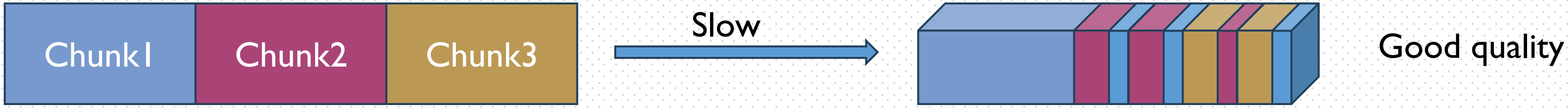


Full KV re-compute

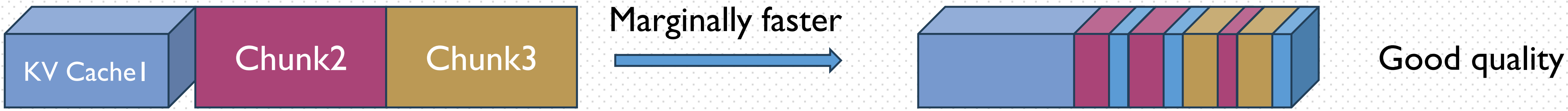


Use same prefix's KV: RAGCache

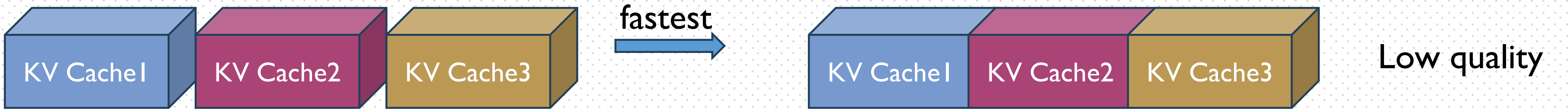
Reuse KV in RAG : Existing Solutions



Full KV re-compute

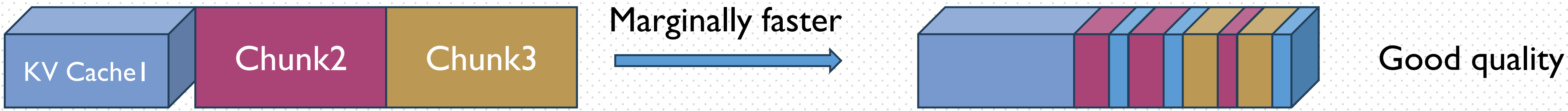


Use same prefix's KV: RAGCache



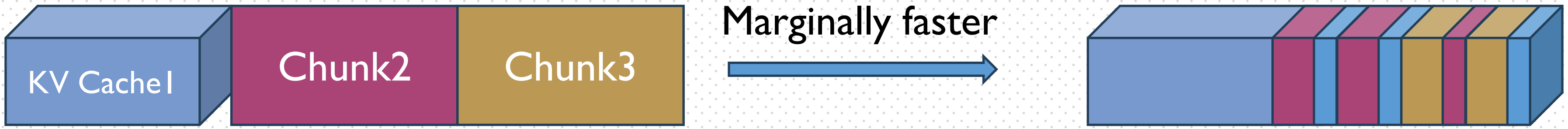
Full KV reuse (ignore cross-chunk attention)**:** PromptCache

Reuse KV in RAG : Existing Solutions



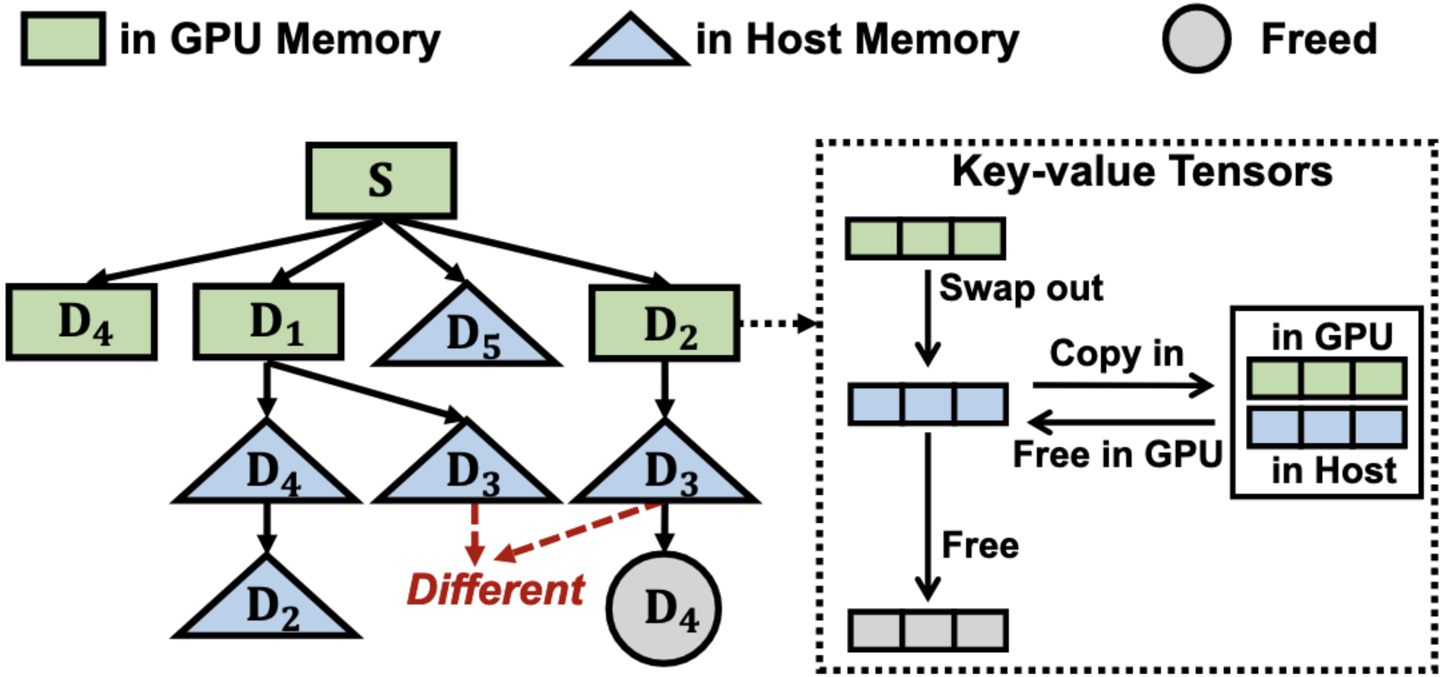
Use same prefix's KV: RAGCache

Reuse KV in RAG : Existing Solutions



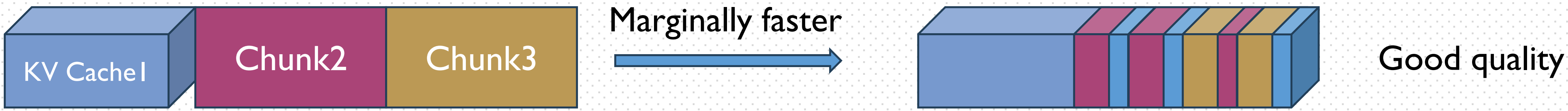
Good quality

Use same prefix's KV: RAGCache

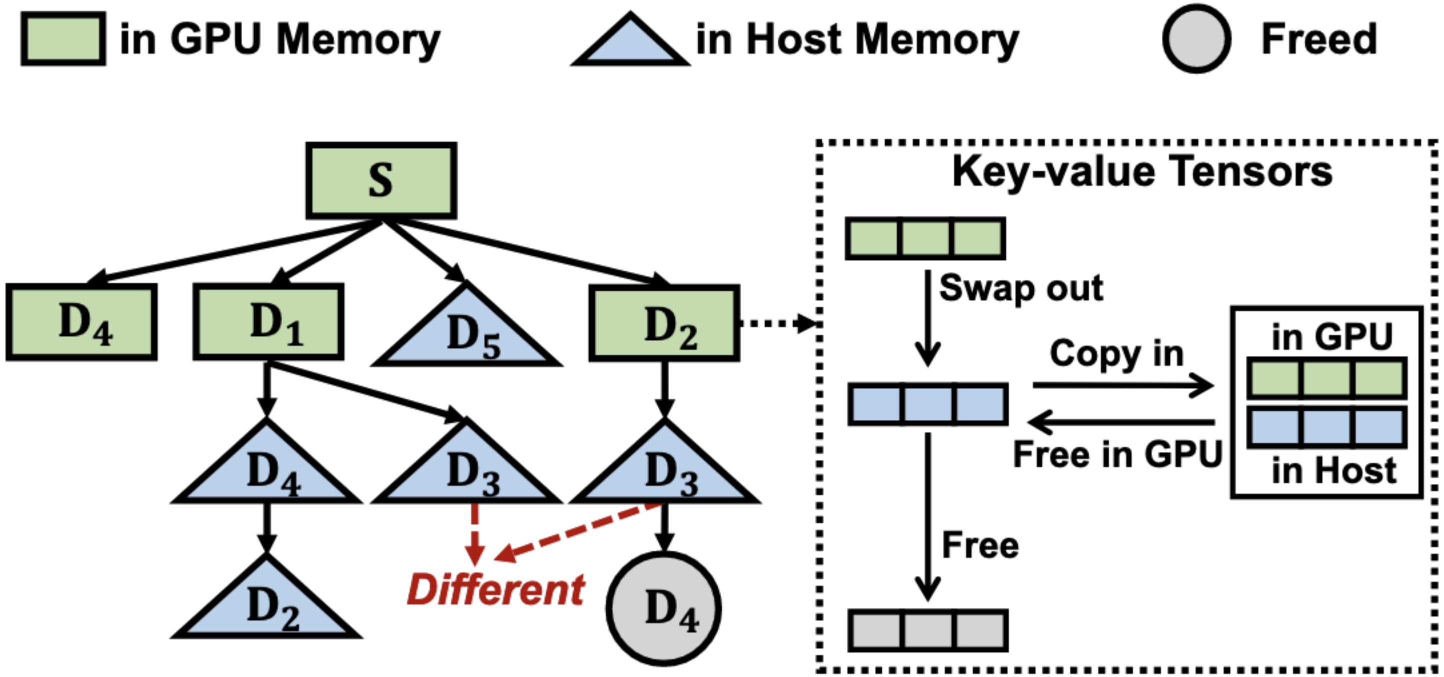


Prefix Caching: RAGCache adopts knowledge tree to cache hot prefix's KV

Reuse KV in RAG : Existing Solutions



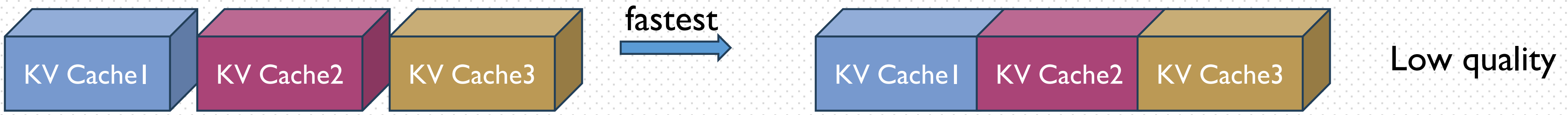
Use same prefix's KV: RAGCache



Prefix Caching: RAGCache adopts knowledge tree to cache hot prefix's KV

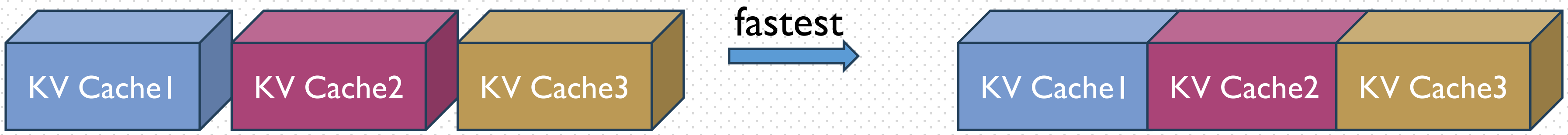
Cons: cannot work when prefix is not the same

Reuse KV in RAG : Existing Solutions



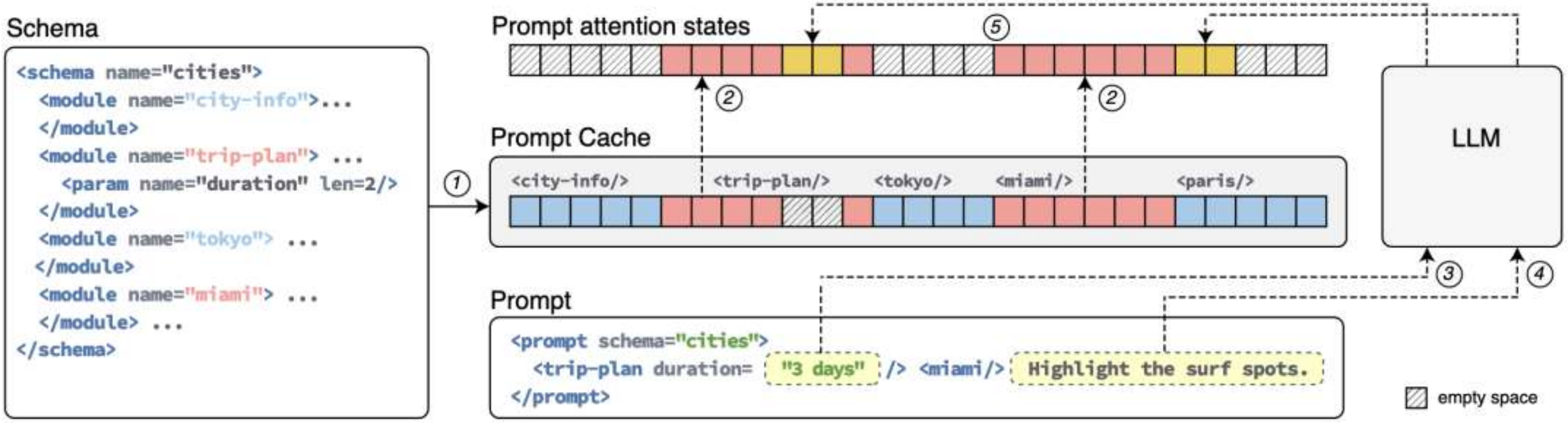
Full KV reuse (ignore cross-chunk attention): PromptCache

Reuse KV in RAG : Existing Solutions

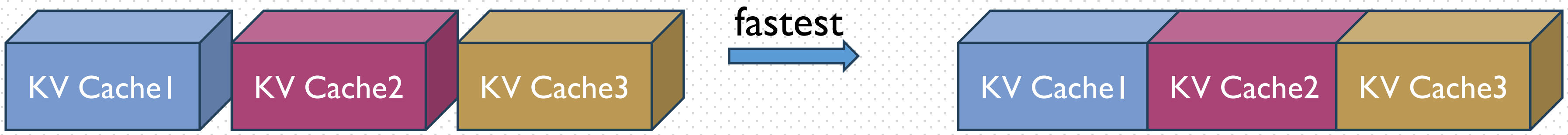


Low quality

Full KV reuse (ignore cross-chunk attention): PromptCache

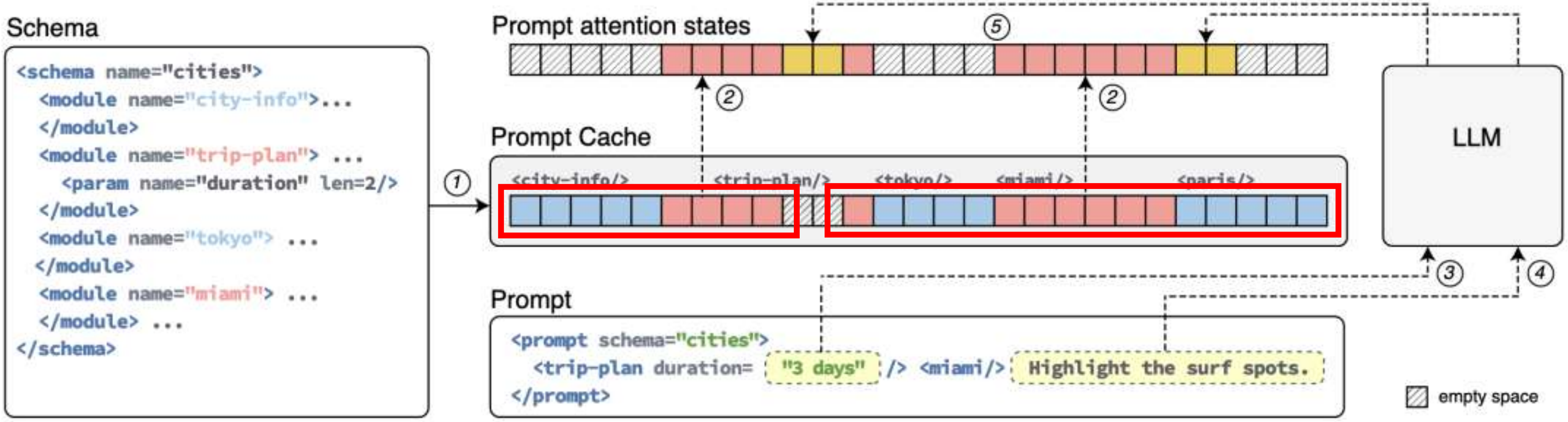


Reuse KV in RAG : Existing Solutions



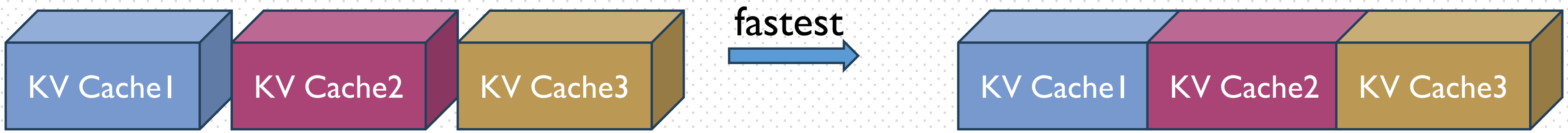
Low quality

Full KV reuse (ignore cross-chunk attention): PromptCache



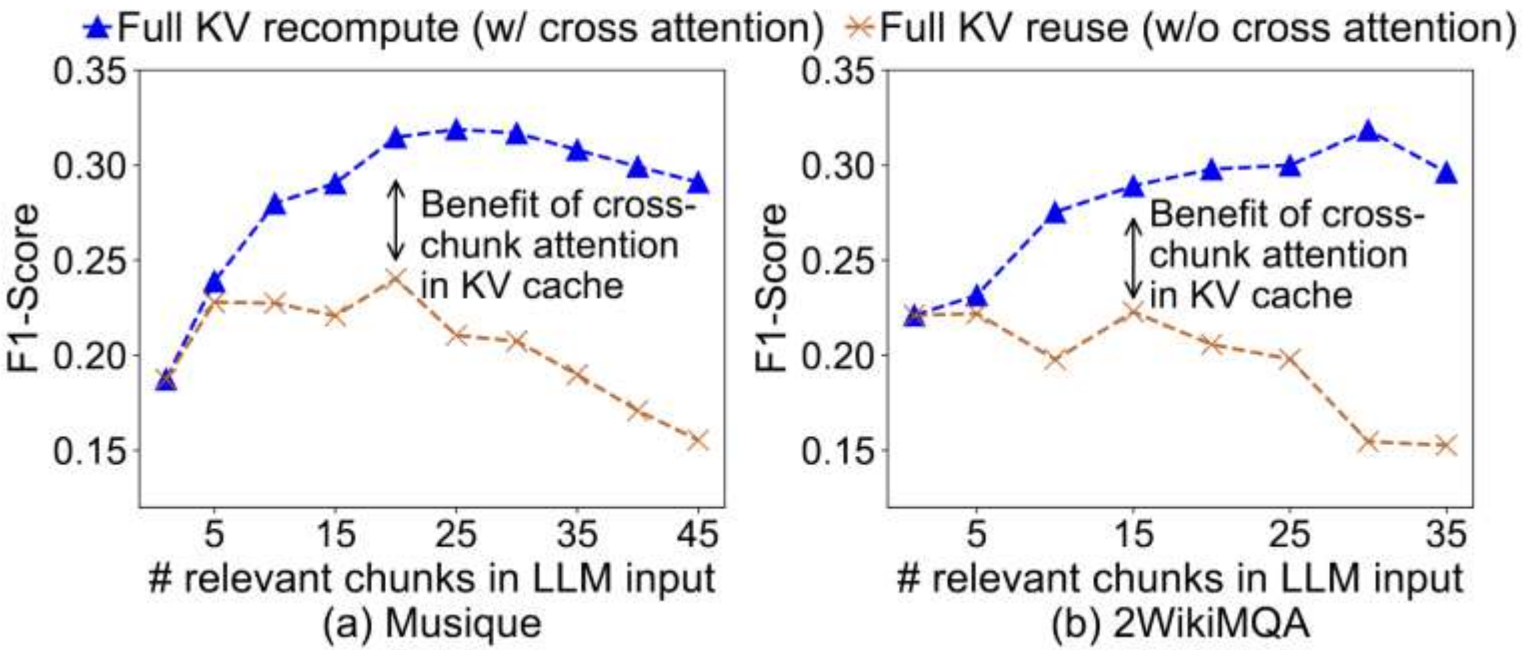
Precompute most chunk's KV with position information, ignore cross-chunk attention

Reuse KV in RAG : Existing Solutions



Low quality

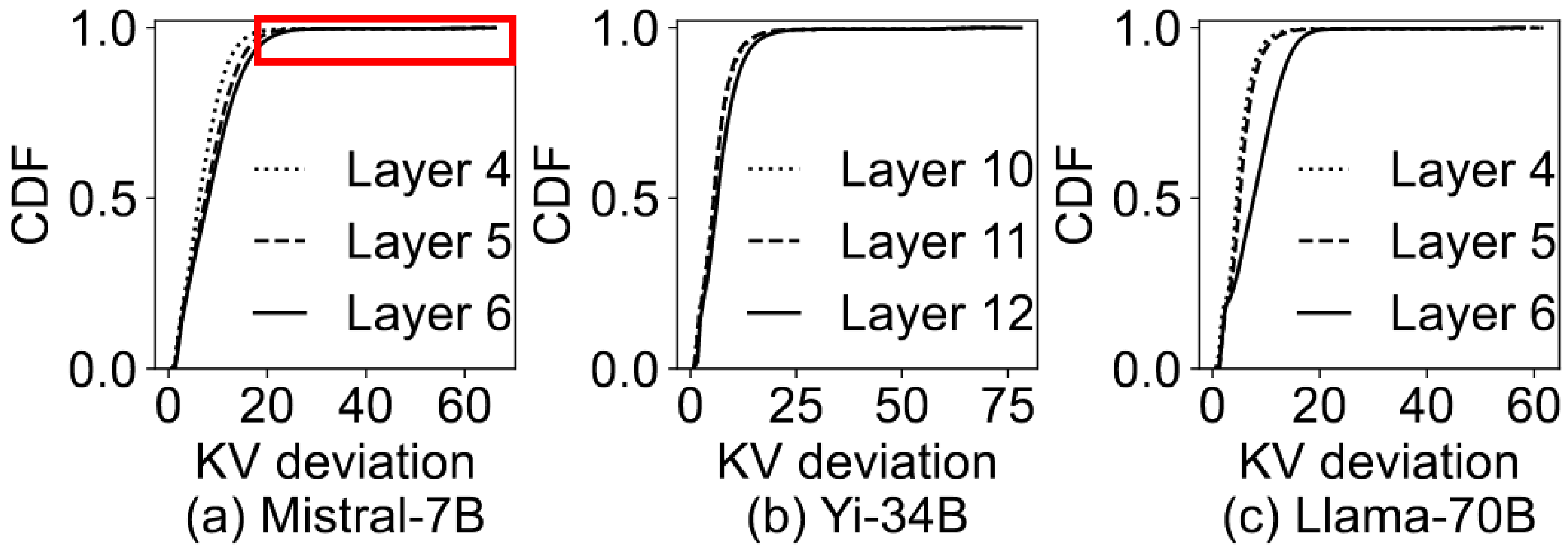
Full KV reuse (ignore cross-chunk attention): PromptCache



Cons: low quality when #chunk is large

Reuse KV in RAG : Insight 1

- Only a small fraction of tokens have large KV deviation



Definition of KV deviation:

$$\|\mathrm{KV}_{\mathrm{pre}} - \mathrm{KV}_{\mathrm{re}}\|_2$$

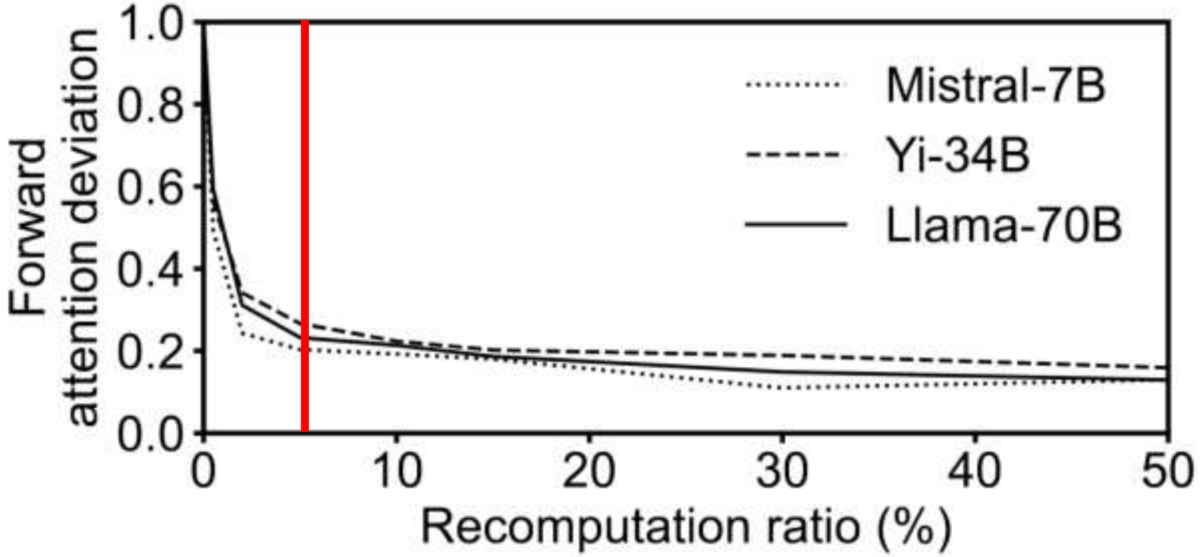
$\mathrm{KV}_{\mathrm{pre}}$: **pre-**computed KV cache

$\mathrm{KV}_{\mathrm{re}}$: **re-**computed KV cache

Distribution of KV deviation of different tokens on one layer.

Reuse KV in RAG : Insight 2

- Recompute only a small number of token's KV cache can get high generation quality



Definition of forward attention deviation:

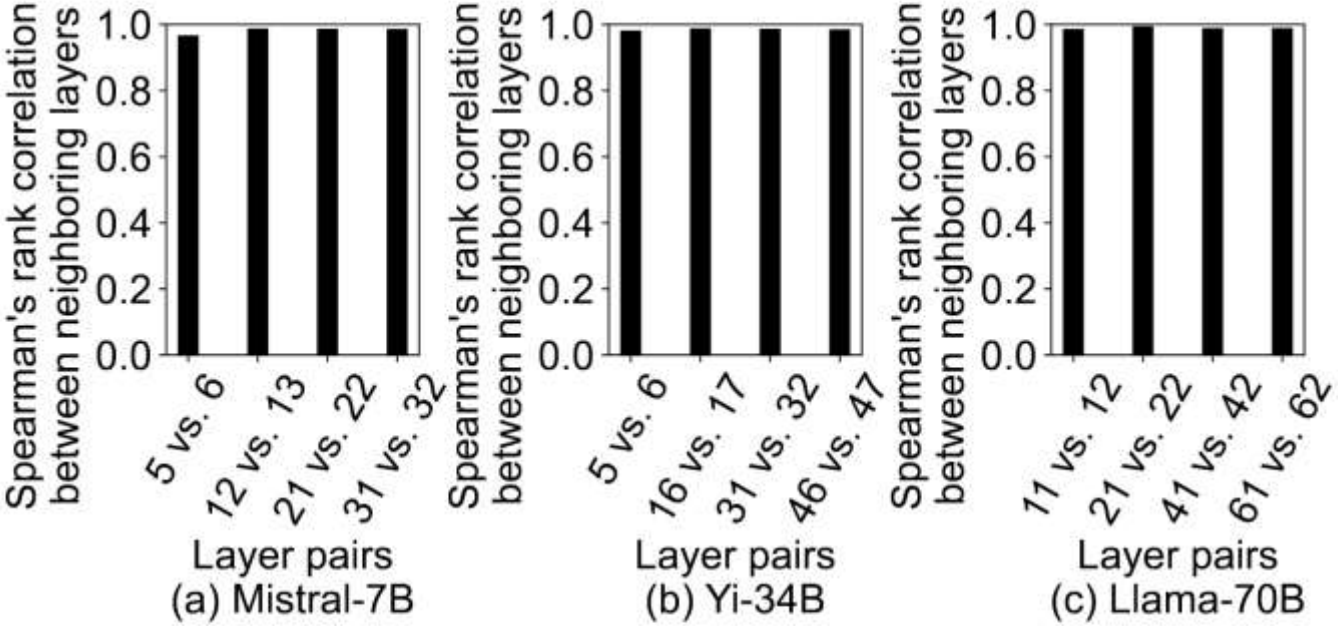
$$\|\mathrm{Attn}(\mathrm{KV}_{\mathrm{pre}}, Q) - \mathrm{Attn}(\mathrm{KV}_{\mathrm{re}}, Q)\|_2$$

Q is the user's query

Recompute the KV of the tokens with the highest KV deviation (HKVD)

Reuse KV in RAG : Insight 3

❑ **Tokens with the highest KV deviations on one layer are likely to have the highest KV deviations on the next layer.**



Definition of Spearman's rank correlation:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

Rank of KV deviation at layer l:
[1, 2, 3, 4, 5]

Rank of KV deviation at layer l+1:
[2, 1, 3, 5, 4]

n = 5

d = [1, 2, 3, 4, 5] - [2, 1, 3, 5, 4] = [-1, 1, 0, -1, 1]

$\rho = 0.8$

Rank correlation of the KV deviation per token between two consecutive layers.

Outline

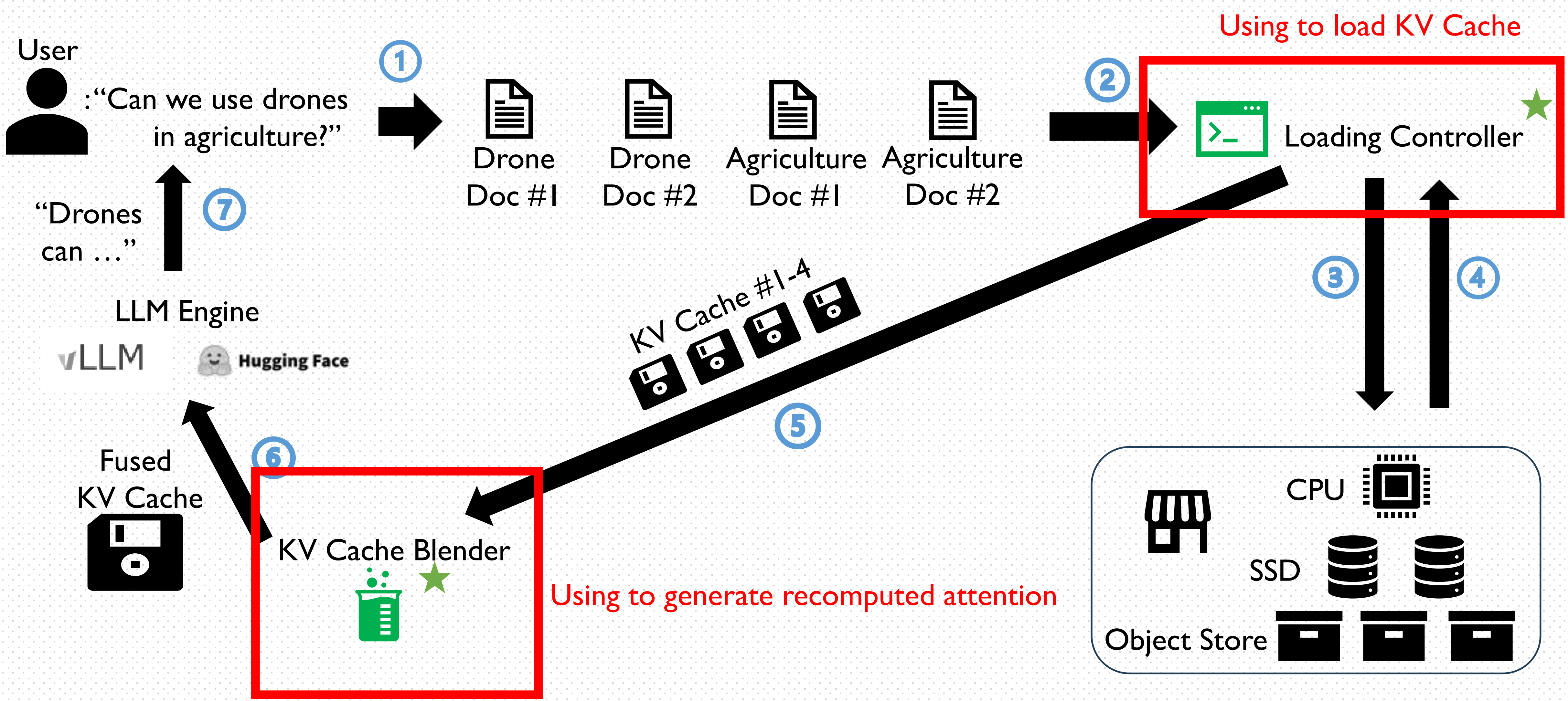
Design: Key Idea

Recompute a small number of tokens' KV cache

Pipeline the small recompute overhead with loading delay

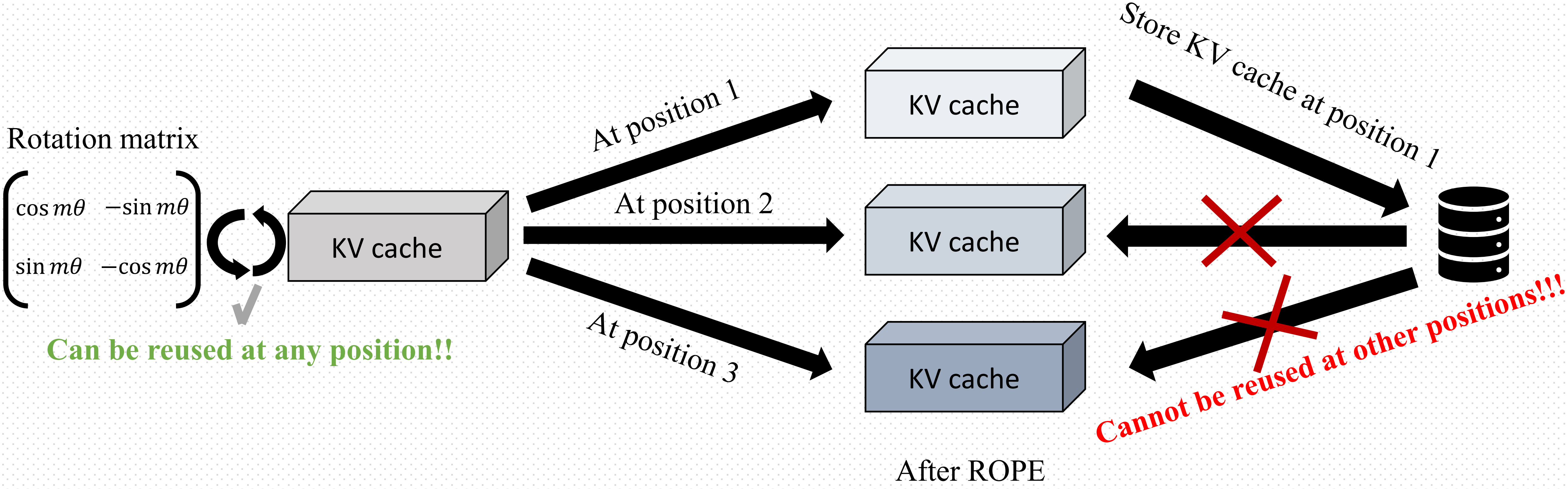
Maintain both **HIGH generation quality** and **LOW recompute overhead**

Design: System Overview



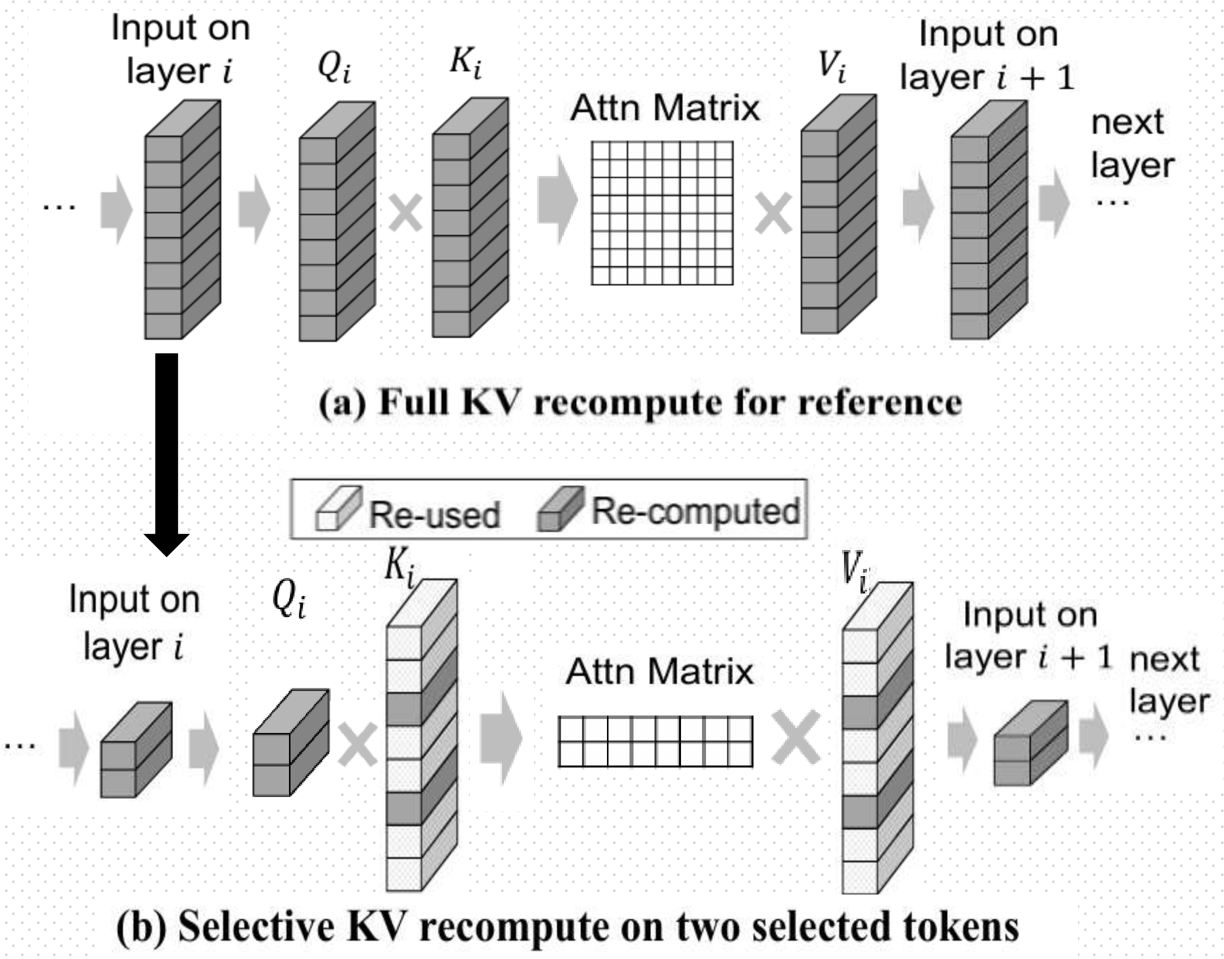
KV Cache Blender: Recover positional embedding

- ❑Positional information can be done easily by multiplying the Key vector by a rotation matrix



KV Cache Blender: Selective re-computation

❑Cross-chunk Attention effect



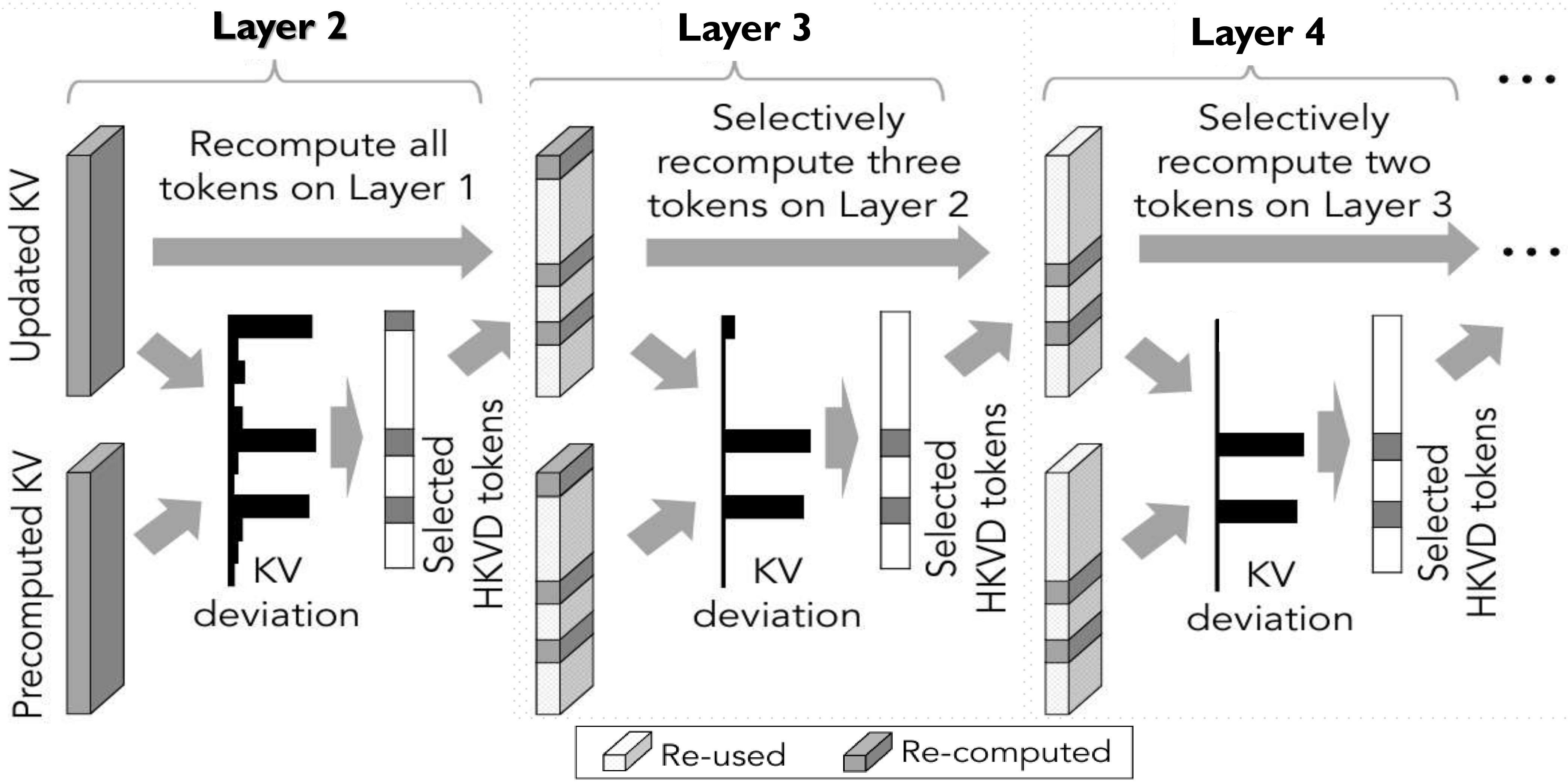
total tokens

selected tokens

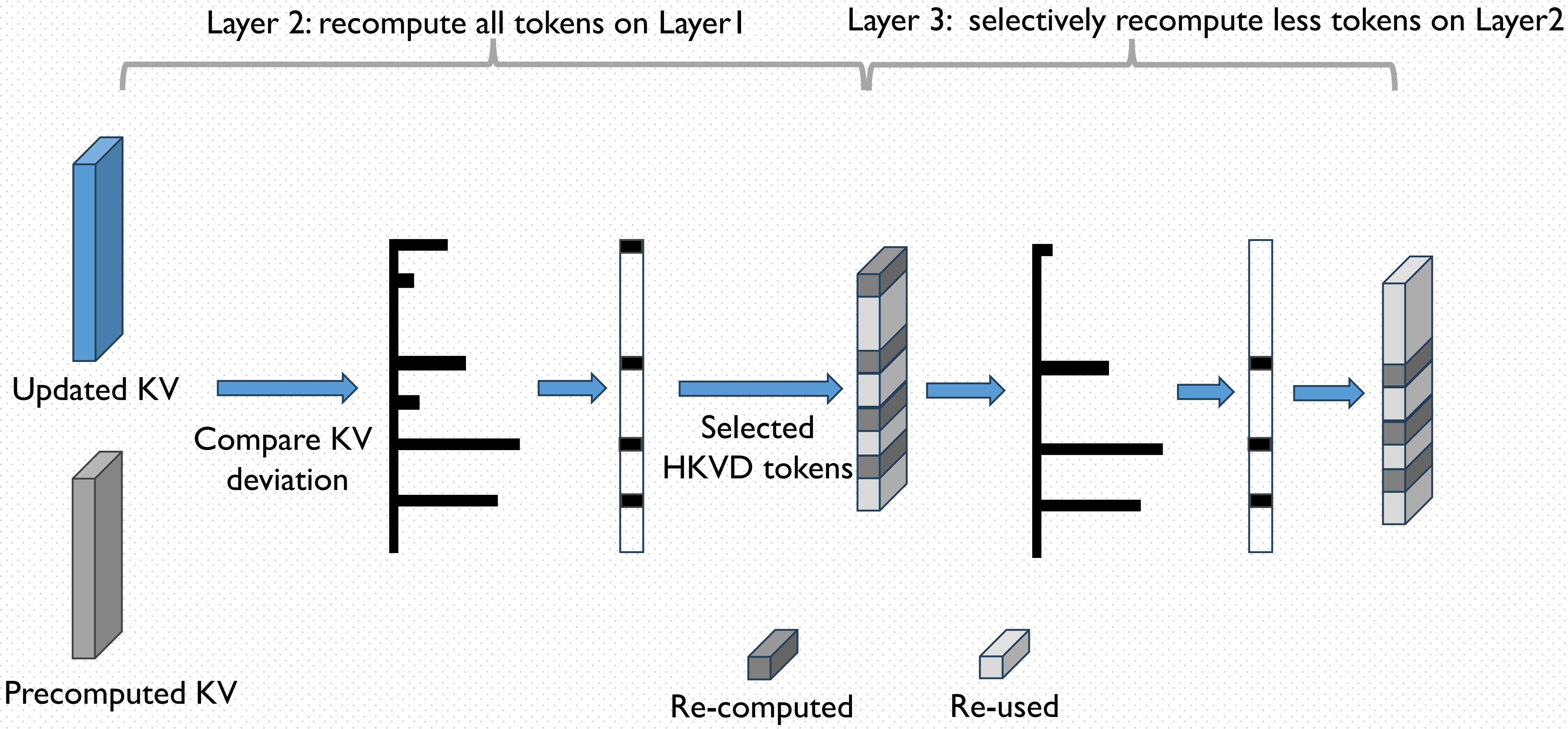
How to select tokens?

KV Cache Blender: Selective recomputation

Layer i selected token is slightly larger than layer i+1

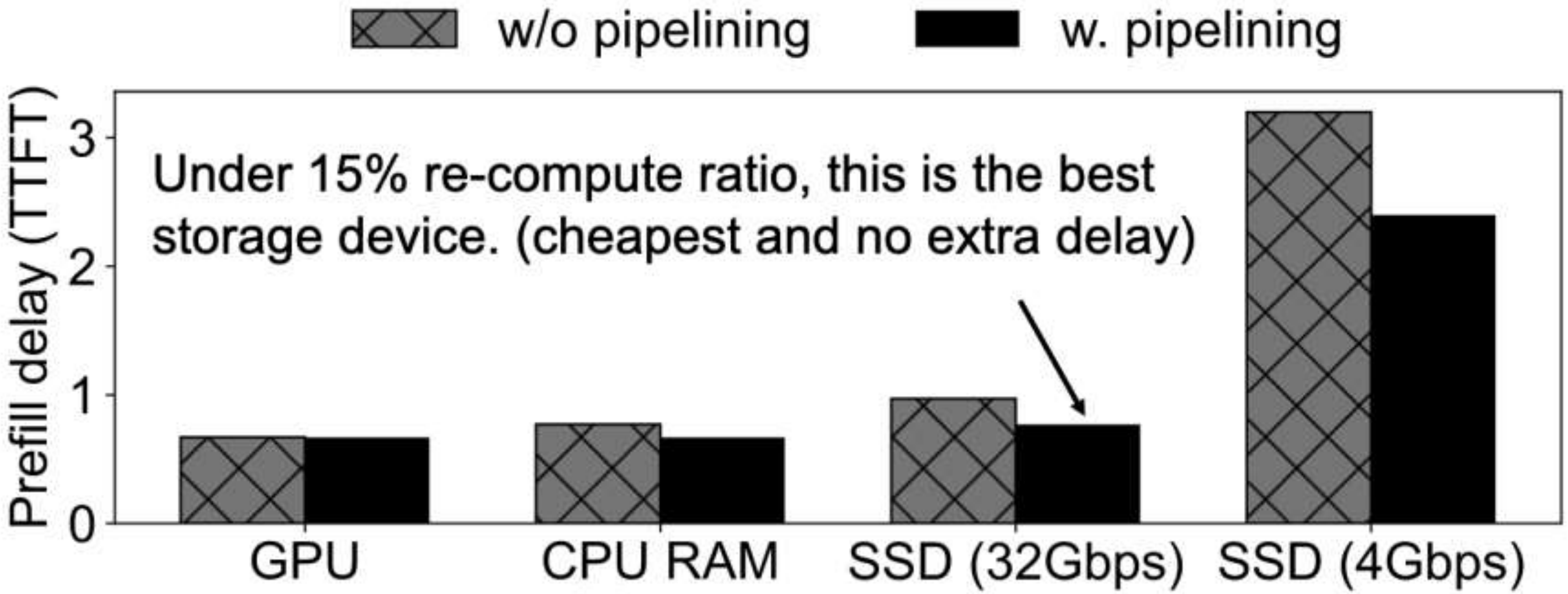


KV Cache Blender: Selective re-computation



Loading Controller

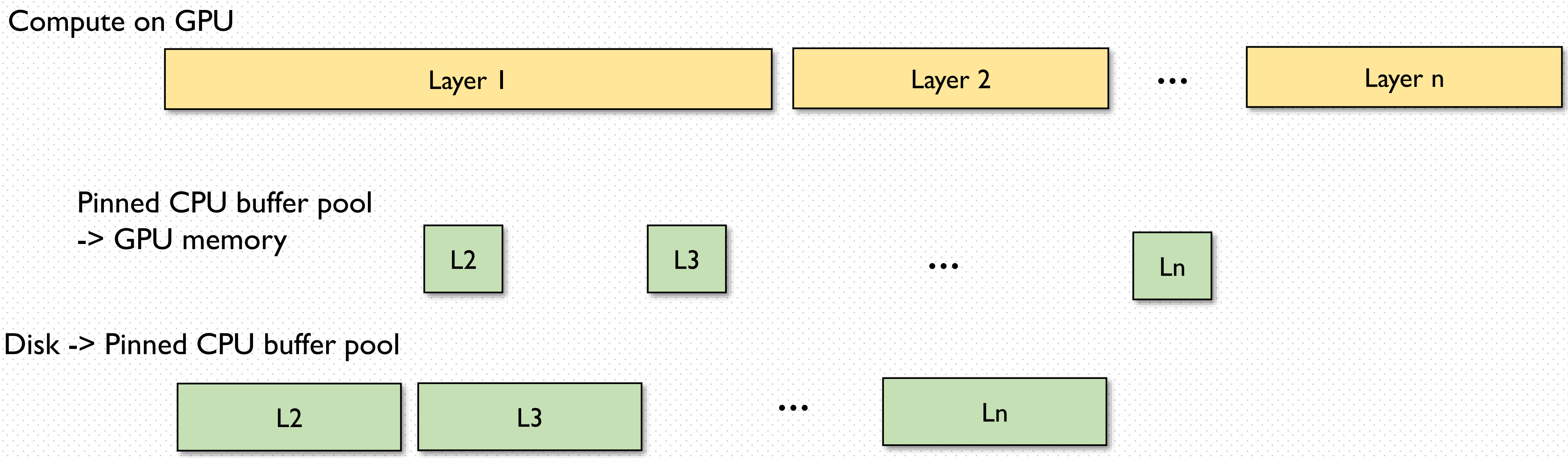
- ❑**Pipline KV Cache loading and Recompute**
- ❑Time of selective KV recomputed should be the same as loading KV into GPU



(b) Storage Devices

Loading Controller

❑Pipline KV Cache loading and Recompute



Outline

1. Background & Motivation
2. Design
3. Evaluation
4. Discussion

Setup

❑Hardware

- ❖128 GB RAM
- ❖2 Nvidia A40 GPUs
- ❖1TB NVME SSD (4.8GB/s thpt)

❑Models

- ❖Mistral-7B, run on 1 GPU
- ❖Yi-34B, run on 1 GPU, applied 8-bit model quantization
- ❖Llama-70B, run on 2 GPUs, applied 8-bit model quantization

Setup

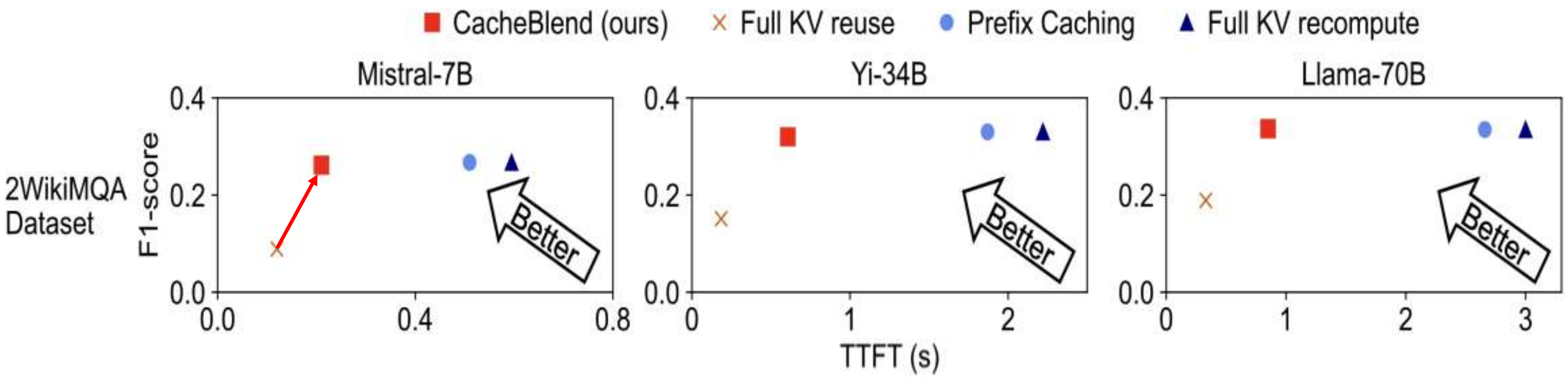
- **Basic Dataset:**
  - 2WikiMQA: 200 test cases
  - Musique: 150 test cases
  - SAMSum: 200 test cases
  - MultiNews: 60 sampled test cases
- **Simulate RAG chunk reuse**
  - extended Musique and 2WikiMQA: generating 6000 queries (1500 original + 3 GPT-4-augmented per query) with contexts split into 512-token chunks
  - retrieving top-6 chunks per query to simulate chunk reuse in RAG scenarios

Setup

- **Metrics:**
  - F1-Score: for 2WikiMQA and Musique datasets
  - Rouge-L score: for MultiNews and SAMSum datasets
- **Baselines:**
  - Full KV Recompute
  - Prefix caching: adopt technique from SGLang
  - Full KV reuse: adopt approach proposed by PromptCache
  - MapReduce: generate result per chunk then reduce
  - MapRerank: generate result per chunk then return the result with best score

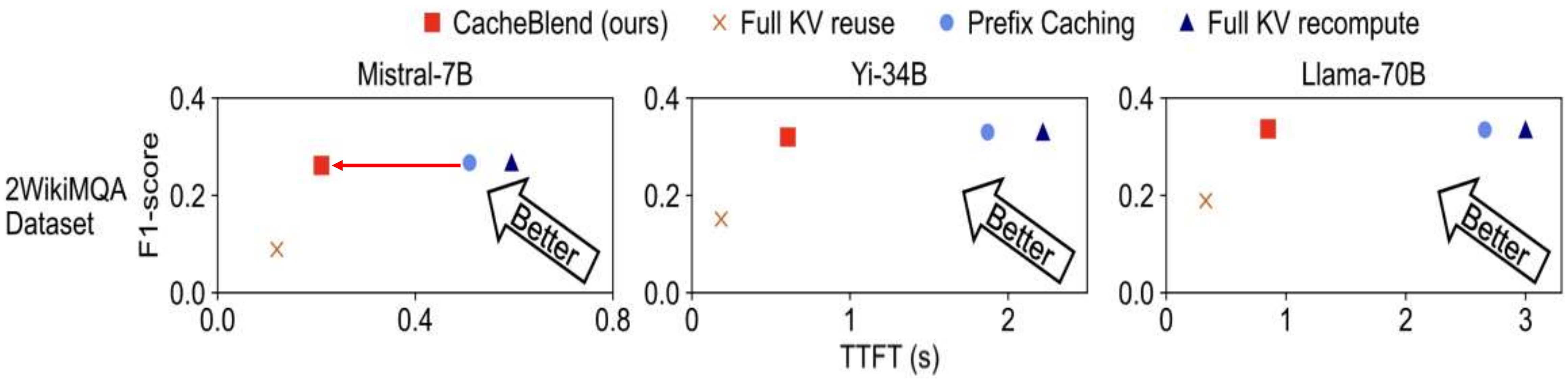
Overall Improvement

- Compared with full KV reuse, CacheBlend is slower but its quality stably outperforms



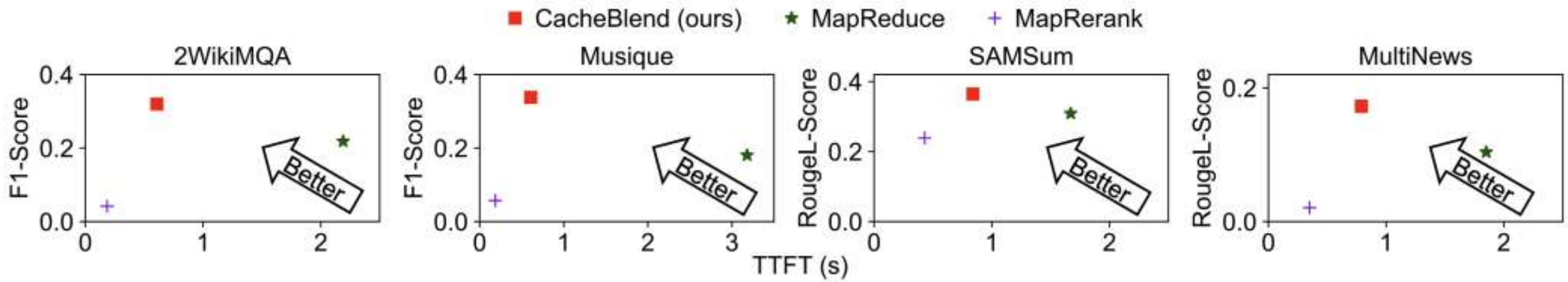
Overall Improvement

- Compared to Prefix Caching and Full KV recompute, CacheBlend's significantly reduces the TTFT by 2.2-3.3× with little reduction in F1-score



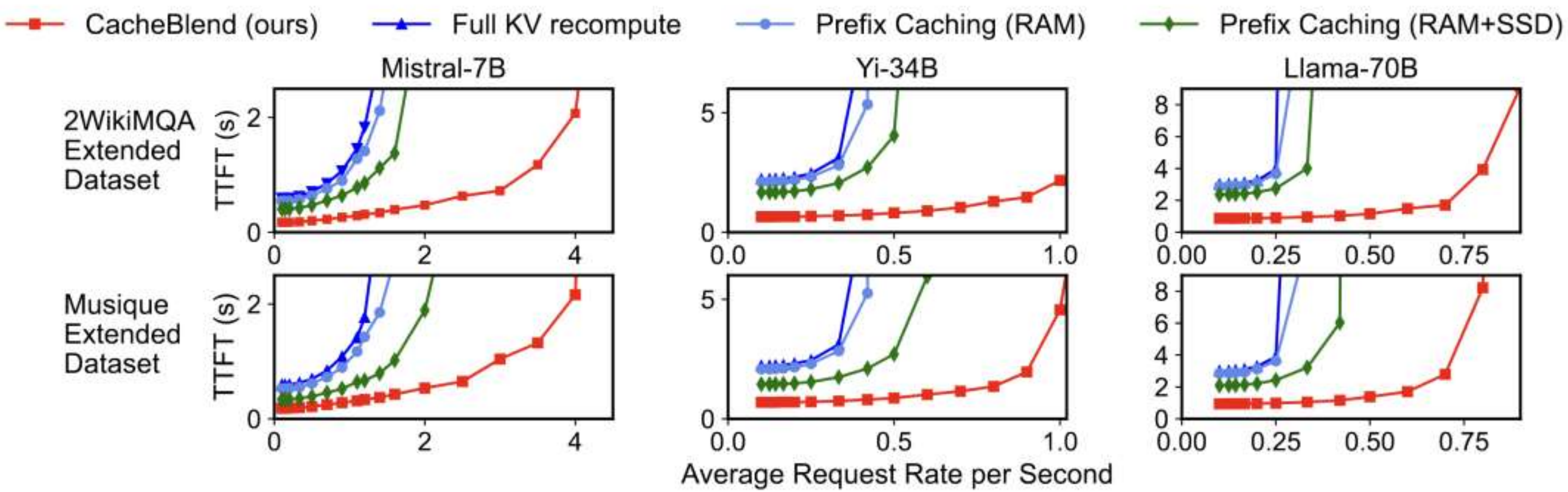
Overall Performance

- Compared to MapReduce, CacheBlend has a 2-5× lower TTFT and higher F1 score.



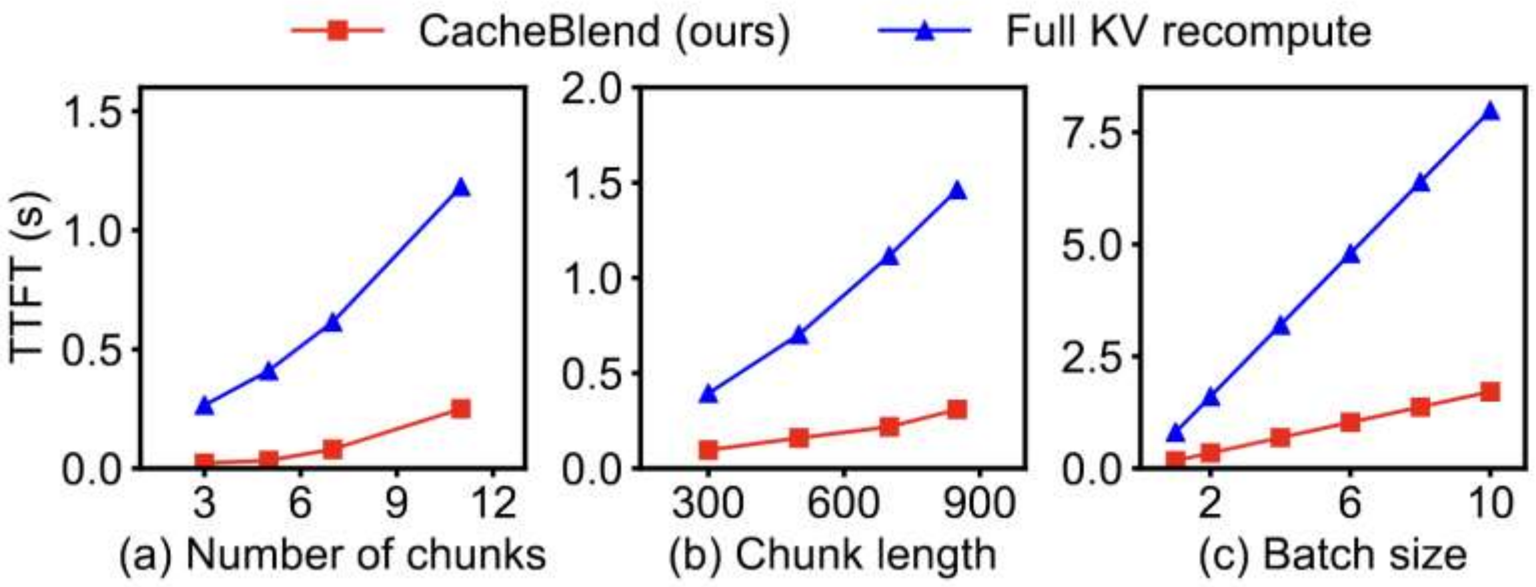
Overall Performance

- ❑CacheBlend achieves higher throughput and lower delay
  - ❖on Musique extended and 2WikiMQA datasets under different request rates.



Sensitivity Analysis

❑Varying chunk numbers and lengths

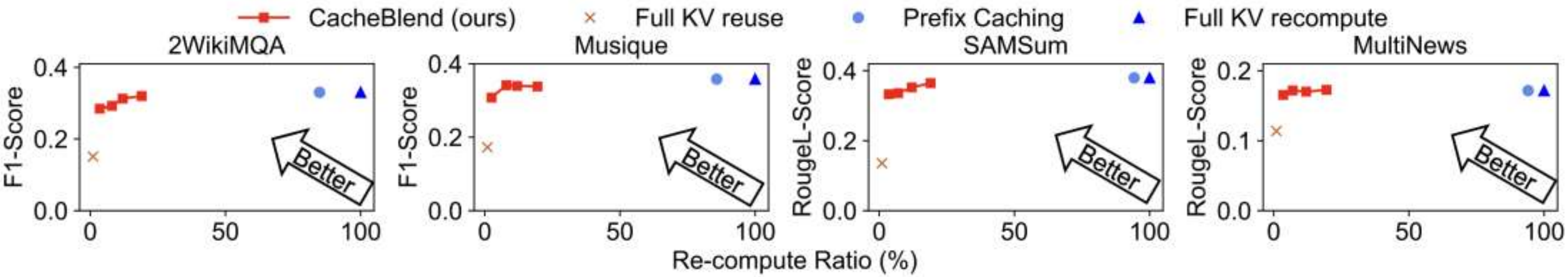


conducted on 2WikiMQA

with Mistral-7B model

Sensitivity Analysis

- ❑Varying recompute ratio, with 5%-18% recompute ratio



Across all dataset on Yi-34B model

Outline

1. Background & Motivation
2. Design
3. Evaluation
4. Discussion

Discussion

- CacheBlend uses KV cache's sparsity in a novel way
- But using SSD to store/load KV cache and without GPU direct is a little weird

Thanks!