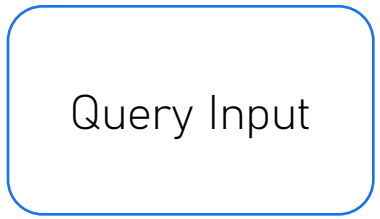
RAG & CacheBlend

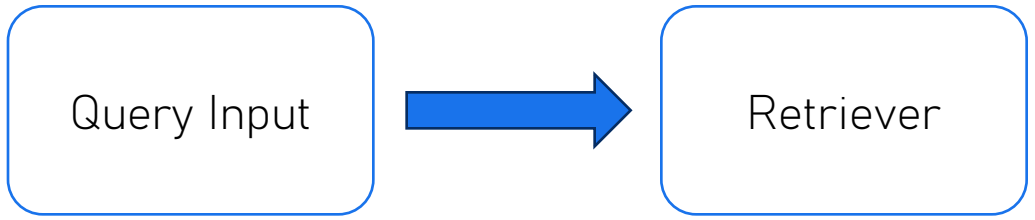
INTRODUCTION TO LLM
INFERENCE SERVING SYSTEMS

CHUHONG YUAN

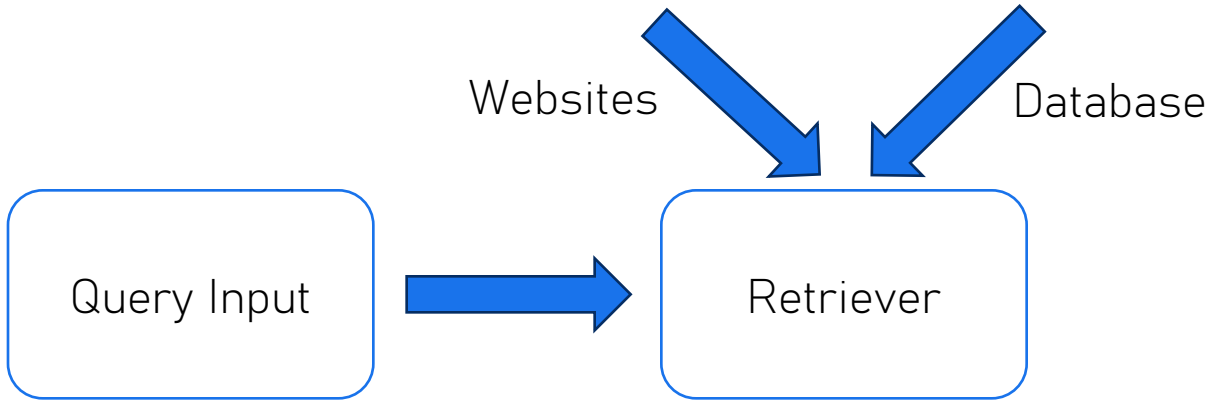
RAG (Retrieval-Augmented Generation)



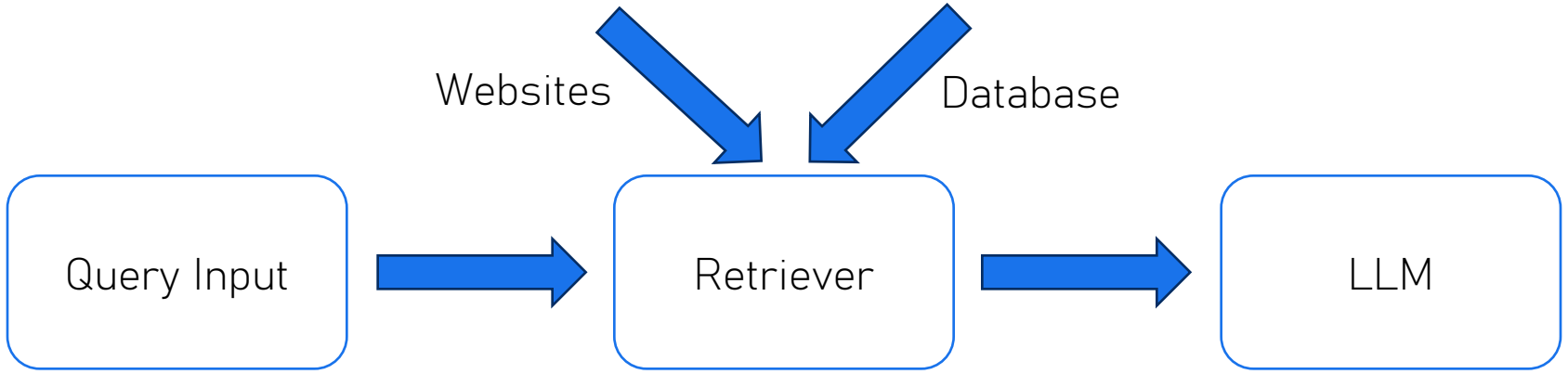
RAG (Retrieval-Augmented Generation)



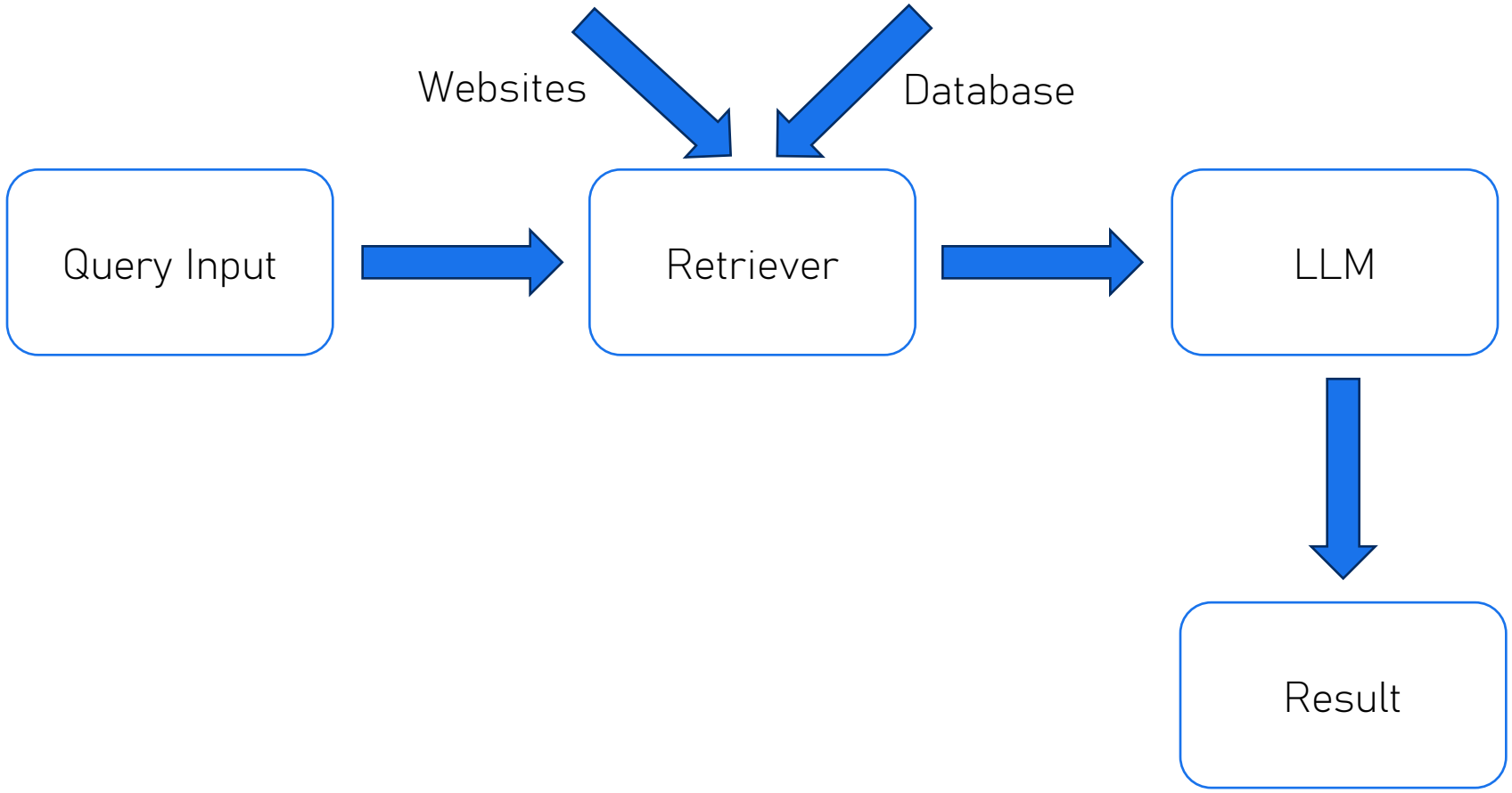
RAG (Retrieval-Augmented Generation)



RAG (Retrieval-Augmented Generation)



RAG (Retrieval-Augmented Generation)



Prefill Strategies For Shared Contexts

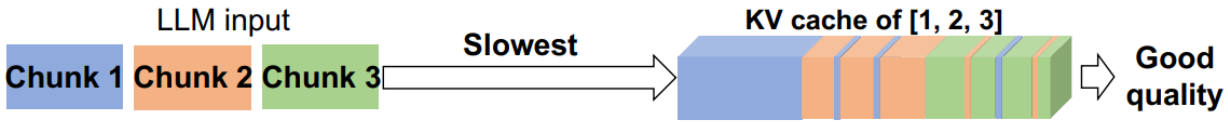
- Some retrieved contexts can be used for multiple chats
- Can we save time in prefill for these shared contexts?

Prefill Strategies For Shared Contexts

- Some retrieved contexts can be used for multiple chats
- Can we save time in prefill for these shared contexts?
  - Cross-attention exists across the retrieved text chunks

Prefill Strategies For Shared Contexts

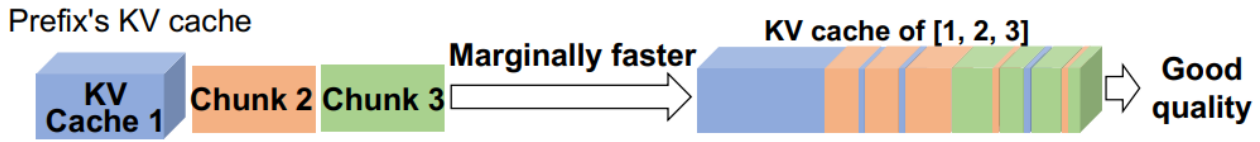
- Some retrieved contexts can be used for multiple chats
- Can we save time in prefill for these shared contexts?
  - Cross-attention exists across the retrieved text chunks
- KV full-recompute
  - Accurate but slow



(a) Default: Full KV re-compute. Prefill on entire input

Prefill Strategies For Shared Contexts

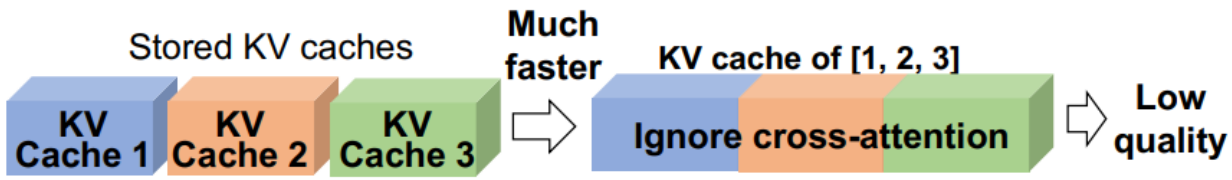
- Some retrieved contexts can be used for multiple chats
- Can we save time in prefill for these shared contexts?
  - Cross-attention exists across the retrieved text chunks
- KV full-recompute
- Prefix caching
  - Doesn't help much



(b) Prior work: Prefix caching.
Only reusing *prefix's* KV cache

Prefill Strategies For Shared Contexts

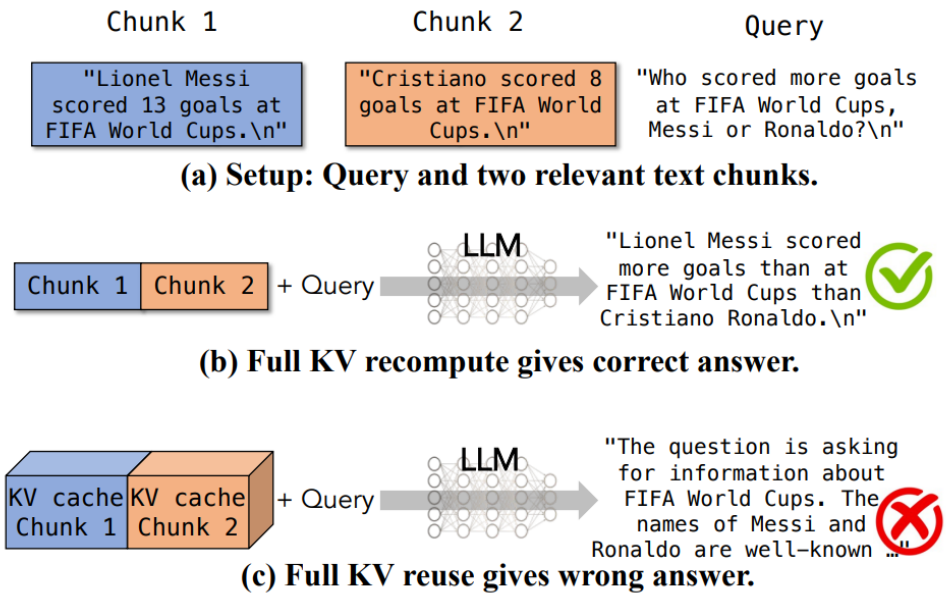
- Some retrieved contexts can be used for multiple chats
- Can we save time in prefill for these shared contexts?
  - Cross-attention exists across the retrieved text chunks
- KV full-recompute
- Prefix caching
- KV full-reuse
  - Fast but low quality



(c) Prior work: Full KV reuse.
Reusing all KV caches, ignoring cross-attention

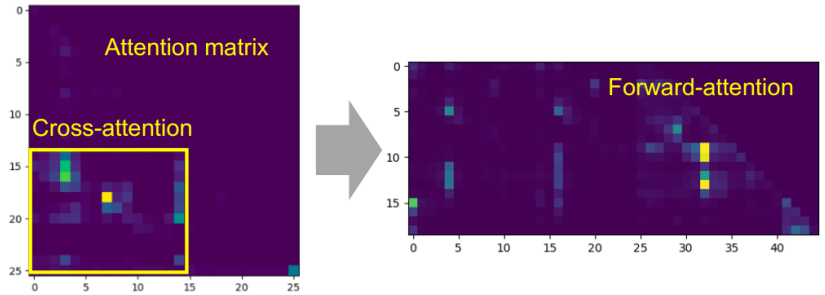
Prefill Strategies For Shared Contexts

- Some retrieved contexts can be used for multiple chats
- Can we save time in prefill for these shared contexts?
  - Cross-attention exists across the retrieved text chunks
- KV full-recompute
- Prefix caching
- KV full-reuse
  - Fast but low quality

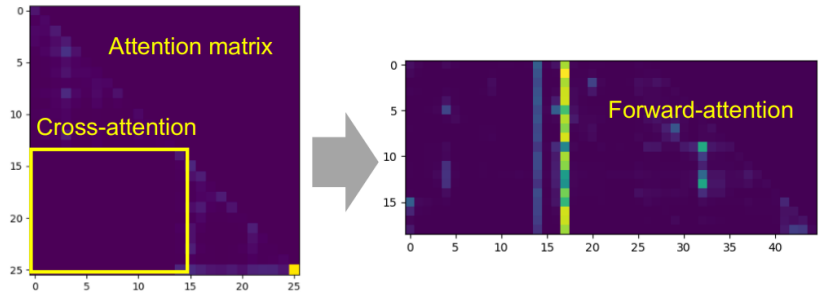


Prefill Strategies For Shared Contexts

- Some retrieved contexts can be used for multiple chats
- Can we save time in prefill for these shared contexts?
  - Cross-attention exists across the retrieved text chunks
- KV full-recompute
- Prefix caching
- KV full-reuse
  - Fast but low quality



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



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

Prefill Strategies For Shared Contexts

- Some retrieved contexts can be used for multiple chats
- Can we save time in prefill for these shared contexts?
  - Cross-attention exists across the retrieved text chunks
- KV full-recompute
- Prefix caching
- KV full-reuse
- Challenge: tradeoff between accuracy and efficiency

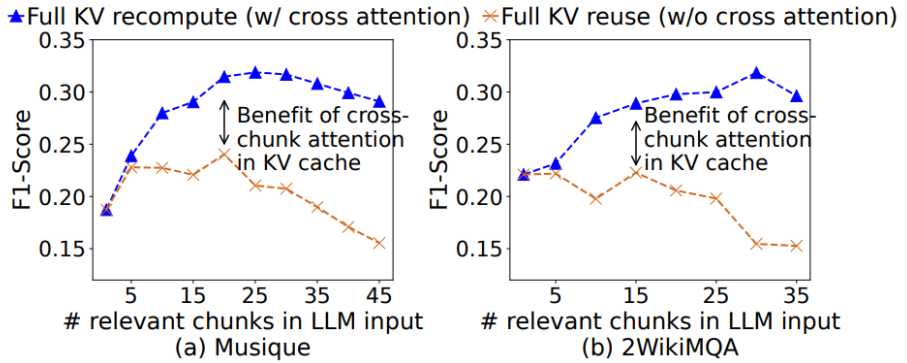


Figure 2. *Generation quality improves as more text chunks are retrieved.*

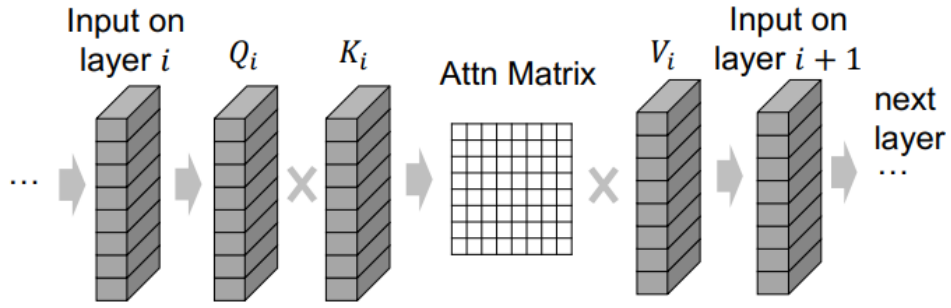
Core Idea – Selective KV Cache Recomputation

- Balance between accuracy and efficiency
  - Less KV Cache computation but maintain an acceptable accuracy
- Goal
  - With multiple reused text chunks, quickly update KV Cache, making the forward-attention matrices have the smallest difference from the one under full KV-recomputation
- Terminology
  - KV Deviation – difference between the KV under selective recomputation and full recomputation
  - Attention Deviation – difference between the forward attention matrices under two recomputations

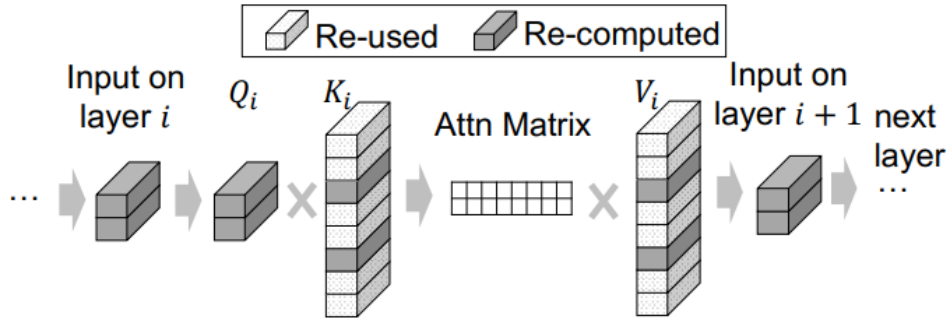
Challenges For Selective Recomputation

- How many tokens should be selected for recomputation?
- Which tokens should be selected?
  - How to achieve the minimum accuracy loss?
  - How to get the ground truth of the forward attention matrix without large costs?

Which Tokens To Recompute?



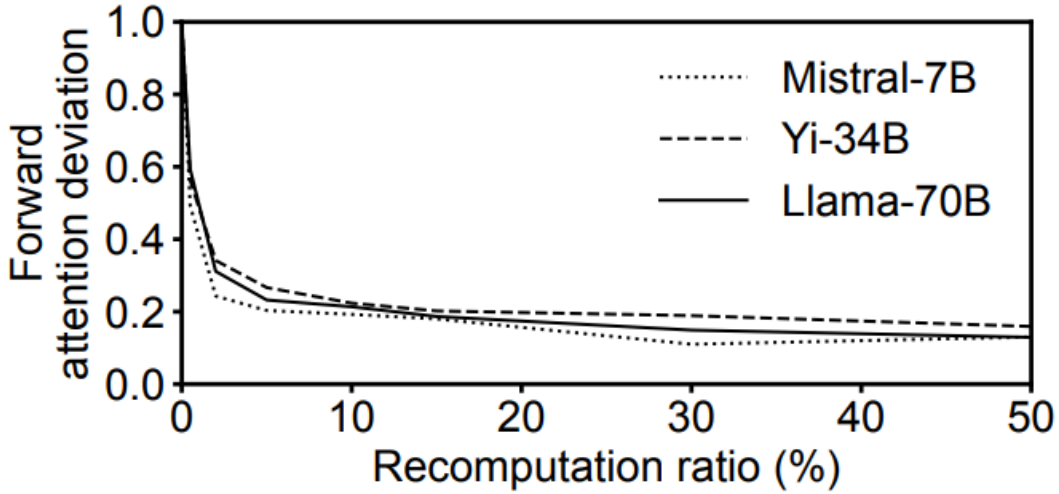
(a) Full KV recompute for reference



(b) Selective KV recompute on two selected tokens

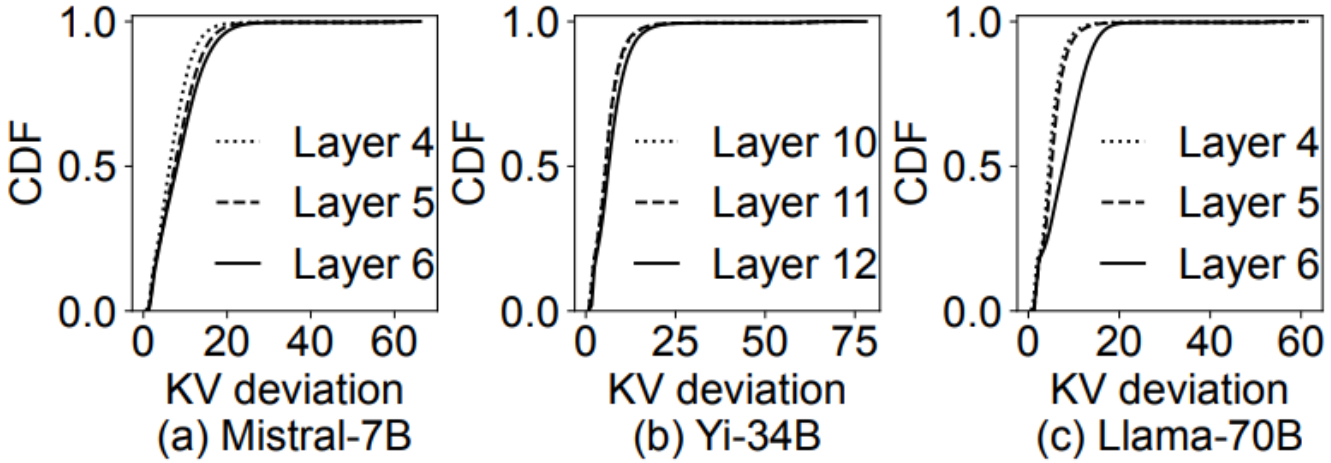
Which Tokens To Recompute?

- The tokens with higher KV deviation may also cause higher attention deviation
  - Recomputing the KV Cache of such tokens is intuitively more efficient
- How many tokens should be recomputed?



Which Tokens To Recompute?

- The tokens with higher KV deviation may also cause higher attention deviation
  - Recomputing the KV Cache of such tokens is intuitively more efficient
- How many tokens should be recomputed?

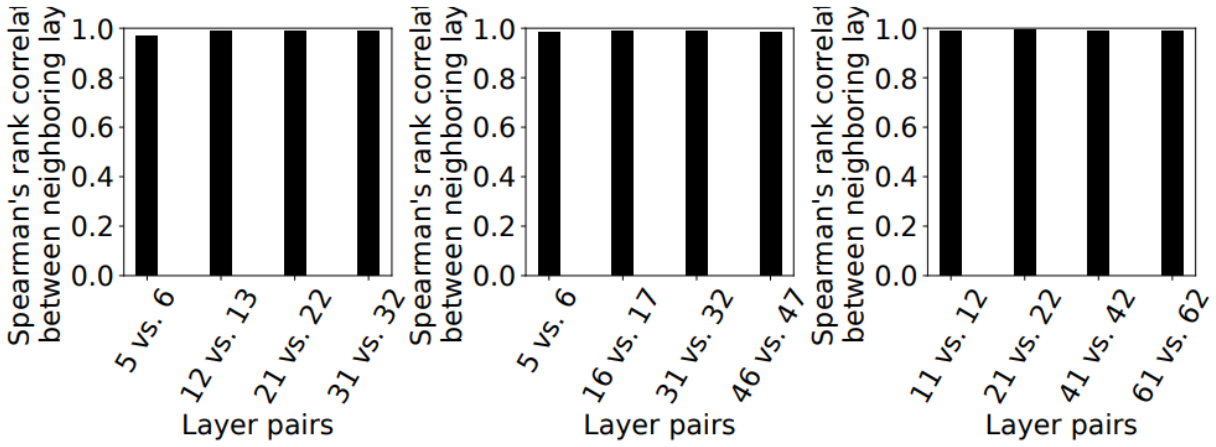


Which Tokens To Recompute?

- The tokens with higher KV deviation may also cause higher attention deviation
  - Recomputing the KV Cache of such tokens is intuitively more efficient
- How many tokens should be recomputed?
  - Attention sparsity – only part of the tokens are highlighted

How To Identify HKVD Tokens?

- The ground truth can only be known under full recomputation
  - Too time costly
- Tokens with high KVD on one layer are likely to be HKVD tokens in the next layer

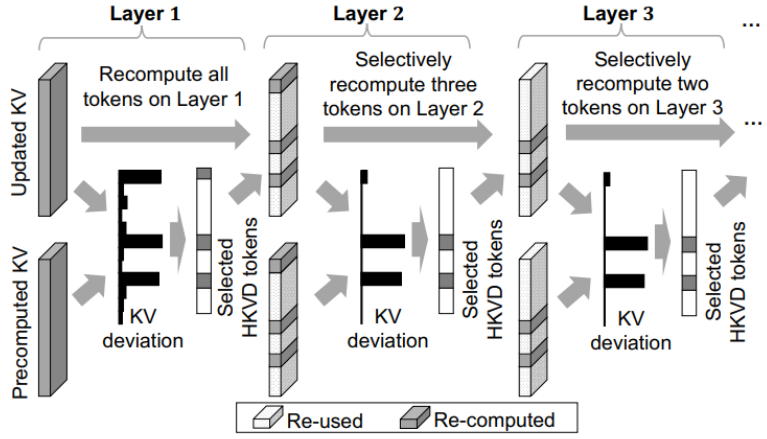


How To Identify HKVD Tokens?

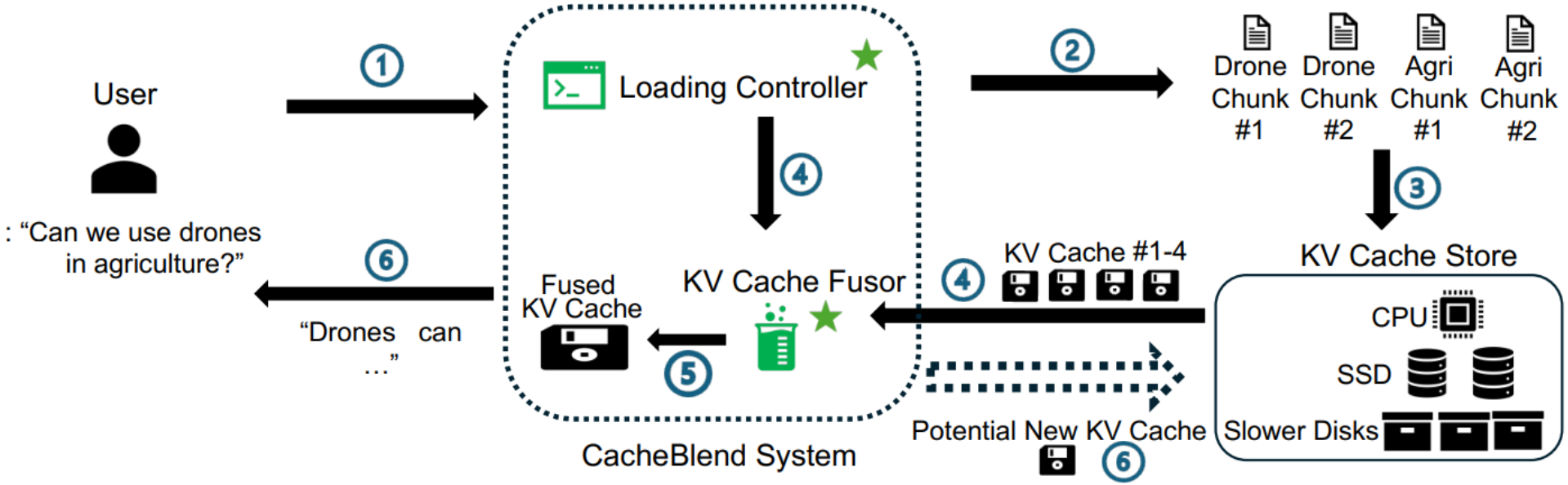
- The ground truth can only be known under full recomputation
  - Too time costly
- Tokens with high KVD on one layer are likely to be HKVD tokens in the next layer
- Gradual filtering
  - First layer all recomputation
  - Then select the ones with the highest r% KV deviation in the next layer
  - In the next layer, filter out the tokens with low KV deviation

How To Identify HKVD Tokens?

- The ground truth can only be known under full recomputation
  - Too time costly
- Tokens with high KVD on one layer are likely to be HKVD tokens in the next layer
- Gradual filtering
  - First layer all recomputation
  - Then select the ones with the highest r% KV deviation in the next layer
  - In the next layer, filter out the tokens with low KV deviation

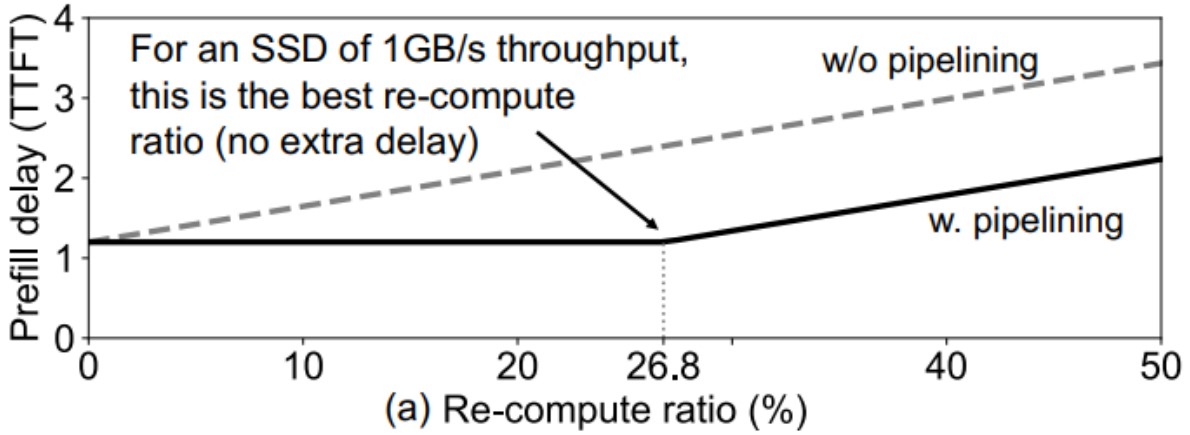


CacheBlend Design



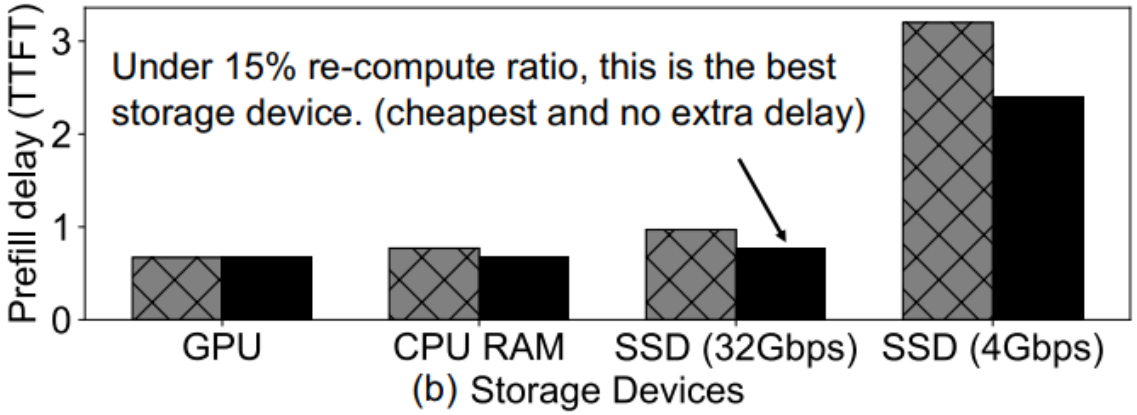
Loading Controller

- Pipeline KV Cache loading and recomputation
  - Avoid extra latency
- Device fixed, decide the recomputation ratio so that $T_{recomputation} < T_{load}$



Loading Controller

- Pipeline KV Cache loading and recomputation
  - Avoid extra latency
- Device fixed, decide the recomputation ratio so that $T_{recomputation} < T_{load}$
- Recomputation ratio fixed, decide the device so that $T_{recomputation} > T_{load}$



KV Cache Store & Fusor

- KV Cache store
  - Inputs are split into text chunks
  - Indexing with a hashing value
  - LRU policy to eviction
- Fusor
  - Merge the precomputed KV cache and the new one

Evaluation Settings

- Models: Mistral-7B, Yi-34B, Llama-70B
- GPUs: Runpod 128GB RAM, A40
- Storage: 1TB NVME SSD
- Baselines
  - Full KV recompute, prefix caching, full KV reuse
  - MapReduce: summarize the chunks then concatenate together
  - MapRerank: independently generate answers with each chunk then select the one with highest score

Workloads & Metrics

- Datasets
  - 2WikiMQA, Musique, SAMSum, MultiNews
  - Contexts split into chunks
  - Generate a database with the queries
- Metrics
  - Efficiency: TTFT, throughput (request rate)
  - Accuracy: F1-score (num of overlapped words), Rouge-L score (longest common sequence)

Evaluation – Latency vs. Accuracy

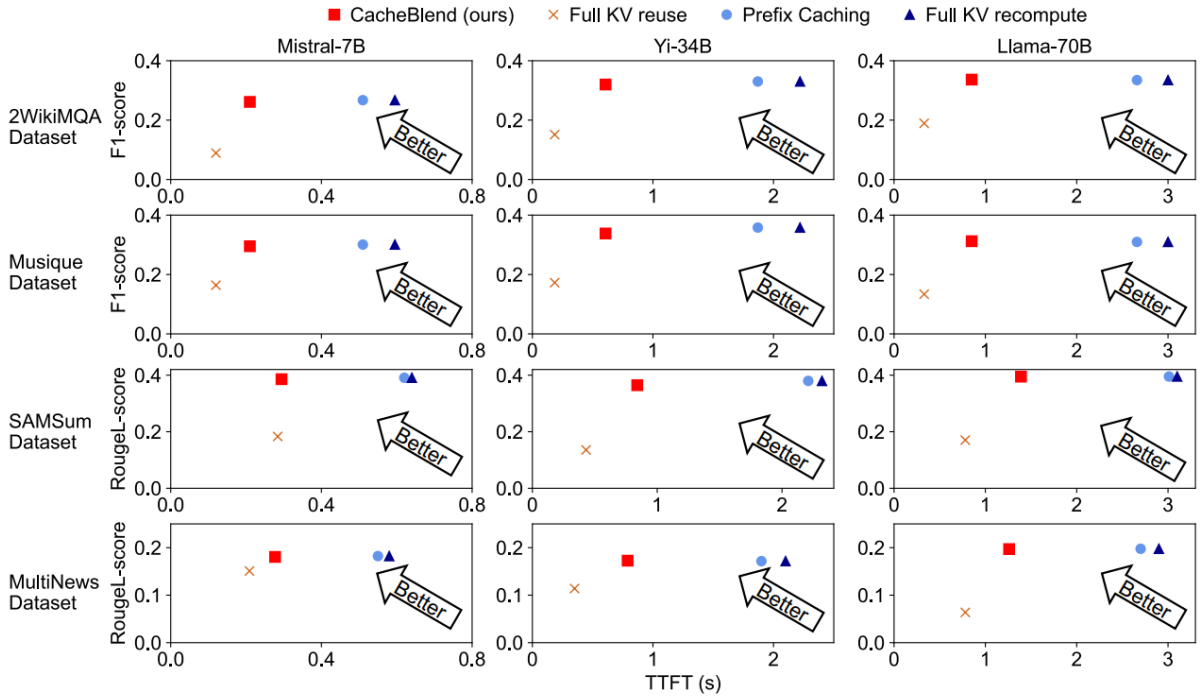


Figure 12. *CacheBlend reduces TTFT by 2.2-3.3× compared to full KV recompute with negligible quality drop across four datasets and three models.*

Evaluation – Latency vs. Accuracy

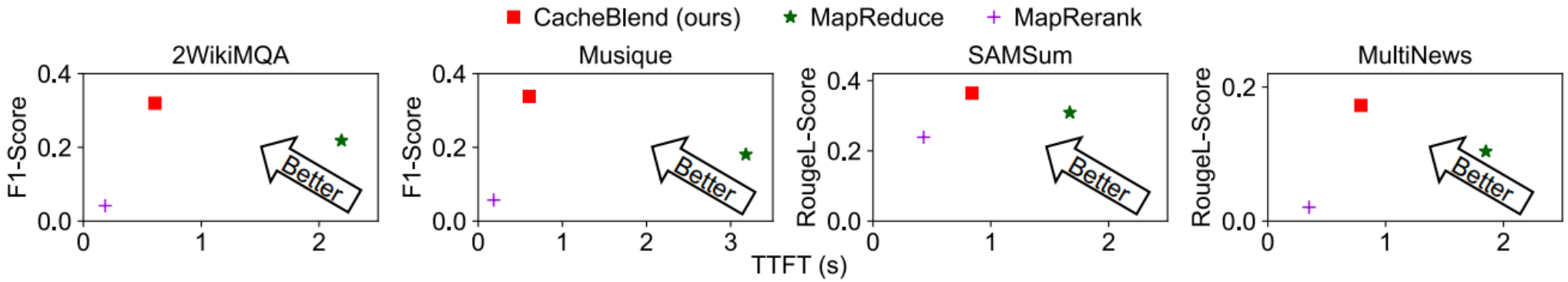


Figure 13. *Generation quality of CacheBlend with Yi-34B vs MapReduce and MapRerank.*

Evaluation – Throughput

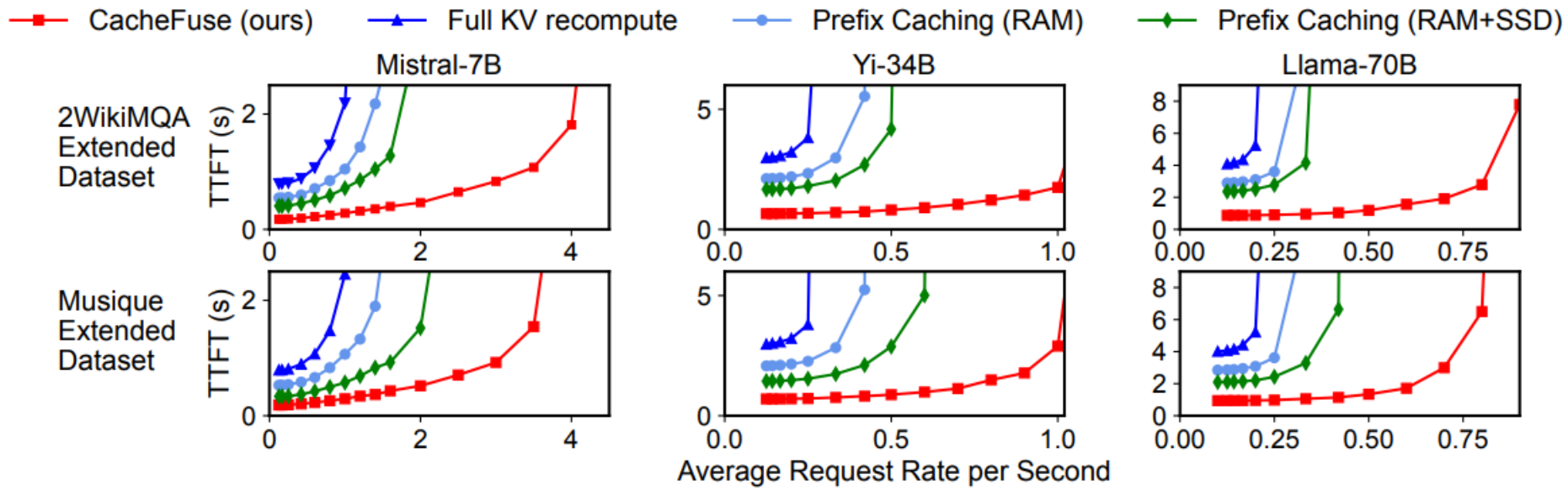


Figure 14. *CacheBlend achieves lower TTFT with higher throughput in RAG scenarios compared with baselines of similar quality.*

Evaluation – Sensitivity Analysis

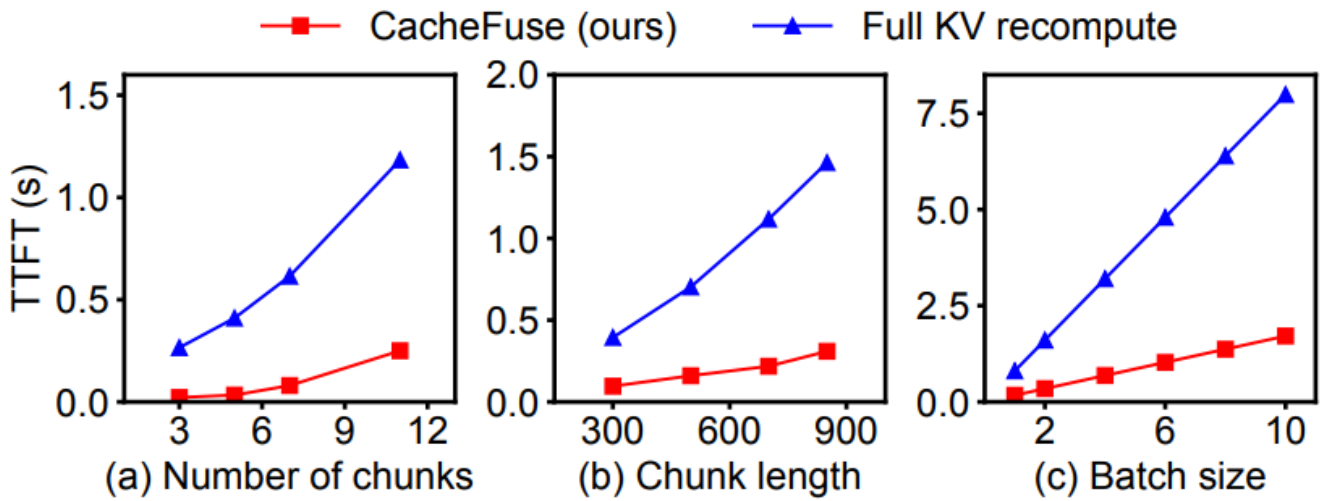


Figure 15. *CacheBlend outperforms baseline with varying chunk numbers, chunk lengths, and batch sizes.*

Evaluation – Sensitivity Analysis

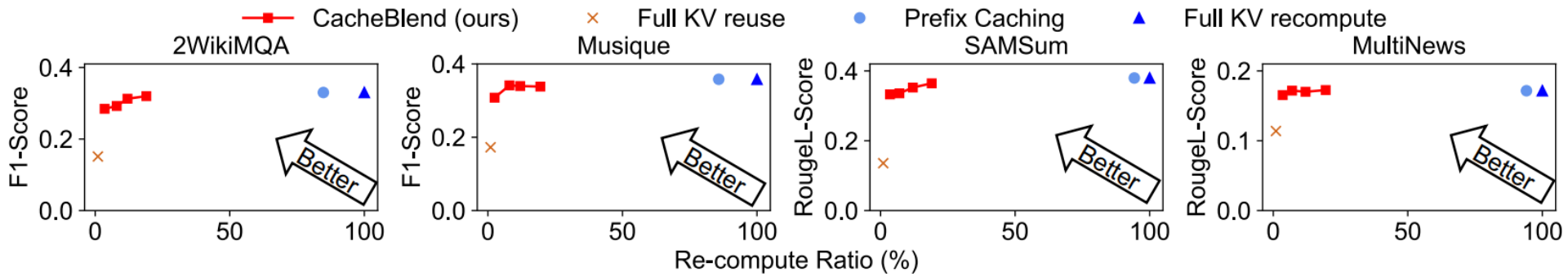


Figure 16. *CacheBlend has minimal loss in quality compared with full KV recompute, with 5%–18% selective recompute ratio, with Yi-34B.*

Evaluation – Sensitivity Analysis

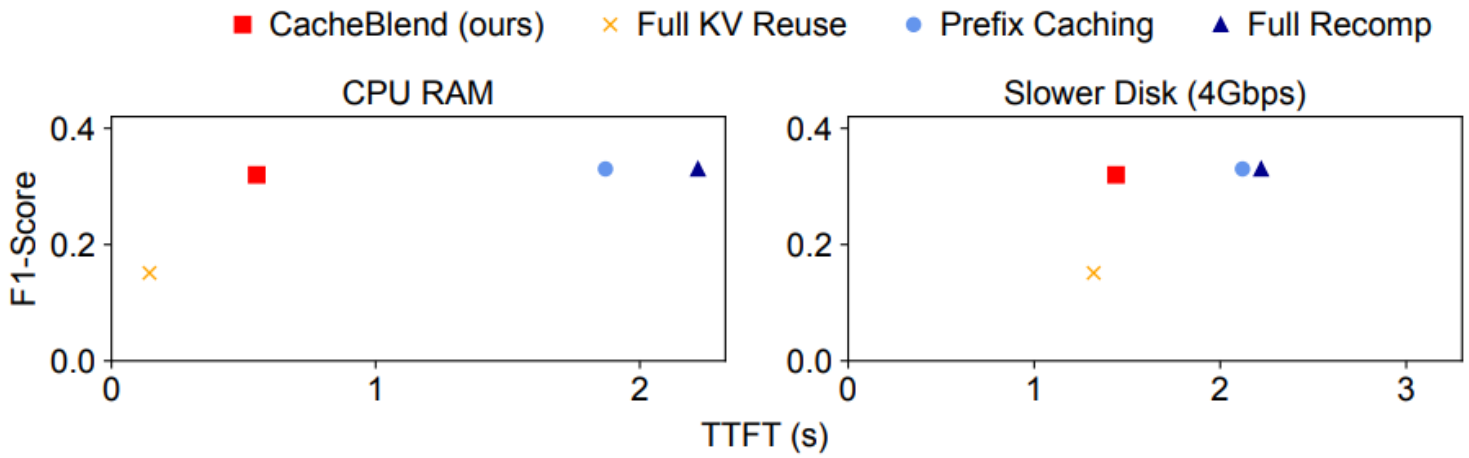


Figure 17. *CacheBlend's outperforms baselines when using RAM and slower disks*

Homework Review

- Challenges
  - Accuracy vs. Efficiency
  - Which tokens should be selected?
  - How to identify HKVD tokens?
- Design
  - Selective KV Cache recomputation
  - Select ~15% HKVD tokens
  - Gradual filtering
- Evaluation
  - Accuracy, efficiency for proving the successful tradeoff between two factors
  - Sensitivity: avoid the influence of other factors

Homework

- Read the paper **SpecInfer: Accelerating Large Language Model Serving with Tree-based Speculative Inference and Verification**, summarize the paper, specifically, including the points below:
  - What are the motivations/challenges of this work?
  - How does the design of this paper address the challenges?
  - How does the paper evaluate its design (experiment settings, workloads, metrics)?
  - How does the evaluation prove its claims?
- Note that it is essential to logically connect the motivation, design, and evaluation, rather than merely listing some points.
- Related link:
  - Paper of SpecInfer: https://dl.acm.org/doi/pdf/10.1145/3620666.3651335

Q & A