REALM & CacheBlend

Saad Sher Alam (saadsa2), Andrew Zuo (bzuo2), Yuhang Li (yuhang8)

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Outline

- **Paper 1: REALM - Retrieval Augmented Language Model Pre-Training (Saad)**
  - Background and Motivation
  - Insights
  - Method & System design
  - Evaluation
  - Limitation

- **Paper 2: CacheBlend - Fast Large Language Model Serving for RAG with Cached Knowledge Fusion (Andrew and Yuhang)**
  - Background
  - Previous Work
  - Insights
  - Method & System design
  - Evaluation
  - Limitation

REALM: Retrieval Augmented Language Model Pre-Training

OpenQA: Traditional Language Models

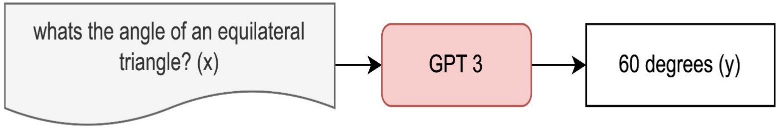


Fig 1: OpenQA Example

OpenQA: Traditional Language Models

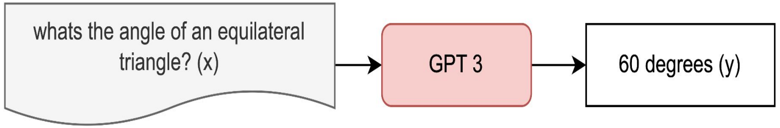


Fig 1: OpenQA Example

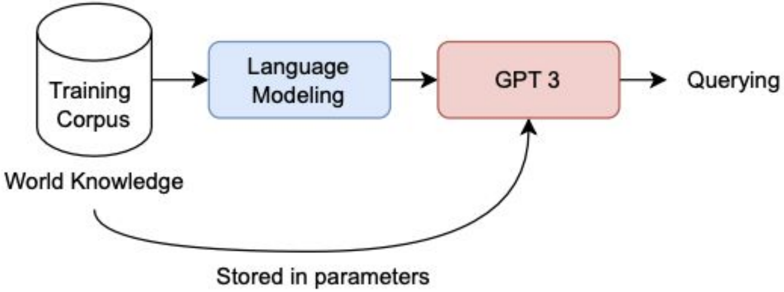


Fig 2: How traditional LMs perform Open QA

OpenQA: Traditional Language Models

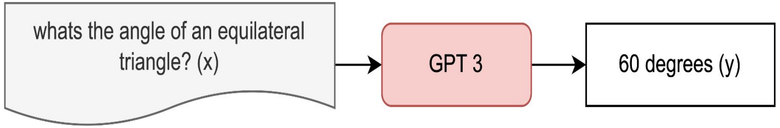


Fig 1: OpenQA Example

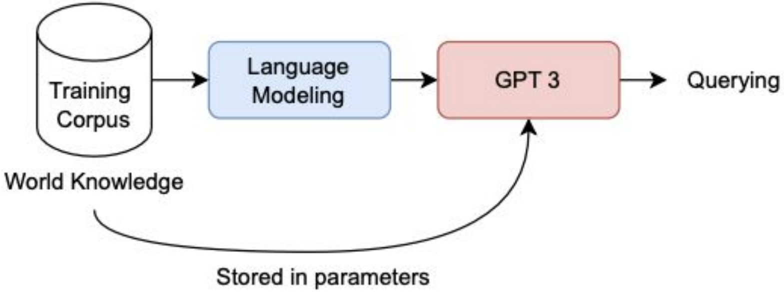


Fig 2: How traditional LMs perform Open QA

Problems:

- The knowledge is stored implicitly in the parameters of the network.
- To increase facts/knowledge, the size of the network needs to be increased.

OpenQA: Retrieval Based Approach

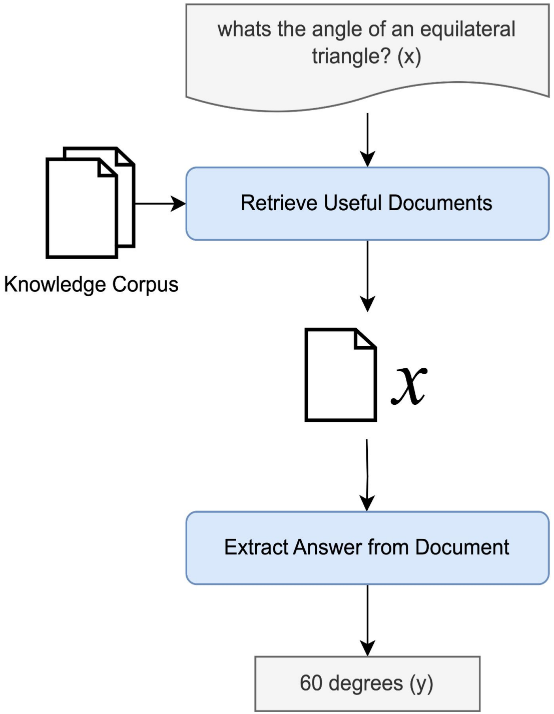


Figure 3: Retrieval Approach Overview

OpenQA: Retrieval Based Approach

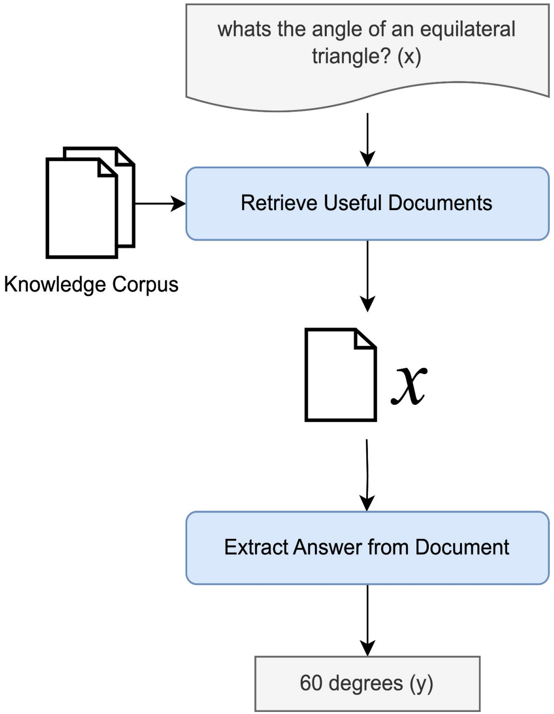


Figure 3: Retrieval Approach Overview

This is exactly what the paper aims to achieve!

OpenQA: Retrieval Based Approach

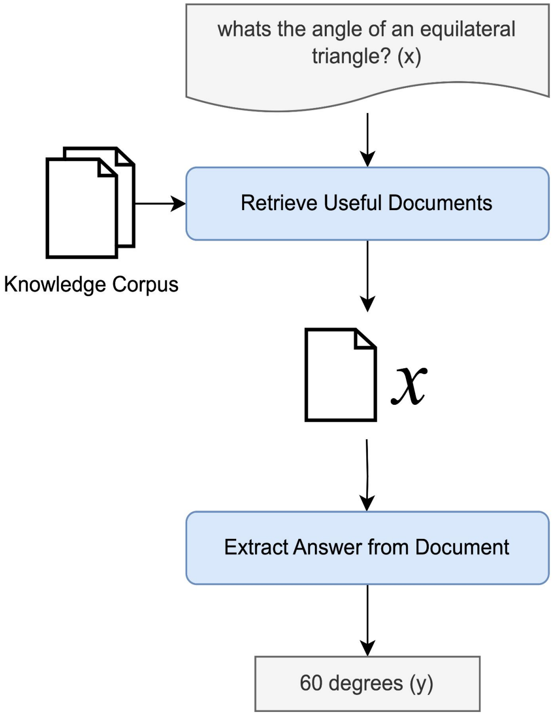


Figure 3: Retrieval Approach Overview

This is exactly what the paper aims to achieve!

Why would you want to do this?

- Reduce model size
- Increase model accuracy
- Make knowledge more modular and interpretable.

OpenQA: Retrieval Based Approach

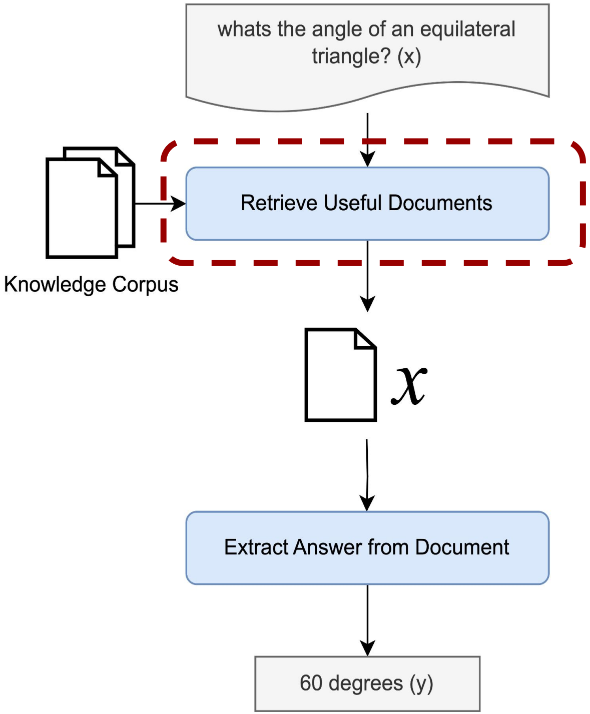


Figure 3: Retrieval Approach Overview

This is exactly what the paper aims to achieve!

Why would you want to do this?

- Reduce model size
- Increase model accuracy
- Make knowledge more modular and interpretable.

Two Important Components:

- A learned model to retrieve documents

OpenQA: Retrieval Based Approach

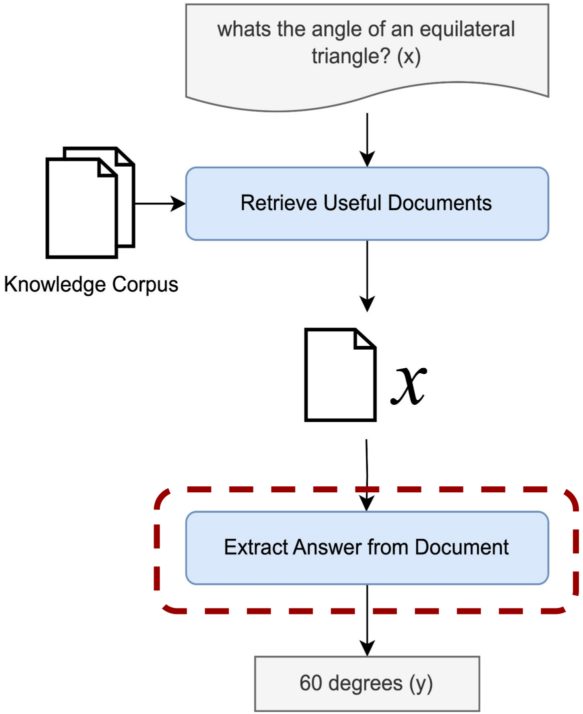


Figure 3: Retrieval Approach Overview

This is exactly what the paper aims to achieve!

Why would you want to do this?

- Reduce model size
- Increase model accuracy
- Make knowledge more modular and interpretable

Two Important Components:

- A learned model to retrieve documents
- A learned model to answer using documents

Pre-Training Task: Masked Language Modeling (MLM)

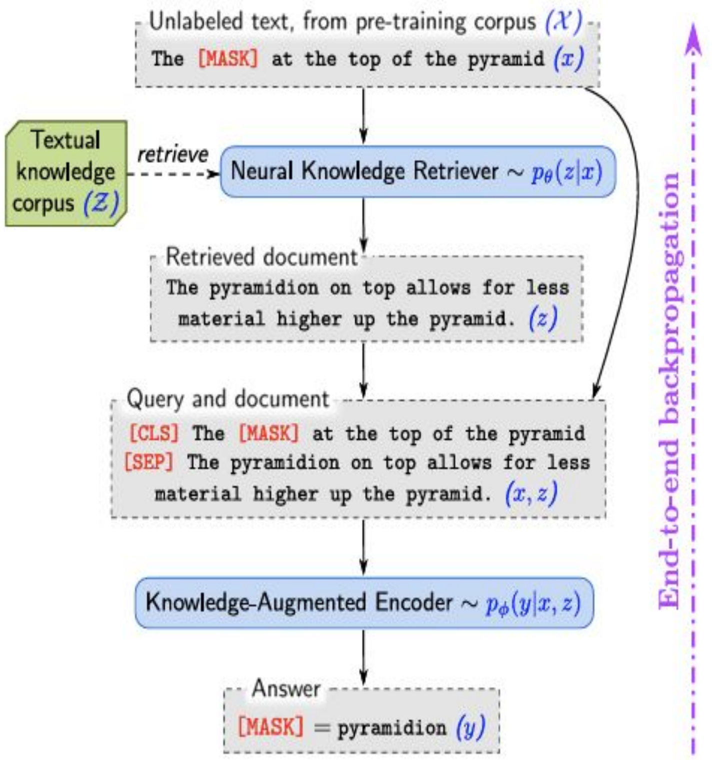


Figure 4: Pre-training for REALM

Neural Knowledge Retriever ~ $p_\theta(z|x)$

Goal: Train a model to extract most ‘relevant’ documents.

- How do you compare an input text and documents from the knowledge corpus?

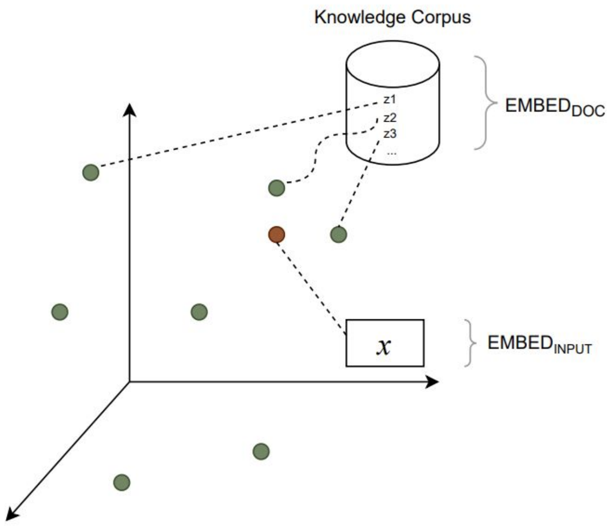
  Embed them into a d-dimensional vector space - **BERT**

Neural Knowledge Retriever ~ $p_\theta(z|x)$

Goal: Train a model to extract most 'relevant' documents.

- How do you compare an input text and documents from the knowledge corpus?

  Embed them into a d-dimensional vector space - **BERT**



Neural Knowledge Retriever ~ $p_\theta(z|x)$

Goal: Train a model to extract most 'relevant' documents.

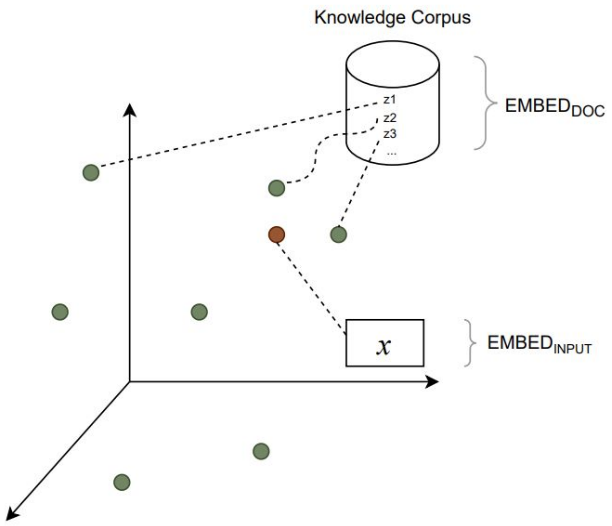
- How do you compare an input text and documents from the knowledge corpus?

  Embed them into a d-dimensional vector space - **BERT**

- How to assign a high score to similar vectors?

  Inner product

  $f(x, z) = (\mathrm{EMBED}_{\mathrm{input}})^T\,(\mathrm{EMBED}_{\mathrm{doc}})$



Neural Knowledge Retriever ~ $p_\theta(z|x)$

Goal: Train a model to extract most 'relevant' documents.

- How do you compare an input text and documents from the knowledge corpus?

  Embed them into a d-dimensional vector space - **BERT**

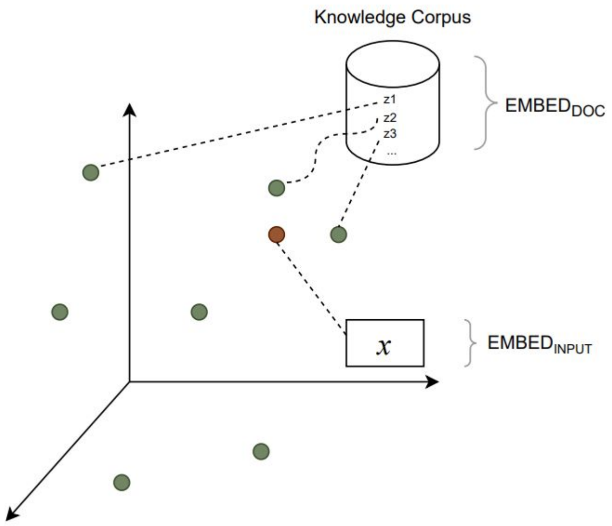
- How to assign a high score to similar vectors?

  Inner product

  $f(x, z) = (EMBED_{input})^T (EMBED_{doc})$

- Finally, to learn a probability distribution:

$$p(z \mid x) = \frac{\exp f(x, z)}{\sum_{z'} \exp f(x, z')}$$



- What are the parameters to learn?

  θ = $EMBED_{input,}$ $EMBED_{doc}$

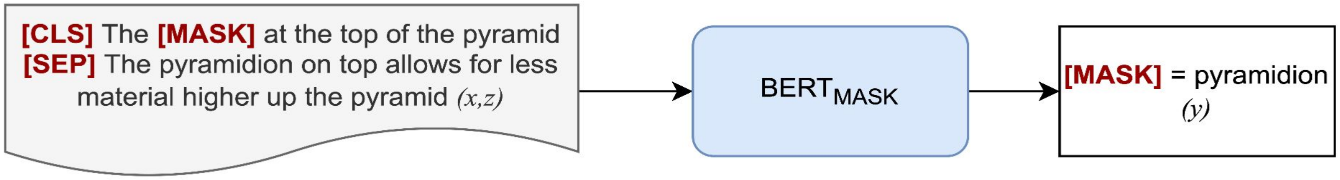
Knowledge Augmented Encoder ~ $p_{\phi}(y \mid z,x)$

Goal: Given input x and document z, predict the mask value.
Note: This BERT model is a text model. (It does not use the embeddings from previous models).

Knowledge Augmented Encoder ~ $p_\phi(y \mid z,x)$

Goal: Given input x and document z, predict the mask value.
Note: This BERT model is a text model. (It does not use the embeddings from previous models).

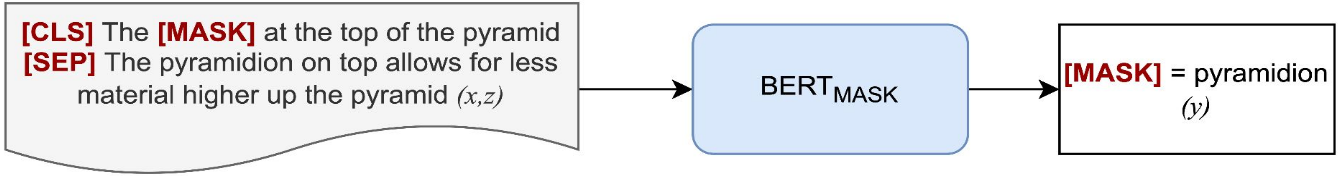
- Pre-training: predict mask as a classification task across all vocabulary.



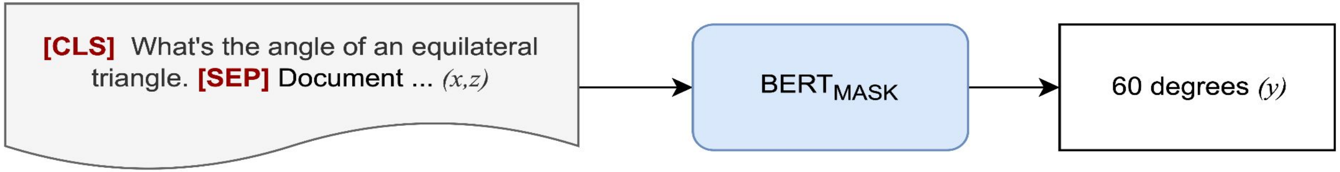
Knowledge Augmented Encoder ~ $p_{\phi}(y \mid z,x)$

Goal: Given input x and document z, predict the mask value.
Note: This BERT model is a text model. (It does not use the embeddings from previous models).

- Pre-training: predict mask as a classification task across all vocabulary.



- Fine-tuning: (**Assumption -** answer is as a span in the document) Classify the start and the end of the span in z.



Training

Neural Knowledge Retriever

$$p(y \mid x) = \sum_{z \in \mathcal{Z}} p(y \mid z, x) \boxed{p(z \mid x)}.$$

Training

Knowledge Augmented Encoder

Neural Knowledge Retriever

$$p(y \mid x) = \sum_{z \in \mathcal{Z}} \boxed{p(y \mid z, x)}\, \boxed{p(z \mid x)}.$$

Training

Knowledge Augmented Encoder

Neural Knowledge Retriever

$$p(y \mid x) = \boxed{\sum_{z \in \mathcal{Z}}} \boxed{p(y \mid z, x)} \boxed{p(z \mid x)}.$$

Marginalize over all documents

- **Loss:** log p(y | x) - Maximize the log-likelihood
- Everything is differentiable!

Training

Knowledge Augmented Encoder

Neural Knowledge Retriever

$$p(y \mid x) = \boxed{\sum_{z \in \mathcal{Z}}} \boxed{p(y \mid z, x)} \boxed{p(z \mid x)}.$$

Marginalize over all documents

- **Loss:** log p(y | x) - Maximize the log-likelihood
- Everything is differentiable!
- Loss derivative:

$$\nabla \log p(y \mid x) = \sum_{z \in \mathcal{Z}} r(z) \nabla f(x, z)$$

$$r(z) = \left[\frac{p(y \mid z, x)}{p(y \mid x)} - 1 \right] p(z \mid x).$$

Training

Knowledge Augmented Encoder

Neural Knowledge Retriever

$$p(y \mid x) = \sum_{z \in \mathcal{Z}} p(y \mid z, x)\, p(z \mid x).$$

Marginalize over all documents

- **Loss:** log p(y | x) - Maximize the log-likelihood
- Everything is differentiable!
- Loss derivative:

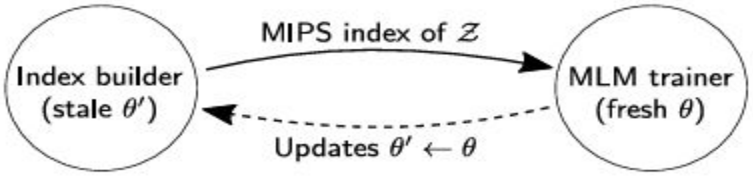
$$\nabla \log p(y \mid x) = \sum_{z \in \mathcal{Z}} r(z) \nabla f(x, z)$$

$$r(z) = \left[\frac{p(y \mid z, x)}{p(y \mid x)} - 1 \right] p(z \mid x).$$

+ve if p(y | z,x) > p(y|x)
-ve otherwise

Computational Overhead of Training

- Backpropagating over 13M possible documents for each pre-training iteration is computationally infeasible!
- **Solution:** Use only top k documents for each pre-training run (k = 5).
- How to find top k documents efficiently?
  - Maximum Inner Product Search (MIPS) algorithm.
  - Precompute $\text{EMBED}_{\text{DOC}}(z)$ for all documents.
  - Construct an efficient search index over these embeddings.
  - But, θ is being updated every epoch. The index goes stale after every gradient update.
  - **Solution:** Refresh index asynchronously, every several hundred training steps.



Experimental Setup

- Pretraining:
  - Steps: 200k
  - 64 Google Cloud TPUs
  - Batch Size: 512
  - Learning Rate: 3e-5
  - Optimizer: BERT’s default optimizer
- Fine-tuning:
  - ORQA fine-tuning approach.
  - Knowledge Corpus: Wikipedia (Dec 20, 2018)
  - 13M retrieval documents
  - k = 5
  - Entire model run on a single machine, 12GB GPU

Experimental Results

Name	Architectures	Pre-training	NQ (79k/4k)	WQ (3k/2k)	CT (1k /1k)	# params
BERT-Baseline (Lee et al., 2019)	Sparse Retr.+Transformer	BERT	26.5	17.7	21.3	110m
T5 (base) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	27.0	29.1	-	223m
T5 (large) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	29.8	32.2	-	738m
T5 (11b) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	34.5	37.4	-	11318m
DrQA (Chen et al., 2017)	Sparse Retr.+DocReader	N/A	-	20.7	25.7	34m
HardEM (Min et al., 2019a)	Sparse Retr.+Transformer	BERT	28.1	-	-	110m
GraphRetriever (Min et al., 2019b)	GraphRetriever+Transformer	BERT	31.8	31.6	-	110m
PathRetriever (Asai et al., 2019)	PathRetriever+Transformer	MLM	32.6	-	-	110m
ORQA (Lee et al., 2019)	Dense Retr.+Transformer	ICT+BERT	33.3	36.4	30.1	330m
ORQA (more fine-tune epochs)	Dense Retr.+Transformer	ICT+BERT	34.8	35.4	28.7	330m
Ours ($\mathcal{X}$ = Wikipedia, $\mathcal{Z}$ = Wikipedia)	Dense Retr.+Transformer	REALM	39.2	40.2	**46.8**	330m
Ours ($\mathcal{X}$ = CC-News, $\mathcal{Z}$ = Wikipedia)	Dense Retr.+Transformer	REALM	**40.4**	**40.7**	42.9	330m

Experimental Results

Name	**Architectures**	**Pre-training**	**NQ** (79k/4k)	**WQ** (3k/2k)	**CT** (1k /1k)	**# params**
BERT-Baseline (Lee et al., 2019)	Sparse Retr.+Transformer	BERT	26.5	17.7	21.3	110m
T5 (base) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	27.0	29.1	-	223m
T5 (large) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	29.8	32.2	-	738m
T5 (11b) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	34.5	37.4	-	11318m
DrQA (Chen et al., 2017)	Sparse Retr.+DocReader	N/A	-	20.7	25.7	34m
HardEM (Min et al., 2019a)	Sparse Retr.+Transformer	BERT	28.1	-	-	110m
GraphRetriever (Min et al., 2019b)	GraphRetriever+Transformer	BERT	31.8	31.6	-	110m
PathRetriever (Asai et al., 2019)	PathRetriever+Transformer	MLM	32.6	-	-	110m
ORQA (Lee et al., 2019)	Dense Retr.+Transformer	ICT+BERT	33.3	36.4	30.1	330m
ORQA (more fine-tune epochs)	Dense Retr.+Transformer	ICT+BERT	34.8	35.4	28.7	330m
Ours ($\mathcal{X}$ = Wikipedia, $\mathcal{Z}$ = Wikipedia)	Dense Retr.+Transformer	REALM	39.2	40.2	**46.8**	330m
Ours ($\mathcal{X}$ = CC-News, $\mathcal{Z}$ = Wikipedia)	Dense Retr.+Transformer	REALM	**40.4**	**40.7**	42.9	330m

Outperforms the standard language models in terms of: (1) accuracy, (2) model size

Experimental Results

Name	Architectures	Pre-training	NQ (79k/4k)	WQ (3k/2k)	CT (1k /1k)	# params
BERT-Baseline (Lee et al., 2019)	Sparse Retr.+Transformer	BERT	26.5	17.7	21.3	110m
T5 (base) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	27.0	29.1	-	223m
T5 (large) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	29.8	32.2	-	738m
T5 (11b) (Roberts et al., 2020)	Transformer Seq2Seq	T5 (Multitask)	34.5	37.4	-	11318m
DrQA (Chen et al., 2017)	Sparse Retr.+DocReader	N/A	-	20.7	25.7	34m
HardEM (Min et al., 2019a)	Sparse Retr.+Transformer	BERT	28.1	-	-	110m
GraphRetriever (Min et al., 2019b)	GraphRetriever+Transformer	BERT	31.8	31.6	-	110m
PathRetriever (Asai et al., 2019)	PathRetriever+Transformer	MLM	32.6	-	-	110m
ORQA (Lee et al., 2019)	Dense Retr.+Transformer	ICT+BERT	33.3	36.4	30.1	330m
ORQA (more fine-tune epochs)	Dense Retr.+Transformer	ICT+BERT	34.8	35.4	28.7	330m
Ours ($\mathcal{X}$ = Wikipedia, $\mathcal{Z}$ = Wikipedia)	Dense Retr.+Transformer	REALM	39.2	40.2	**46.8**	330m
Ours ($\mathcal{X}$ = CC-News, $\mathcal{Z}$ = Wikipedia)	Dense Retr.+Transformer	REALM	**40.4**	**40.7**	42.9	330m

Outperforms other retrieval based modes in terms of accuracy

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

Background - Prefill

User's query is prepended with text chunks for better response quality

User: Please help me plan a trip that includes Tokyo,Miami, and Paris

Application: [Tokyo-info] [Miami-info] [Paris-info] Please help me plan a trip that includes Tokyo,Miami, and Paris

Background - Prefill

LLM go through the entire input to produce the KV Cache before generating any token.

KV Cache

[Tokyo-info] [Miami-info] [Paris-info] **Please help me plan a trip that includes Tokyo,Miami, and Paris**

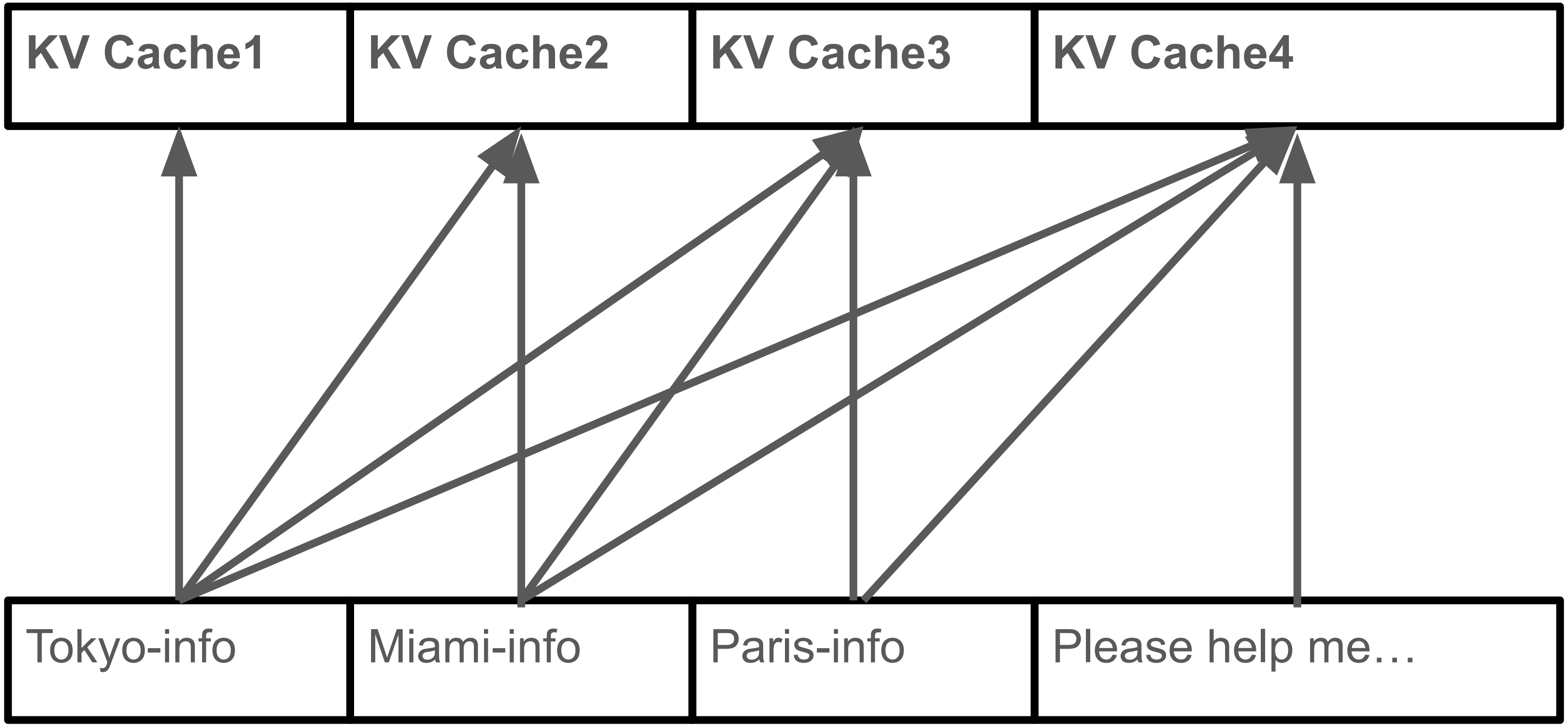
Background - Why KV Cache

The attention output at the t-th time step is computed as follows:

$$z_t = \mathrm{softmax}\left(\frac{q_t \cdot K^T}{\sqrt{d_k}}\right) V$$

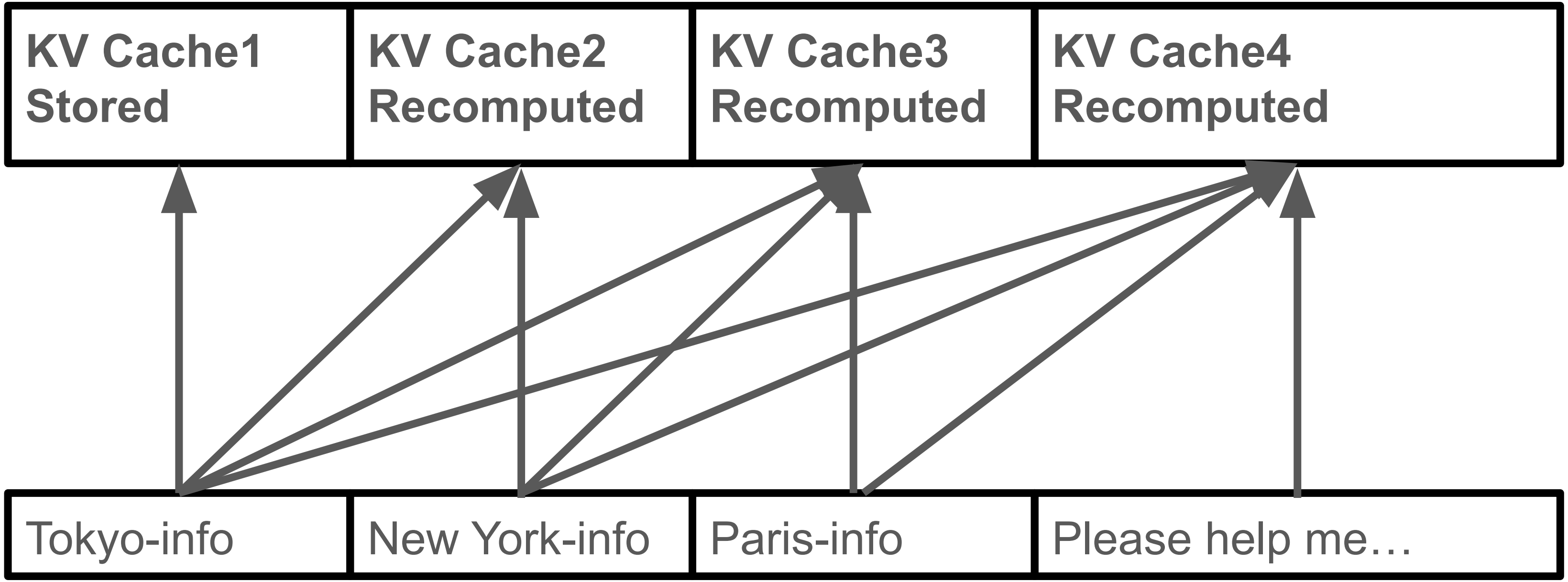
- q_t: The query vector at the t-th time step.
- K: The matrix of keys from all previous time steps, typically represented as $[k_1, k_2, \ldots, k_{t-1}]$.
- V: The matrix of values from all previous time steps, represented as $[v_1, v_2, \ldots, v_{t-1}]$.

Background - KV Cache



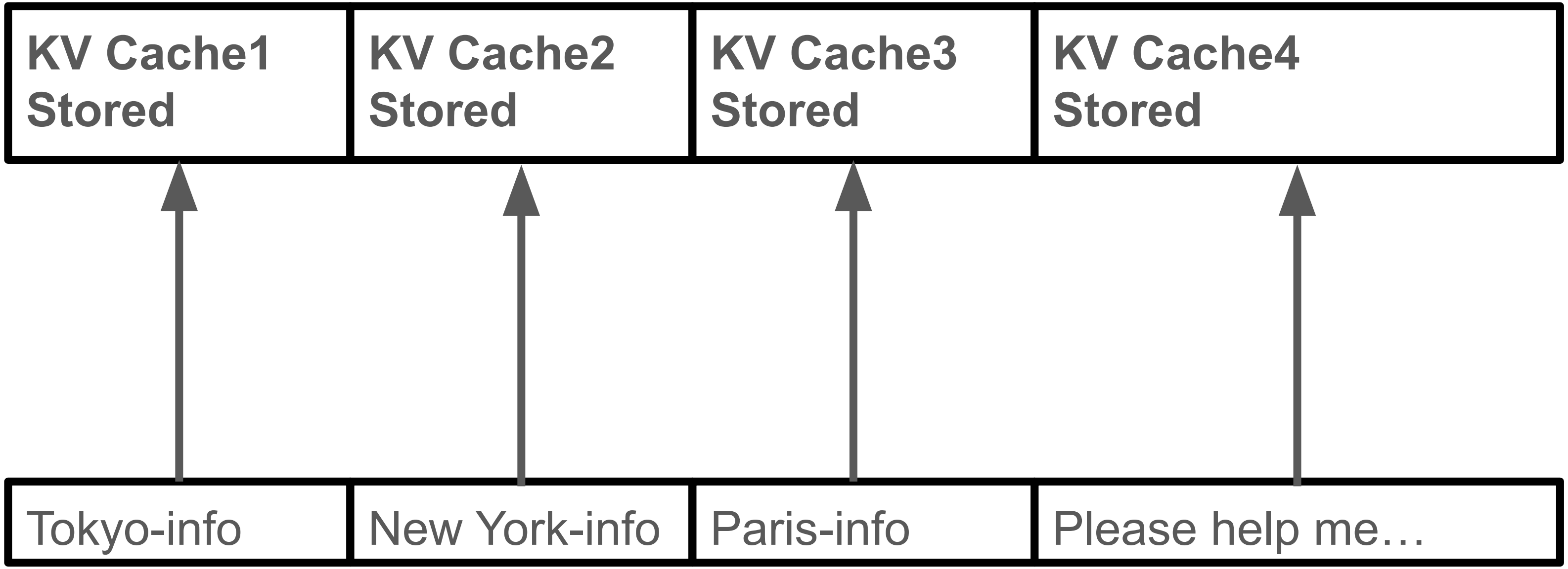
Background - Prefix Caching

High performance
Only reuse the first chunk's KV cache

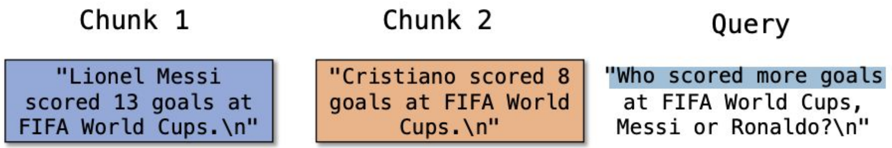


Background - Full KV Reuse

Low performance (ignore cross-attention)
Reuse all KV caches



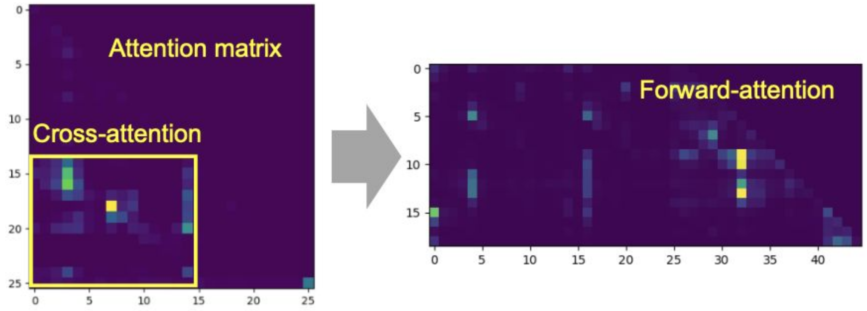
Background - Full KV Reuse Gives Wrong Answer



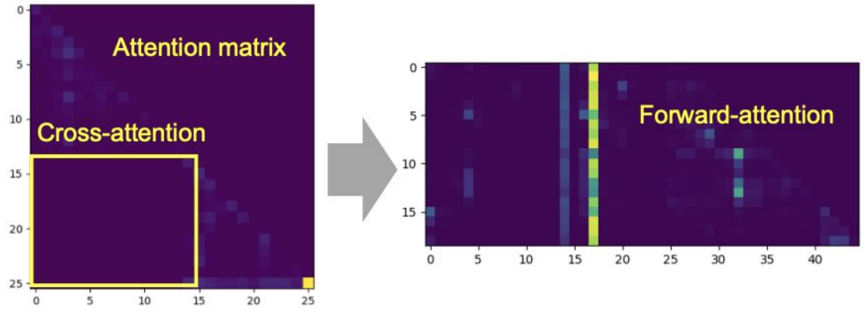
(a) Setup: Query and two relevant text chunks.

(b) Full KV recompute gives correct answer.

(c) Full KV reuse gives wrong answer.



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



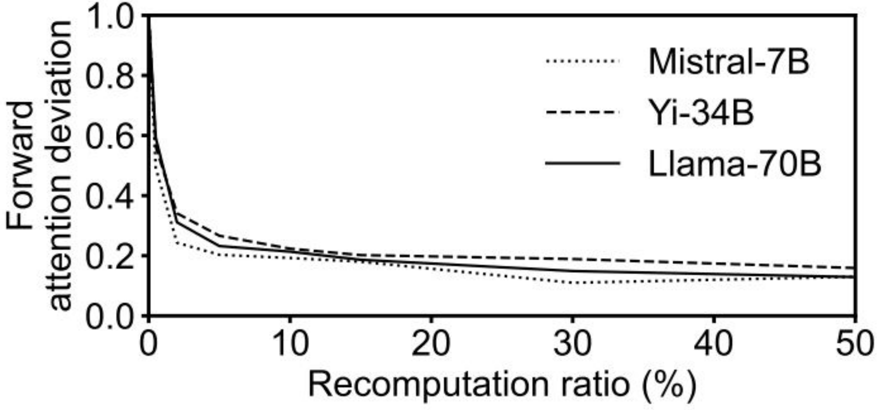
Question:

When an LLM input includes multiple re-used text chunks, how to *quickly* update the pre-computed KV cache, such that the forward attention matrix has *minimum difference* with the one produced by full KV recompute.

Terminology

- **KV Deviation**: Absolute difference between the precomputed KV cache and full recomputed KV cache
- **Attention Deviation**: L2 norm of the difference between the attention matrix of precomputed KV cache and the attention matrix of recomputed KV cache

Insight 1

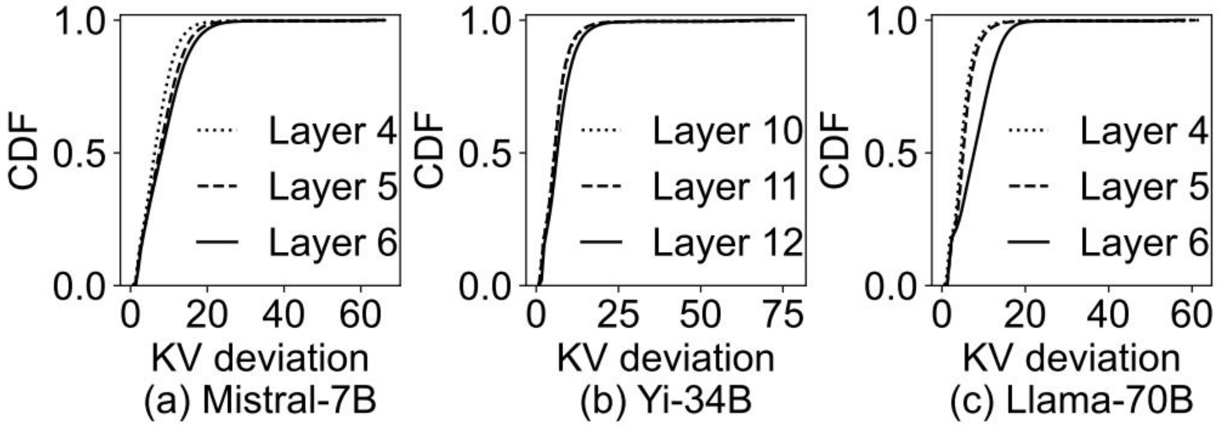


Attention deviation reduces as we recompute KV for more tokens.

The biggest drop results from recomputing the KV of the tokens with the highest KV deviation.

- **Recomputing the KV of tokens with a higher KV deviation reduces the attention deviation by a greater amount.**

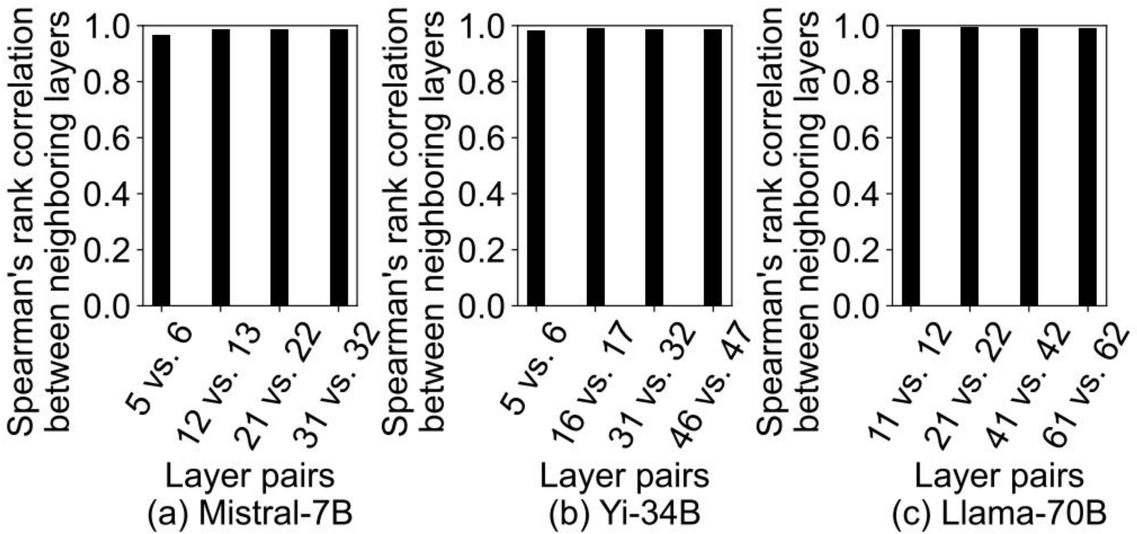
Insight 1



Do we need to recompute KV for most tokens?

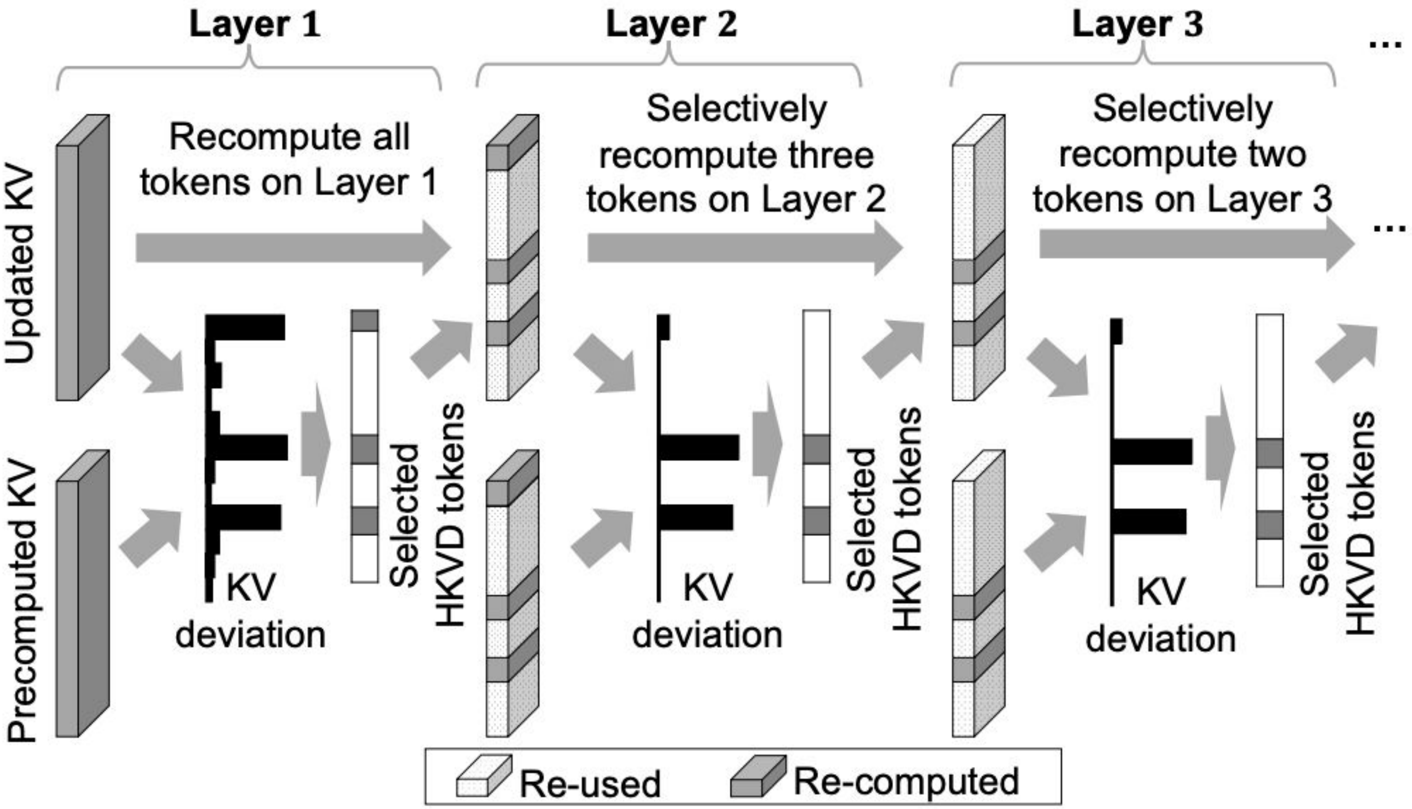
– A small fraction of tokens have much higher KV deviations than others

Insight 2



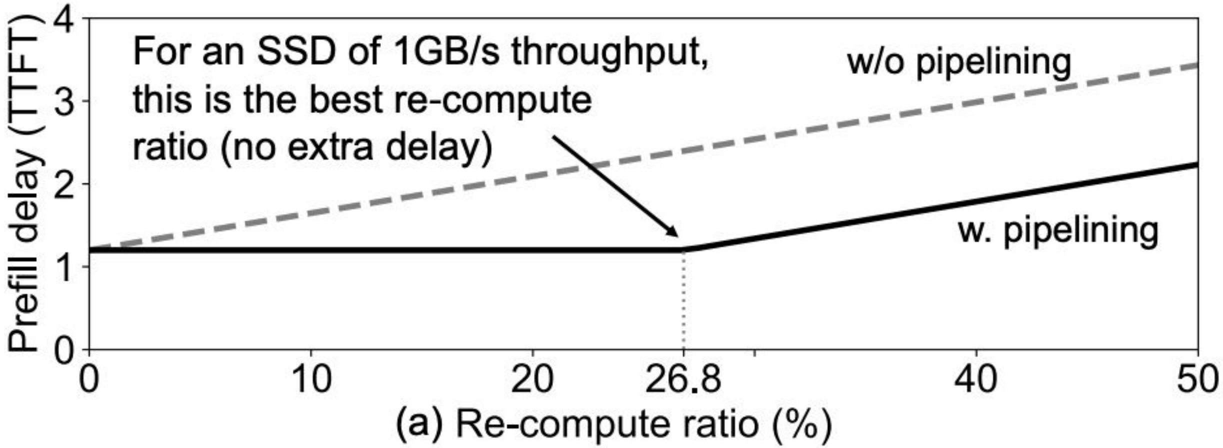
- **Tokens with the highest KV deviations on one layer are likely to have the highest KV deviations on the next layer.**

Method



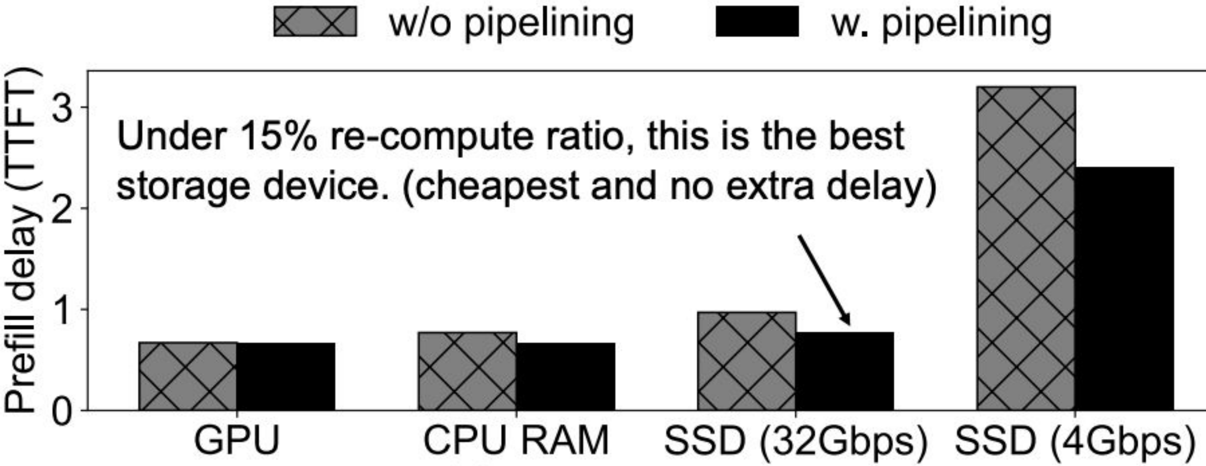
Layer 1
Layer 2
Layer 3
...
Recompute all tokens on Layer 1
Selectively recompute three tokens on Layer 2
Selectively recompute two tokens on Layer 3
...
Updated KV
Precomputed KV
KV deviation
Selected HKVD tokens
KV deviation
Selected HKVD tokens
KV deviation
Selected HKVD tokens
Re-used
Re-computed

System Design



If the delay for selective KV recompute is faster than the loading of KV into GPU memory, then properly pipelining the selective KV recompute and KV loading makes the extra delay of KV recompute negligible.

System Design



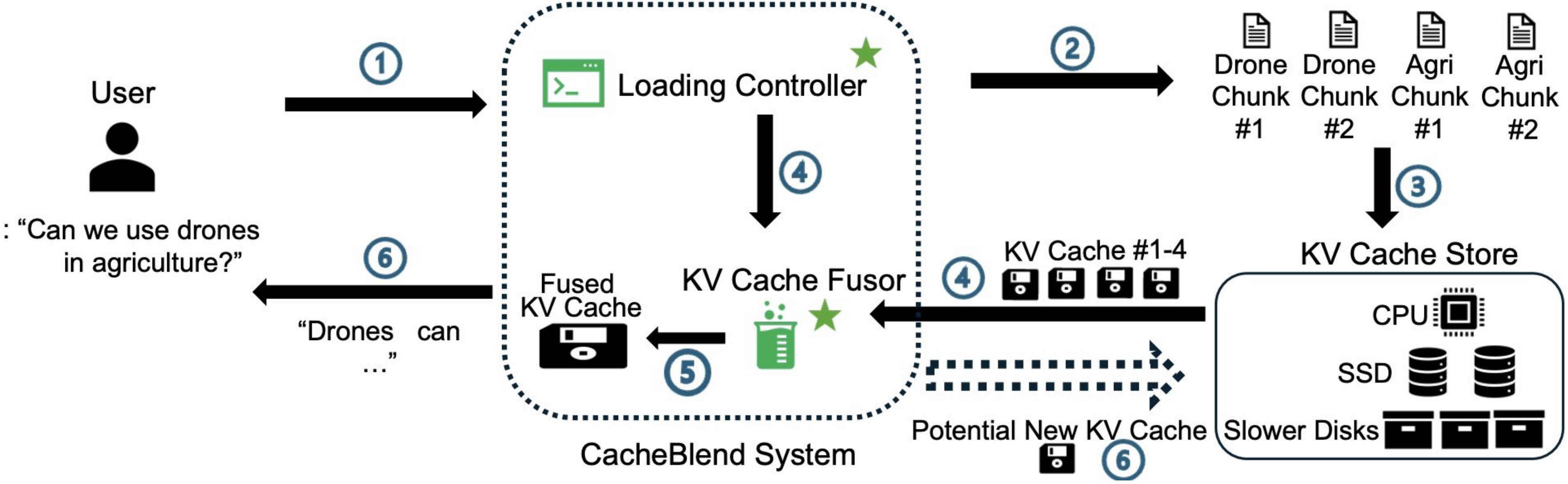
(b) Storage Devices

Smartly picking storage device to store KVs saves cost while not increasing delay.

System Design

- Loading Controller: Determine the recompute ratio and the storage device for KV cache
- KV Cache Store: Split LLM inputs into multiple text chunks and map into KV caches
- Fusor: Merge pre-computed KV caches via selective recompute

System Design



Evaluation

TTFT: start after input is received till first token is output.

F1-score for QA and **Rouge-L score** for Summarization task:

F1-score=2×(Precision+Recall)/Precision×Recall

Rouge-L = F1(Longest Common Subsequence)

Throughput(under same TTFT)

Baseline

Full KV recompute: Raw text

Prefix caching(mentioned before): KV cache of frequently used prefix chunks store both RAM and SSD + idealized assumption: no delay from RAM or SSD to GPU

Full KV reuse(mentioned before)

Langchain default: **MapReduce**(recursively summarize) and **MapRerank**(choose the best answer by model itself)

Evaluation Setup

Mistral-7B, Yi-34B, Llama-70B

“Runpod GPUs with 128 GB RAM, 2 Nvidia A40 GPUs, and 1TB NVME SSD whose measured throughput is 4.8 GB/s. We use 1 GPU to serve Mistral7B and Yi-34B, and 2 GPUs to serve Llama-70B.”

- **2WikiMQA7**: reasoning skills
- **Musique7**: multi-hop reasoning ability
- **SAMSum**: summary ability, conversational text
- **MultiNews**: professional summary, multi-document summarization

Query turned into 512-token chunks with Langchain

Use GPT4 API to generate 3 more similar queries.

Top-6 chunks based on L2 distance.

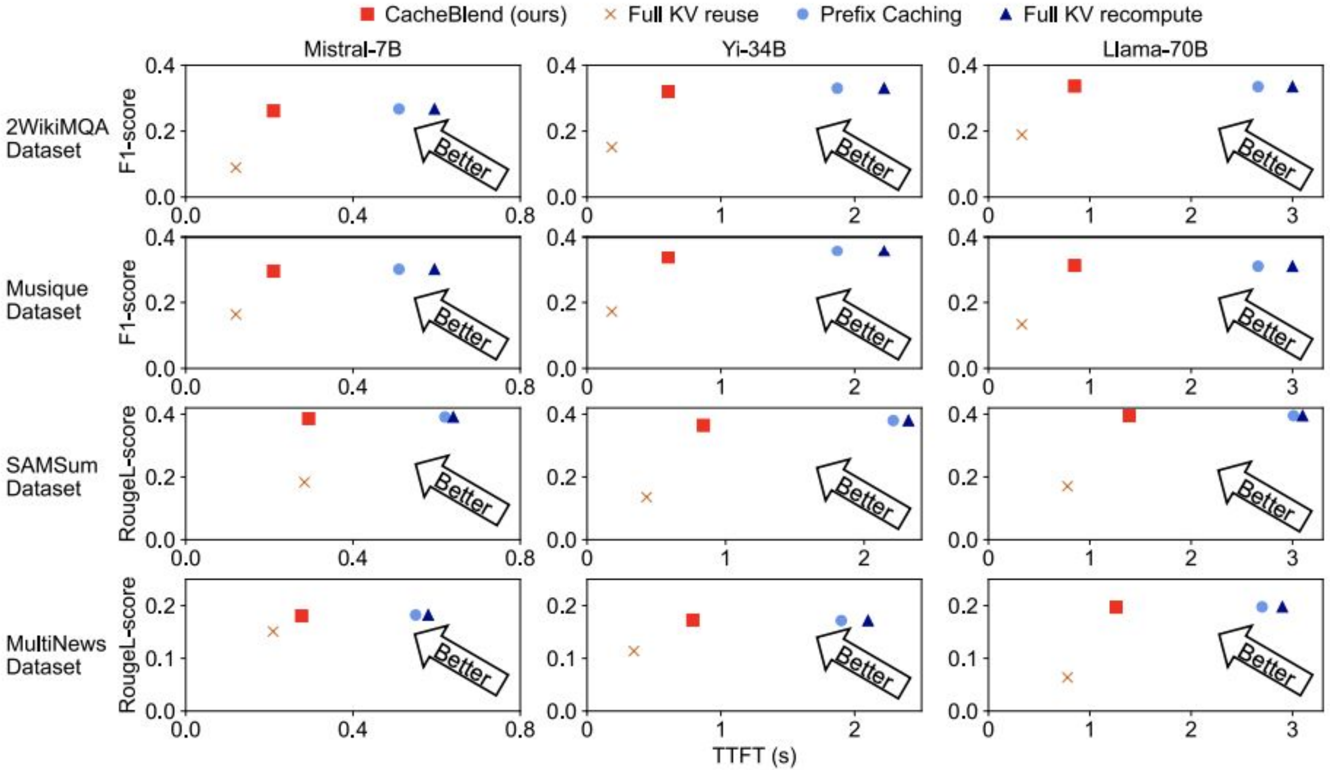


Figure 12. *CacheBlend reduces TTFT by 2.2-3.3× compared to full KV recompute with negligible quality drop across four datasets and three models.*

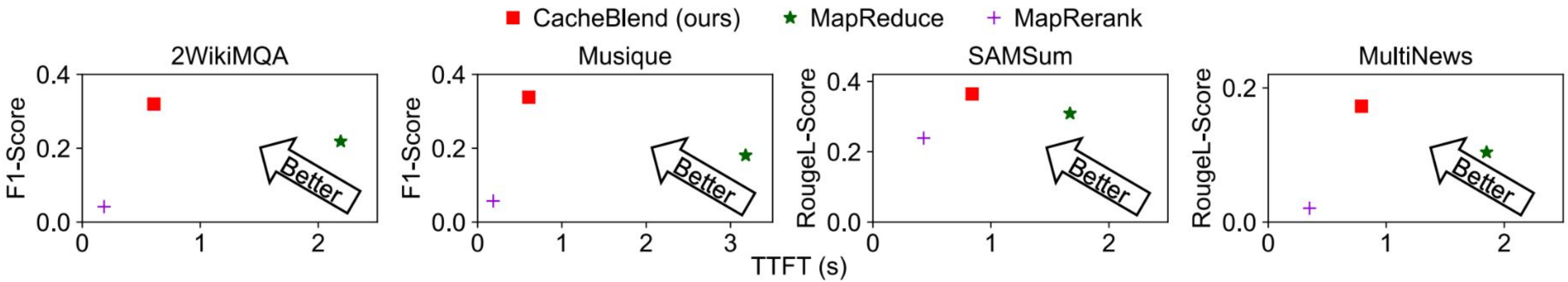


Figure 13. *Generation quality of CacheBlend with Yi-34B vs MapReduce and MapRerank.*

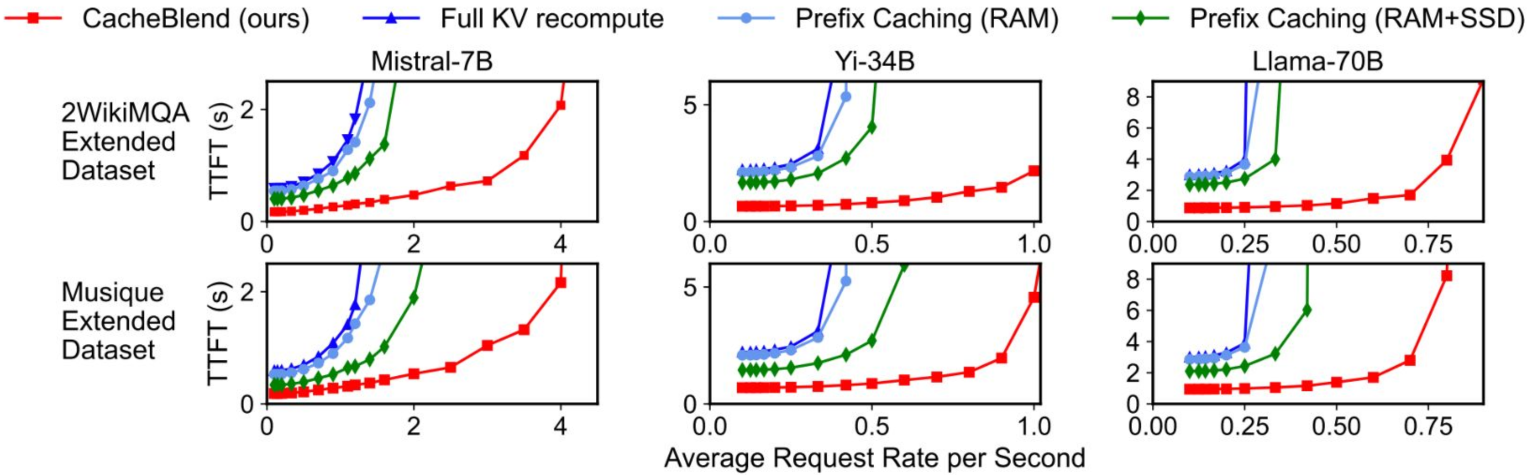


Figure 14. *CacheBlend achieves lower TTFT with higher throughput in RAG scenarios compared with baselines of similar quality.*

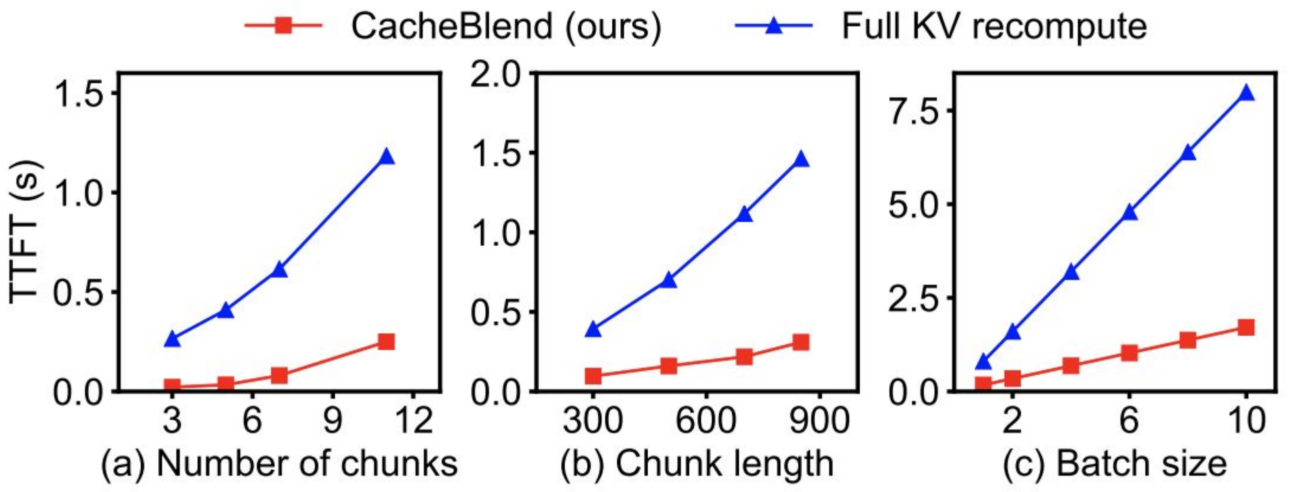


Figure 15. *CacheBlend outperforms baseline with varying chunk numbers, chunk lengths, and batch sizes.*

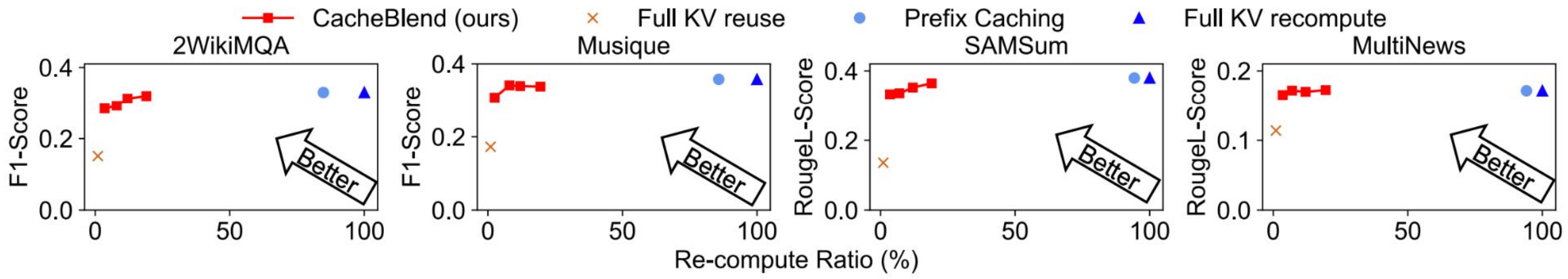
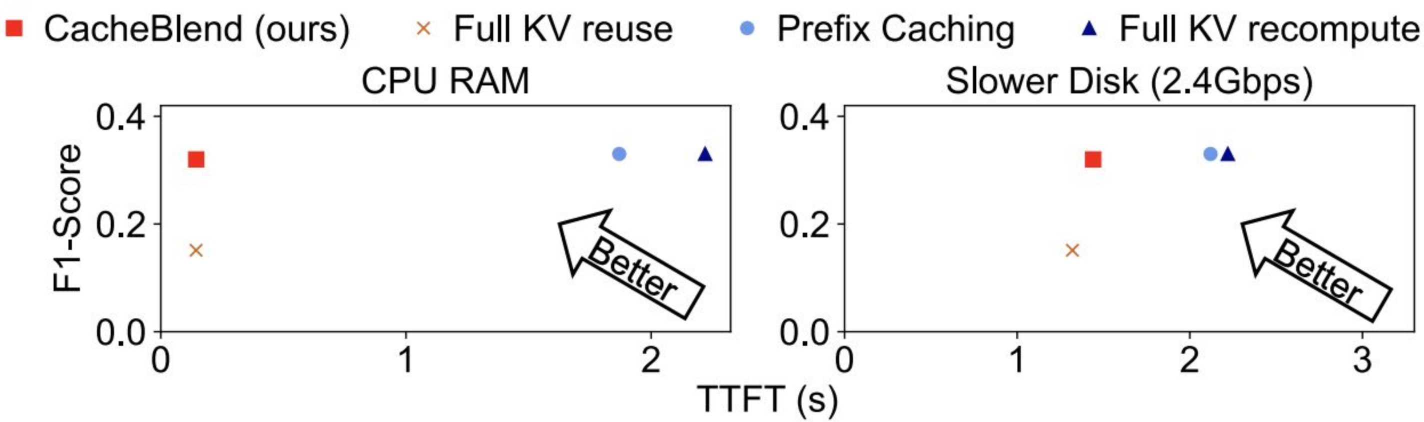


Figure 16. *CacheBlend has minimal loss in quality compared with full KV recompute, with 5%–18% selective recompute ratio with Yi-34B.*



Limitations

- Cannot work on non-transformer model
- May not work for MOE
- Have not tested in distributed & stable inference framework, DistServe or StableGen
- Text chunks before question. E.g. Multi-turn

- Others
- Can be used for non RAG task

Thank you!