SemShareKV: Efficient KVCache Sharing for Semantically Similar Prompts via Fuzzy Token Matching

Anonymous ACL submission

Abstract

As large language models (LLMs) continue to scale, the memory footprint of key-value (KV) caches during inference has become a significant bottleneck. Existing approaches primarily focus on compressing KV caches within a single prompt or reusing shared prefixes or frequently ocurred text segments across prompts. However, such strategies are limited in scenarios where prompts are semantically similar but lexically different, which frequently occurs in tasks such as multi-document summarization and conversational agents. We propose *SemShareKV*, a KV cache sharing and compression framework that accelerates LLM inference by reusing KVCache in semantically similar prompts. Instead of relying on exact token matches, SemShareKV applies fuzzy token matching using locality-sensitive hashing (LSH) on token embeddings and incorporates Rotary Position Embedding (RoPE) to better preserve positional information. By selectively reusing relevant key-value pairs from a reference prompt's cache, SemShareKV reduces redundant computation while maintaining output quality. Experiments on diverse summarization datasets show up to 6.25× speedup and 42% lower GPU memory usage with 5k tokens input, with negligible quality degradation. These results highlight the potential of semantic-aware cache sharing for efficient LLM inference. The code is available at `https://anonymous.4open.science/r/SemShareKV-B53C`.

1 Introduction

Large Language Models (LLMs) have exhibited strong capability to understand and process human languages, and have been proved to perform comparably as human beings in several fields, such as math inference, text memorization, information extraction, story telling (Naveed et al., 2023). Recently released LLMs have significantly advanced in processing and comprehending extremely long prompts. However, this progress introduces a notable challenge: increased computational demand due to the quadratic time complexity of their Decoder-Only Transformer architecture when handling lengthy text sequences. The issue is further compounded during inference, as the auto-regressive decoding process repeats the computation for each newly generated token (Luohe et al., 2024).

Existing KVCache optimization approaches primarily focus on single-prompt compression through various techniques: Yang et al. (2024a) leverage decaying key-value importance across layers for selective extraction (though with limited small-batch gains), Gim et al. (2024) employ restrictive markup schemas for text chunk reuse, and Yao et al. (2024) propose deviation-based recomputation that requires impractical per-chunk precomputation for long inputs. Crucially, these methods operate within the constrained paradigm of single-prompt optimization, failing to exploit the substantial efficiency potential of cross-prompt cache reuse, a significant oversight given the prevalence of semantically similar queries in real-world applications where shared computational savings could be substantial.

Motivated by this challenge, we aim to address the following research question: ***Can we reuse the precomputed KVCache for another semantically similar prompt?***

To answer this question, we proposed ***SemShareKV***, a KVCache framework that can reuse the cache from one prompt for another that is semantically similar to each other via fuzzy token match. It speeds up prefill phase and compress KV cache in memory. We show that our method can reduce the pre-fill phase time by 6.25× and save 42% GPU memory space. We make the following contributions.

- We introduce SemShareKV, which, to the best of our knowledge, is the first to explore

KVCache sharing across semantically similar prompts.

- We evaluate SemShareKV across multiple datasets, demonstrating its effectiveness in accelerating the prefill phase while simultaneously reducing KVCache size.

- We proposed another angle of studying the importance position encoding in KVCache, by introducing position encoding into vector embeddings.

2 Related Work

Prior research on KVCache optimization can be categorized into three key directions: (i) **Conventional KVCache Compression**, which focuses on reducing the storage and computational overhead of KVCache by applying quantization, pruning, or other compression techniques; (ii) **KVCache Sharing**, which explores methods to reuse KVCache across different queries or tasks to improve efficiency while maintaining response quality; and (iii) **KVCache Reusing**, which investigates strategies to adapt and repurpose precomputed KVCache for semantically similar inputs, minimizing redundant computation while preserving model accuracy.

2.1 Conventional KVCache Compression

To address long-context processing, many works propose optimizing inference by retaining only informative tokens. Token-level compression often uses attention-based token selection (Zhang et al., 2023; Xiao et al., 2024; Li et al., 2024; Yang et al., 2024a; Zhong et al., 2024), low-rank decomposition (Sun et al., 2024), or quantization (Zhang et al., 2024; Wang et al., 2024). Model-level approaches redesign architectures to improve reuse (Sun et al., 2025; Yan et al., 2024), while system-level methods focus on memory and scheduling (Kwon et al., 2023; Sheng et al., 2023). Recent work has highlighted the use of value vectors to facilitate compression (Guo et al., 2024).

2.2 KVCache Sharing

Some also emphasize reusing portions of the cache for future or similar queries and prompts. For example, PromptCache (Gim et al., 2024) stores text segments that appear frequently on an inference server using a schema, although this approach hampers usability, as users must conform their natural language to the schema format. Mooncake (Qin et al., 2024), KVSharer (Yang et al., 2024b) and MiniCache (Liu et al.) exploit the high similarity of attention scores among adjacent transformer layers to improve KVCache reuse. By consolidating or sharing Key-Value pairs between similar layers, these methods improve memory efficiency and streamline token processing. However, their approaches are restricted to sharing in the layer or text segment within adjacent layers or the same LLM, limiting the broader applicability; GPTCache (Regmi and Pun, 2024), (Rasool et al., 2024) and (Bang, 2023) utilize similarity search among queries to reuse KVCache. However, they have a high probability of missing a hit and require the entire query to be similar, offering limited flexibility.

2.3 KVCache Reusing

Limited attention has been directed toward the sharing of KVCache in LLMs. DroidSpeak (Liu et al., 2024b) improves context sharing between fine-tuned LLMs by identifying critical KV cache layers and selectively re-computingg them for efficient reuse while maintaining accuracy. LMCache (Cheng et al., 2024) introduces a Knowledge Delivery Network (KDN) to optimize KV cache storage and transfer, allowing cost-effective knowledge injection in LLM inference.

3 Observations and Insights

We present three key insights derived from our experiments on three LLMs: Mistral-7B (Jiang et al., 2023), LLaMA-3.1-8B (Grattafiori et al., 2024), and MPT-7B (Team, 2023). These insights show consistent patterns across different LLMs, supporting the generality of our observations.

Insights 1 *HD tokens stay consistent across layers.*

When reusing KV caches from semantically similar prompts, we ensure the reused cache maintains high fidelity with fully recomputed caches to prevent performance degradation. To compare the similarity between two KV matrices, we used our augmented MultiNews dataset, where each sample consists of a pair of semantically similar prompts: the ***Target Prompt***, which serves as the primary input to the model, and the ***Reference Prompt***, which acts as the semantically similar counterpart. For each of the aforementioned LLMs, we first computed the KV caches for the prompt pairs independently. Subsequently, we calculated the deviations between the KV caches of the target and reference prompts using the previously mentioned L2 norm.

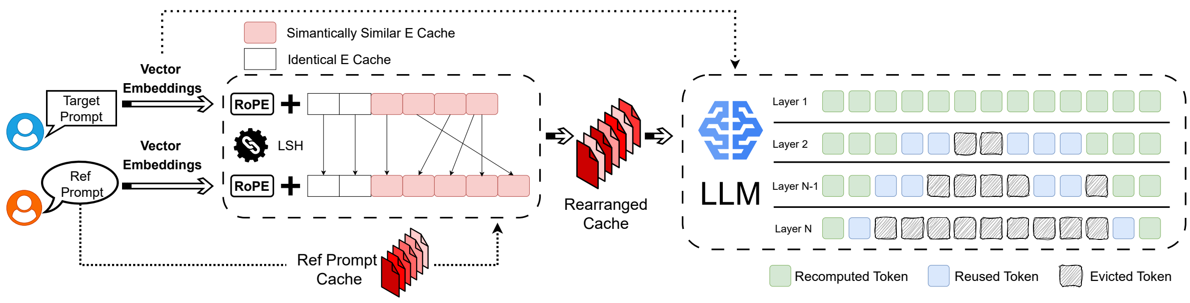


Figure 1: Comparison of retention patterns for Llama3.1-8B, Mistral-7B, and MPT-7B.

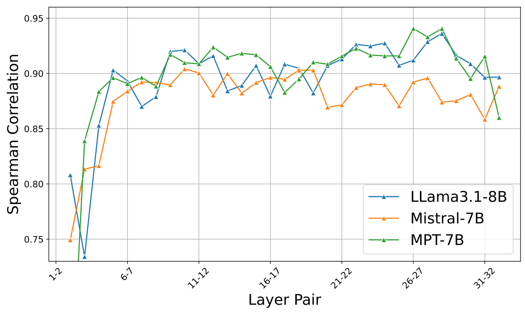


Figure 2: Insight 1: High-deviation tokens remain consistent across layers.

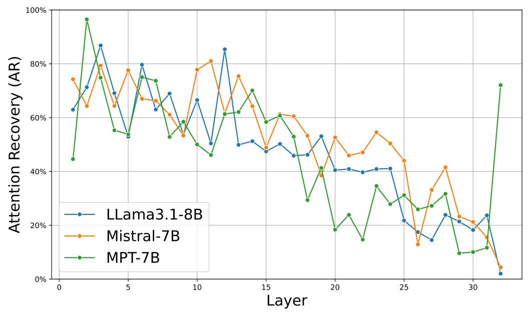


Figure 3: Insight 2: Deeper layers attend to fewer tokens.

Tokens with the highest 40% deviation were identified as ***High Deviation*** (HD) tokens.

To further quantify this observation, we compute the Spearman correlation of HD tokens between adjacent layers. As shown in Figure 2, adjacent layers exhibit relatively high consistency in HD token positions.

Insights 2 *Self-attention from deeper layers focus on fewer tokens.*

To analyze attention patterns across layers, we first averaged the attention scores across all heads in each layer and then computed the mean along the first dimension, resulting in a one-dimensional vector per layer. To quantify this behavior, we introduce ***Attention Recovery*** (AR), defined as follows:

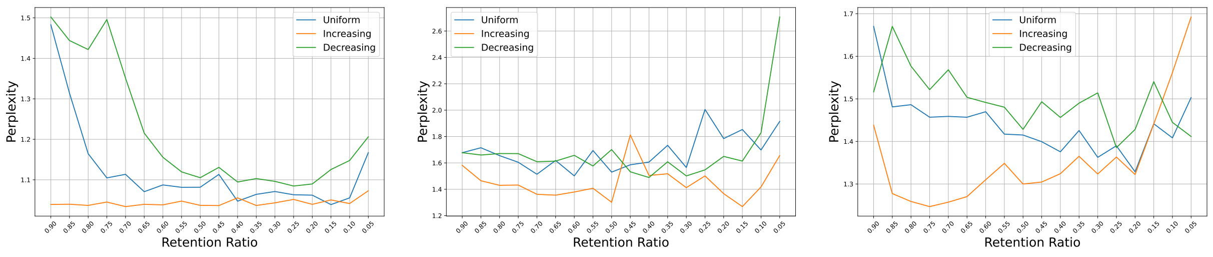
$$S_{\text{total}} = \sum_{i=1}^{n} T_i \qquad \frac{\sum_{i=k}^{n} T_i}{S_{\text{total}}} > \text{Thres} \tag{1}$$

Where T is a sorted vector of average attention scores for each token, S_{total} represents the total attention score derived from the averaged self-attention matrices, and $Thres$ indicates the threshold of attention score. AR indicates the number of tokens that must be summed from highest to lowest based on their average attention scores in order to cover $Thres\%$ of the total attention score. We computed AR for each layer, and the results (Figure 3) reveal a consistent trend: as depth increases, AR decreases across all three LLMs, despite minor fluctuations. This suggests that deeper layers concentrate attention on progressively fewer tokens, reflecting more selective focus.

Insights 3 *Deeper layers have more redundant information.*

To reduce memory overhead from the KV cache, a key optimization strategy is to remove tokens containing redundant information. Such tokens contribute minimally to the prediction of next tokens during decoding but occupy substantial GPU memory. However, selective token retention risks information loss, necessitating careful trade-offs between memory savings and generation quality. To address this, we evaluated three token retention strategies and assessed their impact on model performance using perplexity as our primary metric.

Figure 5 illustrates three retention patterns for KVCache: **Constant Retention**, where each layer retains the same percentage of KVCache; **Exponential Growth Retention**, where the shallow layers retain more KVCache and the retention ratio decreases in the deeper layers; and **Exponential Decay Retention**, where shallow layers retain less KVCache, with the retention ratio increasing in



(a) Llama3.1-8B Retention Pattern (b) Mistral-7B Retention Pattern (c) MPT-7B Retention Pattern

Figure 4: Insight 3: Deeper layers contain more redundant information.

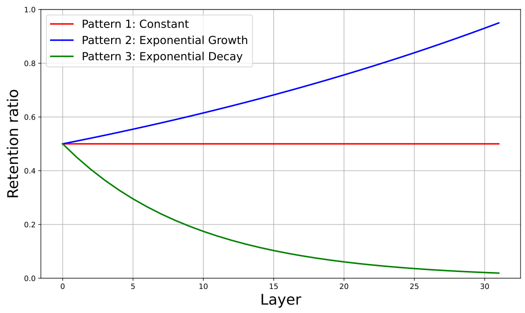


Figure 5: The three retention patterns start from the same retention ratio.

deeper layers.

We applied these three retention patterns to LLMs, and utilized perplexity to benchmark the generation performance, shown in Figure 5. For all three LLMs, the Exponential Decay pattern achieves the lowest perplexity score, indicating the best generation performance. This finding further validates that this pattern aligns with how LLMs interpret knowledge from prompts.

4 Methodology

4.1 Model Overview

The design of SemShareKV, illustrated in Figure 1, is based on three key insights from Section 3. Our approach employs two core strategies:

- **Recomputation Strategy (Insights 1 & 2):** Prioritize the recomputing of more tokens in shallow layers while reducing the recomputation in deeper layers, reflecting the varying importance of the layer depth in attention mechanisms.

- **Retention Strategy (Insights 1 & 3):** Preserve more tokens in shallow layers while evicting tokens from deeper layers, optimizing memory usage without significant accuracy degradation.

When the LLM processes the target prompt, its vector embedding is first integrated with RoPE position encoding. Using Locality-Sensitive Hashing (LSH), each token in the target prompt is then matched to the most similar tokens from the reference prompt. Based on these LSH mappings, the precomputed KVCache of the reference promptrompt is rearranged token by token and injected into LLM transformer layers. On the first transformer layer, all tokens undergo full recomputation. The recomputed outputs are compared with the rearranged cache values via L2 norm, identifying high-deviation tokens for prioritized recomputation in subsequent layers. Simultaneously, the system evicts tokens with the lowest attention scores from recent computations, optimizing KVCache memory usage dynamically.

4.2 Relevant Concepts

Our work focuses on three critical cache components in modern LLMs:

- **Key Cache (K):** Key vectors encode the structural relationships among tokens in a sequence.

- **Value Cache (V):** Value vectors containing the actual content representations aggregated through attention weights. These preserve the contextual information of each token.

- **Embedding Cache (E):** Caches static word embeddings capturing fundamental semantic and syntactic relationships (Mikolov et al., 2013), providing the foundational token representations before transformer processing.

4.2.1 Fuzzy Token Match

4.3 KVCache Sharing Challenge

The primary challenge in cross-prompt KVCache sharing stems from length disparity between prompts. Inspired by (Liu et al., 2024b), we incorporates positional encoding within the E Cache to enable accurate token alignment while preserving contextual relationships.

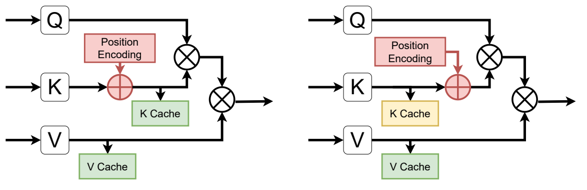
Specifically, we use LSH to identify, for each token in the target prompt, the most similar token in the reference prompt based on their vector representations. We use Locality-Sensitive Hashing (LSH) for efficient token similarity search. Additional details on LSH are provided in Appendix A.3. This process allowed us to reorder the KVCache of the reference prompt to align with the token sequence of the target prompt. Consequently, the reordered KVCache matches the target prompt's length, with its key-value pairs entirely derived from the original KVCache of the reference prompt. The reordered KVCache is then input into the LLM, enabling the transfer of cached values to the target prompt.

Use Relative Position Encoding to Facilitate Fuzzy Token Match

A fundamental limitation of naive fuzzy matching using the E cache arises from the absence of positional context in its representation. Since raw vector embeddings lack inherent positional information, LSH fails to maintain crucial sequential relationships when identifying reference-target token correspondences. This positional agnosticism in the E cache consequently produces semantically inferior mapping results.

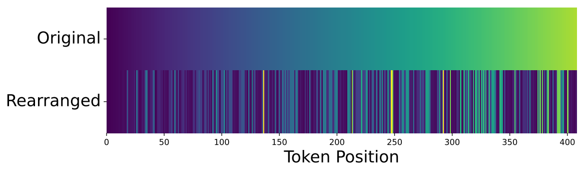
To address this, we introduced position encoding into E cache to improve fuzzy token match performance. Two commonly used position encoding mechanisms are absolute position encoding (Vaswani et al., 2017), which directly embeds position information into the cache, and relative position encoding (Su et al., 2024), which captures positional relationships between tokens. In this work, we incorporate RoPE in the E cache, we evaluate the impact of incorporating positional information into the E cache. Specifically, we compare the perplexity of a plain E cache, paired with the default precomputed KVCache from the reference prompt, with that of an E cache encoded with RoPE. The results, shown in Figure 8, indicate using RoPE in the E cache reduces perplexity, demonstrating its effectiveness in preserving positional context.

These findings support our claim that encoding the E cache with RoPE helps preserve meaningful semantics when using LSH for token matching. By maintaining positional relationships, RoPE improves alignment accuracy, leading to better token retrieval and overall performance, which is further illustrated in Figure 7. Figure 7a compares the token positions in the original E cache with the rearranged positions after passing through the fuzzy token matching block, while Figure 7b presents the results when using an E cache with positional encoding. Notably, the first 20 tokens remain in their original positions, as they represent query tokens. This demonstrates that LSH correctly identifies and preserves query positions.

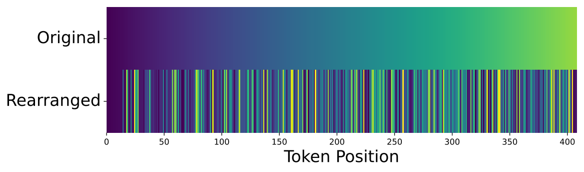


(a) Standard KVCache Store (b) Modified KVCache Store

Figure 6: Comparison of default and modified KV cache storage mechanisms.



(a) Only E Cache



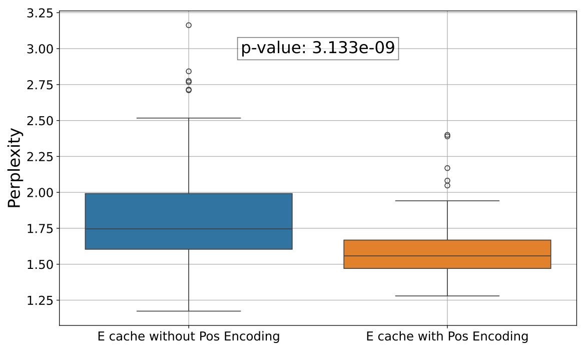
(b) E Cache with Relative Position Encoding

Figure 7: Comparison of fuzzy token matching with and without position encoding.

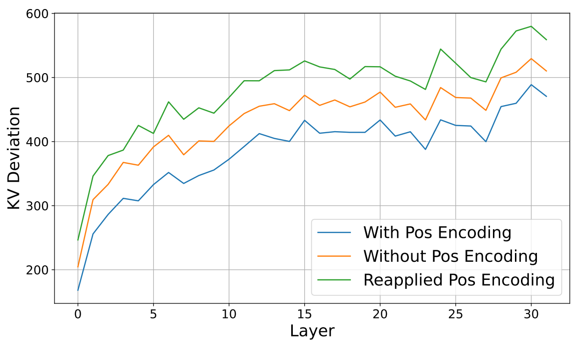
Beyond the initial tokens, a key difference emerges: without positional encoding, many tokens in the target prompt map to the initial tokens, whereas with positional encoding, they align more accurately with later tokens. We interpret this as a manifestation of "attention sink"—a phenomenon in self-attention mechanisms where a significant portion of attention scores is consistently assigned to the initial tokens, regardless of their actual relevance to the task (Xiao et al., 2023; Fei et al., 2025). Incorporating positional encoding into the E cache effectively mitigates this issue, leading to more accurate token matching and improved performance.

4.3.1 Impact from Position Encoding in KVCache

The second challenge in KVCache mapping from a reference prompt to target prompts is the impact of position encoding. Rearranging tokens based



(a) Comparison of PPL With and Without Position Encoding



(b) KV deviation with three position encoding strategies.

Figure 8: Impact of position encoding on E cache and KV cache deviation.

on LSH search alters their order in the precomputed KVCache, which in turn affects the position information encoded in the KV matrices during the prefill phase. The impact of different position encoding mechanisms is discussed in previous studies (Gao et al., 2024), position encoding could be easily removed from KVCache by moving the emcoding function after storing KVCache, which is shown in Figure 6.

Building on our previous discussion on the role of RoPE in E, we compare three configurations: (i) KVCache with position encoding, which follows the standard approach in LLMs; (ii) KVCache without position encoding, where the KVCache is stored before applying position encoding; and (iii) KVCache without position encoding but with reapplied position encoding after reordering, which is similar to (ii) but with RoPE reapplied to the KVCache after rearrangement. Ideally, the KVCache rearranged using LSH should closely match the ground-truth KVCache of the target prompt. To quantify this deviation, we compute the L_2 norm between the rearranged and ground-truth caches, with results shown in Figure 8. The results indicate that KVCache with position encoding exhibits the lowest deviation, followed by KVCache without position encoding, while KVCache with reapplied RoPE has the highest deviation. This highlights the importance of aligning position information between E and KV—only when they share the same position encoding can the LLM fully utilize them, leading to the best generation quality.

4.3.2 Recomputation Strategy

We divide the layers into two groups: the first and subsequent layers. Given that LLMs tend to focus more on later tokens (Liu et al., 2024a; Yang et al., 2024a), we categorize tokens into Cold (c) and Hot (h) using a dynamic ratio $\boldsymbol{r_{dynamic}}$ from Attention Recovery with a threshold of 55%, meaning $\boldsymbol{r_{dynamic}}\%$ of tokens with the highest cumulative attention are selected as Hot, and the rest as Cold. The total number of recomputed tokens is defined as Recomputed $= \omega_c \cdot c + \omega_h \cdot h$, where $\boldsymbol{\omega_c} = 0.1$ and $\boldsymbol{\omega_h} = 0.5$ in the SemShareKV setting.

Starting from the second layer, token selection follows this rule: based on *Insight 1*, the tokens selected in the next layer are derived from those chosen in the previous layer based on a recompute ratio $\boldsymbol{\alpha_{recomp}}\%$ of this layer. Based on *Insight 2*, $\boldsymbol{\alpha_{recomp}}\%$ in shallow layers will be relatively small while in deeper layers will be relatively large. Therefore, the total number of tokens being recomputed on layer i is represented as

$$\text{Recomp}[i] = \text{T} \prod_{j=1}^{i} \alpha_{\text{recomp}}[j] \tag{2}$$

Where T denotes the total number of tokens, $\boldsymbol{i}$ represents the layer index.

4.3.3 Retention Strategy

Similar to token recomputation, we categorize the layers into two groups: the first and the subsequent layers. On the first layer, the retention ratio is determined also by $\boldsymbol{r_{dynamic}}$, follows $Retained = \max(0.8, \boldsymbol{r_{dynamic}})$. And retained tokens are selected based on average attention scores across the last $(1 - \boldsymbol{r_{dynamic}})\%$ tokens, and only retain the top $\boldsymbol{r_{dynamic}}\%$ tokens with highest avg attention scores. In detail, the intuition behind selecting retained tokens is as follows: In the first layer, all hot tokens will be retained. Token eviction occurs only among Cold tokens that are not marked as recomputed. The underlying principle is that recomputed tokens provide better

Method	SAMSum	MultiNews	PubMed	BookSum	BigPatent	% from Best (Avg ↓)
Fully Recompute	18.79	22.10	**24.66**	22.44	25.47	–4.73%
SemShareKV	**21.22**	**23.15**	24.30	22.50	**26.62**	**–0.91%**
SnapKV	20.16	23.07	24.58	**23.22**	25.78	–1.77%
H2O	20.50	23.04	23.53	22.77	24.99	-3.30%

Table 1: Benchmarking SemShareKV with ROUGE-L score (%). The rightmost column reports the average percentage gap to the best-performing method per dataset.

representations of the target prompt. If these tokens are evicted, the computational resources and time spent on recomputing them will be wasted. In subsequent layers, based on *Insight 3*, we should retain fewer tokens, which is defined as:

$$\text{Retain}[i] = \text{T} \prod_{j=1}^{i} \alpha_{\text{retain}}[j] \quad (3)$$

Where T denotes the total number of tokens, i represents the layer index, and $\alpha_{\text{retain}}[j]$ is the token retention ratio at layer j; the remaining tokens are evicted from the KV cache. Intuitively, α_{retain} is typically larger in shallower layers and smaller in deeper ones.

5 Evaluation and Results

5.1 Experiments Setup

We select datasets to evaluate various types of summarization tasks: **MultiNews** from LongBench (Bai et al., 2023) for summarization across multiple passages; **SAMsum** (Gliwa et al., 2019) for understanding multi-turn conversations; and **BookSum**, **PubMed**, and **BigPatent** from M4LE (Kwan et al., 2023) to assess comprehension of narrative stories, scientific medical papers and patent documents, respectively. Further details on data preparation are provided in Appendix B.

We compared SemShareKV against three baselines: (i) **Fully Recompute**: standard inference using the unmodified model from the Transformers library, where the entire prompt is input without any KVCache reuse; (ii) **SnapKV** (Li et al., 2024): a KVCache management method that accelerates the prefill phase by efficient caching but does not compress the KVCache; (iii) **H2O** (Zhang et al., 2023): a dynamic KVCache eviction strategy that compresses KV memory by prioritizing important tokens, but does not optimize the prefill phase. We employ a modified version of H2O that compresses 10% of the cache at each layer. We selected SnapKV and H2O as baselines to separately represent optimization for the prefill phase and KVCache compression. All experiments were conducted on a single A100 GPU. The details of the implementation are discussed in the Appendix D.

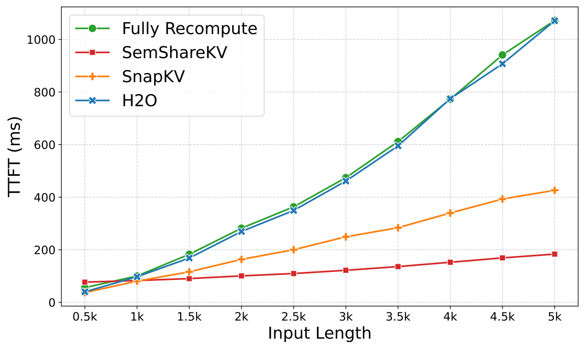
To the best of our knowledge, no existing dataset benchmarks LLMs on KVCache sharing across semantically similar prompts. To bridge this gap, we constructed evaluation samples by randomly selecting portions of entries from existing datasets and rewriting them using the Llama3 model. Then, these rewritten samples were manually verified to ensure that they remained semantically close to the originals. For each dataset, we curated 100 such samples. More details on the data preparation are provided in Appendix B.

5.2 Benchmarking Evaluation

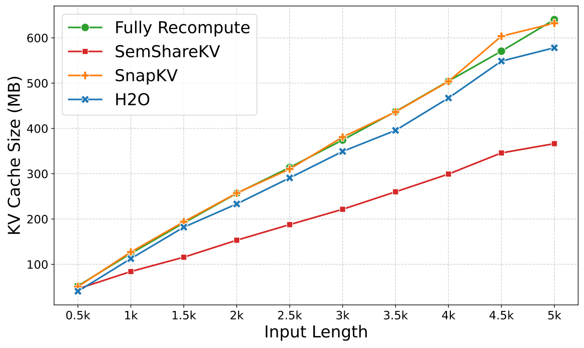
We argue that using Fuzzy Token Match introduces only a negligible overhead to model inference. Table 1 reports the ROUGE-L scores (Lin, 2004). Benchmarking results show that SemShareKV achieves performance comparable to or better than other baseline methods. Notably, in 4 out of the 5 evaluated datasets, *Fully Recompute* fails to attain the highest performance scores. We attribute this phenomenon to the token eviction mechanisms employed by SemShareKV, SnapKV, and H2O. By selectively retaining only the most semantically significant tokens for self-attention computation, these methods effectively reduce redundant information in the semantic representation, thereby enhancing the model's generation quality.

5.3 Efficiency Evaluation

We evaluate SemShareKV based on Time To First Token (TTFT) and KV Cache GPU KV memory usage, benchmarking it against Fully Recompute, SnapKV, and the unmodified H2O model. Figure 9 demonstrates the efficiency advantages of SemShareKV on the MultiNews dataset, showing consistent improvements over baseline methods: SemShareKV achieves **6.25**× faster Time-To-First-Token (TTFT) than Fully Recompute and H2O, **2.23**× faster TTFT than SnapKV, while reducing memory usage by **42%**. However, as illustrated



(a) TTFT Comparison



(b) KVCache Size Comparison

Figure 9: Efficiency Evaluation Results.

Method	SAMSum(↑)	MultiNews(↑)
SemShareKV	**21.22**	**23.15**
Ablation-Zero	14.63	17.71
Ablation-Random	5.38	12.67

Table 2: Ablation study on ROUGE-L for SemShareKV and its ablations across datasets.

in Figure 9b, SemShareKV offers limited performance improvements for shorter prompts (fewer than 700 tokens), which we attribute to the overhead caused by fuzzy token matching and the rearrangement of tokens from the precomputed cache of the reference prompt. In future work, our goal is to minimize this overhead.

5.4 Ablation Study

We conducted two ablation studies to assess whether the rearranged cache produced by the fuzzy token matching process helps the model better capture semantic information.

In the first ablation, we examined the role of semantically matched tokens by altering the contents of the KV cache. Specifically, we evaluated model performance (measured by ROUGE-L) under two conditions: (i) zeroing out the KV values of the matched tokens after fuzzy matching, and (ii) replacing the matched tokens with random tokens from the precomputed cache. As shown in Table 2, both settings result in significantly lower ROUGE-L scores compared to the full SemShareKV method, confirming that the fuzzy token matching mechanism contributes meaningfully to semantic understanding.

The second ablation investigates the relationship between the KVCache compression ratio and the performance of the model during the token retention phase based on Multinews dataset. As illustrated in Figure 10, we observe that retaining too much of the cache introduces redundant information and degrades performance, while retaining too little leads to information loss. This highlights the importance of striking an effective balance in cache retention to preserve useful semantic context.

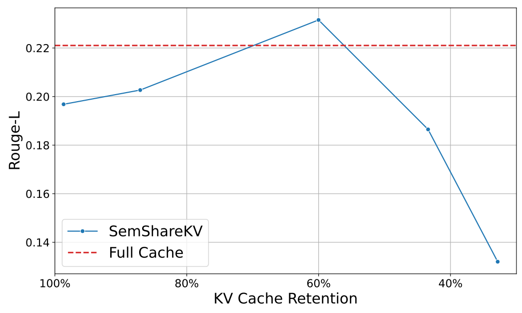


Figure 10: Ablation Study of KV Cache Compression Ratio

6 Conclusion

We proposed SEMSHAREKV, a KVCache sharing framework that enables reuse across semantically similar prompts through fuzzy token matching using locality-sensitive hashing. SemShareKV achieves a speed of 6.25× and saves up to 42% kvcahce memory space compared to conventional KVCache, with a minimum performance drop.

Limitations

While SEMSHAREKV demonstrates effective KV cache sharing for semantically similar prompts, our current evaluation is limited to summarization tasks. It exhibits diminishing speedups for shorter prompts due to computational overhead from fuzzy token matching and introduces several hyper-parameters requiring careful optimization for new domains. These limitations suggest opportunities for future work.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. 2023. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*.

Fu Bang. 2023. Gptcache: An open-source semantic cache for llm applications enabling faster answers and cost savings. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pages 212–218.

Yihua Cheng, Kuntai Du, Jiayi Yao, and Junchen Jiang. 2024. Do large language models need a content delivery network? *arXiv preprint arXiv:2409.13761*.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library.

Weizhi Fei, Xueyan Niu, Guoqing Xie, Yingqing Liu, Bo Bai, and Wei Han. 2025. Efficient prompt compression with evaluator heads for long-context transformer inference. *arXiv preprint arXiv:2501.12959*.

Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. 2024. Cost-efficient large language model serving for multi-turn conversations with cachedattention. In *USENIX Annual Technical Conference*.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. 2024. Prompt cache: Modular attention reuse for low-latency inference. *Proceedings of Machine Learning and Systems*, 6:325–338.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. 2019. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pages 70–79, Hong Kong, China. Association for Computational Linguistics.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. 2024. Attention score is not all you need for token importance indicator in kv cache reduction: Value also matters. *arXiv preprint arXiv:2406.12335*.

Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and 1 others. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Wai-Chung Kwan, Xingshan Zeng, Yufei Wang, Yusen Sun, Liangyou Li, Lifeng Shang, Qun Liu, and Kam-Fai Wong. 2023. M4LE: A Multi-Ability Multi-Range Multi-Task Multi-Domain Long-Context Evaluation Benchmark for Large Language Models.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Akide Liu, Jing Liu, Zizheng Pan, Yefei He, Gholamreza Haffari, and Bohan Zhuang. Minicache: Kv cache compression in depth dimension for large language models, 2024b. *URL https://arxiv.org/abs/2405.14366*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Yuhan Liu, Esha Choukse, Shan Lu, Junchen Jiang, and Madan Musuvathi. 2024b. Droidspeak: Enhancing cross-llm communication. *arXiv preprint arXiv:2411.02820*.

Shi Luohe, Zhang Hongyi, Yao Yao, Li Zuchao, and Zhao Hai. 2024. Keep the cost down: A review on methods to optimize llm's kv-cache consumption. *arXiv preprint arXiv:2407.18003*.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*.

Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2023. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2024. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *arXiv preprint arXiv:2407.00079*.

Zafaryab Rasool, Scott Barnett, David Willie, Stefanus Kurniawan, Sherwin Balugo, Srikanth Thudumu, and Mohamed Abdelrazek. 2024. Llms for test input generation for semantic caches. *arXiv preprint arXiv:2401.08138*.

Sajal Regmi and Chetan Phakami Pun. 2024. Gpt semantic cache: Reducing llm costs and latency via semantic embedding caching. *arXiv preprint arXiv:2411.05276*.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pages 31094–31116. PMLR.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. 2024. Shadowkv: Kv cache in shadows for high-throughput long-context llm inference. *arXiv preprint arXiv:2410.21465*.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. 2025. You only cache once: Decoder-decoder architectures for language models. *Advances in Neural Information Processing Systems*, 37:7339–7361.

MosaicML NLP Team. 2023. Introducing mpt-7b: A new standard for open-source, commercially usable llms. Accessed: 2023-05-05.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Neural Information Processing Systems*.

Zihao Wang, Bin Cui, and Shaoduo Gan. 2024. Squeezeattention: 2d management of kv-cache in llm inference via layer-wise optimal budget. *arXiv preprint arXiv:2404.04793*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, and 3 others. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. 2024. Duoattention: Efficient long-context llm inference with retrieval and streaming heads. *arXiv preprint arXiv:2410.10819*.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Ruiqing Yan, Linghan Zheng, Xingbo Du, Han Zou, Yufeng Guo, and Jianfei Yang. 2024. Recurformer: Not all transformer heads need self-attention. *arXiv preprint arXiv:2410.12850*.

Dongjie Yang, XiaoDong Han, Yan Gao, Yao Hu, Shilin Zhang, and Hai Zhao. 2024a. Pyramidinfer: Pyramid kv cache compression for high-throughput llm inference. *arXiv preprint arXiv:2405.12532*.

Yifei Yang, Zouying Cao, Qiguang Chen, Libo Qin, Dongjie Yang, Hai Zhao, and Zhi Chen. 2024b. Kvsharer: Efficient inference via layer-wise dissimilar kv cache sharing. *arXiv preprint arXiv:2410.18517*.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. 2024. Cacheblend: Fast large language model serving with cached knowledge fusion. *arXiv preprint arXiv:2405.16444*.

Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. 2024. Pqcache: Product quantization-based kv-cache for long context llm inference. *arXiv preprint arXiv:2407.12820*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. *arXiv preprint arXiv:1904.09675*.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, and 1 others. 2023. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710.

Meizhi Zhong, Xikai Liu, Chen Zhang, Yikun Lei, Yan Gao, Yao Hu, Kehai Chen, and Min Zhang. 2024.

Zigzagkv: Dynamic kv cache compression for long-context modeling based on layer uncertainty. *arXiv preprint arXiv:2412.09036*.

A Formula and Inference

A.1 Rotary Position Encoding

In our methodology, we introduced the application of **Rotary Position Embedding (RoPE)** to the E cache, which improves the performance of fuzzy token matching. RoPE is designed to incorporate positional information directly into embeddings, allowing for improved alignment between tokens in a sequence. This is particularly important in natural language processing tasks where the order of words can significantly impact the meaning and context.

The formula of RoPE in a 2-D case is shown below:

$$\text{RoPE}(\mathbf{x}) = \begin{bmatrix} \cos(\theta_k) & -\sin(\theta_k) \\ \sin(\theta_k) & \cos(\theta_k) \end{bmatrix} \begin{bmatrix} x_{2k} \\ x_{2k+1} \end{bmatrix} \quad (1)$$

In this equation, $\theta_k = 10000^{-2k/d}$, where d represents the embedding dimension. The use of RoPE allows for the effective encoding of relative positional information, enabling the model to better capture the relationships between tokens in a sequence. Integrating RoPE into the E cache facilitates the identification of semantically similar tokens using LSH, leading to more accurate and efficient fuzzy token matching. This enhancement helps the model perform more accurately on tasks that require strong semantic understanding.

A.2 Key-Value Deviation

We define Key-Value Deviation with L2 norm as below:

$$\begin{aligned} \sigma_K &= \|K^{\text{reused}} - K^{\text{recomputed}}\|_2, \\ \sigma_V &= \|V^{\text{reused}} - V^{\text{recomputed}}\|_2, \\ \sigma_{KV} &= \sigma_K + \sigma_V \end{aligned} \quad (4)$$

Where K^{reused} and V^{reused} represent the Key and Value matrices in cache reused from the semantic similar prompt; $K^{\text{recomputed}}$ and $V^{\text{recomputed}}$ refer to the Key and Value matrices recomputed at the current layer.

A.3 Locality-Sensitive Hashing (LSH)

Locality-Sensitive Hashing (LSH) is a technique that enables efficient approximate nearest neighbor searches in high-dimensional spaces by ensuring similar input items are hashed into the same bucket with high probability (Indyk and Motwani, 1998). This reduces the number of distance computations required, making LSH particularly useful for large datasets in applications such as image retrieval and natural language processing (Datar et al., 2004). In LSH for Euclidean distance, a common hash function is:

$$h(x) = \lfloor \frac{x \cdot r + b}{w} \rfloor$$

where r is a random vector, b is a random offset, and w is the hash width. This overview encapsulates the theory and practical application of LSH in our framework.

The LSH (Locality-Sensitive Hashing) in SemShareKV is implemented using the FAISS Python library (Douze et al., 2024). Further configuration details can be found in the provided code repository.

B Data Preparation

We categorize these five datasets into two groups based on how semantically similar samples are prepared, all datasets are in English, with a focus on English-language text:

1. **MultiNews (Bai et al., 2023):** This datasets contain samples composed of multiple independent passages or articles. To generate semantically similar samples, we randomly select one passage or article from each sample and use the Llama 3 model to rewrite it while preserving the original semantics. The rewritten passage is constrained to have a similar length to the original (within a 10% difference in token count). We then replace the original passage with the rewritten one to construct a semantically similar prompt. The position of the rewritten passage naturally varies across samples, appearing at the beginning, middle, or end of the context.

2. **SAMSum (Gliwa et al., 2019), PubMed, BigPatent and BookSum (Kwan et al., 2023):** These datasets consist of semantically continuous text. For each sample, we divide the context into individual sentences and randomly select a contiguous segment of the total sentence count. This segment is rewritten using the Llama 3 model, with the constraint that the token count deviates by less

Metric	SAMSum	MultiNews	PubMed	BookSum	BigPatent
N of Samples	100	100	100	100	100
Rewrite % (Avg)	46.75	54.58	45.64	44.31	28.55
ROUGE-L(%)	50.90	84.71	88.15	78.15	90.03
BERTScore(%)	86.97	95.85	95.48	95.58	96.07
BLEU(%)	24.68	90.40	89.22	81.16	89.29

Table B1: Similarity evaluation on benchmarking datasets using ROUGE-L, BERTScore, and BLEU. All datasets contain 100 semantically similar rewritten samples.

than 10% from the original. The rewritten segment replaces the original to create a semantically similar prompt, with its position varying within the context in a similar manner.

Table B1 presents the results of the similarity evaluation, measured using ROUGE-L (Lin, 2004), BLEU (Papineni et al., 2002), and BERTScore (Zhang et al., 2019). We include both longest common subsequence-based metrics (ROUGE-L), n-gram-based metrics (BLEU) and embedding-based metrics (BERTScore) to provide a comprehensive evaluation of semantic similarity across rewritten datasets.

C Artifact Use and Compliance with Intended Purpose

The datasets used in this study are publicly available and are consistent with their intended use, as specified by the respective sources. In preparing the data, we adhered to ethical guidelines and ensured that the use of these publicly released datasets was for research purposes only.

For the created artifacts, such as the semantically similar samples, we have ensured that the use of these modified datasets remains consistent with the original intended research purpose. The generated data serves the purpose of advancing research in semantic similarity and does not extend beyond the intended scope of the original datasets.

D Implementation and Hyperparameters

SemShareKV is implemented in Python using the `transformers` library (Wolf et al., 2020), with the monkeypatching technique. We use the Locality Sensitive Hashing from FAISS (Douze et al., 2024) library. The code is available at: `https://anonymous.4open.science/r/SemShareKV-B53C`. Details of the key functions and their roles are outlined below:

- **conventional_forward_simp**: A modified version of `MistralAttention.forward` from the `transformers` library, incorporating the SemShareKV mechanism. The hyperparameters used in our experiments are also specified in this function.
- **replace_mistral_forward**: Applies monkey-patching to substitute the original attention forward function in the `transformers` library with our customized SemShareKV implementation.
- **prepare_fuzzy_caches**: Encodes ROPE into E caches and performs fuzzy token matching using locality-sensitive hashing (LSH).

Overall, SemShareKV is built on the transformer architecture and consists of fewer than 300 new lines of code, making it lightweight and easily transferable to other LLMs.