Self-Indexing KVCache: Predicting Sparse Attention from Compressed Keys

Xu Yang[1*], **Jiapeng Zhang**[1*], **Dongyang Zhao**[1], **Guo Chen**[1], **Zhuo Tang**[1, 2†]

[1]College of Computer Science and Electronic Engineering, Hunan University, Changsha, China
[2]Shenzhen Research Institute, Hunan University, Shenzhen, China
yangxv, zhangjp, ztang@hnu.edu.cn

Abstract

The KV cache in self-attention has emerged as a major bottleneck in long-context and large-batch inference for LLMs. Existing approaches often treat sparsity prediction and compression as separate modules—relying on auxiliary index structures to select relevant tokens, and on complex quantization schemes to reduce memory usage. This fragmented design introduces redundant overhead and limits scalability.

In this paper, we propose a novel paradigm: treating the compressed key representation not merely as storage, but as a self-indexing structure that directly enables efficient sparse attention. By designing a sign-based 1-bit vector quantization (VQ) scheme, our method unifies compression and retrieval in a single, hardware-friendly format. This approach eliminates the need for external indices or learning-based predictors, offering a lightweight yet robust solution for memory-constrained inference. All components are designed to be hardware-efficient and easy to implement. By implementing custom CUDA kernels, our method integrates seamlessly with FlashAttention, minimizing additional runtime and memory overhead. Experimental results demonstrate that our approach delivers both effectiveness and efficiency.

Code — https://github.com/LfieLike/selfindexingkv

Introduction

Transformer-based large language models (LLMs) (Dubey et al. 2024) have driven significant progress in NLP, achieving breakthroughs in numerous fields (Yang et al. 2023; Liu et al. 2023; Chen et al. 2021). Self-attention (Vaswani et al. 2017), as the core mechanism of Transformer, uses KV cache that consumes large amounts of memory and grows linearly with context length. This makes it a major bottleneck in large-batch inference and long-text inference.

Many efficient KV cache management strategies and optimization techniques have been proposed to ensure that the models can handle long contexts without exceeding resource limits. LLMs have adopted techniques like Group-Query Attention (GQA) (Ainslie et al. 2023), which enables multiple attention heads to share and process the same KV cache, reducing memory by several times. However, despite these advances, efficient KV cache management remains a key challenge in long-context inference scenarios.

To further alleviate the KV cache bottleneck, solutions such as sparsification, quantization, and low-rank factorization have been developed. However, optimizing the KV cache must balance three critical aspects: memory usage, latency, and inference accuracy. Each of the above approaches presents its own trade-offs in this balance. For example, static sparsification (Li et al. 2024; Zhang et al. 2023) can significantly reduce memory usage and computational overhead but leads to noticeable drops in inference accuracy on certain tasks. Dynamic sparsification (Tang et al. 2024; Yang et al. 2025a) introduces non-negligible memory overhead due to the need for constructing token-wise indices that guide top-k selection during decoding. These indices improve relevance but increase memory usage, thereby affecting the overall memory-latency balance. Quantization (Liu et al. 2024c; Hooper et al. 2024; Kang et al. 2024) and Low-rank factorization face inherent trade-offs between latency and memory reduction, and often suffer from format-specific constraints that limit optimization flexibility. In addition, some approaches rely on specialized data layouts or operators that are not readily supported by mainstream hardware, which may limit their practical deployment efficiency.

Additionally, learning-based methods (Yang et al. 2025a; Mazaré et al. 2025) have been proposed, but their effectiveness is often tightly coupled with the distribution of the training data. This strong data dependency limits their generalizability across diverse tasks and scenarios. These intertwined challenges underscore the current bottlenecks in KV cache management for LLM inference and call for more unified and principled solutions. To this end, an effective KV cache optimization method must satisfy three key requirements: **(i) Unbiased**. The optimization algorithm should generalize well across a wide range of downstream tasks. However, learning-based methods (Yang et al. 2025a) rely heavily on auxiliary data, which compromises generalization. **(ii) Hardware-friendly**. Algorithms must be designed with hardware constraints in mind. An improperly designed complex algorithm can incur overheads that outweigh its intended benefits. **(iii) Low Overhead**. Beyond theoretical efficiency, practical deployment requires minimal additional

[*]These authors contributed equally.
[†]Zhuo Tang is the corresponding author.

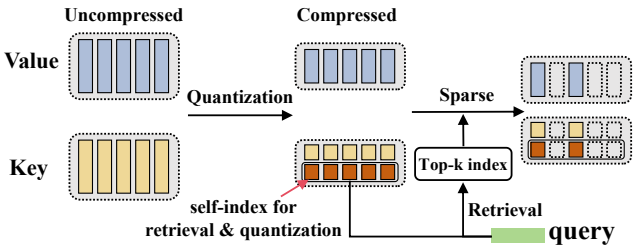

Figure 1: The core idea of our method: Find a **hardware-friendly** compressed representation for the key. This compressed data is designed to split into two parts, one of which enables fast retrieval with accuracy comparable to the full-precision key.

memory and compute overhead. However, many methods reduce one type of cost but increase others, and combining such techniques often leads to accumulated or conflicting overheads that negate the expected benefits.

In summary, these observations prompt a natural question: **Is it possible to develop a practical joint optimization strategy that preserves the complementary advantages of multiple techniques while minimizing redundant overhead**? To this end, as illustrated in Figure 1, we design a unified optimization paradigm that integrates compression and sparsity in a mutually compatible manner. Unlike prior work (Yang et al. 2025b), which decouples compression and retrieval, our method co-designs both components and uses a single compact format that supports top-k selection directly in the compressed domain, thereby fully leveraging the existing information while avoiding redundant indexing or metadata overhead.

To instantiate this unified paradigm, we propose the Self-Indexing KVCache, a novel method that jointly optimizes dynamic sparsity and quantization. The key insight is that sparsity prediction in attention is essentially a top-k retrieval problem based on cosine similarity. To this end, we introduce a 1-bit vector quantization (VQ)-based retrieval algorithm, which allows fast and accurate token selection directly in the compressed domain. This retrieval process is accelerated by a custom CUDA kernel, ensuring low latency. Meanwhile, our compression format is designed to support token-wise random access, making it fully compatible with attention acceleration frameworks such as FlashAttention (Dao et al. 2022). Crucially, the same 1-bit VQ indices used for retrieval also serve as the quantization representation, enabling efficient reuse and significantly reducing quantization error. By integrating sparsity prediction and quantization into a unified design, Self-Indexing KVCache achieves memory savings, speed improvements, and precision preservation simultaneously. It significantly outperforms the trade-offs of existing methods that optimize only a single dimension, thus realizing an all-round enhancement in both efficiency and effectiveness. Our contributions are summarized as follows:

- We propose a novel KV cache optimization paradigm that directly leverages compressed key cache for retrieval of the most valuable KV caches, thereby avoiding the indexing overhead inherent in dynamic sparsification.
- We design a novel one-pass sign-bit-based VQ clustering strategy for retrieval index construction, enabling fast and expressive codebooks without iterative optimization.
- Our method leverages custom-designed LUT-GEMV and sparse FlashAttention CUDA kernels for hardware-friendly optimization, minimizing computational overhead and memory traffic, achieving up to a 5× reduction in KV cache memory, 6.7× acceleration in sparse attention computation, and 2× speedup in end-to-end inference latency over FlashAttention v2.

Background and Preliminaries

Self-attention and KV Cache. Self-attention is the core component of Transformer-based LLMs, formally expressed as $softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$. During inference, the attention module undergoes two distinct stages: the prefill stage and the decode stage. Taking a single attention head as an example, in the prefill stage, the dimensions of Q, K, and V are $L \times D$, where L is the context token length. As the context gets long, this stage is usually compute-intensive and fully utilizes GPU computational power. The K and V tensors are then cached as KV cache for computations in the decode stage. In the decode stage, the dimension of each Q becomes $1 \times D$. Since it requires storing and processing large context windows of tokens while generating predictions sequentially, this stage is memory-bound. The computational overhead can account for over 80% of the total generation time. Therefore, optimizing the KV cache's storage and computational overhead during the decode stage is a critical research and engineering challenge.

KV Cache Compression and Sparsity. Quantization is a standard technique for reducing KV cache memory, with KIVI (Liu et al. 2024c) achieving 2-bit compression while preserving accuracy. More aggressive methods (Zandieh, Daliri, and Han 2024; Zhang et al. 2024b; Kang et al. 2024; Xiao et al. 2024b; Liu et al. 2024b) achieve higher ratios but introduce non-trivial overhead. In parallel, sparsity-based approaches reduce compute by selecting a token subset. Static sparsity (Han et al. 2024; Xiao et al. 2024c; Li et al. 2024) often suffers from degraded accuracy, while dynamic methods (Tang et al. 2024; Liu et al. 2024a; Xiao et al. 2024a) improve adaptivity via top-k selection but rely on additional indexing structures. Recent work explores tighter integration between compression and retrieval. Lserve (Yang et al. 2025b) integrates sparse attention with low-bit KV cache compression, but treats them as separate components. Without joint optimization, this design introduces redundant overhead during inference. In contrast, our method directly reuses 1-bit quantized sign codes both for retrieval and reconstruction, eliminating redundancy between indexing and compression. This enables compressed-domain top-k selection in a unified and hardware-efficient format, avoiding auxiliary predictors or metadata.

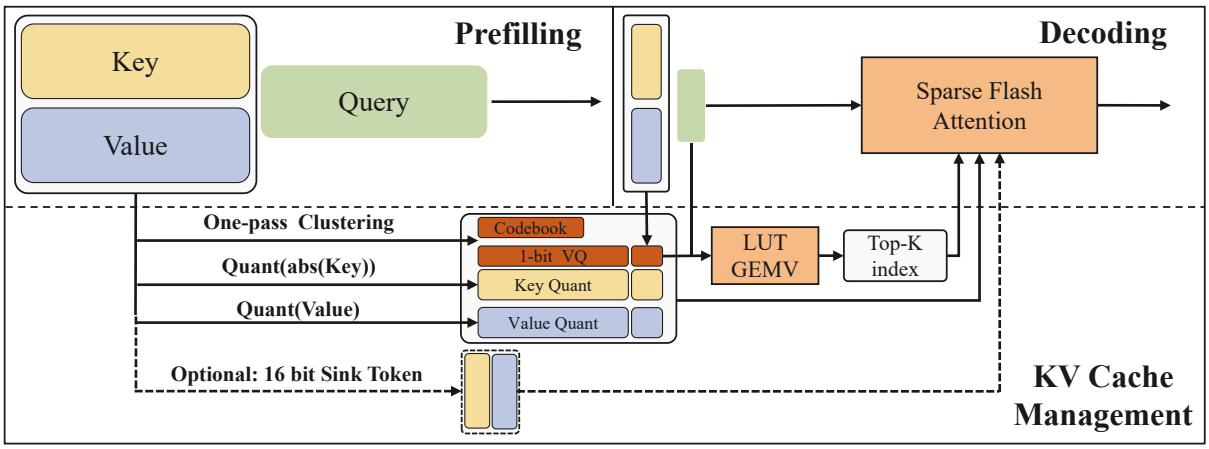

Figure 2: Overview of Self-Indexing KVCache. By leveraging custom kernels, we significantly reduce the runtime overhead typically associated with retrieval and dequantization. Under low-bit quantization, we optionally preserve a fixed number of full-precision sink tokens during the prefill stage to improve inference robustness, **without sacrificing efficiency**.

Implementation

In this section, we focus on the overall algorithmic workflow of our proposed Self-Indexing KVCache. Figure 2 illustrates the Self-Indexing KVCache workflow. During the prefill stage, we perform one-pass sign-based vector quantization to construct a lightweight codebook and quantize the key and value tensors. This design avoids iterative clustering. As a result, it introduces minimal overhead and is well-suited for latency-sensitive applications. Optionally, 64 sink tokens can be retained in full precision. These tokens, along with the sink tokens, participate in the attention computation. By fusing dequantization into the flash attention kernel through a custom implementation, we significantly reduce overhead. Implementation details, including kernel-level optimizations, are deferred to the source code.

One-Pass Sign-Based Clustering

Sparsity prediction in attention can be formulated as a top-k retrieval problem based on cosine similarity. To accelerate retrieval, VQ is commonly used to compress key vectors (Zhang et al. 2024a; Liu et al. 2024a). A key component of VQ is the construction of a codebook that clusters high-dimensional vectors into representative centroids. Traditional codebook construction methods, such as K-means clustering, typically involve multiple iterations and introduce non-trivial computational overhead. While effective in some offline settings, their iterative nature can introduce considerable overhead during the prefill stage of LLM inference, where efficiency is critical.

To address this, we propose a one-pass, sign-based clustering method for efficient codebook generation. Instead of iteratively optimizing a reconstruction loss, we group vectors purely on the basis of their sign patterns. **In cosine similarity space, the sign of a vector conveys meaningful information about its direction**. Thus, clustering by sign patterns preserves the angular structure while drastically reducing computational overhead. Experimental results demonstrate that this simple yet effective strategy enables the construction of a lightweight and expressive codebook in a single pass, making it well-suited for compressed-domain sparse attention under stringent inference-time constraints.

Codebook Construction. We begin by constructing a lightweight codebook based on sign patterns. We partition the key cache $K \in \mathbb{R}^{L\times D}$ along the last dimension into $G = D/4$ groups, each consisting of 4-dimensional subvectors:

$$K = [K^{(1)}, K^{(2)}, \ldots, K^{(G)}], \quad K^{(g)} \in \mathbb{R}^{L\times 4}. \tag{1}$$

Each subvector $k \in \mathbb{R}^4$ is encoded using its sign pattern:

$$s = \text{sign}(k) \in \{-1, +1\}^4. \tag{2}$$

We define a deterministic mapping function $\text{Code}(k) : \mathbb{R}^4 \to \{0, \ldots, 15\}$ that converts the sign pattern s into a 4-bit binary index by mapping $+1 \to 1$ and $-1 \to 0$. The resulting integer code is computed as:

$$\text{Code}(k) = \sum_{i=1}^{4} \left(\frac{1+s_i}{2}\right) \cdot 2^{4-i}. \tag{3}$$

This assigns each subvector to one of the 16 sign-defined clusters. For each cluster, we compute a centroid by averaging the vectors assigned to it:

$$c_j = \frac{1}{|C_j|} \sum_{k\in C_j} k, \tag{4}$$

where C_j denotes the set of subvectors sharing the same sign pattern. This process yields a compact codebook of size 16 for each group. The choice of 16 is a hardware-aware optimization: **a larger codebook would exceed the limited capacity of on-chip shared memory in modern accelerators, leading to higher latency and lower throughput.** The entire procedure requires only a single pass and avoids costly iterative methods such as K-means (Zhang et al. 2024a), resulting in minimal overhead while preserving the directional information essential for cosine-based sparsity prediction.

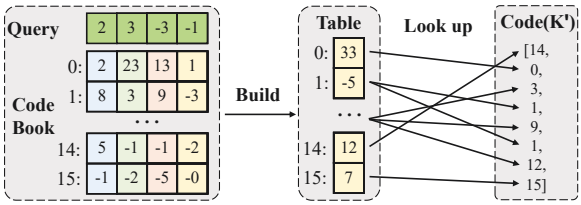

Figure 3: Overview of the LUT-GEMV. We compute the dot product between the query and each codeword in the codebook to generate a lookup table of size 16.

Entropy-Aware Normalization. While sign-based clustering enables efficient 1-bit VQ, applying it directly on raw key cache often results in unbalanced sign distributions, limiting the expressiveness of the resulting codes. To address this, we propose an entropy-aware normalization scheme that enhances the capacity of 1-bit representations from an information-theoretic perspective (Shannon 1948).

Specifically, we apply channel-wise mean normalization to the original key cache $K \in \mathbb{R}^{L\times D}$, subtracting the mean of each channel across all tokens:

$$K'_{i,d} = K_{i,d} - \mu_d, \quad \mu_d = \frac{1}{L}\sum_{i=1}^{L} K_{i,d}. \tag{5}$$

This transformation ensures that each dimension has zero mean, which leads to a more balanced distribution of positive and negative signs.

From an information theoretic viewpoint, a binary variable $B = \text{sign}(U) \in \{-1, +1\}$ achieves its maximum entropy when the probabilities of both outcomes are equal:

$$P(B = +1) = P(B = -1) = \frac{1}{2}, \quad \Rightarrow \mathcal{H}(B) = \log 2. \tag{6}$$

Therefore, normalization increases the entropy of the sign patterns, improving the effective capacity of the 1-bit codes.

Importantly, this preprocessing does not affect the output of the attention mechanism, as the softmax function is invariant to additive shifts:

$$\text{softmax}(x - \mu) = \text{softmax}(x), \quad \forall \mu \in \mathbb{R}. \tag{7}$$

Thus, our normalization improves quantization robustness without altering attention semantics.

Compressed-Domain Top-k Retrieval

To enable efficient sparse attention, we perform similarity search between queries and the compressed key cache entirely in the quantized domain. As illustrated in Figure 3, this process avoids full-precision computation and instead relies on fast table lookups.

Each query vector is first split into 4-dimensional subvectors, consistent with the group-wise structure used during key quantization. Each subvector is performed dot-produced with the 16 centroids in the codebook to compute similarity scores. These scores are precomputed and stored in a lookup table, where the index corresponds to one of the $2^4 = 16$ possible sign patterns per group.

The key cache K, already compressed into sign-pattern codes, stores only the index values representing which centroid each key subvector corresponds to. During retrieval, similarity scores between the query and all cached keys are efficiently approximated by performing simple table lookups followed by additions:

$$\text{score}(q, K) = qK^T \approx \sum_{g=1}^{G} \text{Table}^{(g)}[\text{Code}(K')^{(g)}], \tag{8}$$

where $\text{Table}^{(g)}[\cdot]$ denotes the group-g lookup table, and $\text{Code}(K')^{(g)}$ is the index value of key k' in that group. The aggregated scores are then used to select the top-k most relevant keys for sparse attention.

Overall, this LUT-GEMV based method is hardware-efficient, easily parallelizable on GPUs. By replacing full-precision computations with lightweight table lookups and additions, our approach significantly reduces floating-point operations and memory bandwidth usage—two major bottlenecks in large-scale inference. Despite this simplification, it maintains high retrieval accuracy, demonstrating the effectiveness of compressed-domain similarity computation.

Token-Wise Quantization Format

Token-Wise vs. Channel-Wise. In random-access or token-level decoding scenarios, token-wise quantization offers clear efficiency advantages. Channel-wise quantization, as used in prior work (Liu et al. 2024c), stores quantization parameters per channel, requiring all dimensions to be read in order to reconstruct a single token, resulting in high latency and memory bandwidth. This design may be effective in dense settings, but becomes inefficient for sparse access. In contrast, token-wise quantization stores parameters per token, significantly reducing parameter lookup overhead, making it naturally suited for efficient sparse token retrieval.

Quantization Function. Given an input value cache $\mathbf{V}$, we compute the quantization scale qs and zero point zp as:

$$qs = \frac{V_{\max} - V_{\min}}{2^B - 1}, \quad zp = V_{\min}. \tag{9}$$

The quantized values are obtained by:

$$Q(V) = \text{clamp}\left(\text{round}\left(\frac{V - zp}{qs}\right), 0, 2^B - 1\right). \tag{10}$$

Dequantization is then performed as:

$$D(V) = qs \cdot Q(\mathbf{V}) + zp. \tag{11}$$

Given a key cache matrix $\mathbf{K}$, we first take the absolute value and normalize it across each channel (dimension) using the per-channel maximum:

$$\hat{\mathbf{K}} = \frac{|\mathbf{K}|}{\alpha}, \quad \text{where} \quad \alpha_j = \max\left(|\mathbf{K}_{:,j}|\right). \tag{12}$$

As the sign information is already extracted, it is excluded from the subsequent quantization. Only the absolute values are quantized and stored.

Method	Cache Bits (K,V,Index)	SD-QA		MD-QA		Summarization		Few-shot		Synthetic	Code		Avg.
		Qasper	MF-en	HPQA	2WQA	GVRpt	QMSum	TREC	TrivQA	PR-en	Lcc	RB-P	
Llama3.1-8B	16, 16, 0	45.5	53.8	54.7	47.1	34.9	25.3	73.0	91.6	99.5	63.4	56.7	58.7
SnapKV	16, 16, 0	33.9	46.4	54.4	44.7	21.8	22.6	48.0	90.5	78.0	58.0	50.1	49.9
Quest	16, 16, 2	44.9	51.6	55.0	46.9	31.2	24.0	71.0	90.6	95.5	60.6	50.5	56.5
DoubleSparse	16, 16, 2	43.8	51.7	54.4	45.9	31.7	24.0	67.5	91.9	97.0	50.5	44.7	54.8
Ours(16 bits)	16, 16, 1	45.4	53.9	55.4	47.0	34.0	24.8	72.5	91.7	99.5	63.2	55.3	58.4
Ours	2, 2, 1	44.8	54.6	55.7	46.0	33.1	24.7	72.5	92.1	99.5	63.2	54.4	58.2
Qwen2.5-14B	16, 16, 0	46.5	54.0	65.0	64.3	31.9	23.6	81.0	89.0	100.0	33.1	38.1	56.9
SnapKV	16, 16, 0	29.4	47.6	60.1	54.0	19.3	19.8	54.5	86.0	61.0	24.9	31.1	44.3
Quest	16, 16, 2	45.7	52.0	62.6	62.3	28.8	21.3	70.5	88.1	94.5	31.7	35.2	53.9
DoubleSparse	16, 16, 2	44.7	53.1	61.0	62.1	30.3	21.9	79.0	89.0	94.1	27,5	31.8	54.1
Ours(16 bits)	16, 16, 1	46.0	53.5	63.7	62.4	30.5	22.6	80.5	88.5	97.5	32.6	37.4	55.9
Ours	2, 2, 1	45.8	53.5	63.0	63.1	30.5	22.0	80.5	88.5	96.4	32.3	37.4	55.7

Table 1: Performance comparison of LongBench. The best result is highlighted in underline.

The per-channel scaling factors α are also **reused during the decoding stage**. Finally, the dequantized key values are recovered as:

$$D(|K|) = \alpha \cdot (qs \cdot Q(|\hat{K}|) + zp). \tag{13}$$

Experimental results demonstrate that our method enables effective 2-bit token-wise quantization without significant loss in performance.

Full Precision Sink Tokens. We observe that certain tokens are consistently selected and are particularly sensitive to low-bit quantization. To address this, we adopt the SnapKV (Li et al. 2024) in the prefill stage, where we fix the selection of 64 tokens in full precision. These tokens are guaranteed to always participate in the sparse attention computation, mitigating quantization-induced errors. This trick provides robustness without sacrificing efficiency: the overhead is negligible relative to total KV cache size, and the fixed, contiguous layout allows seamless integration into our FlashAttention kernel.

Overhead Analysis

For memory overhead. Take the KV cache of an attention head in LlaMA-3.1-8B with a context length of L, where $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{L\times 128}$. The memory required under our method includes the following components: **(i)** Sign bits: Each key vector requires $L \times 128$ bits for storing sign information. **(ii)** Quantized values and parameters: Both keys and values are 2-bit quantized, requiring $2 \times (2 \times L \times 128) = 512L$ bits. In addition, the quantization is applied per 32 elements, yielding $4L$ groups for each of key and value. Each group stores a 16-bit scale and zero point, leading to another $2 \times 4L \times 2 \times 16 = 256L$ bits. In total, this occupy $768L$ bits. **(iii)** Fixed overhead: The codebook, normalization parameters α and μ, as well as memory for handling outlier tokens, contribute a fixed overhead. As L increases, this overhead becomes negligible. Therefore, our method achieves up to 78% memory savings for the KV cache without degrading inference accuracy. Furthermore, We implement all components with custom CUDA kernels to minimize memory transfers and maximize runtime efficiency.

Evaluation

Experimental Setup

Benchmark and Module. We evaluate our method using LongBench (Bai et al. 2024), which covers six categories: Single/Multi-Document QA, Summarization, Few-shot Learning, Synthetic Tasks, Code Completion. For each category, we select two representative datasets. Additionally, we use the Ruler benchmark (Hsieh et al. 2024) to assess ultra-long context performance. We evaluate our method on two open-source models: Llama3.1-8B-Instruct (Dubey et al. 2024) (128K context with GQA), and Qwen2.5-14B-1M (Yang et al. 2024a) (1M context with aggressive GQA). Through experiments on these models, we demonstrate the robustness of our approach.

Baseline. We consider SnapKV (Li et al. 2024), Quest (Tang et al. 2024), and DoubleSparse (Yang et al. 2024b) as our baselines. We implement all methods to ensure fair comparison: SnapKV serves as a standard one-shot KV cache pruning baseline, known for its simplicity and efficiency. Quest employs dynamic block-wise sparse attention with effective prediction strategies. DoubleSparse offers fine-grained token-wise sparsity with low overhead. Together, these methods span a spectrum from static pruning to dynamic sparsity, forming a comprehensive comparison baseline. Recent methods such as SAAP (Mazaré et al. 2025) adopt fundamentally different designs, relying on auxiliary encoders, separate indices, or heavy offline preprocessing. These methodological gaps make direct and fair comparisons infeasible. Instead, we evaluate against open-sourced, lightweight baselines that align with our inference setting.

Hyperparameter Settings. To ensure a fair comparison, we meticulously align extraneous variables. For Quest, we set the chunk size to 16. For DoubleSparse, we use 16 channels for token selection, which is equivalent to building a 2-bit index per parameter over the key cache. For all methods, sparse attention is applied at every layer, and tokens generated during the decode stage are always included in the attention computation by default. Experiments are conducted on NVIDIA RTX 4090 and A100-40G.

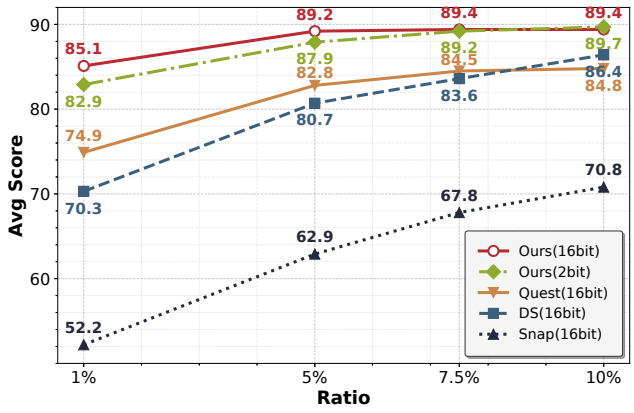

Figure 4: Average task performance under different sparsity ratios on the 32K prompt Ruler Benchmark. Our method outperforms other method, with the 2-bit quantization at 7.5% sparsity.

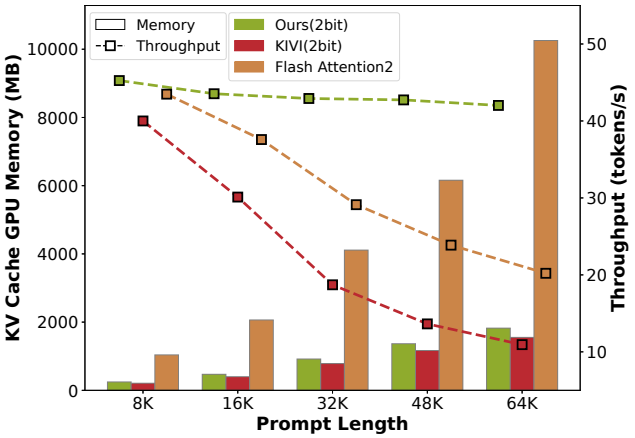

Figure 5: KV cache memory overhead and latency of LlaMA3.1-8B at different prompt lengths. Our method dynamically select 7.5% tokens.

Performance Study

LongBench. In the performance comparison experiment of LongBench with a budget of 160 tokens, our method retains the 64 sink tokens, thus only dynamically select 96 tokens. We compared the performance of various methods on different tasks for different models. As shown in Table 1, under the same settings, our method achieves lower accuracy degradation. Even with 2-bit quantization, the accuracy loss remains minimal, thanks to the use of sign-bit assistance.

RULER. In long-context experiments, we retain a proportion of tokens for attention computation rather than a fixed number. As shown in Figure 4, our method achieves optimal accuracy even with only 7.5% of tokens preserved. Table 2 presents results across all tasks under this 7.5% sparsity setting, focusing on 32K-token prompts. We choose this length as both models show noticeable degradation beyond 32K, making comparisons less meaningful. Even under such extreme sparsity, our method consistently outperforms all baselines. Notably, it shows more stable performance on reasoning tasks like FWE and CWE, suggesting that our dynamic retrieval better captures critical information.

Efficiency Study

End to End Efficiency. We implemented the Self-index Cache based on the Transformers framework (Wolf et al. 2019) and conducted end-to-end efficiency comparisons. To ensure fairness and strong baseline performance, we compared against full cache, Flash Attention2, and KIVI 2-bit quantization, all using the official implementations from the Transformers library. We adopt Time To 2nd Token (TT2T), as the evaluation metric in the prefill stage, and report KV cache memory footprint and throughput as metrics in the decode stage. Notably, our method retains only 7.5% of the tokens in the cache.

Table 3 presents the TT2T evaluation results, from which we draw two key conclusions: **(i)** Compared to Flash Attention2, our method introduces only 5% additional overhead, indicating that both the quantization process and codebook construction are highly efficient and lightweight. **(ii)** Compared to KIVI, our method supports significantly longer context lengths, attributed to our custom sparse Flash Attention kernel.We fuse dequantization and sparse memory access into a single compute pass, minimizing memory traffic and maximizing execution efficiency. These results highlight not only the scalability of our approach under longer sequences, but also the importance of low-level kernel optimization in achieving high-performance sparse attention.

Figure 5 illustrates the performance of the three methods during the decode stage, leading to the following key observations: Storage Efficiency: Our method achieves nearly 5× reduction in KV cache memory footprint, matching the compression ratio of 2-bit KIVI while maintaining retrieval compatibility. Throughput Advantage: Thanks to aggressive sparsification and a fused kernel design, our approach delivers up to 2× higher decoding throughput compared to full-cache FlashAttention2. In contrast, KIVI suffers from degraded performance due to its naive decompress-then-compute strategy, underscoring the efficiency of our integrated implementation.

Head to Head Efficiency. Table 4 demonstrates that our method achieves strong efficiency across all major components. **(i)** The proposed one-pass sign-based clustering achieves over 20× speedup compared to traditional KMeans while preserving sufficient representational quality for downstream retrieval. Here, we follow prior work on KV cache clustering, which typically adopts 20–50 KMeans iterations for convergence (Zhang et al. 2024a). **(ii)** Our retrieval procedure, which includes table construction and LUT-GEMV computation, provides more than 4× speedup over full dot-product attention and outperforms Quest under identical settings. **(iii)** In the attention stage, our custom sparse FlashAttention kernel achieves a 6–7× speedup relative to full FlashAttention2, while maintaining comparable efficiency to Page Attention, which is commonly used in prior sparse attention methods such as Quest to accelerate block-level computation.

Method	Cache Bits (K,V,Index)	Task													Avg.
		NS1	NS2	NS3	NM1	NM2	NM3	NV	NQ	VT	CWE	FWE	QA1	QA2	
Llama3.1-8B	16,16,0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	98.4	59.2	94.67	72.0	56.0	90.8
SnapKV	16,16,0	100.0	100.0	28.0	100.0	32.0	8.0	100.0	100.0	96.8	43.6	90.0	60.0	56.0	70.8
Quest	16,16,2	100.0	100.0	100.0	100.0	100.0	56.0	98.0	100.0	97.6	31.6	90.7	68.0	56.0	84.5
DoubleSparse	16,16,2	100.0	100.0	100.0	100.0	68.0	68.0	100.0	100.0	97.6	34.8	90.6	64.0	64.0	83.6
Ours(16 bits)	16,16,1	100.0	100.0	100.0	100.0	100.0	100.0	98.0	100.0	98.4	53.2	89.3	68.0	60.0	89.4
Ours	2,2,1	100.0	100.0	100.0	100.0	100.0	96.0	100.0	100.0	96.8	49.2	85.3	68.0	64.0	89.2
Qwen2.5-14B	16,16,0	100.0	100.0	100.0	100.0	100.0	100.0	99.0	100.0	100.0	90.8	94.7	80.0	72.0	95.1
SnapKV	16,16,0	100.0	100.0	52.0	100.0	0.0	4.0	94.0	100.0	100.0	75.0	94.67	68.0	60.0	72.9
Quest	16,16,2	100.0	100.0	100.0	100.0	100.0	96.0	99.0	100.0	100.0	76.4	90.67	80.0	76.0	93.7
DoubleSparse	16,16,2	100.0	100.0	92.0	100.0	88.0	100.0	100.0	98.0	100.0	89.6	93.33	80.0	72.0	93.3
Ours(16 bits)	16,16,1	100.0	100.0	100.0	100.0	100.0	100.0	99.0	100.0	100.0	93.6	94.6	80.0	68.0	95.0
Ours	2,2,1	100.0	100.0	100.0	100.0	100.0	100.0	98.0	99.0	100.0	83.6	92.0	80.0	68.0	93.9

Table 2: Detail experimental results of Llama3.1-8B on the 32K prompt Ruler Benchmark.

Prompt Length	**Ours**	**KIVI**	**Flash Attention2**
8K	1.26	1.23	1.18
16K	2.75	2.66	2.62
32K	6.51	6.32	6.18
48K	11.12	10.88	OOM
64K	17.70	OOM	OOM

Table 3: Comparison of Time TT2T (seconds) across different methods and prompt lengths.

Module	Method	Time (ms)
Clustering	Ours	88.85
	KMeans (20 iterations)	2113.9
Retrieval	Ours	0.039
	Quest (page size = 16)	0.041
	Full $K \cdot q^\top$	0.166
Attention	Ours (7.5%)	0.116
	Page Attention (7.5%)	0.101
	Flash Attention2 (Full)	0.776

Table 4: Performance comparison across different modules under a 16K token input with batch size 10.

These results highlight the effectiveness of our design in minimizing latency at every stage of the inference pipeline.

Ablation Study

The results are shown in Table 5. To assess the contributions of key components in our design, we perform ablation experiments on sign-bit quantization, retrieval mechanism, and the use of sink tokens. Removing the sign bit during quantization (w/o sign in quant) consistently reduces accuracy, confirming that sign information carries essential directional cues that mitigate low-bit quantization error. The sign-only retrieval variant, which computes similarity using only the sign bit without magnitude-based VQ, results in even larger degradation. This indicates that the magnitude component plays a critical role in accurate token selection. Removing sink tokens has minor impact on most datasets but slightly reduces robustness in low-redundancy settings like GovRpt. Overall, these results highlight the dual role of 1-bit VQ and the complementary effect of sink tokens in inference quality under aggressive compression.

Setting	MF-en	HPQA	GovRpt	RB-P
Ours	**54.6**	**55.7**	**33.1**	**54.4**
w/o sign in quant	52.5	54.9	32.2	50.9
sign-only retrieval	52.8	53.2	31.9	52.1
w/o sink tokens	52.6	55.2	32.5	54.2

Table 5: Ablation study on the effect of components.

Conclusion

In this work, we presented Self-Indexing KVCache, a practical and hardware-friendly method that unifies compression and sparse retrieval for attention in large language models. By using the compressed representation itself as a functional index, our design directly supports top-k token selection in the attention computation process. Leveraging sign-bit-based 1-bit vector quantization, entropy-aware normalization, and a custom LUT-GEMV kernel, our method achieves up to 5$\times$ memory reduction and consistently lower latency, with negligible accuracy loss. Experiments demonstrate that Self-Indexing KVCache outperforms strong baselines in both long-context reasoning and end-to-end inference efficiency, all without requiring extra training or auxiliary indexing structures. These findings suggest that compression in LLM systems can be more than a storage optimization; when co-designed with retrieval, it can act as a computation-aware, index-equivalent representation, opening new opportunities for unified and efficient inference in future large-scale models.

Acknowledgments

The work is supported by the National Natural Science Foundation of China (Grant Nos. 62225205, 62302160, 62222204), the Natural Science Foundation of Hunan Province (Grant Nos. 2024JJ6154), the Science and Technology Program of Changsha (kh2301011), the Major Science and Technology Research Projects of Hunan Province (Grant Nos. 2024QK2010, 2024QK2009), the Yunnan Provincial Major Science and Technology Special Plan Projects (No.202502AD080009), the Shenzhen Basic Research Project (Natural Science Foundation) under Grant JCYJ20210324140002006.

References

Ainslie, J.; Lee-Thorp, J.; de Jong, M.; Zemlyanskiy, Y.; Lebrón, F.; and Sanghai, S. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*.

Bai, Y.; Lv, X.; Zhang, J.; Lyu, H.; Tang, J.; Huang, Z.; Du, Z.; Liu, X.; Zeng, A.; Hou, L.; Dong, Y.; Tang, J.; and Li, J. 2024. LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding. arXiv:2308.14508.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; de Oliveira Pinto, H. P.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; Ray, A.; Puri, R.; Krueger, G.; Petrov, M.; Khlaaf, H.; Sastry, G.; Mishkin, P.; Chan, B.; Gray, S.; Ryder, N.; Pavlov, M.; Power, A.; Kaiser, L.; Bavarian, M.; Winter, C.; Tillet, P.; Such, F. P.; Cummings, D.; Plappert, M.; Chantzis, F.; Barnes, E.; Herbert-Voss, A.; Guss, W. H.; Nichol, A.; Paino, A.; Tezak, N.; Tang, J.; Babuschkin, I.; Balaji, S.; Jain, S.; Saunders, W.; Hesse, C.; Carr, A. N.; Leike, J.; Achiam, J.; Misra, V.; Morikawa, E.; Radford, A.; Knight, M.; Brundage, M.; Murati, M.; Mayer, K.; Welinder, P.; McGrew, B.; Amodei, D.; McCandlish, S.; Sutskever, I.; and Zaremba, W. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374.

Dao, T.; Fu, D.; Ermon, S.; Rudra, A.; and Ré, C. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35: 16344–16359.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Han, C.; Wang, Q.; Peng, H.; Xiong, W.; Chen, Y.; Ji, H.; and Wang, S. 2024. LM-Infinite: Zero-Shot Extreme Length Generalization for Large Language Models. arXiv:2308.16137.

Hooper, C.; Kim, S.; Mohammadzadeh, H.; Mahoney, M. W.; Shao, Y. S.; Keutzer, K.; and Gholami, A. 2024. KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization. arXiv:2401.18079.

Hsieh, C.-P.; Sun, S.; Kriman, S.; Acharya, S.; Rekesh, D.; Jia, F.; Zhang, Y.; and Ginsburg, B. 2024. RULER: What's the Real Context Size of Your Long-Context Language Models? *arXiv preprint arXiv:2404.06654*.

Kang, H.; Zhang, Q.; Kundu, S.; Jeong, G.; Liu, Z.; Krishna, T.; and Zhao, T. 2024. Gear: An efficient kv cache compression recipefor near-lossless generative inference of llm. *arXiv preprint arXiv:2403.05527*.

Li, Y.; Huang, Y.; Yang, B.; Venkitesh, B.; Locatelli, A.; Ye, H.; Cai, T.; Lewis, P.; and Chen, D. 2024. SnapKV: LLM Knows What You are Looking for Before Generation. In Globerson, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J.; and Zhang, C., eds., *Advances in Neural Information Processing Systems*, volume 37, 22947–22970. Curran Associates, Inc.

Liu, G.; Li, C.; Zhao, J.; Zhang, C.; and Guo, M. 2024a. ClusterKV: Manipulating LLM KV Cache in Semantic Space for Recallable Compression. *arXiv preprint arXiv:2412.03213*.

Liu, N. F.; Lin, K.; Hewitt, J.; Paranjape, A.; Bevilacqua, M.; Petroni, F.; and Liang, P. 2023. Lost in the Middle: How Language Models Use Long Contexts. arXiv:2307.03172.

Liu, Y.; Li, H.; Cheng, Y.; Ray, S.; Huang, Y.; Zhang, Q.; Du, K.; Yao, J.; Lu, S.; Ananthanarayanan, G.; et al. 2024b. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, 38–56.

Liu, Z.; Yuan, J.; Jin, H.; Zhong, S.; Xu, Z.; Braverman, V.; Chen, B.; and Hu, X. 2024c. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*.

Mazaré, P.-E.; Szilvasy, G.; Lomeli, M.; Massa, F.; Murray, N.; Jégou, H.; and Douze, M. 2025. Inference-time sparse attention with asymmetric indexing. *arXiv preprint arXiv:2502.08246*.

Shannon, C. E. 1948. A mathematical theory of communication. *The Bell system technical journal*, 27(3): 379–423.

Tang, J.; Zhao, Y.; Zhu, K.; Xiao, G.; Kasikci, B.; and Han, S. 2024. Quest: Query-Aware Sparsity for Efficient Long-Context LLM Inference. *arXiv preprint arXiv:2406.10774*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. arXiv:1706.03762.

Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M.; et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*.

Xiao, C.; Zhang, P.; Han, X.; Xiao, G.; Lin, Y.; Zhang, Z.; Liu, Z.; and Sun, M. 2024a. InfLLM: Training-Free Long-Context Extrapolation for LLMs with an Efficient Context Memory. arXiv:2402.04617.

Xiao, G.; Lin, J.; Seznec, M.; Wu, H.; Demouth, J.; and Han, S. 2024b. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. arXiv:2211.10438.

Xiao, G.; Tian, Y.; Chen, B.; Han, S.; and Lewis, M. 2024c. Efficient Streaming Language Models with Attention Sinks. arXiv:2309.17453.

Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; Wei, H.; et al. 2024a. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Yang, Q.; Wang, J.; Li, X.; Wang, Z.; Chen, C.; Chen, L.; Yu, X.; Liu, W.; Hao, J.; Yuan, M.; et al. 2025a. AttentionPredictor: Temporal Pattern Matters for Efficient LLM Inference. *arXiv preprint arXiv:2502.04077*.

Yang, S.; Guo, J.; Tang, H.; Hu, Q.; Xiao, G.; Tang, J.; Lin, Y.; Liu, Z.; Lu, Y.; and Han, S. 2025b. Lserve: Efficient long-sequence llm serving with unified sparse attention. *arXiv preprint arXiv:2502.14866*.

Yang, S.; Sheng, Y.; Gonzalez, J. E.; Stoica, I.; and Zheng, L. 2024b. Post-training sparse attention with double sparsity. *arXiv preprint arXiv:2408.07092*.

Yang, X.; Zhan, R.; Wong, D. F.; Wu, J.; and Chao, L. S. 2023. Human-in-the-loop Machine Translation with Large Language Model. arXiv:2310.08908.

Zandieh, A.; Daliri, M.; and Han, I. 2024. Qjl: 1-bit quantized jl transform for kv cache quantization with zero overhead. *arXiv preprint arXiv:2406.03482*.

Zhang, H.; Ji, X.; Chen, Y.; Fu, F.; Miao, X.; Nie, X.; Chen, W.; and Cui, B. 2024a. PQCache: Product Quantization-based KVCache for Long Context LLM Inference. arXiv:2407.12820.

Zhang, T.; Yi, J.; Xu, Z.; and Shrivastava, A. 2024b. KV Cache is 1 Bit Per Channel: Efficient Large Language Model Inference with Coupled Quantization. In Globerson, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J.; and Zhang, C., eds., *Advances in Neural Information Processing Systems*, volume 37, 3304–3331. Curran Associates, Inc.

Zhang, Z.; Sheng, Y.; Zhou, T.; Chen, T.; Zheng, L.; Cai, R.; Song, Z.; Tian, Y.; Ré, C.; Barrett, C.; Wang, Z.; and Chen, B. 2023. H_2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. arXiv:2306.14048.