

Seer: Efficient KV Cache Management for LLM Acceleration via Mitigating Semantic Search Inefficiency

Yiwei Hu^{1,2,*}, Fangxin Liu^{1,2,*†}, Yue Liang^{1,2}, Xuan Wang³, Cheng Li³,
Gezi Li³, Li Jiang^{1,2,†}, Haibing Guan¹

¹Shanghai Jiao Tong University, ²Shanghai Qi Zhi Institute

³Huawei Technologies Co., Ltd.

{arikara666, liufangxin, ljiang_cs, hbguan}@sjtu.edu.cn

Abstract

Large language models (LLMs) have made significant progress in various NLP tasks, especially with the rise of long-context models that handle extended inputs. However, the growing size of the key-value (KV) cache in Transformer architectures imposes heavy memory demands during decoding, becoming a major bottleneck. Existing sparse attention methods aim to address this issue but face two main limitations: (1) they introduce high overhead when retrieving important tokens for attention, and (2) they often fail to significantly reduce memory usage in practice. This paper presents **Seer**, a simple yet effective system for fast and accurate LLM decoding on long-context sequences. Seer leverages semantic space representations to manage the KV cache more efficiently. It dynamically estimates cluster-level importance to reduce both computation and memory access, while preserving output quality. Through offline training, Seer learns semantic centroids to enable fast token importance estimation. To further improve efficiency, it integrates product quantization with sparse attention for KV cache compression and uses an asymmetric distance lookup table to reduce computational cost. Evaluation on a diverse set of LLMs and tasks shows that Seer reduces theoretical attention bandwidth by $40\times$ with minimal accuracy loss. Seer achieves a $5.57\times$ attention speedup and a $2\times$ End-To-End speedup over full attention at 32K sequence length. Our code is available at <https://github.com/Acaciasama/Seer>.

1 Introduction

Large language models (LLMs) (Brown et al., 2020; OpenAI et al., 2024) have transformed natural language processing (NLP), achieving state-of-the-art performance across diverse applications. As LLMs advance, they are increasingly applied

*Equal contribution.

†Corresponding author.

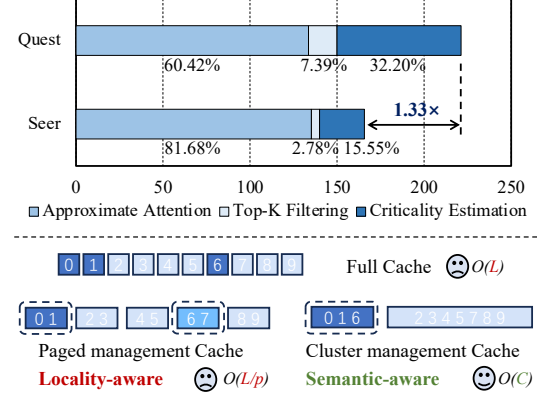


Figure 1: **Top:** Seer self-attention time breakdown compared with Quest (Tang et al., 2024). **Bottom:** Cache management paths for different approaches. Semantic cache organization reduces search and access overhead relative to page-based selection. L denotes sequence length, p denotes page size, and C denotes the number of clusters.

to tasks involving long contexts, such as Chain-of-Thought reasoning (Wei et al., 2023), document summarization, and retrieval-augmented generation (Lewis et al., 2021; Jin et al., 2025). However, efficiently serving long-context LLMs remains challenging due to memory and computational bottlenecks inherent in Transformer architectures (Vaswani et al., 2017; Liu et al., 2024b, 2025, 2026).

LLM inference consists of two stages (Zhong et al., 2024): prefilling and decoding. During prefilling, activations for all input tokens are computed, and their keys and values (KV) are stored in the KV cache. This allows the LLM to reuse these KV pairs for computing attention in subsequent decoding steps. In the decoding stage, the model generates one token at a time, leveraging both input tokens and previously generated ones. The KV cache size grows linearly with sequence length. Moreover, the decoding stage is memory-bound, as generating each new token requires accessing all prior tokens

in the KV cache. This makes KV cache access the primary bottleneck for long-context LLM decoding, severely limiting scalability and efficiency.

To mitigate this issue, recent research (Xiao et al., 2024; Zhang et al., 2023; Tang et al., 2024) has introduced sparse attention mechanisms. These approximate full attention by focusing on a subset of tokens with the highest attention scores, reducing computational and memory overhead while maintaining generative performance. Building on this, dynamic sparse attention methods have emerged. These approaches retain the full KV cache in memory but selectively load or unload tokens based on their semantic importance during decoding.

Dynamic sparse attention techniques can be broadly categorized into two types: selection-based methods and semantic-space search-based methods. **Selection-based sparsity** maintains all tokens in the KV cache, estimates their attention scores, and selects a small subset for computation in each decoding step. **Semantic-space search-based sparsity** (Yuan et al., 2025; Malkov and Yashunin, 2018), on the other hand, clusters tokens to evaluate their importance at the cluster level, enabling the dynamic selection of varying token subsets. This approach aligns naturally with the heterogeneity observed among attention heads in KV caches, making it a promising direction for optimization.

Despite advancements in semantic-similarity-based sparse decoding acceleration, notable gaps remain between current methods and practical deployment: (1) **Inefficient Clustering Overhead**. Clustering-based methods require recomputing centroids for each new input, increasing computational overhead rather than reducing it. (2) **Discontinuous Memory Access**. Dynamic token selection leads to discontinuous memory access patterns, degrading GPU efficiency due to frequent gather operations and sparse-to-dense transitions. (3) **Memory Management Challenges**. Clustering-based sparse still requires storing the full KV cache, often involving data unloading to CPU memory, which is impractical for large contexts.

To address these challenges, we introduce **Seer**, an algorithm and system designed for fast and accurate LLM decoding, as shown in Figure 1. Seer leverages a semantic space search-based dynamic sparse attention mechanism to optimize efficiency and accuracy. In conclusion, our contributions are:

- **Offline Index Construction:** We construct lightweight cluster indices via offline profiling on representative calibration datasets, decoupling the indexing overhead from runtime inference. To accommodate diverse dynamic workloads without expensive online re-indexing, we encode KV cache pages independently into structured centroid representations. Furthermore, to mitigate centroid assignment drift caused by out-of-distribution (OOD) activation patterns, we prune degenerated and empty clusters post-encoding, thereby ensuring robust search precision and high retrieval fidelity during decoding.
- **Memory-Efficient Sparse Attention Kernel:** We design a sparse self-attention kernel that reorganizes KV caches via reordering after encoding. This aligns the data with Inverted File indices, enabling fast sparse token retrieval. The operator’s thread scheduling is optimized based on the number of clusters searched, facilitating operator fusion and acceleration.
- **Integration with Product Quantization:** We integrate the semantic search-based sparse attention mechanism with product quantization, storing only compressed codes during generation. During decoding, asymmetric distance tables are computed at runtime, enabling approximate matching that closely approximates full attention while significantly reducing both computational and memory overhead.

2 Related Works

KV Cache Eviction. Several studies have focused on reducing memory footprint by discarding historically less important tokens. StreamingLLM (Xiao et al., 2024) identifies the "attention sink" phenomenon and addresses it by retaining initial tokens to enhance window attention. H2O (Zhang et al., 2023) leverages historical tokens to predict future ones, evicting less critical tokens. SnapKV (Li et al., 2024) and Keyformer (Adnan et al., 2024) employ heuristic-based methods during the prefilling stage to select which tokens to preserve for decoding. However, token eviction methods (Liu et al., 2023) permanently remove historical tokens, leading to significant accuracy degradation in scenarios requiring long-term memory.

Dynamic Sparse Attention. To overcome the limitations of eviction methods, many approaches

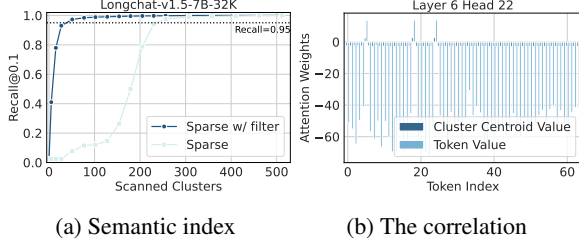


Figure 2: Semantic indexing for KV cache retrieval. (a) Empty-cluster filtering improves recall under the same search scope. (b) Semantic similarity between tokens and cluster centroids correlates with attention behavior, supporting cluster-level importance estimation.

retain the full KV cache and dynamically select a subset of KV pairs for inference. Quest (Tang et al., 2024) and ArkVale (Chen et al., 2024a) estimate page-level importance from key cache, reducing the search space from individual tokens to pages. InfiniGen (Lee et al., 2024) and ShadowKV (Sun et al., 2024) use low-rank representations to identify important cached states. MagicPIG (Chen et al., 2024b) uses locality-sensitive hashing to approximate token selection.

Semantic search offers a more structured way to select relevant KV tokens. RetrievalAttention (Liu et al., 2024a) builds graph indices for KV tokens in main memory. SqueezedAttention (Hooper et al., 2024a) accelerates fixed-prompt tasks by using K-means clustering during the offline phase and semantic similarity to predict relevant keys during inference. ClusterKV (Liu et al., 2024c) and PQ-Cache (Zhang et al., 2025) use cluster representations to retrieve similar cached states during decoding. However, online clustering for certain keys requires repeated processing for each input, incurring substantial computational overhead, and the associated latency is difficult to conceal in practice.

KV Cache Quantization. To reduce data volume in memory-bound scenarios, many quantization techniques (Liu et al., 2021) have been specifically optimized for the KV cache (Hooper et al., 2024b; Lin et al., 2024). KIVI (Liu et al., 2024d) applies different quantization strategies for keys and values, quantizing keys per-channel and values per-token to 2 bits. MILLION (Wang et al., 2025) addresses outlier challenges by applying product quantization (PQ) (Jegou et al., 2010) to the KV cache. These methods aim to balance memory efficiency with minimal performance degradation.

Upon improvement of previous work, **Seer** reduces computational overhead and memory access

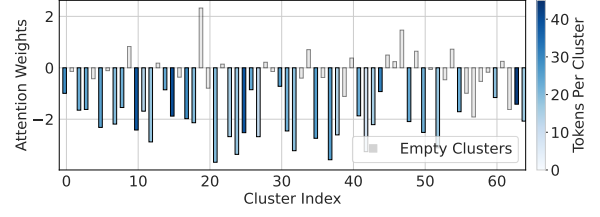


Figure 3: Cluster Importance and Imbalance Study. Due to the out-of-distribution (OOD) issue, clusters with high semantic similarity between centroid and current query may lack keys.

by using offline training and separating KV cache encoding. It introduces a memory-efficient sparse attention kernel that integrates product quantization to store compressed key codes, minimizing memory footprint while maintaining performance.

3 Rethinking Semantic Search

3.1 Problem Formulation

During LLM decoding, self-attention retrieves information from the KV cache:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d}})V \quad (1)$$

where we view KV are sets of vectors. Here, d is the head dimension, and n is the context length. The KV cache can be partitioned into M mutually disjoint subsets (clusters):

$$\mathcal{K} = \{\mathcal{K}_j\}_{j=1}^M, \bigcup_{j=1}^M \mathcal{K}_j = K, \mathcal{K}_i \cap \mathcal{K}_j = \emptyset \text{ for } i \neq j. \quad (2)$$

Our goal is to identify the union of as few subsets as possible such that the approximate attention results maintain accuracy:

$$\mathcal{I}_T = \bigcup_{j \in S} \mathcal{K}_j \quad S \subseteq \{1, 2, \dots, M\} \quad (3)$$

As a sparse attention mechanism, the attention scores s , computed using the selected subset, approximate the full attention calculation:

$$softmax(\frac{s_i}{\sqrt{d}})_{i \in \mathcal{I}_T} \cdot V_{\mathcal{I}_T} \approx softmax(\frac{s_i}{\sqrt{d}})_{i=1}^N \cdot V \quad (4)$$

In the following section, we analyze the computational complexity of existing semantic search methods and study the distribution of KV cache vectors in high-dimensional space, identifying the root causes of their inefficiency.

3.2 Semantic Search for Approximate Attention

KV cache naturally supports semantic-space retrieval because cached keys encode contextual information. We cluster keys by representation similarity and use centroids for coarse-grained importance estimation. Although centroid attention weights (QK^T) are lower than token-level weights in high-dimensional space, Figure 2(b) shows that important tokens often belong to clusters that are also important to the query. This suggests that cluster-level scores can approximate token importance, motivating semantic-search-based sparse attention.

3.3 Inefficiency of online Clustering

For dynamic sparse attention mechanisms, importance evaluation must be performed for each token to select the Top-K most significant tokens. Quest reduces this overhead by computing the maximum and minimum values per page of tokens, lowering the complexity to $O(Ld/page_size)$. However, traditional semantic search methods require clustering the KV caches for each prompt, with the K-means algorithm’s complexity being $O(n_iCLd)$, where n_i represents the number of training iterations (typically 20-25). This makes naive semantic search methods inefficient for KV cache sparsification. A critical question arises: **Is offline clustering feasible using irrelevant keys?**

3.4 Semantic Search in KV cache

Our analysis suggests that the answer is yes, but only when the mismatch introduced by offline clustering is explicitly handled. Naive cluster-based indexes are inefficient for KV cache retrieval (Liu et al., 2024a). As shown in Figure 2(a), to retrieve the Top-10% tokens, traditional IVF indexes require searching 50% of clusters to achieve a 95% recall rate. This inefficiency arises from Out-Of-Distribution (OOD) discrepancies between offline-trained keys and online-encoded keys.

Analyzing key distributions in Figure 3, we observe a highly imbalanced clustering using offline training: most tokens reside in a few clusters with low centroid importance, while many empty clusters have high centroid importance. This misalignment reduces retrieval efficiency, as dominant clusters contribute minimally to attention scores, and empty clusters hinder effective token selection. In extreme cases, the top 20% of clusters contain no

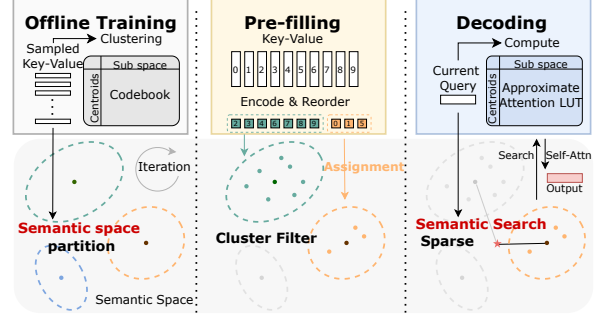


Figure 4: Overview of Seer for efficient KV cache management.

tokens whatsoever.

To address this, we exclude empty clusters from the search scope. Our experiments show that searching only 10% of remaining clusters achieves a recall rate of 0.95, validating this approach. To minimize overhead, we perform offline training and encode the KV cache once after prefill, reducing complexity to $O(CLd)$. During decoding, we compute distances between clusters and the query, further lowering complexity to $O(Cd)$. In long-text scenarios where $C \ll L$, this method substantially reduces computational overhead to a constant, while preserving accuracy.

4 METHODOLOGY

4.1 Overview

Figure 4 provides an overview of Seer across the stages of LLM inference. We delineate three key phases—offline, pre-filling, and decoding—to describe an efficient algorithm and system for fast LLM decoding, leveraging KV cache management to mitigate semantic search inefficiency. In the offline stage, sampled KV tokens are used to train reusable centroids and PQ codebooks. In the prefilling stage, the current KV cache is assigned to the learned centroids, reordered by cluster, and compressed into PQ codes. In the decoding stage, Seer selects important active clusters and performs approximate sparse attention over their compressed codes.

4.2 Training and Prefilling Phase

Training Phase. Figure 5 (top) shows that Seer extracts the key representations of each attention head from sampled historical Key-Value tokens during offline training. For the i -th head, the input can be written as $K^{(i)} \in \mathbb{R}^{s \times d_h}$, where s is the number of sampled tokens and d_h is the per-head hidden dimension. Seer performs clustering independently

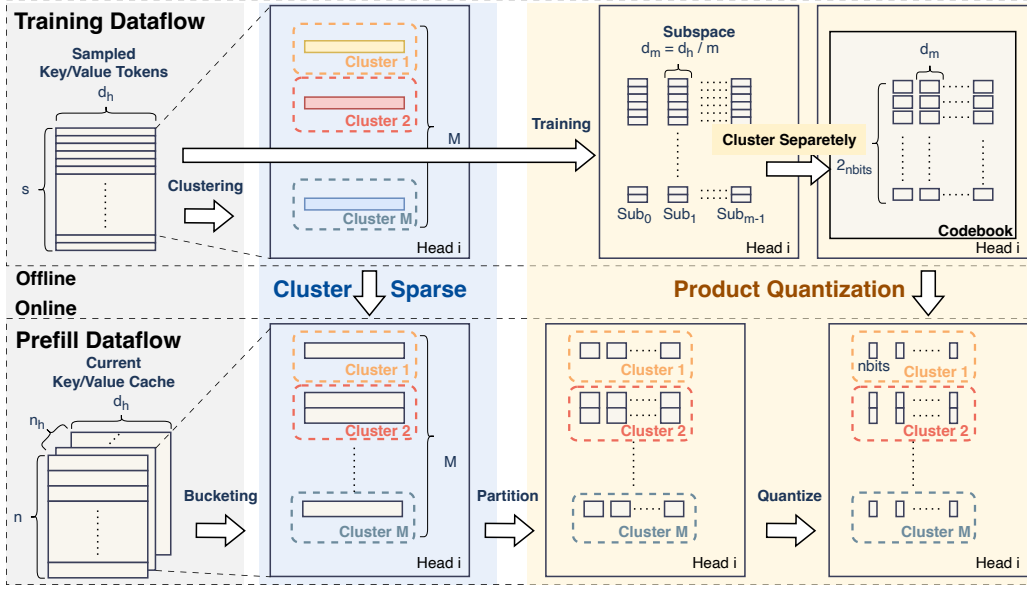


Figure 5: Seer training and prefilling dataflow. **Top:** the offline training stage learns semantic centroids and PQ codebooks from sampled KV tokens. **Bottom:** the prefilling stage assigns the current KV cache to learned clusters, reorganizes tokens into cluster-contiguous buckets, and encodes them into PQ codes for approximate sparse decoding.

for each head and compresses the original token-level representation of size $s \times d_h$ into a fixed set of centroids, where M is the number of clusters and typically satisfies $M \ll s$.

$$\mu^{(i)} \in \mathbb{R}^{M \times d_h}, \quad (5)$$

Meanwhile, Seer also trains the PQ codebook in the offline stage. Specifically, each key vector is partitioned along the head dimension into m subspaces, each of dimension $d_m = d_h/m$. For each subspace, Seer trains a sub-codebook with $2^{n_{bits}}$ codewords. As a result, the PQ codebook for each head can be represented as

$$\mathcal{C}^{(i)} \in \mathbb{R}^{m \times 2^{n_{bits}} \times d_m}. \quad (6)$$

Therefore, the offline stage produces two types of static structures: the centroids $\mu^{(i)}$ for cluster-level semantic routing, and the PQ codebook $\mathcal{C}^{(i)}$ for token-level approximate computation.

Prefilling Phase. As shown in Figure 5(bottom), Seer converts the dense KV cache of the current prompt into a cluster-organized compressed cache. For each head, the input key and value caches are $K^{(i)}, V^{(i)} \in \mathbb{R}^{n \times d_h}$, Seer assigns keys to their nearest offline-trained centroids $\mu^{(i)}$, reorders KV pairs by cluster assignment so that tokens in the same cluster are stored contiguously, records cluster boundaries, and filters out empty clusters. It then applies PQ encoding to the reordered keys

and values, producing compact key/value codes, boundary metadata, and active-cluster indices for decoding.

This one-time transformation avoids online centroid recomputation and prepares the cache for efficient cluster-level sparse access. The theoretical compression ratio is:

$$\frac{d_h \cdot \text{fpbits}}{m \cdot n_{bits}}, \quad (7)$$

excluding the small overhead of codebooks and boundary metadata. With $m = 64$, $n_{bits} = 8$, and FP16 cache storage, Seer achieves $4\times$ KV-cache compression, comparable to 4-bit quantization. The byte-aligned 8-bit codes are also GPU-friendly for lookup-based kernels.

4.3 Decoding Phase

Decoding Algorithm. Algorithm 1 summarizes the decoding procedure in Seer. Given the cluster-organized PQ-compressed KV cache produced during prefilling, decoding first computes the similarity between the current query and all cluster centroids, and then selects the top- k most relevant clusters. Since tokens belonging to the same cluster are stored contiguously, the selected clusters can be gathered efficiently as a small set of contiguous cache segments.

To avoid reconstructing full-precision keys, Seer builds a query-dependent asymmetric lookup table

Algorithm 1 Seer Decoding

Input: $K_{code} \in \mathbb{N}^{n \times m}$, $V_{code} \in \mathbb{N}^{n \times m}$, $q \in \mathbb{R}^{1 \times d}$,
 $\mu \in \mathbb{R}^{M \times d}$, $b \in \mathbb{N}^{M+1}$, k , $C_K \in \mathbb{R}^{2^{nbits} \times m \times d_m}$,
 $C_V \in \mathbb{R}^{2^{nbits} \times m \times d_m}$,
 $\triangleright$ *Compute query-to-centroid attention scores*
 $D \in \mathbb{R}^{1 \times M} \leftarrow \text{MatMul}(q, \mu^\top)$
 $\triangleright$ *Gather tokens from top-k clusters*
 $\mathcal{I} \leftarrow \bigcup_{j \in \text{ArgTopK}(D, k)} [b_j : b_{j+1})$
 $\triangleright$ *Build LUT for query-key similarity*
 $q_{sub} \in \mathbb{R}^{1 \times m \times d_m} \leftarrow \text{Reshape}(q)$
 $L \in \mathbb{R}^{m \times 2^{nbits}} \leftarrow \text{EinSum}(q_{sub}, C_K)$
 $\triangleright$ *Compute approximate attention scores*
 $K_{code}^{sparse} \in \mathbb{N}^{s_q \times m} \leftarrow \text{Gather}(K_{code}, \mathcal{I})$
 $P' \in \mathbb{R}^{s_q \times m} \leftarrow \text{Lookup}(K_{code}^{sparse}, L)$
 $P \in \mathbb{R}^{1 \times s_q} \leftarrow \text{Sum}(P', \text{dim} = -1)$
 $\triangleright$ *Decode values of selected tokens*
 $\hat{V}^{sparse} \leftarrow \text{Decode}(\text{Gather}(V_{code}, \mathcal{I}), C_V)$
 $\triangleright$ *Compute attention output*
 $S \in \mathbb{R}^{1 \times s_q} \leftarrow \text{Softmax}(P / \sqrt{d})$
 $\bar{o} \in \mathbb{R}^{1 \times d} \leftarrow \text{MatMul}(S, \hat{V}^{sparse})$
Return $\bar{o}$

(LUT) from the PQ key codebook C_K . Specifically, we reshape the query $q \in \mathbb{R}^{1 \times d}$ into m sub-vectors, denoted as

$$q = [q^{(1)}, \dots, q^{(m)}], \quad q^{(u)} \in \mathbb{R}^{d_m}, \quad (8)$$

For each subspace u , we compute the inner products between $q^{(u)}$ and all codewords in the corresponding sub-codebook of C_K ,

$$L^{(u)}[r] = \langle q^{(u)}, C_K[r, u, :] \rangle, \quad L \in \mathbb{R}^{m \times 2^{nbits}}. \quad (9)$$

Given a PQ-encoded key $K_{code}[i, :] = [z_i^{(1)}, \dots, z_i^{(m)}]$, its approximate similarity with the query is evaluated by

$$\langle q, \hat{k}_i \rangle \approx \sum_{u=1}^m L^{(u)}[z_i^{(u)}]. \quad (10)$$

Accordingly, for the gathered sparse key codes $K_{code}^{sparse} = \text{Gather}(K_{code}, \mathcal{I})$, the kernel computes attention scores through lookup-and-accumulate rather than full-precision key reconstruction followed by dense dot products.

Kernel Implementation. Figure 1 illustrates the execution breakdown of Seer, the decoding kernel has three fused stages: (1) **Criticality estimation.** Given the current query q , the kernel first computes its similarities to the cluster centroids.

Unlike page-based methods incur a search complexity proportional to $O(L/p)$, Seer operates on a much smaller semantic management space with complexity $O(C)$, where C is the number of clusters. As shown in Fig. 1(bottom), this cluster-level cache management is more aligned with the semantic structure of the KV cache and avoids scanning many locality-preserving but semantically irrelevant cache regions. (2) **Top-k cluster filtering.** The centroid scores are then passed to a top- k selection stage, which identifies the most relevant clusters for the current query. To avoid an additional filtering pass, we implement a masked top- k operator that fuses empty-cluster elimination with top- k selection. (3) **Approximate attention with PQ LUT.** After top- k cluster filtering, the kernel restricts attention computation to the selected cluster intervals. Because Seer stores tokens from the same cluster contiguously, these intervals can be accessed with coalesced memory loads, avoiding the scattered access pattern common in token-level sparse attention. The query-dependent LUT is generated once per decoding step and reused across all selected encoded keys. The kernel then computes approximate scores directly over PQ codes. After sparse score computation, only the selected value codes are decoded and accumulated like Flash-Decoding (Dao et al., 2022; Dao, 2024).

5 Evaluation

5.1 Accuracy Experiment

Setup. Our methods are applied to LongChat-7b-v1.5-32k (Li et al., 2023) and Llama-3-8B-Instruct \ Llama-3.1-8B-Instruct model (Grattafiori et al., 2024). We validated our approach on both the MHA model and the GQA model, demonstrating its continued effectiveness across the latest models. We evaluate our approach on long-context benchmarks: Longbench (Bai et al., 2024) and Needle-In-A-Haystack(NIAH) ([Kamradt, 2023]). We evaluated two configurations of Seer: **F-Seer**, which preserves the entire cache, and **Seer**, which utilizes encoding-based compression.

Baseline. Besides full attention, we employed various configurations of Quest (Tang et al., 2024) and ClusterKV (Liu et al., 2024c) as our baseline for comparison. The initial 4 tokens are preserved, although local tokens are not retained. For Quest, we set the page size to 16 and statically retaining the first layer, according to the original experimental setup. For ClusterKV, we use the same K-means

Table 1: LongBench evaluation results of Seer using full cache (F-Seer) and compressed cache (Seer). The "Budget" row shows the expected percentage of KV cache memory loaded into shared memory. We compare Quest, ClusterKV and Seer under the same budget 0.1 and 0.025.

Area	Dataset	LongChat-7b-v1.5-32k							Llama-3.1-8B-Instruct						
		Full	Quest (0.1)	ClusterKV (0.1)	F-Seer (0.1)	Quest (0.025)	ClusterKV (0.025)	Seer (0.025)	Full	Quest (0.1)	ClusterKV (0.1)	F-Seer (0.1)	Quest (0.025)	ClusterKV (0.025)	Seer (0.025)
Single-Doc QA	NQA	20.99	20.52	19.74	20.95	20.45	19.10	21.63	17.76	17.28	18.26	18.78	13.76	11.14	13.99
	Qasper	30.29	28.91	28.96	29.41	22.49	26.35	29.26	45.25	43.12	44.38	44.91	36.33	41.06	44.09
	MFQA	42.98	40.37	42.91	43.47	32.18	34.70	37.59	49.27	49.31	49.93	50.49	45.55	46.72	49.84
Multi-Doc QA	HotpotQA	33.04	35.84	32.58	33.05	30.64	28.74	31.89	50.08	51.38	48.61	49.12	48.66	46.38	49.25
	Musique	13.98	14.28	14.21	14.75	13.04	10.31	12.88	17.47	16.76	17.05	17.58	15.87	11.92	14.74
	Dureader	22.31	20.20	21.02	21.55	15.82	16.28	19.09	27.52	23.64	27.34	27.99	19.12	22.09	25.14
Few-shot	Trec	67.00	66.00	64.50	66.50	62.00	64.60	67.50	72.00	69.50	72.00	72.00	65.00	69.00	72.00
	TriviaQA	83.21	83.77	83.53	83.99	79.23	78.66	81.80	90.19	88.57	89.66	90.08	86.49	89.60	89.57
	Samsum	41.55	40.94	25.96	40.87	33.72	37.91	40.78	44.23	43.55	43.62	44.14	42.02	39.18	42.10
Summarization	GRep	31.13	28.76	29.72	31.05	20.04	25.74	28.52	32.91	29.25	32.12	32.76	23.64	27.02	30.07
	Qmsum	23.03	22.67	22.67	22.69	21.45	19.86	22.63	23.94	22.95	23.62	24.01	21.18	19.91	22.83
	MNews	26.35	24.10	25.87	26.57	8.78	23.40	26.41	26.86	26.46	26.81	27.23	21.48	27.10	27.06
	VCSum	6.61	5.98	5.82	6.30	4.25	3.34	5.91	15.22	13.28	15.01	15.58	12.10	11.62	14.58
Synthetic	Pc	0.00	0.00	0.00	0.00	1.50	0.00	0.00	1.71	2.05	1.02	1.50	1.62	0.00	1.78
	PRz	10.50	7.50	10.50	10.50	8.46	2.51	5.14	99.00	95.00	93.97	98.00	80.25	88.56	95.50
Code	LCC	52.96	44.28	52.41	53.07	14.77	50.48	53.45	65.69	62.10	63.73	65.72	50.93	63.72	63.51
	RepoB	56.86	56.58	56.31	56.88	43.74	52.82	55.74	57.74	55.55	58.26	58.92	50.20	53.48	56.39
Avg.	Score	33.11	31.81	31.57	33.04	25.44	29.11	31.78	43.34	41.75	42.67	43.46	37.31	39.32	41.91
Avg.	Loss	0.00	-1.30	-1.54	-0.07	-7.66	-4.00	-1.33	0.00	-1.59	-0.67	0.12	-6.04	-4.02	-1.44

configurations as Seer, setting the number of cluster centroids to 512 and the number of training iterations to 20. We also employed StreamingLLM as our baseline, setting the recent context budget to 20% in NIAH evaluations.

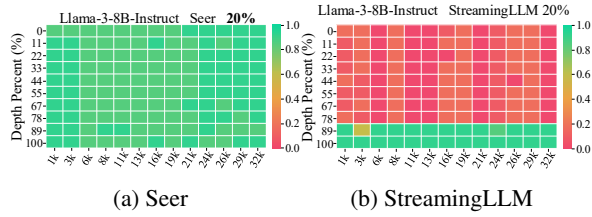


Figure 6: Needle-In-A-Haystack results. Since most of the historical tokens have been evicted, StreamingLLM are no longer effective for long-context retrieval. In contrast, Seer can identify key information through semantic recognition.

LongBench. We validated the effectiveness of Seer across multiple downstream tasks, including single- and multi-document QA, few-shot learning, document summarization, synthetic tasks and code tasks. At a compression rate of 0.1, we compared Quest and ClusterKV using a semantically managed, full-precision KV cache with a search over 10% of the clusters. For a compression rate of 0.025, we use compressed KV cache through product quantization while keeping the search range unchanged. The subspace dimension m was set to 64, with each code occupying 8 bits, achieving $4\times$ compression ratio, equivalent to 4-bit quantization. According to the last row of Table 1, Seer’s average degradation is less than 3%, maintaining high accuracy while utilizing only 1/40 of the theoretical bandwidth compared to full attention. Further-

more, F-Seer ensures lossless accuracy when the sparsity rate is 0.1. Regardless of the different configurations, F-Seer consistently outperforms Quest and ClusterKV by an accuracy improvement of 3–10%. Moreover, Seer still surpasses Quest (0.1) and ClusterKV (0.1) in performance, while reducing memory access by $4\times$, thereby demonstrating the superior efficacy of our approach.

NIAH. We demonstrated that Seer is capable of preserving key information within the KV cache when applied to long-text retrieval tasks, effectively filtering out irrelevant information. As shown in Figure 6, Seer shows the ability to process information at different positions across various context windows, ranging from 1k to 32k tokens. And at the budget of 0.05(sparse ratio 20%), Seer’s overall score on NIAH dataset is 0.872.

5.2 Performance Experiment

Setup. Our hardware configuration comprises an NVIDIA A40 48GB PCIe GPU, and the software environment is configured with Pytorch version 2.5.1, CUDA version 12.4. We evaluated the performance of Seer, with a budget of 0.025, using Llama-3.1-8B-Instruct model. This configuration implies the utilization of encoded KV cache management.

Baseline. Our baselines for performance experiment are full attention, Quest and ClusterKV. All of them are implemented by FlashInfer (Ye et al., 2025), a fast kernel library for LLM inference. The budget of Quest and ClusterKV is 0.1, achieving

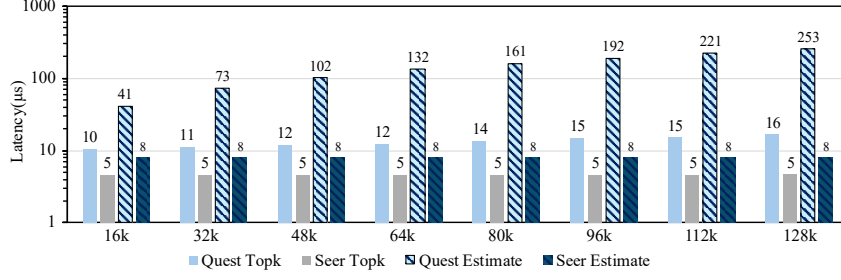


Figure 7: Dynamic sparse attention cost kernel(Topk and Estimate) latency. As sequence length increases, the relative criticality estimation latency of Quest increases linearly. The additional sparse overhead introduced by Quest is $21\times$ greater than Seer at 128K.

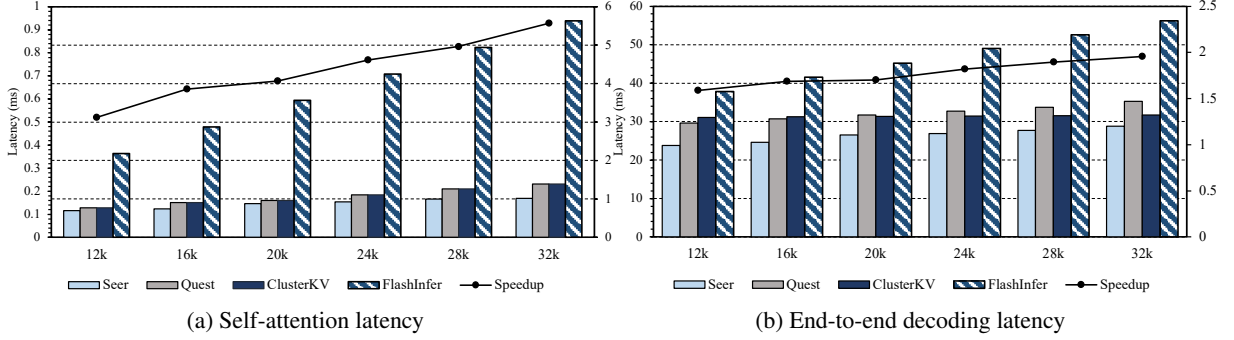


Figure 8: Latency comparison of Seer with baselines under the different sequence length. Regardless of attention latency or e2e latency, the performance advantage of Seer over Quest and ClusterKV becomes more pronounced as the prefilling length increases.

performance comparable to that of Seer in Section 5.1. The baselines utilize page-based KV cache management, whereas our approach employs cluster-based encoded KV cache management.

Kernel Evaluation. We illustrate the proportion of attention latency in decoding stage in Figure 1. Seer outperforms Quest primarily due to reductions in token importance estimation time and top-k selection time, which are attributable to our semantic-based efficient KV cache management approach. Additionally, we further evaluated how the overhead of dynamic sparsity varies with context length in Figure 7. The time complexity disadvantage of Quest becomes increasingly amplified as context-length grows, whereas Seer consistently maintains a low search overhead across different prefill sizes.

As shown in Figure 8(a), Seer significantly outperforms FlashInfer. Benefiting from fewer tokens after sparse, Seer’s latency exhibits minimal increase with longer sentences, resulting in an increasingly higher acceleration ratio compared to FlashInfer. Seer also outperforms ClusterKV in terms of clustering cost. At a sequence length of 32K, Seer achieves $5.57\times$ speedup over FlashInfer, $1.35\times$ speedup over Quest and ClusterKV.

End-To-End Evaluation. We evaluate the End-To-End latency of Seer to demonstrate its substantial potential in real-world applications. Using Time Per Output Token (TPOT) as metric, we assess the acceleration performance within a serving system. Through efficient KV cache management, Seer significantly reduces off-chip memory accesses, resulting in a consistent decrease in decoding latency compared to Quest. As shown in Figure 8(b), at a sentence length of 32K, Seer achieves $2\times$ speedup relative to the full cache. Our approach is fully compatible with other acceleration techniques, such as model quantization, enabling further improvements in inference speed.

More comprehensive experimental results are provided in Appendix A. Specifically, we report perplexity evaluation under the prefilling setting in Section A.2, examine the generalization ability of the offline-trained index across related datasets in Section A.3, present ablation studies on semantic search filtering and KV-cache management in Section A.4, and study scalability with respect to batch size, model size, and search range in Section A.5.

6 Conclusion

In this work, we first analyze the inefficiencies of existing dynamic sparse attention mechanisms

in terms of computational complexity and memory consumption. We then identify the limitations of current semantic search methods and demonstrate how relocating clustering operations to an offline stage can significantly improve semantic search efficiency. To address the suboptimal memory management in dynamic sparse attention techniques, we employ vector compression to optimize the storage of KV caches and implement semantic-based cache management, enabling high-efficiency off-chip memory access. Finally, we validate that Seer achieves high-generation quality across various long-text scenarios.

7 Limitation

Our approach requires pre-training and we discuss the cost of offline training in Section A.1. Although our results show that offline-trained indices exhibit a certain degree of generalization, they may still fail when there is a significant semantic gap in the KV cache—such as during transitions from English to Chinese.

Seer employs semantic-driven KV cache management, where cluster sizes vary. Despite optimizing memory access through reordering, load imbalance issues persist. To address this, we dynamically partition task workloads within CUDA kernels, which partially alleviates the problem; however, some performance degradation remains. In future research, we intend to explore advanced cluster segmentation strategies to improve load balancing and support inference systems such as vLLM.

Acknowledgments

This work was partially supported by the National Key Research and Development Program of China (2024YFE0204300), the National Natural Science Foundation of China (Grant No. 62402311), the Natural Science Foundation of Shanghai (Grant No. 24ZR1433700), and the Key Research and Development Program of Shanghai (25LN3201200).

References

- Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant J. Nair, Ilya Soloveychik, and Purushotham Kamath. 2024. [Keyformer: Kv cache reduction through key tokens selection for efficient generative inference](#). *Preprint*, arXiv:2403.09054.
- 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. 2024. [Longbench: A bilingual, multitask benchmark for long context understanding](#). *Preprint*, arXiv:2308.14508.
- Tom B. Brown, Benjamin Mann, and et al. Nick Ryder. 2020. [Language models are few-shot learners](#). *Preprint*, arXiv:2005.14165.
- Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. 2024a. [Arkval: Efficient generative llm inference with recallable key-value eviction](#). In *Advances in Neural Information Processing Systems*, volume 37, pages 113134–113155. Curran Associates, Inc.
- Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, and Beidi Chen. 2024b. [Magicpig: Lsh sampling for efficient llm generation](#). *Preprint*, arXiv:2410.16179.
- Tri Dao. 2024. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*.
- Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. [Flashattention: Fast and memory-efficient exact attention with io-awareness](#). *Preprint*, arXiv:2205.14135.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, and et al. Angela Fan. 2024. [The llama 3 herd of models](#). *Preprint*, arXiv:2407.21783.
- Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, June Paik, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. 2024a. [Squeezed attention: Accelerating long context length llm inference](#). *Preprint*, arXiv:2411.09688.
- Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. 2024b. [Kvquant: Towards 10 million context length llm inference with kv cache quantization](#). *Preprint*, arXiv:2401.18079.
- Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1):117–128.
- Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Shufan Liu, Xuanzhe Liu, and Xin Jin. 2025. [Ragcache: Efficient knowledge caching for retrieval-augmented generation](#). *ACM Trans. Comput. Syst.*, 44(1).
- Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.
- Gregory [Kamradt]. 2023. [Needle in a haystack - pressure testing llms](#).

- Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. [Infinigen: Efficient generative inference of large language models with dynamic kv cache management](#). *Preprint*, arXiv:2406.19707.
- Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. [Retrieval-augmented generation for knowledge-intensive nlp tasks](#). *Preprint*, arXiv:2005.11401.
- Dacheng Li, Rulin Shao, Anze Xie, Ying Sheng, Lianmin Zheng, Joseph Gonzalez, Ion Stoica, Xuezhe Ma, and Hao Zhang. 2023. [How long can context length of open-source LLMs truly promise?](#) In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*.
- 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](#). *Preprint*, arXiv:2404.14469.
- Yujun Lin, Haotian Tang, Shang Yang, Zhekai Zhang, Guangxuan Xiao, Chuang Gan, and Song Han. 2024. [Qserve: W4a8kv4 quantization and system co-design for efficient llm serving](#). *Preprint*, arXiv:2405.04532.
- Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. 2024a. [Retrievalattention: Accelerating long-context llm inference via vector retrieval](#). *Preprint*, arXiv:2409.10516.
- Fangxin Liu, Junjie Wang, Ning Yang, Zongwu Wang, Junping Zhao, Li Jiang, and Haibing Guan. 2025. [Aster: Adaptive dynamic layer-skipping for efficient transformer inference via markov decision process](#). In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 11853–11861.
- Fangxin Liu, Ning Yang, Haomin Li, Zongwu Wang, Zhuoran Song, Songwen Pei, and Li Jiang. 2024b. [Spark: Scalable and precision-aware acceleration of neural networks via efficient encoding](#). In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 1029–1042. IEEE.
- Fangxin Liu, Ning Yang, Jingkui Yang, Zongwu Wang, Chenyang Guan, Yu Feng, Li Jiang, and Haibing Guan. 2026. [Earth: An efficient moe accelerator with entropy-aware speculative prefetch and result reuse](#). In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 633–646.
- Fangxin Liu, Wenbo Zhao, Zhezhi He, Yanzhi Wang, Zongwu Wang, Changzhi Dai, Xiaoyao Liang, and Li Jiang. 2021. [Improving neural network efficiency via post-training quantization with adaptive floating-point](#). In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5281–5290.
- Guangda Liu, Chengwei Li, Jieru Zhao, Chenqi Zhang, and Minyi Guo. 2024c. [Clusterkv: Manipulating llm kv cache in semantic space for recallable compression](#). *Preprint*, arXiv:2412.03213.
- Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrilidis, and Anshumali Shrivastava. 2023. [Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time](#). *Preprint*, arXiv:2305.17118.
- Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen (Henry) Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024d. [Kivi: a tuning-free asymmetric 2bit quantization for kv cache](#). In *Proceedings of the 41st International Conference on Machine Learning, ICML’24*. JMLR.org.
- Yu. A. Malkov and D. A. Yashunin. 2018. [Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs](#). *Preprint*, arXiv:1603.09320.
- OpenAI, Josh Achiam, and et al. Steven Adler. 2024. [Gpt-4 technical report](#). *Preprint*, arXiv:2303.08774.
- 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](#). *Preprint*, arXiv:2410.21465.
- Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. 2024. [Quest: Query-aware sparsity for efficient long-context llm inference](#). *Preprint*, arXiv:2406.10774.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. [Attention is all you need](#). In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.
- Zongwu Wang, Peng Xu, Fangxin Liu, Yiwei Hu, Qingxiao Sun, Gezi Li, Cheng Li, Xuan Wang, Li Jiang, and Haibing Guan. 2025. [Million: Mastering long-context llm inference via outlier-immunized kv product quantization](#). *Preprint*, arXiv:2504.03661.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. [Chain-of-thought prompting elicits reasoning in large language models](#). *Preprint*, arXiv:2201.11903.
- Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2024. [Efficient streaming language models with attention sinks](#). *Preprint*, arXiv:2309.17453.

- Zihao Ye, Lequn Chen, Ruihang Lai, Wuwei Lin, Yinneng Zhang, Stephanie Wang, Tianqi Chen, Baris Kasikci, Vinod Grover, Arvind Krishnamurthy, and Luis Ceze. 2025. [Flashinfer: Efficient and customizable attention engine for llm inference serving](#). *arXiv preprint arXiv:2501.01005*.
- Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. 2025. [Native sparse attention: Hardware-aligned and natively trainable sparse attention](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23078–23097, Vienna, Austria. Association for Computational Linguistics.
- Hailin Zhang, Xiaodong Ji, Yilin Chen, Fangcheng Fu, Xupeng Miao, Xiaonan Nie, Weipeng Chen, and Bin Cui. 2025. [Pqcache: Product quantization-based kvcache for long context llm inference](#). *Preprint*, arXiv:2407.12820.
- Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yundong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. [H₂O: Heavy-hitter oracle for efficient generative inference of large language models](#). *Preprint*, arXiv:2306.14048.
- Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. [Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving](#). *Preprint*, arXiv:2401.09670.

A Appendix

A.1 Offline Training and Generalization

Offline training protocol. Seer employs a one-shot, lightweight offline training procedure to construct model-specific semantic centroids and PQ codebooks. Our approach requires offline pre-training, whose cost depends mainly on the model size and PQ hyperparameters. We first run inference once to collect training samples, but only a small fraction of tokens is needed. This allows the collected samples to follow the model’s natural autoregressive distribution. We exclude the initial positions associated with attention sinks and randomly sample historical KV tokens from the remaining positions. The same samples are then used to train the semantic centroids and PQ codebooks separately for each layer and attention head. All evaluation prompts and datasets, including LongBench, are strictly excluded from offline training.

The required number of samples is determined primarily by the PQ configuration. For example, when $\text{nb_bits} = 8$, each PQ subspace contains $2^8 = 256$ centroids. We use at most 256 training samples per centroid, resulting in a maximum of 65,536 sampled tokens. In practice, these samples can be collected from only 2–3 sufficiently long prompts. We use a fixed random seed for token sampling and apply the same sampling protocol across all evaluated models.

Training cost. The offline training cost depends mainly on the model size and PQ hyperparameters. Because each attention head in each layer uses an independent set of semantic centroids and PQ codebooks, the total cost scales with the number of layers and attention heads. Using the FAISS library (Johnson et al., 2019) on an AMD EPYC 7542 32-core CPU, training the index for Llama-3.1-8B takes approximately one CPU hour in our experiments. This is a one-time cost and can be further reduced using FAISS-GPU. Once trained, the index can be reused across input sequences and downstream tasks without additional fine-tuning.

Cross-task transfer. To determine whether Seer relies on dataset-specific index training, we evaluate a plug-and-play setting in which the index is trained only once on PG19, a generic long-text corpus that is entirely disjoint from LongBench, and is then directly applied to multiple LongBench tasks. Table 2 reports the results for Llama-3.1-8B-Instruct at an attention budget of 0.1.

Despite using no LongBench data during index training, Seer-PG19 achieves an average score of 44.98, compared with 44.52 for the target-domain index and 46.64 for full attention. The PG19-trained index therefore generalizes at least as well as the target-domain index on average, indicating that Seer learns reusable semantic structures rather than relying on task-specific tuning.

Cross-model validation. We further evaluate the same transfer setting on Qwen2.5-7B to verify that the observed generalization is not specific to Llama-3.1-8B-Instruct. As shown in Table 3, the PG19-trained index achieves an average score of 47.20, which is comparable to the 47.14 obtained using the target-domain index. These results consistently demonstrate that an index trained on a generic, held-out corpus can be directly applied to different downstream tasks and model families.

A.2 Perplexity

Seer primarily aims to optimize inference throughput during the decoding stage of LLMs; however, its applicability extends to the pre-filling stage as well. In scenarios such as prefix sharing and multi-turn dialogue, part of the key-value (KV) cache is precomputed and later merged with the KV cache generated from the current input to produce the output during pre-filling. Therefore, in situations where latency constraints are less strict, offline clustering can be employed, and historical caches can be sparsely loaded during pre-filling.

For our experiments, we utilize the WikiText-103 dataset, setting the model’s context window to 2,048. Considering the characteristics of the pre-filling phase, we perform index searches based on the average query token value across the textual length dimension. Additionally, we include an optimization by incorporating neighboring token windows of 10% of the pre-filling length to improve causal masking.

A.3 Generalization Ability

Although we train separate indices for each dataset, the results in Table 5 indicate that indices trained on samples from similar domains demonstrate notable generalization capabilities. By incorporating key-value tokens in few-shot learning datasets, we develop a more universal and versatile index.

The results in Table 5 show that the offline-trained index of Seer has favorable generalization ability across related datasets. Although training on

Table 2: Cross-task generalization results on LongBench using Llama-3.1-8B-Instruct. All Seer variants use an attention budget of 0.1. “Target” denotes indices trained using target-domain samples, “Seer-M” uses a few-shot universal index, and “PG19” denotes an index trained exclusively on PG19. The best result among the Seer variants is shown in bold.

Method / index training data	NQA	Qasper	MFQA	HotpotQA	MuSiQue	DuReader	TREC	TriviaQA	SAMSum	Avg.
Full attention	17.76	45.25	49.27	50.08	17.47	27.52	72.00	90.19	44.23	46.64
Seer, target-domain index	13.99	44.09	49.84	49.25	14.74	25.14	72.00	89.57	42.10	44.52
Seer-M, few-shot universal index	17.33	43.09	48.47	48.38	15.36	22.85	69.50	89.17	44.20	44.26
Seer-PG19, PG19 index	13.97	44.71	50.63	49.26	16.80	24.87	72.00	89.89	42.65	44.98

Table 3: Cross-task generalization results on LongBench using Qwen2.5-7B. Both Seer variants use an attention budget of 0.1. The better result between the two Seer variants is shown in bold.

Method / index training data	NQA	Qasper	MFQA	HotpotQA	MuSiQue	DuReader	TREC	TriviaQA	SAMSum	Avg.
Full attention	28.55	43.37	52.72	58.08	30.29	30.00	72.50	89.47	45.72	50.08
Seer, target-domain index	26.92	41.76	48.80	53.71	26.13	22.00	72.50	88.85	43.59	47.14
Seer-PG19, PG19 index	26.92	41.76	48.80	53.71	26.13	21.63	72.50	89.17	44.14	47.20

Table 4: PPL results for Llama3.1-8B-Instruct and Longchat-v1.5-7B model in WikiText-103 dataset.

	Llama3.1-8B-Instruct	Loss	Longchat-v1.5-7B	Loss
Baseline(FP16)	7.23		7.60	
Ours(0.05 budget)	7.66	5.94%	7.69	1.18%

Table 5: The experiment results for the generalization ability of Seer in Longbench dataset. "Seer-M" employs data sampled from the dataset area of few-shot learning to train a universal index.

Dataset Area	Single-Document QA			Multi-Document QA			Few-shot Learning		
Dataset	NQA	qasper	MFQA	hotpotqa	musique	dureader	trec	triviaqa	samsum
model	Llama-3.1-8B-Instruct								
Full	17.76	45.25	49.27	50.08	17.47	27.52	72	90.19	44.23
Seer (0.1)	13.99	44.09	49.84	49.25	14.74	25.14	72	89.57	42.1
Seer-M (0.1)	17.33	43.09	48.47	48.38	15.36	22.85	69.5	89.17	44.2

the target dataset still gives the best performance in most cases, Seer-M, which is trained only on few-shot learning data, remains competitive on both single-document and multi-document QA tasks. This suggests that the index mainly captures transferable key-value matching patterns rather than dataset-specific statistics, indicating that representative data from related domains is sufficient to train a practical shared index.

A.4 Ablation Study

Accurate semantic search. In Section 3.4, we filter empty cluster after encode. Empirical results show this to be important for search efficiency. We reduced the search range required to achieve 0.95 recall rate from 50% to 10%. We also evaluated the change in Seer’s accuracy before and after implementing filtering for empty clusters, as shown in Table 6. The results indicate that the filtering

Table 6: Downstream accuracy

Model	Longchat-7B-v1.5-32K		
Dataset	Hotpotqa	Musique	2Wikimqa
Full	33.04	13.98	23.82
Seer	31.89	12.88	23.16
Seer w/o Filter	25.46	11	20.21

Table 7: Efficiency

Seq	Seer	PQ	Full
8k	0.10	0.17	0.24
16k	0.12	0.29	0.46
24k	0.15	0.40	0.68
32k	0.17	0.51	0.90

mechanism is crucial for maintaining the quality of the model’s outputs.

Efficient KV cache management. We accelerate LLM decoding by reducing memory accesses and replacing computations with Lookup table. We present PQ and Seer attention latency(ms) in Table 7. PQ represent for encoded KV cache without sparse. Lookup approximate contributes to the acceleration significantly.

Table 8: Document QA and Few-shot learning tasks in Longbench. Seer has not accuracy loss for 70B level models. The config of Seer is (512, 51).

	Llama-3.1-70B-Instruct				
Method	2wikimqa	trec	hotpotqa	triviaqa	Avg.
Base.	61.02	75.5	51.75	83.99	68.07
Ours	61.03	75.0	51.84	83.98	67.96
Loss	+0.01	-0.50	+0.09	-0.01	-0.10

A.5 Scalability Study

A.5.1 Larger batch size

We implement kernels with different batch sizes and demonstrate that our method remains effective as the batch size increases.

We evaluate the attention latency of the Llama-3.1-8B-Instruct model in comparison to full attention at a sequence length of 32K. The Scaled

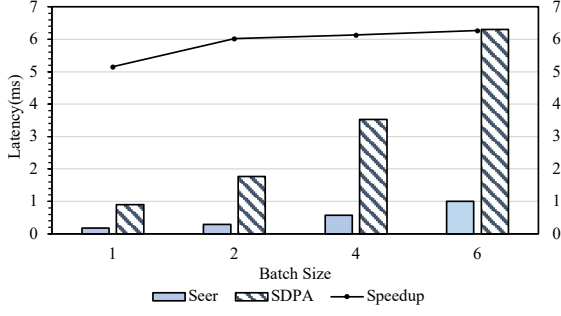


Figure 9: Efficiency comparison of Seer with the baseline across different batch sizes. The acceleration effect of Seer relative to full attention remains stable as batch sizes vary.

Dot-Product Attention (SDPA) kernel serves as our baseline, as it exhibits similar decoding performance to FlashInfer. Our results, shown in Figure 9, demonstrate that Seer achieves a significant speedup across all batch sizes. Notably, the latency of Seer increases approximately linearly with batch size, which is consistent with the baseline attention kernel. This indicates that the computational and memory access overhead introduced by Seer scales proportionally with the number of requests, without introducing additional batch-dependent bottlenecks. As a result, the speedup of Seer remains stable as batch size increases, demonstrating that our method preserves its efficiency advantage under higher-throughput serving scenarios.

A.5.2 Larger model size

To demonstrate that our approach remains effective with larger models, we apply Seer to Llama-3.1-70B-Instruct on the LongBench dataset. We use TriviaQA as a sampled dataset to train the index and evaluate performance on the WikimQA, Trec, HotpotQA, and TriviaQA datasets.

As shown in Table 8, Seer continues to achieve lossless acceleration in accuracy when scaling from 8B to 70B parameters. This result suggests that the effectiveness of Seer is not tied to a specific model size. Since Seer primarily reduces the memory access and retrieval overhead of long-context attention, its benefits remain applicable even for larger models, where long-context decoding is still constrained by memory bandwidth and cache access. These results demonstrate the robustness and scalability of Seer for large-scale LLM inference.

A.5.3 Different Search Range

As a supplementary analysis to the main experiments in Table 1, we examined the effect of varying the number of probes on accuracy, as detailed in

Table 9: Longbench experiments for different config setting of Seer. As the search range increases, the accuracy of model also improves; however, the decoding latency correspondingly increases.

Dataset Area	Single-Document QA			Multi-Document QA		
Dataset	NQA	Qasper	MFQA	Hotpotqa	Musique	Dureader
Model	Llama-3.1-8B-Instruct					
Full	17.76	45.25	49.27	50.08	17.47	27.52
Seer(0.1)	13.99	44.09	49.84	49.25	14.74	25.14
Seer(0.2)	13.58	43.71	51.92	48.99	15.24	25.98

Table 9. When the search range for clusters was increased from 10% to 20%, the accuracy improvement was minimal, indicating that the configuration used in our main experiments is already adequate.