FINDING THE NEEDLES IN LONG-CONTEXT ATTENTION THROUGH SPARSE FEATURES

Anonymous authors
Paper under double-blind review

ABSTRACT

In long-context decoding, high attention scores often concentrate on a small fraction of historical tokens scattered throughout the context. Attending only to these tokens can preserve generation quality, but finding them efficiently is a needle-in-a-haystack problem. We show that these tokens are not arbitrarily distributed in feature space: although they may differ in many respects, they share query-relevant features in a high-dimensional sparse representation. To expose this structure, we develop a shared query-key sparse autoencoder (SAE) that decomposes dense queries and keys into a common dictionary of sparse features. Most high-scoring keys share active features with the query, whereas such overlap is uncommon among the remaining keys. Building on this finding, we introduce PrismKV, which indexes historical keys by their active SAE features to form overlapping buckets. Compact bucket summaries identify relevant tokens without scanning the entire history, allowing attention to operate only on the selected original KV entries. Across RULER and LongBench on the 27B model, PrismKV retains an average of 99.90% of full-attention performance. Relative to optimized full attention, it achieves attention-path speedups of 3.37x on Qwen3-8B at a 128K context length and 2.78x on the 27B model with a 256K total window.

1 INTRODUCTION

In long-context decoding, high attention scores often concentrate on a small fraction of historical tokens, and attending only to these tokens can preserve near-full-attention generation quality (Liu et al., 2024; Hooper et al., 2024). However, these tokens are scattered throughout a long context and finding them efficiently is difficult. Pruning and compression methods (Ribar et al., 2023; Singhania et al., 2024) make individual token comparisons cheaper but still scan the entire history, so retrieval cost grows with context length. Bucket-summary methods (Tang et al., 2024; Hooper et al., 2024) reduce comparisons through block or cluster summaries but can miss relevant tokens when the data lack clearly separated groups. Both approaches remain limited because they doesn't answer a fundamental question: *What feature structure characterizes tokens with high attention scores?*

This paper identifies that structure: **high-scoring tokens share features with the query in a high-dimensional sparse space, even when their other features differ.** Relevance can arise from one shared feature within otherwise different combinations of features. This suggests that token-level pruning can leave the underlying feature structure unresolved and naive cluster summaries can hide a query-relevant feature among unrelated content. The challenge is therefore to represent each token by the individual features through which it may become relevant to a future query. The following paragraphs describe how this structure can be made explicit with a sparse autoencoder (SAE) (Anthropic, 2023) and how it relates to attention scores.

Features mixed in dense representations can be separated in a shared high-dimensional sparse space. Viewing query–key alignment through feature matching motivates examining which features each representation carries. An SAE (Anthropic, 2023) provides a tool for this analysis by learning a high-dimensional dictionary that reconstructs dense representations from sparse activations. Each token activates only a small subset of the learned features, making its feature composition explicit. Figure 1 illustrates this mapping from dense query and key vectors to a 3,000-dimensional sparse space with only a small subset of active features, in a measured Qwen3.8-27B example.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

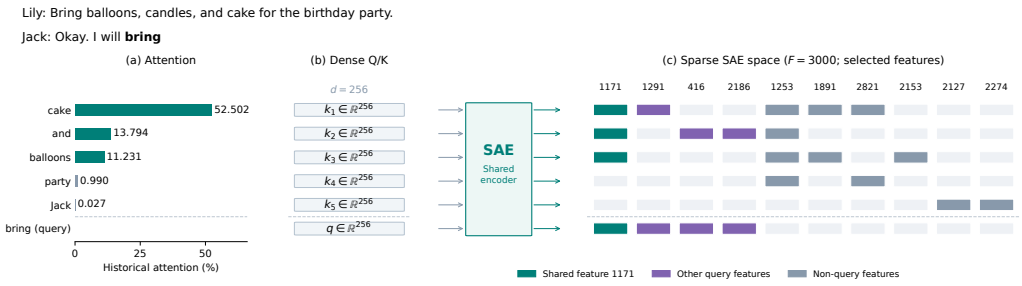


Figure 1: **Measured attention and shared SAE features on Qwen3.8-27B.** The shared SAE maps dense query and key vectors into a common 3,000-dimensional sparse space. Filled cells denote measured retained activations in selected SAE dimensions; gray cells denote features that are inactive for the query.

In this sparse space, high-scoring keys share query features substantially more often than other keys. Across the sampled layers and contexts, 66.5% of the highest-scoring 1% of keys share at least one active feature with the query, while only 5.3% of the remaining keys do so (Section 2). Figure 1 illustrates this association: the query *bring* shares feature 1171 with the high-scoring keys *cake*, *and*, and *balloons*. The key *cake* also shares feature 1291, while *and* shares features 416 and 2186. By contrast, the lower-scoring keys *party* and *Jack* share none of the retained query features. High-scoring keys can therefore agree with the query on a few relevant features while differing in their other features.

This finding motivates **PrismKV**, which organizes historical keys by the sparse features they activate. Because a token may be relevant through any one of its features, PrismKV indexes each token under every active feature, producing overlapping buckets. At decoding time, the query scores compact summaries of the original keys in each bucket and selects the highest-scoring buckets without scanning individual historical tokens. For a fixed feature count and bucket-selection budget, this scoring and selection cost is independent of context length. Attention is then computed over the retrieved original KV entries.

PrismKV preserves generation quality while accelerating attention. PrismKV achieves the highest average score among the evaluated sparse methods on several benchmarks, including RULER (Hsieh et al., 2024) and LongBench (Bai et al., 2024). On the 27B model, it retains an average of 99.90% of full-attention performance across these two benchmarks. On LongBench, its historical-token read fraction averages 8.67% across tasks. Relative to optimized full attention, PrismKV achieves attention-path speedups of 3.37× on Qwen3-8B at 128K context length and 2.78× on the 27B model at a 256K total window.

2 Sparse Feature Structure of Attention

High-scoring tokens can share query-relevant features while differing in their other features. A sparse-feature view of query–key alignment motivates examining this relationship, and SAE representations provide a way to test it empirically.

2.1 A Sparse-Feature View of Query–Key Alignment

A query and a key may align through shared features even when their other features differ. Each dense representation can be viewed as a mixture of features, suggesting that query–key alignment may reflect matches between some of its components. Research on *superposition* (Elhage et al., 2022) shows how neural networks can represent more features than they have dimensions. Although a d-dimensional space admits at most d mutually orthogonal directions, a larger dictionary can contain approximately orthogonal directions. This motivates an idealized picture in which each query or key activates only a few such directions.

A sparse decomposition formalizes this intuition and motivates examining feature overlap. Consider a dictionary $D = [d_1, \ldots, d_F] \in \mathbb{R}^{d \times F}$ with $F > d$ and unit-norm directions. In this

picture, the original attention vectors are written as

$$q = Da + r_q, \qquad k_i = Db_i + r_i, \qquad a, b_i \in \mathbb{R}^F_{\geq 0}, \tag{1}$$

where a, b_i are sparse activations and r_q, r_i collect the components not represented by the dictionary, including offsets and reconstruction errors; scaling is absorbed into the coefficients. Approximate orthogonality, expressed as $|d_f^\top d_g| \leq \epsilon$ for $f \neq g$ with small ϵ, makes distinct active directions weakly aligned in this idealized picture. The unscaled attention score can then be expressed as

$$q^\top k_i = \underbrace{a^\top b_i}_{\text{shared features}} + \underbrace{\sum_{f \neq g} a_f b_{ig} d_f^\top d_g}_{\text{cross-feature interactions}} + \underbrace{(Da)^\top r_i + r_q^\top (Db_i) + r_q^\top r_i}_{\text{residual contributions}}. \tag{2}$$

The shared-feature term contributes positively when the query and key activate the same directions; sparsity restricts the number of cross-feature terms, and near-orthogonality limits their inner-product factors. The decomposition serves as a motivation for examining feature overlap in measured query and key representations. It leads to the empirical question: can such features be identified in actual representations, and do high-scoring keys share them with the query more frequently than other keys?

2.2 Examining Feature Sharing with Sparse Autoencoders

A sparse autoencoder provides a tool for examining this feature-level view of attention. SAEs (Anthropic, 2023) learn high-dimensional feature dictionaries that reconstruct dense representations from a few active features. This makes them suited to identifying the components suggested by the preceding intuition and examining their relationship to attention scores. For a key $k \in \mathbb{R}^d$, write $\tilde{k} = (k - \bar{k})/s_K$ for its centered and scaled input. The encoder computes

$$z(k) = \mathcal{S}\Big(\mathrm{ReLU}\Big(W_{\mathrm{enc}}\tilde{k} + b_{\mathrm{enc}}\Big)\Big), \qquad W_{\mathrm{enc}} \in \mathbb{R}^{F \times d}. \tag{3}$$

Here $b_{\mathrm{enc}} \in \mathbb{R}^F$ is the encoder bias, and $\mathcal{S}$ retains a sparse set of positive activations. The nonzero entries of $z(k) \in \mathbb{R}^F_{\geq 0}$ identify the features active in the key. The corresponding reconstruction is

$$\widehat{\tilde{k}} = Dz(k) + b_{\mathrm{dec}}, \qquad D = [d_1, \ldots, d_F] \in \mathbb{R}^{d \times F}, \tag{4}$$

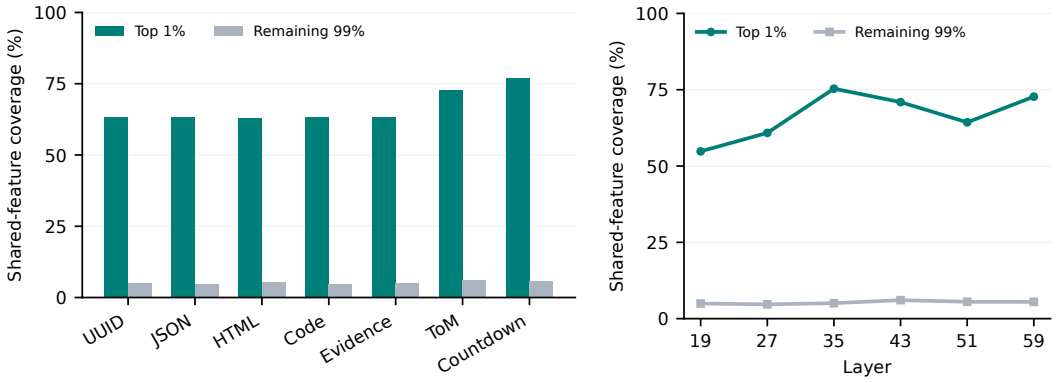
where the learned decoder combines feature directions d_f with activation weights $z_f(k)$ and adds a bias $b_{\mathrm{dec}} \in \mathbb{R}^d$.

The SAE is trained on query and key vectors collected from the frozen language model. The analysis in this section uses SAEs trained with reconstruction, query–key matching, and feature-utilization objectives, detailed in Section 3.1. Feature sharing among high-scoring tokens is also observed in an SAE without query–key matching supervision (Appendix B.1), and becomes more pronounced with this supervision.

These sparse representations make shared features directly observable across different token compositions. Different keys can activate different feature subsets while sharing individual features with the query. In Figure 1(b–c), the keys *cake*, *and*, and *balloons* share feature 1171 with the query *bring*, while activating different additional features. This representation allows feature sharing to be compared between high- and low-scoring keys.

High-scoring tokens share query-relevant features across different feature combinations. The 27B model interleaves full and linear attention; we sample full-attention layers 19, 27, 35, 43, 51, and 59 (zero-based), including all query heads. For each query, historical keys are ranked by their full-attention scores, and the top 1% are compared with the remaining 99% by the fraction sharing at least one active SAE feature with the query. Figure 2a shows that across different data, the coverage is 66.497% for the top 1% and 5.298% for the remaining 99% of keys. Figure 2b shows the same pattern across the six sampled layers. Figure 1 provides a token-level example of this pattern: the query *bring* shares specific SAE features with high-scoring keys across different feature combinations, while the displayed low-scoring keys share none.

These observations support using sparse feature memberships to organize token retrieval, motivating the overlapping buckets introduced in the next section.



(a) Across data.

(b) Across layers.

Figure 2: **High-attention keys are more likely to share active SAE features with the query.** The comparison is between the top 1% of keys by full-attention score and the remaining 99%. Detailed statistics appear in Appendix B.

3 PrismKV: Sparse Feature Indexing

PrismKV turns shared sparse features into an index for retrieving relevant KV entries (Figure 3). An SAE assigns each historical key to a few overlapping feature buckets, and compact summaries of those buckets allow each query to select candidates without scanning the full history. Attention then operates on the selected original keys and values. The pretrained model remains frozen throughout SAE training and inference.

3.1 Learning sparse feature membership

PrismKV fits the SAE defined in Equations (3) and (4) to preserve representation information and align sparse features with attention relevance. Each layer and KV group has a separate SAE, with a shared encoder for its queries and keys and unit-norm decoder columns. Only SAE parameters are optimized, using query and key vectors collected after positional encoding from the frozen language model.

Normalization and sparse activation. Keys use the normalized input $\tilde{k} = (k - \bar{k})/s_K$ from Section 2.2, where $\bar{k}$ is the mean key for the corresponding context. Queries use $\tilde{q} = q/s_Q$ without centering; both scales are estimated from training data and saved with the SAE. The notation $z(q)$ denotes the sparse query code produced by the same encoder. With $F = 3000$, the key sparsification operator $\mathcal{S}$ retains the largest positive activations under a total budget of at most $8N_K$ nonzero entries across a batch of N_K keys. This batch-level sparsity constraint follows the principle of BatchTopK (Bussmann et al., 2024): individual keys can activate different numbers of features, averaging at most eight per key. Query codes retain at most 16 positive activations each.

Reconstruction and query–key matching. The reconstruction loss $\mathcal{L}_{\text{rec}}$ is the mean squared reconstruction error over normalized query and key samples, divided by their mean squared input magnitude with a floor of 10^{-8}. To align sparse features with attention relevance, a positive key k^+ is sampled from the top 2% of full-attention scores in proportion to its attention probability, and 16 negative keys k_j^- are sampled from outside the top 10%. A cosine-margin loss favors the positive key over the hardest sampled negative:

$$\mathcal{L}_{\text{pair}} = \mathbb{E}_q \left[\max \left(0,\, 0.1 + \max_{j=1,\ldots,16} \cos\big(z(q), z(k_j^-)\big) - \cos\big(z(q), z(k^+)\big) \right) \right]. \tag{5}$$

Auxiliary objectives. We add matching pairs from selected evidence regions when those regions receive at least 0.02 teacher attention mass. These regions are intermediate evidence spans in the 27B recipe and the middle 20%–80% of the history in the 8B recipe. Their loss $\mathcal{L}_{\text{mid}}$ uses the same

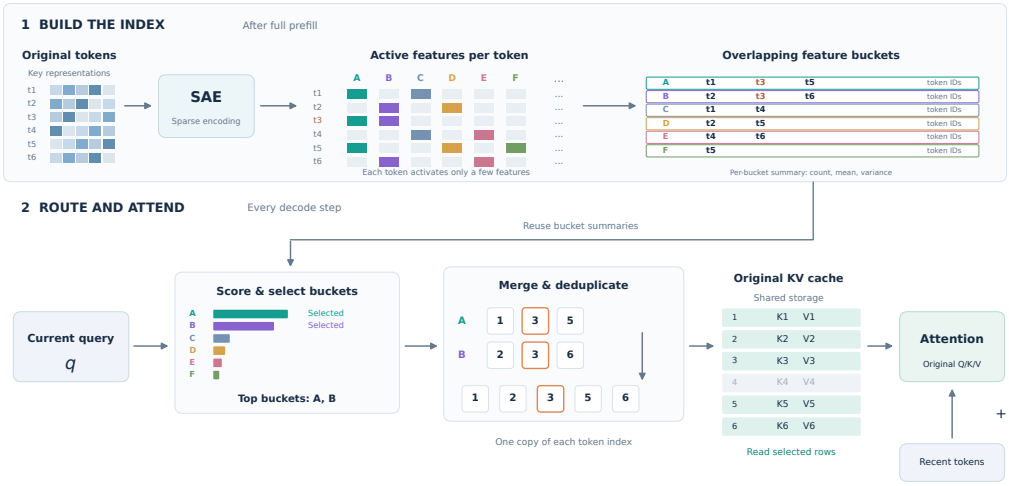


Figure 3: **PrismKV organizes tokens by sparse features and retrieves them through key summaries.** After prefill, SAE activations define overlapping feature buckets containing token indices and summaries of the original keys. During decoding, the query scores these summaries, selects buckets, and merges their indices before attention over the selected KV entries and the generated tail. Memberships and scores are illustrative; two selected buckets are shown for clarity.

margin objective. Together with $\mathcal{L}_{\text{pair}}$ and temporary soft matching variants, this loss forms $\mathcal{L}_{\text{match}}$; Appendix C describes the soft matching schedule. Two complementary losses encourage feature utilization: $\mathcal{L}_{\text{dead}}$ uses auxiliary residual reconstruction for inactive features (Gao et al., 2024), while $\mathcal{L}_{\text{balance}}$ penalizes concentration on frequently activated features using a moving average of feature usage. The total objective combines the three purposes:

$$\mathcal{L} = \underbrace{\mathcal{L}_{\text{rec}}}_{\text{reconstruction}} + \underbrace{\mathcal{L}_{\text{match}}}_{\text{query–key matching}} + \underbrace{\tfrac{1}{32}\mathcal{L}_{\text{dead}} + 0.5\mathcal{L}_{\text{balance}}}_{\text{feature utilization}}. \tag{6}$$

3.2 Building Overlapping Feature Buckets

After full prefill, PrismKV encodes the H historical keys using a total budget of at most $8H$ nonzero activations and creates one inverted list per feature:

$$\mathcal{B}_f = \{i : z_f(k_i) > 0\}. \tag{7}$$

Each list stores token indices, so tokens can participate in multiple feature buckets while their original KV vectors remain in shared storage. For every nonempty bucket, the index stores its size and the mean and elementwise variance of its original keys:

$$n_f = |\mathcal{B}_f|, \qquad \mu_f = \frac{1}{n_f}\sum_{i\in\mathcal{B}_f} k_i, \qquad \nu_f = \frac{1}{n_f}\sum_{i\in\mathcal{B}_f} (k_i - \mu_f)^{\odot 2}. \tag{8}$$

Feature activations determine membership, while these original-key statistics retain information for query-dependent scoring. The decoder is needed only during training.

3.3 Indexing Keys and Computing Attention

Bucket summaries estimate how much attention each feature group can attract. For a query q_h, PrismKV scores each nonempty bucket by

$$s_{hf} = \log n_f + \frac{q_h^\top \mu_f}{\sqrt{d}} + \frac{(q_h^{\odot 2})^\top \nu_f}{2d}. \tag{9}$$

This approximates the bucket's log unnormalized attention mass, $\log \sum_{i\in\mathcal{B}_f} \exp(q_h^\top k_i/\sqrt{d})$, using its size, mean alignment, and variation along the query dimensions. A finite SAE dictionary provides a useful but approximate feature decomposition, making its features suitable for organizing

overlapping buckets. These memberships narrow the search, while bucket scoring and final attention use the original query and key representations. Scoring bucket summaries costs $O(Fd)$, the same order as projecting a query into the SAE space. Computing attention over the selected original KV entries preserves their original attention scores without introducing additional approximation from SAE reconstruction error.

Under grouped-query attention (GQA) (Ainslie et al., 2023), multiple query heads share a KV head; PrismKV also shares the selected candidates within each group. Let B denote the bucket-selection number, a indexing hyperparameter. For the R query heads sharing a KV head, scores are normalized separately and then averaged:

$$p_f = \frac{1}{R}\sum_{h=1}^{R} \mathrm{softmax}_f(s_{h:}), \qquad \mathcal{A} = \mathrm{TopK}_B(p), \qquad \mathcal{U} = \bigcup_{f\in\mathcal{A}} \mathcal{B}_f. \tag{10}$$

Selection includes at most B nonempty buckets, and the union removes duplicate token indices. Because buckets overlap, p_f serves as a score rather than a partition of token-level attention probability. The unindexed generated tail $\mathcal{T}$ remains fully accessible, giving the candidate set $\mathcal{C} = \mathcal{U}\cup\mathcal{T}$. During decoding, PrismKV keeps newly generated tokens in the fully accessible tail and adds them back to the feature buckets every M tokens (e.g., $M = 4096$) for efficiency. PrismKV assigns the new keys to buckets using the SAE used during prefill, updates bucket counts, means, and variances, and resets the unindexed tail. Each query head computes its own attention weights over the original KV entries.

3.4 Computational cost

Indexing scores cost $O(RFd)$ per decoding step, independent of historical length for fixed dictionary size and head configuration. Bucket selection likewise operates on the fixed set of F features. If L denotes the total number of indices in the selected lists before deduplication, candidate assembly processes L entries, and attention costs $O(R|\mathcal{C}|d)$. The index stores at most $8H$ memberships and $O(Fd)$ summary statistics, adding a small storage overhead relative to the $O(Hd)$ KV cache in long-context settings. Initial encoding and index construction scale with history length and are amortized over subsequent decoding steps.

4 Experiments

We evaluate benchmark quality in Section 4.2, runtime in Section 4.3, and the contribution of query–key matching supervision in Section 4.4. Quality comparisons and latency measurements use the model-specific protocols below.

4.1 Experimental Settings

Models and baselines We evaluate Qwen3-8B and Qwen3.8-27B, with frozen BF16 backbones. The 8B model has 36 attention layers, eight KV heads per layer, four query heads per KV group, and head dimension 128. The 27B model has 16 standard attention layers with four KV heads, six query heads per group, and head dimension 256; its 48 linear-attention layers remain unchanged. We use full attention as the quality reference, and compare with Quest (Tang et al., 2024), SnapKV (Li et al., 2024b), PyramidKV (Cai et al., 2024), H_2O (Zhang et al., 2023), and KVzip (Kim et al., 2025). All methods use the same token reading budget, and baselines follow their default implementations unless otherwise specified.

Benchmarks. RULER (Hsieh et al., 2024) covers 13 tasks at 8K, 16K, 32K, 64K, and 128K, with 50 examples per task. LongBench (Bai et al., 2024) covers 16 tasks and 3,750 examples at natural input lengths. On 27B, LongProc (Ye et al., 2025) evaluates long-form procedural generation from 16 task difficulty subsets. SCBench (Li et al., 2024a) evaluates context reuse in both multi-request and multi-turn modes, with model-generated dialogue history in the latter.

PrismKV configuration. The total number of SAE features is set to 3,000, and key activations use a global budget of at most eight nonzero entries per key on average; individual keys may activate different numbers of features. 3000 here is an empirical choice that balances bucket size and

Table 1: **Benchmark averages for both models.** Higher is better; bold marks the highest displayed sparse-method score, including rounded ties. All scores are on a 0–100 scale; these aggregates combine different task metrics.

	Qwen3-8B		Qwen3.8-27B			
Method	RULER	LongBench	RULER	LongBench	LongProc	SCBench
Full attention	85.444	49.342	96.750	54.031	68.704	59.796
PrismKV	**81.995**	**48.833**	**96.673**	**53.963**	**67.297**	**59.082**
Quest	80.650	46.884	91.350	53.092	23.895	54.265
SnapKV	61.211	47.194	82.550	53.086	17.349	32.894
PyramidKV	55.415	46.076	57.130	50.889	15.455	27.218
H_2O	22.249	43.754	37.133	50.204	12.044	31.149
KVzip	26.112	42.274	36.520	45.886	29.557	27.238

Table 2: **LongBench category scores and PrismKV historical-token usage on the 27B model.** Higher quality scores are better; bold denotes the best sparse method.

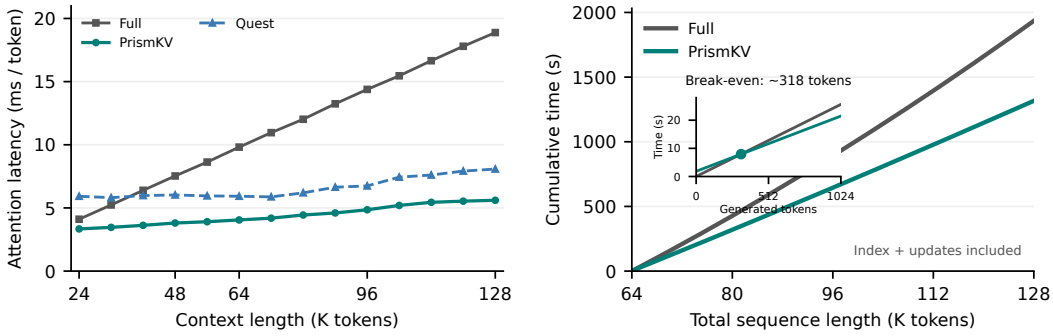
Category	Full	PrismKV	Quest	SnapKV	PyramidKV	H_2O	KVzip	Read (%)
Single-document QA	47.934	**47.464**	46.424	46.892	42.647	38.667	39.203	8.55
Multi-document QA	64.534	**64.663**	63.614	64.524	63.285	59.444	58.887	8.86
Summarization	26.878	**26.590**	26.026	23.549	21.969	25.855	23.278	9.69
Few-shot learning	71.770	72.423	72.181	**72.577**	71.416	68.859	54.744	8.79
Synthetic	58.167	**58.167**	**58.167**	**58.167**	**58.167**	57.917	54.250	6.56
Code	57.408	**56.830**	54.201	55.211	49.967	54.475	48.668	8.97
Task average	54.031	**53.963**	53.092	53.086	50.889	50.204	45.886	8.67

indexing cost. We set the bucket-selection number to $B = 16$. For GQA, probabilities are averaged over the query heads sharing each KV group before selecting buckets and deduplicating their token union. For the 27B model, the main SAE training pool contains 280 synthetic contexts, totaling approximately 9.2M input tokens across long-context retrieval and generation tasks.

Latency Measurement. Latency is measured in BF16 on a PPU-ZW810 96 GiB device, without dynamic or continuous batching, to isolate the contribution of selective KV access. For each metric, full attention uses the faster of grouped SDPA and native FlashAttention. We focus the sparse-baseline latency comparison on Quest, because it offers strong overall benchmark performance(Table 1), while other baselines show a too large performance drop or require specific kernel support. To match the evaluated model's GQA architecture, Quest (Tang et al., 2024) is adapted to share retrieval across query heads within each KV group. PrismKV and Quest use the same indexed sparse-attention kernel and are timed on the same device.

4.2 Benchmark Performance

Aggregate quality. Table 1 summarizes benchmark performance. On 8B, PrismKV has the highest sparse-method average on both benchmarks: 81.995 on RULER and 48.833 on LongBench, compared with full attention's 85.444 and 49.342. On 27B, it retains approximately 99.92% and 99.87% of full-attention RULER and LongBench scores. PrismKV has the highest average among the evaluated sparse baselines on all four 27B benchmarks. On LongProc and SCBench, it scores 67.297 and 59.082, respectively, compared with full attention's 68.704 and 59.796, exceeding the strongest sparse baseline by 37.740 and 4.817 points. These results extend the benefit of feature-based indexing to sustained generation and repeated access to shared context. Detailed LongBench scores for individual tasks are provided in Appendix A.1, with separate tables for the 27B and Qwen3-8B models (Tables 4 and 5).



(a) Measured attention-path latency.

(b) Cumulative time.

Figure 4: **PrismKV accelerates decoding across prefill lengths and amortizes indexing during continued generation.** (a) Measured attention-path latency, including indexing and selection, for full attention, PrismKV, and Quest after separate prefills of 24K–128K tokens. (b) Cumulative time for generating another 64K tokens after a single 64K prefill, including 1.844 s initial indexing and 0.269 s bucket updates every 4K generated tokens.

4.3 Latency

Decoding after different prefill lengths. PrismKV achieves lower attention-path latency than the Quest adaptation across the displayed 24K–128K prefill lengths (Figure 4(a)). Each point corresponds to a separate prefill followed by a 256-token autoregressive decoding run. At 128K, attention-path latency is 5.605 ms/token for PrismKV, compared with 18.883 for full attention, 8.077 for Quest. These savings translate into whole-model decoding latencies of 20.863, 34.092, and 23.232 ms/token, respectively, giving PrismKV speedups of $1.63\times$ over full attention and $1.11\times$ over Quest. Appendix A.2 provides the corresponding whole-model latency curves (Figure 5). The 27B evaluation in Appendix A.3 (Figure 6) reports separate attention-path and whole-model decoding speedups, reaching $2.78\times$ and $1.16\times$, respectively, at a 256K total window.

Continued generation with periodic bucket updates. The decoding savings also amortize index construction and periodic maintenance as generation extends the context (Figure 4(b)). This scenario starts with a single 64K prefill and generates another 64K tokens, incorporating newly generated keys into the buckets every 4K tokens while preserving existing memberships. PrismKV recovers the initial indexing cost after approximately 318 generated tokens and remains faster. By 128K total sequence length, cumulative time falls from 1,935.05 to 1,316.70 seconds, a $1.47\times$ speedup including indexing and periodic updates. Our evaluation isolates the benefit of selective KV access; the reported times characterize this setup rather than an optimized production serving system. In production scenarios, PrismKV can still accelerate attention with optimizations like low-bit quantization, multi-device parallelism, and dynamic batching.

4.4 SAE Ablations

Table 3 shows that PrismKV outperforms random bucket membership on Qwen3-8B, while query–key matching further improves quality and reduces token reads. Feature sharing also appears without matching supervision and becomes more pronounced with it (Appendix B.1).

Training cost and reuse. Offline SAE training is lightweight: optimizing all 64 SAEs for the 27B model requires approximately 1.7 GPU-hours on PPU-ZW810 hardware, excluding backbone Q/K collection (Appendix C.1). For each model, the resulting frozen dictionaries are reused across all reported benchmarks and latency settings without task- or context-length-specific retraining. These results support generalization of the learned feature dictionaries across the evaluated tasks and context lengths. Appendix C provides training configurations, Qwen3-8B training curves, and a NarrativeQA example of feature-bucket loads.

Table 3: **SAE ablations on Qwen3-8B.** Random membership assigns each token to eight distinct buckets, with token budgets calibrated from a separate PrismKV reference run. Read measures unique historical-token access, excluding the generated tail.

	RULER		LongBench	
Method	Score ↑	Read (%)	Score ↑	Read (%)
PrismKV	**81.995**	7.04	**48.833**	7.22
Without Q–K matching	81.590	7.67	48.782	7.56
Random bucket membership	21.885	7.04	45.689	7.22

5 Related Works

Sparse attention and selective KV access. Sparse attention methods differ in how they identify relevant history and distribute the access budget. SparQ (Ribar et al., 2023) and Loki (Singhania et al., 2024) reduce the cost of token-level scoring through coordinate selection and low-rank key representations, respectively. Routing Transformer (Roy et al., 2021) uses content-based clustering to organize sparse interactions, whereas Quest (Tang et al., 2024) estimates query-dependent page importance from key extrema and RetrievalAttention (Liu et al., 2024) uses attention-aware vector retrieval. DeepSeek Sparse Attention (DSA) (DeepSeek-AI, 2025) learns a lightweight indexer to score preceding tokens and selects the highest-scoring KV entries for attention. Its indexer is trained to match the main attention distribution, establishing attention supervision as a precedent; index evaluation nevertheless remains dependent on the growing history. Complementary work adapts which KV states remain accessible: SnapKV (Li et al., 2024b) identifies important positions from a prompt observation window, Ada-KV (Feng et al., 2024) allocates cache budgets across heads, and DuoAttention (Xiao et al., 2024) distinguishes retrieval heads from streaming heads. Squeezed Attention (Hooper et al., 2024) groups keys by clustering and uses their centroids to select relevant groups.

Sparse feature representations. Superposition (Elhage et al., 2022) provides an account of how neural representations can encode more features than their dimensionality through overlapping directions. SAE dictionary learning (Anthropic, 2023) recovers sparse feature activations from these dense representations. TopK SAEs (Gao et al., 2024) control activation sparsity directly and use an auxiliary reconstruction objective for inactive features, while BatchTopK (Bussmann et al., 2024) allocates the activation budget across samples. PrismKV uses sparse feature membership to index original keys in overlapping buckets and augments reconstruction with query–key matching to support attention retrieval.

6 Conclusion

Efficient long-context decoding requires finding the few relevant historical tokens without scanning the entire KV cache. The core observation is that high-scoring tokens share query-relevant SAE features even when their remaining features differ. PrismKV exploits this structure through overlapping feature buckets, using compact key summaries to select candidates at an indexing cost independent of context length for a fixed dictionary and head configuration. It retains 99.90% of full-attention performance averaged across RULER and LongBench on the 27B model and accelerates the attention path by 3.37× and whole-model decoding by 1.63× on Qwen3-8B at 128K context length.

AI use statement

Large language models were used to translate and polish manuscript text, improve its organization and clarity, assist with literature discovery and summarization, and draft sections from author-provided outlines, notes, and results. They also assisted in refining the conceptual presentation, formulating the shared-feature analysis, and drafting its mathematical derivation. The authors retain responsibility for the final manuscript, including its claims, citations, mathematical arguments, and reported results.

REFERENCES

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 4895–4901. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.298. URL https://aclanthology.org/2023.emnlp-main.298/.

Anthropic. Towards monosemanticity: Decomposing language models with dictionary learning, 2023. URL https://www.anthropic.com/research/towards-monosemanticity-decomposing-language-models-with-dictionary-learning.

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. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Bart Bussmann, Patrick Leask, and Neel Nanda. Batchtopk sparse autoencoders, 2024. URL https://arxiv.org/abs/2412.06410.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Yucheng Li, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Junjie Hu, and Wen Xiao. PyramidKV: Dynamic KV cache compression based on pyramidal information funneling, 2024. URL https://arxiv.org/abs/2406.02069.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, Roger Grosse, Sam McCandlish, Jared Kaplan, Dario Amodei, Martin Wattenberg, and Christopher Olah. Toy models of superposition, 2022. URL https://arxiv.org/abs/2209.10652.

Yuan Feng, Junlin Lv, Yukun Cao, Xike Xie, and S. Kevin Zhou. Ada-KV: Optimizing KV cache eviction by adaptive budget allocation for efficient LLM inference, 2024. URL https://arxiv.org/abs/2407.11550.

Leo Gao, Tom Dupré la Tour, Henk Tillman, Gabriel Goh, Rajan Troll, Alec Radford, Ilya Sutskever, Jan Leike, and Jeffrey Wu. Scaling and evaluating sparse autoencoders, 2024. URL https://arxiv.org/abs/2406.04093.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Monishwaran Maheswaran, Sebastian Zhao, June Paik, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. Squeezed attention: Accelerating long context length LLM inference, 2024. URL https://arxiv.org/abs/2411.09688.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, and Boris Ginsburg. RULER: What's the real context size of your long-context language models? In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=kIoBbc76Sy.

Jang-Hyun Kim, Jinuk Kim, Sangwoo Kwon, Jae W. Lee, Sangdoo Yun, and Hyun Oh Song. KVzip: Query-agnostic KV cache compression with context reconstruction, 2025. URL https://arxiv.org/abs/2505.23416.

Yucheng Li, Huiqiang Jiang, Qianhui Wu, Xufang Luo, Surin Ahn, Chengruidong Zhang, Amir H. Abdi, Dongsheng Li, Jianfeng Gao, Yuqing Yang, and Lili Qiu. SCBench: A KV cache-centric analysis of long-context methods, 2024a. URL https://arxiv.org/abs/2412.10319.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation, 2024b. URL https://arxiv.org/abs/2404.14469.

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. RetrievalAttention: Accelerating long-context LLM inference via vector retrieval, 2024. URL https://arxiv.org/abs/2409.10516.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. SparQ Attention: Bandwidth-efficient LLM inference, 2023. URL https://arxiv.org/abs/2312.04985.

Aurko Roy, Mohammad Saffar, Ashish Vaswani, and David Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021. doi: 10.1162/tacl_a_00353. URL https://aclanthology.org/2021.tacl-1.4/.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention, 2024. URL https://arxiv.org/abs/2406.02542.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context LLM inference, 2024. URL https://arxiv.org/abs/2406.10774.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, Junxian Guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. DuoAttention: Efficient long-context LLM inference with retrieval and streaming heads, 2024. URL https://arxiv.org/abs/2410.10819.

Xi Ye, Fangcong Yin, Yinghui He, Joie Zhang, Howard Yen, Tianyu Gao, Greg Durrett, and Danqi Chen. LongProc: Benchmarking long-context language models on long procedural generation, 2025. URL https://arxiv.org/abs/2501.05414.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang "Atlas" Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 34661–34710. Curran Associates, Inc., 2023. doi: 10.52202/075280-1506. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6ceefa7b15572587b78ecfcebb2827f8-Paper-Conference.pdf.

A Complementary Results

This appendix complements the main experiments with task-level LongBench scores, whole-model decoding measurements, and benchmark results on Llama-3.1-8B-Instruct.

A.1 Detailed LongBench Results

Tables 4 and 5 report individual LongBench task scores for the 27B and Qwen3-8B models, respectively, complementing the benchmark averages in Section 4. Both tables also include PrismKV historical-token read fractions.

A.1.1 Qwen3.8-27B

Table 4 pairs task scores with historical-token read fractions to show quality and selective KV access together.

Table 4: **LongBench task scores and PrismKV historical-token usage on the 27B model.**

Task	n	Full	PrismKV	Quest	SnapKV	PyramidKV	H_2O	KVzip	Read (%)
NarrativeQA	200	35.723	35.789	34.766	**36.123**	36.059	34.210	33.923	9.37
Qasper	200	48.671	47.705	**48.763**	45.857	38.597	37.645	38.090	8.01
MultiFieldQA-en	150	59.406	**58.899**	55.744	58.696	53.284	44.145	45.596	8.26
HotpotQA	200	70.660	**70.760**	69.815	69.910	68.474	67.006	66.257	8.59
2WikiMQA	200	66.377	**66.924**	66.123	66.761	65.252	60.594	58.176	8.98
MuSiQue	200	56.566	56.304	54.904	**56.901**	56.130	50.734	52.227	9.00
GovReport	200	32.974	32.430	**32.446**	28.933	26.409	32.044	28.389	9.64
QMSum	200	23.362	**23.293**	22.835	23.245	22.610	23.193	21.925	9.60
MultiNews	200	24.298	**24.046**	22.797	18.467	16.888	22.329	19.521	9.83
TREC	200	79.500	**81.500**	**81.500**	**81.500**	80.500	72.500	38.000	9.46
TriviaQA	200	90.452	**90.702**	90.390	90.265	89.494	89.406	88.635	8.07
SAMSum	200	45.357	45.068	44.653	**45.965**	44.254	44.671	37.597	8.84
Passage Count	200	16.333	**16.333**	**16.333**	**16.333**	**16.333**	**16.333**	16.000	8.09
Passage Retrieval-en	200	100.000	**100.000**	**100.000**	**100.000**	**100.000**	99.500	92.500	5.02
LCC	500	66.220	**65.632**	63.140	63.200	56.720	62.986	56.420	9.10
RepoBench-P	500	48.596	**48.028**	45.262	47.222	43.214	45.964	40.916	8.83
Task average	–	54.031	**53.963**	53.092	53.086	50.889	50.204	45.886	8.67

A.1.2 QWEN3-8B

Table 5 breaks down the Qwen3-8B benchmark average by task, pairing quality with PrismKV's historical-token read fractions across the 16 LongBench tasks.

Table 5: **LongBench task scores on Qwen3-8B.** Each method is evaluated on the same 3,750 examples across 16 tasks; n is the sample count for each task. Bold marks the highest displayed sparse-method score, including rounded ties. The final row averages the 16 task scores.

Task	n	Full	PrismKV	Quest	SnapKV	PyramidKV	H_2O	KVzip	Read (%)
NarrativeQA	200	27.151	26.288	26.782	27.108	26.617	**27.265**	25.108	8.33
Qasper	200	48.321	**46.652**	43.901	41.866	37.035	36.025	34.074	7.04
MultiFieldQA-en	150	53.473	**53.145**	49.822	51.567	48.334	40.386	40.898	6.69
HotpotQA	200	59.441	**59.260**	57.418	57.384	58.286	55.675	52.328	6.30
2WikiMQA	200	43.038	**42.840**	39.353	42.627	41.311	38.228	35.934	6.44
MuSiQue	200	36.101	34.592	32.953	36.033	**36.135**	31.014	32.407	6.78
GovReport	200	33.790	**32.437**	32.344	26.385	24.734	30.009	26.785	7.70
QMSum	200	23.828	**23.999**	23.724	23.561	23.157	22.422	22.540	7.99
MultiNews	200	24.665	**23.886**	22.811	16.962	15.586	22.371	20.035	8.01
TREC	200	71.500	**71.500**	71.000	67.500	67.500	66.500	65.000	9.35
TriviaQA	200	91.112	**91.112**	88.229	90.862	90.945	89.545	87.933	7.05
SAMSum	200	43.940	**43.861**	38.940	42.469	41.884	41.604	36.998	7.49
Passage Count	200	1.500	1.000	1.267	2.000	**3.000**	1.500	1.292	6.12
Passage Retrieval-en	200	100.000	**100.000**	95.500	**100.000**	**100.000**	72.000	83.500	4.64
LCC	500	67.822	**67.134**	63.258	66.094	62.446	64.750	57.452	7.93
RepoBench-P	500	63.792	**63.618**	62.840	62.688	60.246	60.776	54.104	7.64
Task average	–	49.342	**48.833**	46.884	47.194	46.076	43.754	42.274	7.22

A.2 WHOLE-MODEL DECODING LATENCY

Figure 5 complements the attention-path comparison in Figure 4(a) with whole-model decoding latency from the same paired runs. Across the displayed 24K–128K contexts, PrismKV has lower decoding latency than full attention and baseline adaptations.

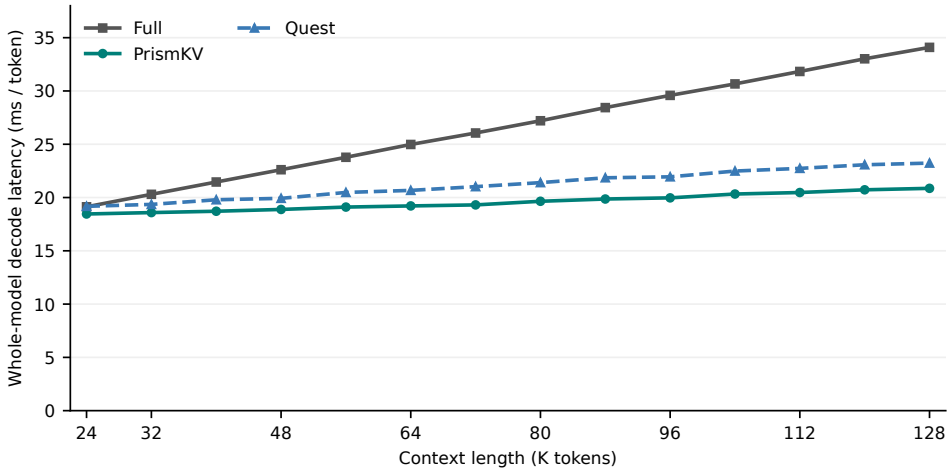


Figure 5: **Whole-model decoding latency on Qwen3-8B.** Measurements use BF16 and 256 autoregressive decode steps on a PPU-ZW810, with medians over six repetitions under graph execution. Quest denotes the GQA adaptation described in Section 4.1. Times include the complete model forward pass during steady-state decoding; prefill, one-time preparation, graph capture, and first replay are excluded.

A.3 Attention and Decoding Acceleration on the 27B Model

Figure 6 reports attention-path and whole-model decoding speedups separately from 24K to 256K. The 27B backbone has 16 standard attention layers and 48 linear-attention layers; PrismKV accelerates the former, while whole-model timing includes all 64 layers.

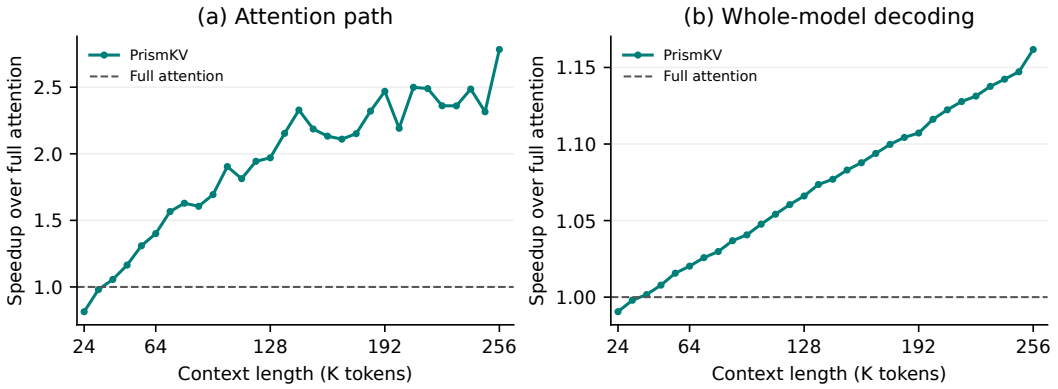


Figure 6: **Attention-path and whole-model decoding speedups on the 27B model.** (a) Attention timing includes indexing and selected attention across the 16 standard attention layers, evaluated on shared Q/K/V snapshots. (b) Whole-model timing includes all 64 layers during autoregressive decoding.

At the 256K total-window boundary, attention-path latency falls from 12.959 to 4.656 ms/token ($2.78\times$), while whole-model decoding falls from 57.342 to 49.359 ms/token ($1.16\times$). The hybrid architecture limits the fraction of decoding that benefits from sparse KV access: only 16 of 64 layers use standard GQA, while the 48 linear-attention layers and other model operations remain unchanged. This architectural difference explains why attention acceleration translates into a smaller whole-model gain.

A.4 RESULTS ON LLAMA-3.1-8B-INSTRUCT

Table 6 evaluates whether the quality advantage extends beyond the Qwen model family. PrismKV also achieves the highest RULER and LongBench averages among the evaluated sparse baselines on Llama-3.1-8B-Instruct, scoring 81.010 and 48.881, respectively. These results support the effectiveness of sparse feature indexing beyond the Qwen model family.

Table 6: **Llama-3.1-8B-Instruct benchmark scores.**

Method	RULER ↑	LongBench ↑
Full attention	85.556	49.548
PrismKV	**81.010**	**48.881**
Quest	79.118	48.818
SnapKV	67.474	46.776
PyramidKV	65.433	46.522
H_2O	20.305	45.878
KVzip	33.235	43.302

B SHARED-FEATURE COVERAGE STATISTICS

Table 7 provides statistics by data and sampled layer; Table 8 reports aggregate coverage across shared-feature thresholds. The 27B model interleaves full and linear attention; we sample full-attention layers 19, 27, 35, 43, 51, and 59 (zero-based).

Table 7: Shared-feature coverage (%) at $m = 1$ by data and sampled full-attention layer.

Task	Keys	Layer 19	Layer 27	Layer 35	Layer 43	Layer 51	Layer 59
UUID	Top 1%	53.950	57.000	72.260	65.500	60.750	71.080
	Rest 99%	5.030	4.500	4.650	5.290	5.540	5.610
JSON	Top 1%	50.820	56.320	71.470	67.770	61.230	71.530
	Rest 99%	4.900	4.570	4.270	5.260	5.020	4.990
HTML	Top 1%	54.410	51.860	72.170	64.810	65.610	67.940
	Rest 99%	5.160	4.860	4.570	5.420	6.350	5.660
Code	Top 1%	49.410	57.830	71.690	69.260	60.840	69.970
	Rest 99%	4.570	4.510	4.510	5.550	5.030	5.030
Evidence	Top 1%	53.190	55.770	71.090	66.880	61.120	72.290
	Rest 99%	5.040	4.560	4.530	5.500	5.490	5.490
ToM	Top 1%	64.800	70.700	81.750	76.310	66.650	75.640
	Rest 99%	5.230	4.840	6.780	7.780	5.720	5.930
Countdown	Top 1%	57.070	76.490	86.960	86.080	74.000	80.610
	Rest 99%	4.700	4.970	6.200	7.620	5.580	5.730

Table 8 varies the minimum number of shared features, m, to show how coverage changes under stricter overlap criteria; values are percentages.

Table 8: Aggregate shared-feature over the six sampled layers.

m	Top 1%	Rest 99%
1	66.497	5.298
2	30.335	0.517
3	8.743	0.056
4	1.978	0.006

B.1 Feature Sharing Without Query–Key Matching Supervision

The association between feature sharing and high attention scores remains observable without explicit query–key matching supervision. We train a Qwen3.8-27B SAE with 3,000 features from random initialization for 6,000 updates on 280 training contexts, retaining reconstruction, dead-feature auxiliary reconstruction, and balance regularization while removing both ordinary and middle-position matching objectives. Evaluation uses the same 14 contexts and full-attention query/key trajectories as the supervised checkpoint, covering KV group 0 in layers 19, 35, 51, and 55 and all six associated query heads. Both checkpoints use positive Top16 query activations and a global $8H$ historical-key activation budget; observations are averaged within each context/dictionary and then equally across the 56 units.

Without matching supervision, 40.572% of the top-scoring 1% of keys share at least one active feature with the query, compared with 5.467% of the remaining 99% of keys, a $7.42\times$ difference in coverage. The rates for the supervised checkpoint are 70.021% and 5.665%. Feature sharing is observable without matching supervision, while matching supervision strengthens its association with high attention scores and improves downstream quality with fewer token reads, as shown in Section 4.4.

C SAE TRAINING AND FEATURE UTILIZATION

This appendix documents the 27B SAE training configuration and offline cost, followed by training curves and feature-bucket loads (Figures 7 and 8).

C.1 27B TRAINING CONFIGURATION AND OFFLINE COST

Table 9 summarizes the configuration used to train 64 SAEs for the 27B model, one per KV group in its 16 full-attention layers. The backbone remains frozen; each SAE encodes the six query heads sharing its KV group.

Table 9: **27B SAE training configuration and cumulative offline cost.** Costs cover the optimization stages contributing to all 64 evaluated SAEs, excluding backbone Q/K trajectory collection; GPU-hours measure cumulative work rather than parallel wall-clock time.

Setting	Configuration
SAE count	16 layers $\times$ 4 KV groups $= 64$
Input / feature / output dimensions	$256/3000/256$
Query / key activation budgets	Positive Top16 / batch-average 8 per key
Optimizer	AdamW, $\beta = (0.9, 0.999)$, $\epsilon = 10^{-8}$
Initial learning rate	3×10^{-4} (joint); 10^{-3} (early reconstruction)
Weight decay / gradient clipping	0 / global norm 1
Learning-rate schedule	50-step warmup; cosine decay to $0.1\times$ initial rate
Reconstruction batch	128 queries and 128 keys
Matching batch	32 queries; 1 positive and 16 negatives per query
Middle-evidence batch	32 additional pairs when eligible
Hardware	PPU-ZW810, 96 GB
SAE optimization cost	Approximately 1.7 GPU-hours
Including loading and preparation	Approximately 2.1 GPU-hours

We collect offline data from different data and tasks. The training pool contains 280 contexts, with target lengths of 16K (112 contexts), 32K (112), and 64K (56). These target lengths sum to 9,175,040 input tokens (approximately 9.2M, with K denoting 1,024); actual tokenized lengths vary slightly with formatting and padding. Full-attention trajectories from the frozen backbone supply post-RoPE queries and keys; attention rankings and annotated evidence spans provide matching supervision.

C.2 TRAINING DYNAMICS

The reconstruction and query–key matching objectives decrease together during SAE training (Figure 7).

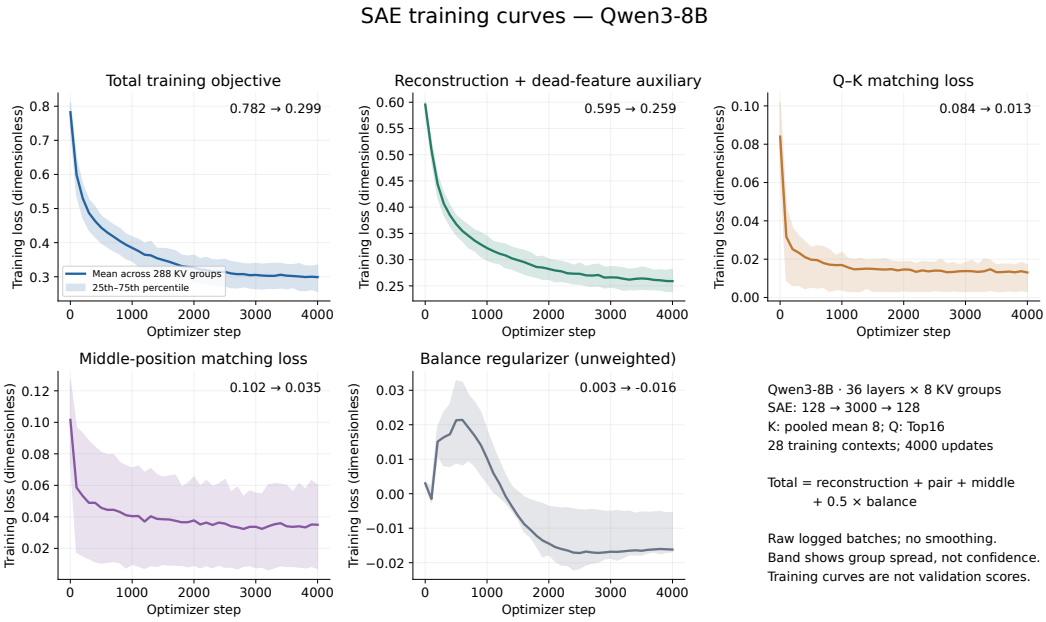


Figure 7: **SAE training losses.** Lines show mean training-minibatch losses across all SAEs; shaded bands show the interquartile range across groups. The reconstruction panel includes the dead-feature auxiliary term with weight $1/32$, and the balance panel shows the regularizer before its weight of 0.5 is applied. The balance regularizer uses lagged feature frequencies and can be negative. Curves connect 41 logged steps without smoothing.

Each update uses 128 queries and 128 keys for reconstruction, and 32 queries with one positive and 16 negative keys per query for matching. We use AdamW with learning rate 3×10^{-4}, zero weight decay, gradient clipping at 1, and a 50-step warmup followed by cosine decay to one tenth of the initial learning rate. The mean total training objective falls from 0.782 to 0.299; the reconstruction term including auxiliary residual reconstruction falls from 0.595 to 0.259, while query–key and middle-position matching losses fall from 0.084 to 0.013 and from 0.102 to 0.035, respectively. The total objective also includes soft matching terms that decay to zero during the first 1,000 updates.

C.3 FEATURE-BUCKET LOADS

Figure 8 illustrates bucket loads on a single NarrativeQA context with 65,055 historical tokens, using Qwen3-8B layer 0, KV group 0. A bucket's load is the percentage of historical tokens it contains, $100|\mathcal{B}_f|/H$. In this context and dictionary, most buckets are small, with a few larger buckets; the mean load is 0.267% and the maximum is 2.270%.

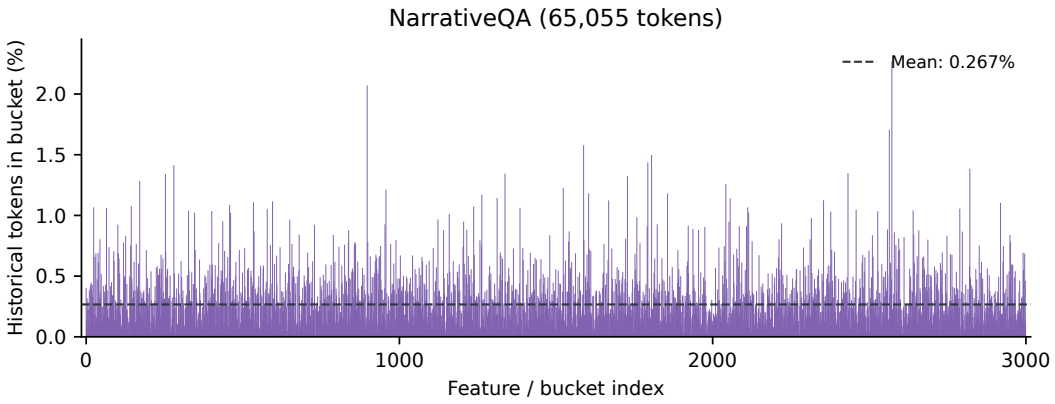


Figure 8: **Individual bucket loads for a NarrativeQA context.** Each vertical line shows one of 3,000 buckets in layer 0, KV group 0, without sorting or averaging across dictionaries. The dashed line marks the mean load of 0.267%, implied by eight memberships per token. The context uses $4\times$ YaRN scaling; all 3,000 bucket loads are displayed in their original feature-index order.