CommunityKV: Efficient Long-Context Decoding via Graph Partitioning

Joe McKenna
Amazon AGI
joemcke@amazon.com

Anastasios Alexandridis
Amazon AGI
aanst@amazon.com

Nathan Susanj
Amazon AGI
nsusanj@amazon.com

Jing Liu
Amazon AGI
jlmk@amazon.com

Abstract

Scaling Transformers to long contexts is constrained by the quadratic cost of self-attention and the linear growth of key-value cache memory transfer. Sparse attention mitigates this by retrieving only relevant tokens, but current approaches either require large-scale training or, within the training-free regime, rely on semantically coarse heuristics or expensive clustering that is difficult to update efficiently during decoding. We introduce CommunityKV, a framework that formulates sparse attention as a community detection problem. CommunityKV constructs a token graph from the QK^T scores already computed during standard prefill, and partitions the graph into communities to enable retrieval of semantically coherent token groups. A local update rule assigns newly generated tokens to communities in constant time, enabling sparse retrieval throughout streaming decoding without global re-partitioning. We evaluate CommunityKV on three Qwen3 model scales across three long-context benchmarks. With one graph per query head, CommunityKV delivers up to $1.25\times$ the end-to-end generation throughput of dense attention, while query-group graph aggregation yields up to $1.71\times$ with comparable accuracy.

1 Introduction

The capacity for Transformers [39] to process long context inputs powers a new generation of applications, ranging from repository-level code understanding [2, 31] and long-horizon agentic planning [10] to multimodal reasoning [41]. However, as the context length grows, standard self-attention encounters prohibitive bottlenecks: the quadratic cost of attention and the linear growth of key-value (KV) cache memory transfer.

Despite the increasing KV cache size for modern applications, empirical studies demonstrate that token prediction accuracy is driven by only a small fraction of relevant past tokens [50, 13]. Sparse attention methods attempt to exploit this redundancy by loading only a portion of the KV cache at each generation step [37, 27]. Recent architectural methods such as DeepSeek Sparse Attention [8] integrate sparsity into the training objective, but require large-scale training. Within the training-free regime, current approaches force a compromise: they either employ retrieval heuristics that suffer from unacceptable accuracy degradation [36] or rely on heavy preprocessing of the KV cache that is difficult to update during decoding [15].

In this work, we propose CommunityKV, a framework that formulates training-free sparse attention as community detection on a graph induced from attention scores. CommunityKV leverages the structure already present in the $QK^\top$ matrix to identify semantically coherent token groups. A local

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

update rule extends the partition to newly generated tokens in constant time, enabling streaming decoding without re-clustering.[1]

Our contributions are as follows:

- **Formulation:** We formulate sparse attention as community detection on a token graph induced from $QK^\top$ scores. A combined graph captures both direct query-key dependencies and second-order co-attention structure, enabling retrieval of semantically coherent token groups without auxiliary embeddings or vector-space clustering. A local update rule extends the partition to newly generated tokens, supporting streaming decoding without global re-partitioning.
- **Complexity analysis:** We show that graph construction is amortized within the standard prefill matrix multiplication, and partitioning adds only $\mathcal{O}(n \log n + nd)$. During decoding, community retrieval is sub-linear at $\mathcal{O}(\sqrt{n}\, d)$ per step, and adding a new token to the partition is $\mathcal{O}(d)$, independent of sequence length, achieving sub-linear retrieval with constant-time updates simultaneously.
- **Empirical results:** We evaluate CommunityKV across three Qwen3 model scales and three long-context benchmarks. In end-to-end measurements, the per-query-head configuration achieves 1.08–$1.25\times$ throughput with 17–29% additional peak memory, while query-group aggregation achieves 1.39–$1.71\times$ throughput with 0.5–8.5% additional peak memory and comparable accuracy.

2 Related work

Architectural vs. training-free sparse attention Recent methods integrate sparsity directly into the training objective, achieving large throughput gains. DeepSeek Sparse Attention (DSA) [8] scores queries against lightweight indexer keys to select a top-k subset of past tokens for full attention, while Native Sparse Attention (NSA) [47] combines block-level token compression with fine-grained token selection. However, these approaches couple sparsity to model training: the upfront cost amortizes only at very high inference volumes, and the resulting sparsity pattern is fixed at training time, limiting retrieval quality to the training distribution.

CommunityKV targets the complementary setting: training-free sparse attention. Within the training-free regime, existing approaches reduce KV cache traffic via two strategies. KV cache compression methods [50, 22, 25] enforce strict memory ceilings through token eviction or merging, but are inherently lossy: once discarded, information cannot be recovered. Selective retrieval methods instead preserve the full KV cache in memory and dynamically load only relevant tokens at each decoding step. The effectiveness of retrieval-based methods depends on the design of the sparsity mask, which typically employs either fixed or dynamic patterns.

Fixed-pattern sparse attention Early work on sparse attention relied on predefined, query-independent sparsity patterns. The Sparse Transformer [6] introduced factorized attention with strided patterns, achieving sub-quadratic $\mathcal{O}(n\sqrt{n}\, d)$ complexity. Longformer [3] and BigBird [48] combined sliding windows, dilated patterns, and global tokens to achieve linear $\mathcal{O}(nd)$ complexity. LongNet [9] extended dilated attention to billion-token sequences, and StreamingLLM [45] showed that retaining initial "attention sinks" stabilizes sliding window attention. Because these patterns are fixed in advance, they miss relevant tokens that fall outside the predefined structure.

Dynamic sparse attention Dynamic sparse attention frames the problem as query-dependent retrieval, typically using auxiliary data structures to index the KV cache and select relevant subsets at each step. A first family uses heuristics over contiguous token blocks. Quest [36] tracks channel-wise min-max values to estimate query-key overlap, while a similar training-based method MoBA [26] applies block-wise gating to filter context chunks. These methods avoid heavy auxiliary data structures but treat blocks of contiguous tokens as the retrieval unit: a high-scoring outlier within a block triggers retrieval of the entire block, including unrelated neighbors. A second family uses Locality Sensitive Hashing [18, 49, 5, 14] to bucket tokens using random projections. Because these projections are data-independent, they fail to account for the distribution of key embeddings; achieving high recall

[1] Code is available at `https://github.com/amazon-science/community-kv`.

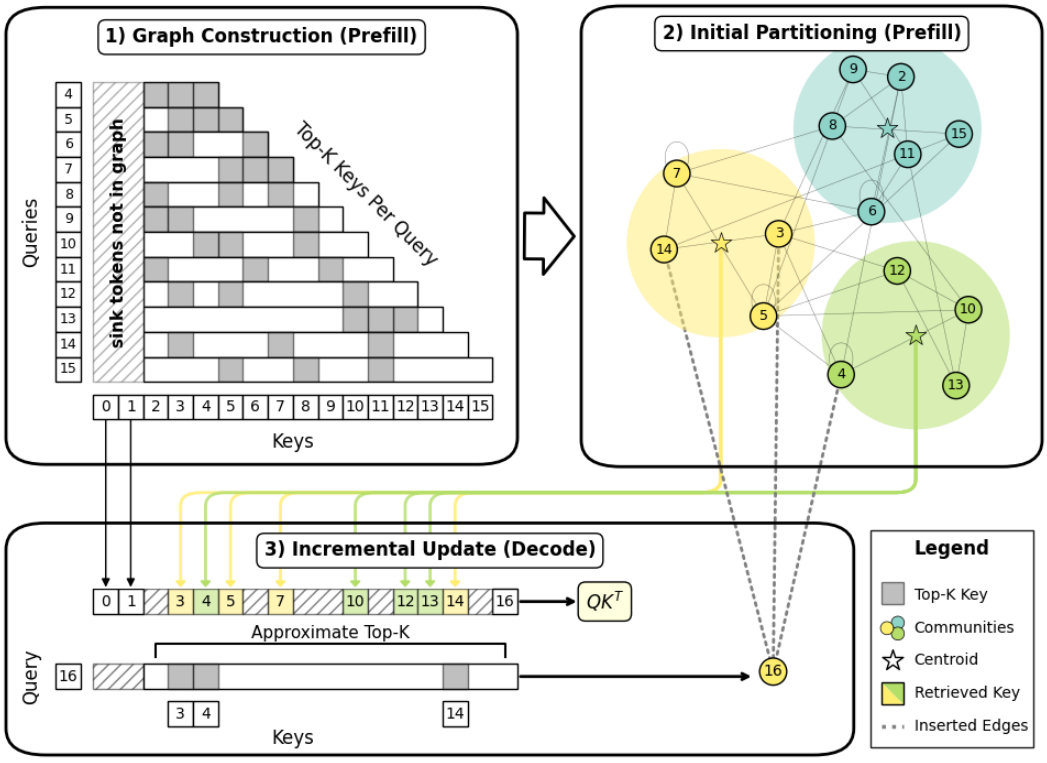

Figure 1: Overview of the CommunityKV framework. **(1) Graph Construction.** During prefill, we induce a sparse graph from the attention matrix by connecting each query to its top-κ keys (gray cells, $\kappa = 3$ shown). Sink tokens (hatched) are excluded from the top-κ pool. **(2) Initial Partitioning.** We partition the graph into semantic communities (colored regions) using the Leiden algorithm and compute a centroid (star) per community. **(3) Incremental Update.** At each decoding step (token 16 shown), we score the query against community centroids to retrieve a candidate subset, run exact attention on this subset together with sink tokens, and use the resulting top-κ keys to update the new token's edges and assign it to a community in constant time. For clarity, only direct attention $w^{(1)}$ relations are depicted; this corresponds to $\lambda = 1$ in the combined graph.

requires scaling the number of hash tables, which re-introduces significant overhead. A third family applies k-means clustering to the key or value vectors [33, 15, 40, 42]. These methods produce semantically coherent clusters, but k must be chosen a priori and centroids drift as new tokens are generated, forcing a trade-off between stale clusters and expensive re-clustering.

Comparison to GraphKV GraphKV [21] is the most closely related work, constructing a sparse attention graph and applying a "decay signal propagation" algorithm to evict semantically redundant tokens. The result is a compressed cache that every subsequent decoding step queries against. CommunityKV adopts the opposite philosophy. Instead of treating semantic similarity as a signal to be decayed, we treat it as the basis for community formation: we retain the full KV cache and dynamically retrieve communities for each query, allowing the model to access locally dense semantic structure without permanently deleting context.

3 Method

Our approach has three phases (Figure 1): (1) graph construction, where we build a sparse token graph from attention scores during prefill; (2) initial partitioning, where we cluster the graph into semantic communities; and (3) incremental update, where newly generated tokens are assigned to communities in constant time during decoding.

3.1 Graph construction

We construct a graph where nodes represent tokens and edges encode two kinds of relations: direct attention, where one token attends strongly to another, and co-attention, where two tokens are jointly attended by a common query. Let $\alpha \in [0,1]^{n\times n}$ be the attention score matrix over an n-token context, where α_{ij} is the score of query i attending to key j, and let $\mathcal{K}_i$ be the top-κ key positions for query i:

$$\mathcal{K}_i = \underset{j=|\mathcal{S}|+1,\ldots,i}{\arg\text{top-}\kappa}\ \alpha_{ij}. \tag{1}$$

We exclude the set of initial "sink token" positions $\mathcal{S}$, as they exhibit disproportionately high attention scores regardless of semantic relevance [44].

Direct attention graph The direct attention graph captures explicit query-key dependencies: a directed edge $i \to j$ records that query i attends strongly to key j:

$$w_{ij}^{(1)} = \alpha_{ij}\,\mathbf{1}\left(j \in \mathcal{K}_i\right) \tag{2}$$

where $\mathbf{1}(\cdot)$ is the indicator function.

Co-attention graph If two keys are strongly attended by the same query, they play similar roles in that query's context. The co-attention graph captures this by linking keys that co-occur in the top-κ set of a common query:

$$w_{ij}^{(2)} = \sum_{m=|\mathcal{S}|+1}^{n} \alpha_{mi}\,\alpha_{mj}\,\mathbf{1}\left(\{i,j\} \subseteq \mathcal{K}_m\right). \tag{3}$$

Combined graph We combine the two adjacency matrices into a single symmetric adjacency matrix:

$$w = \frac{\lambda}{2}\left[w^{(1)} + \left(w^{(1)}\right)^{\top}\right] + (1-\lambda)\,w^{(2)}, \tag{4}$$

with $0 \le \lambda \le 1$ controlling the balance between direct and co-attention relations. By construction, the graph is sparse: each of the n queries contributes κ edges to $w^{(1)}$ and at most κ^2 edges to $w^{(2)}$, so the total edge count satisfies $E = \mathcal{O}(n\kappa(1+\kappa)) = \mathcal{O}(n)$ for constant κ.

Theoretical justification The combined graph w admits a dynamical interpretation that motivates graph partitioning, grounded in the mathematical structure of self-attention. Because α is row-stochastic, it defines the one-step transition kernel of a discrete Markov chain over the token sequence. Symmetrizing as $(\alpha + \alpha^{\top})/2$ and combining with the second-order affinity matrix $\alpha^{\top}\alpha$ yields a symmetric kernel of a time-reversible random walk, and sparsifying via top-κ selection produces w, preserving the dominant transition paths with bounded sparsity ($E = \mathcal{O}(n)$).

In this framework, token communities correspond to metastable states of the walk: groups of tokens the walk is unlikely to escape over time. We partition the graph into these communities by maximizing modularity [28], which on the symmetric graph defined by w is equivalent to maximizing the linearized Markov Stability of the induced walk [20]. In practice, we employ the Leiden algorithm [38] to perform this graph partitioning.

3.2 Initial partitioning

Leiden community detection The Leiden algorithm [38] iterates three phases: fast local moving, refinement, and aggregation, to maximize modularity:

$$Q = \frac{1}{2m}\sum_{ij}\left(w_{ij} - \gamma\frac{w_i w_j}{2m}\right)\mathbf{1}(c_i = c_j), \tag{5}$$

where $m = \frac{1}{2}\sum_{ij} w_{ij}$ is the total graph weight, $w_i = \sum_j w_{ij}$ is the weighted degree of token i, and c_i is the community label of token i. The resolution parameter γ controls community granularity: larger values favor smaller, denser communities. Leiden improves upon Louvain [4] by guaranteeing connected communities.

We choose Leiden over alternative community detection algorithms for several reasons. Leiden empirically exhibits $\mathcal{O}(E)$ runtime [35, 38] and few iterations required for convergence; [4] report fewer than 5 across all tested networks for the analogous Louvain algorithm. We cap iterations at $r = \lfloor \log_{10} n \rfloor$, yielding $r \in [4, 7]$ for our evaluation contexts (10k–10M tokens). With graph sparsity bounded at $E = \mathcal{O}(n)$, graph partitioning costs $\mathcal{O}(rE) = \mathcal{O}(n \log n)$. We then compute community centroids in a single $\mathcal{O}(nd)$ pass, yielding a clustering cost of $\mathcal{O}(n \log n + nd)$ (Table 1). By contrast, Multipole Attention assigns n tokens to $\mathcal{O}(n)$ centroids via k-means, yielding $\mathcal{O}(n^2 d)$ clustering cost. Because pairwise similarity is already captured in the graph edges via $QK^\top$, our clustering operates on scalar weights rather than d-dimensional vectors.

Leiden's local-moving phase is embarrassingly parallel; each node's modularity-gain computation ΔQ (Eq. 7) depends only on its neighbors and community-level aggregates, so all nodes can be evaluated simultaneously. Recent parallel implementations exploit this structure to achieve a processing rate of 403M edges/s on a 3.8B-edge graph using 64 CPU threads [35], and NVIDIA's cuGraph [11] reports significant speedup over CPU baselines.

Leiden admits incremental update schemes that avoid global re-partitioning when edges are added [34, 23]. Our local update rule (Section 3.3) is a domain-specific specialization of this idea, exploiting the structure of autoregressive decoding to assign new tokens in $\mathcal{O}(d)$ time per step.

3.3 Incremental update

Community retrieval During decoding, we retrieve relevant tokens by scoring the latest query q_i against community centroids and aggregating positions from the top-ranking communities into a candidate set $\mathcal{R}_i$. We then form the sparse context $\mathcal{S} \cup \mathcal{R}_i \cup \{i\}$ by augmenting $\mathcal{R}_i$ with the attention sinks $\mathcal{S}$ and the current query position i, truncating to a fixed token budget. Because our graph satisfies $E = \mathcal{O}(n)$, the modularity resolution limit [12] implies the number of communities scales as $\mathcal{O}(\sqrt{n})$ (verified empirically in Figure 2), making retrieval cost $\mathcal{O}(\sqrt{n}\, d)$ (Table 1).

Edge approximation To integrate a new token i into the graph, we approximate its edge weights using only the retrieved set. We compute approximate attention scores and top-κ keys over $\mathcal{R}_i \cup \{i\}$:

$$\begin{aligned} \tilde{\alpha}_{ij} &= \text{softmax}(q_i^\top k_j)_{j \in \mathcal{R}_i \cup \{i\}}, \\ \tilde{\mathcal{K}}_i &= \underset{j \in \mathcal{R}_i \cup \{i\}}{\arg \text{top-}\kappa}\, \tilde{\alpha}_{ij}. \end{aligned} \tag{6}$$

Substituting these approximations into Eqs. (2) and (3) yields the edge updates. We add κ direct attention edges $w_{ij}^{(1)} = \tilde{\alpha}_{ij}$ for each $j \in \tilde{\mathcal{K}}_i$, and increment the κ^2 co-attention edges between pairs $j, k \in \tilde{\mathcal{K}}_i$ by $\Delta w_{jk}^{(2)} = \tilde{\alpha}_{ij}\tilde{\alpha}_{ik}$.

Greedy community assignment After integrating the new token i into the graph, we assign it a community label. We freeze the existing community assignments and consider two options: form a new singleton community $\{i\}$ or join a neighboring community $C \in \{c_j \mid j \in \tilde{\mathcal{K}}_i\}$. We pick the option that maximizes the modularity gain:

$$\Delta Q(i \to C) \propto \sum_{j \in C \cap \tilde{\mathcal{K}}_i} w_{ij} - \gamma \frac{w_i w_C}{2m}, \tag{7}$$

where $w_C = \sum_{j \in C} w_j$ is the total weight of community C.

Each decoding step touches at most $\kappa(\kappa + 1)$ edges, evaluates κ candidate communities, and updates one centroid in $\mathcal{O}(d)$. Since κ is a small constant, the total per-step update cost is $\mathcal{O}(d)$, independent of sequence length n (Table 1).

Stability of incremental updates Our local update rule maintains partition quality throughout decoding without global re-partitioning. Over a 4096-token generation horizon, modularity remains stable while attention mass decays gracefully, staying above 85% at a 4096-token budget (Appendix B.1). The KV cache and the routing graph are decoupled in CommunityKV: updates modify only the graph, leaving past key-value states untouched. A single global re-partitioning restores optimal modularity in the graph. Running Leiden every $\mathcal{O}(\sqrt{n})$ steps adds amortized $\mathcal{O}(\sqrt{n} \log n)$

Table 1: Computational complexity comparison. n is sequence length, d is hidden dimension. Note that our method achieves sub-linear retrieval $\mathcal{O}(\sqrt{n}d)$ without expensive clustering or update costs.

Method	**Initial Clustering**	**Per-step Update**	**Per-Step Retrieval**
Quest	$\mathcal{O}(nd)$	$\mathcal{O}(d)$	$\mathcal{O}(nd)$
Multipole	$\mathcal{O}(n^2 d)$	$\mathcal{O}(nd)$	$\mathcal{O}(nd)$
CommunityKV	$\mathcal{O}(n \log n + nd)$	$\mathcal{O}(d)$	$\mathcal{O}(\sqrt{n}\, d)$

cost per step, making periodic refresh feasible for unbounded generation. Over a shared 16,384-token PG-19 [30] continuation, repartitioning every 512 steps improves CommunityKV's final 512-token window top-1 agreement with dense attention from 68.05% to 82.81% (Appendix B.2).

3.4 Implementation

We fuse top-κ selection into the FlashAttention QK main loop using a WarpSelect-style selector [17]. For each query row, GPU threads maintain small sorted candidate queues, updated via compare-and-insert as each $b_q \times b_k$ score tile is computed. After the final key tile, the queues are merged with warp shuffles to produce the row's top-κ key indices and corresponding weights, which are normalized using the row's final softmax statistics. The total cost is $\mathcal{O}(n^2)$, matching the underlying prefill matrix multiplication (Appendix C.1). Empirically, this adds 6–8% to prefill wall time (Appendix C.3).

Leiden partitioning is pipelined with the forward pass: as soon as layer L's attention scores are available, its graphs are dispatched to asynchronous workers while layers $L{+}1, \ldots$ continue executing (Appendix C.2). The only exposed latency is the final layer's partition (0.37–0.45s across the Qwen3 family). We provide the fused top-κ selection complexity derivation, pipelined graph-partitioning schedule, component-level latency breakdown, and memory accounting in Appendix C.

4 Experiments

We evaluate CommunityKV along three axes: (1) accuracy compared to sparse attention baselines on long context benchmarks, (2) end-to-end throughput and peak GPU memory, and (3) ablation studies over key design parameters. Unless otherwise noted, we fix $\kappa = 8$, $\lambda = 0.5$, exclude the first 10 sink tokens from graph construction, and maintain one independent graph per query head. All models use greedy decoding (single run, deterministic) with a maximum of 128 generated tokens and YaRN RoPE scaling [29] to extend the model context. To ensure a fair comparison, we tune γ to produce a mean community size of 16, matching the baselines' retrieval granularity.

4.1 Experimental setup

4.1.1 Datasets

LongBench v2 We evaluate on LongBench v2 [1], a suite of challenging multiple-choice questions spanning code understanding, long-dialogue history, and multi-document QA.

BABILong We additionally evaluate on BABILong [19] QA1–QA5, a multi-hop reasoning benchmark requiring retrieval of 1–5 supporting facts from long context inputs.

RULER We further evaluate on RULER [16], a configurable long-context benchmark spanning retrieval, multi-hop tracing, aggregation, and question answering.

4.1.2 Models

Models We evaluate the Qwen3 family [46], specifically the 4B, 8B, and 14B variants.

4.1.3 Baselines

FlashAttention-2 (Dense) Exact FlashAttention-2 [7] serves as the lossless reference for both accuracy and wall-clock latency.

Table 2: Accuracy on LongBench v2 across token budgets (4k–32k) for GraphKV and CommunityKV. Dynamic community retrieval matches dense accuracy at $8\times$ lower budget than static eviction.

Qwen3-8B (Dense: 31.4)				
Method	4096	8192	16384	32768
GraphKV	3.4	10.5	18.5	22.5
CommunityKV	**31.1**	**31.2**	**31.4**	**31.4**

Table 3: Accuracy on LongBench v2 across token budgets (128–4096) at cluster size 16 for Qwen3 models (4B, 8B, 14B). CommunityKV accuracy improves with budget and reaches near dense parity at 4096 tokens across all model scales.

Model	**Method**	128	256	512	1024	2048	4096
Qwen3-4B (Dense: 27.4)	Quest	0.2	0.8	3.0	6.6	12.5	14.1
	Multipole	**25.8**	**26.2**	25.6	25.8	25.6	25.8
	CommunityKV	25.4	26.1	**25.8**	**26.6**	**26.2**	**27.4**
Qwen3-8B (Dense: 31.4)	Quest	5.6	9.1	15.1	17.5	22.5	24.7
	Multipole	**29.4**	**30.0**	28.4	29.8	28.2	28.8
	CommunityKV	28.5	28.8	**29.5**	**30.1**	**29.9**	**31.1**
Qwen3-14B (Dense: 32.0)	Quest	9.3	18.1	21.5	24.7	29.0	30.6
	Multipole	**30.4**	29.2	29.2	28.0	28.8	28.8
	CommunityKV	30.2	**30.0**	**30.5**	**30.7**	**29.7**	**32.0**

GraphKV As the most closely related graph-based method, GraphKV [21] constructs a sparse attention graph and permanently evicts low-scoring tokens during prefill. We include it to isolate the benefit of dynamic retrieval over static eviction.

We additionally compare against two training-free retrieval methods that span the efficiency-fidelity spectrum:

Quest Quest [36] partitions consecutive tokens into fixed blocks and tracks channel-wise min-max statistics for retrieval, representing the fast but semantically coarse end of the spectrum.

Multipole Attention Multipole [15] clusters keys via k-means, representing the high-fidelity end. It serves as a performance target that our lighter approach seeks to match without the extra compute cost.

We additionally compare against direct training-free KV retrieval methods (TokenSelect, FreeKV, and SparQ) [43, 24, 32] and eviction/compression methods (H2O, SnapKV, and StreamingLLM) (Appendix D).

4.2 Long context understanding

Graph-based retrieval vs. KV eviction Both CommunityKV and GraphKV construct sparse graphs from attention scores, but differ in how they use the graph: GraphKV evicts tokens permanently, while CommunityKV retrieves dynamically. To isolate this distinction, we compare the two on LongBench v2 with Qwen3-8B, sweeping GraphKV's budget from 4k to 32k tokens (Table 2). GraphKV requires an $8\times$ larger budget (32k tokens) to reach 22.5% accuracy, while CommunityKV achieves 31.1% with only 4k tokens, nearly matching the dense baseline. This confirms that semantic structure should guide dynamic retrieval rather than permanent context deletion.

Comparison between training-free retrieval methods We compare CommunityKV against Quest and Multipole on LongBench v2 across the Qwen3 family (4B, 8B, 14B), sweeping token budgets from 128 to 4096 at cluster size 16 (Table 3). CommunityKV nearly achieves parity with exact FlashAttention-2 across all model scales at a 4096-token budget; for example, 32.0% on Qwen3-14B, matching the dense upper bound exactly. Multipole performs strongly at restricted budgets (128–256 tokens) but plateaus as the budget increases, suggesting that k-means fails to capture the semantic structure required for complex reasoning. Quest scales with budget but trails dense by 1.4–13.3%

Table 4: Accuracy on BABILong QA1–QA5 across sparse-attention baselines. Results use Qwen3-8B with 64k contexts and a matched 4,096-token budget. We compare CommunityKV with direct retrieval methods (TokenSelect, FreeKV, and SparQ) and eviction/compression methods (SnapKV, H2O, and StreamingLLM).

Method	QA1	QA2	QA3	QA4	QA5	Avg.
Dense	63	32	30	55	63	48.6
CommunityKV	48	31	30	53	65	45.4
TokenSelect	58	28	28	48	56	43.6
FreeKV	52	28	23	48	58	41.8
SparQ	49	32	17	51	57	41.2
SnapKV	57	37	35	57	60	49.2
H2O	20	18	22	45	68	34.6
StreamingLLM	7	12	35	47	72	34.6

even at 4k tokens. CommunityKV bridges this gap: its accuracy tends to rise with budget, translating additional retrieval capacity directly into better accuracy.

Multi-hop retrieval At a matched 4,096-token budget, CommunityKV achieves the strongest average among direct retrieval methods and the second-best average among all sparse baselines (Table 4). It trails SnapKV on Qwen3-8B (45.4% versus 49.2%).

4.3 Computational efficiency

We measure end-to-end throughput and peak GPU memory for 32,768-token generations at 64k and 128k input contexts across Qwen3-4B, 8B, and 14B on an NVIDIA H200 (Table 5). Under the per-query-head configuration used for the primary accuracy evaluation, CommunityKV achieves 1.08–$1.25\times$ full-request speedup. Graph and centroid storage is 52–82% of KV-cache size, and total peak memory is 17–29% above dense. Query-group aggregation reduces auxiliary storage to approximately 12–15% of KV-cache size and peak-memory overhead to 0.5–8.5%, while increasing full-request speedup to 1.39–$1.71\times$. Throughput gains increase with context length for every model size. Query-group aggregation nearly matches per-query-head accuracy in the aggregation ablation (31.8% versus 32.0%; Table 6), making it the practical accuracy–systems tradeoff.

4.4 Ablations

We ablate the key design parameters of CommunityKV on LongBench v2 with Qwen3-14B at a 4096-token budget, starting from the default configuration ($\kappa = 8$, $\lambda = 0.5$, graph per query head, 10 sink tokens excluded) (Table 6).

Graph sparsity (κ) Accuracy peaks at $\kappa = 8$ (32.0%) and drops sharply at higher values (27.8% at $\kappa = 32$), suggesting that excessive connectivity obscures community boundaries. The lower accuracy with too few edges ($\kappa = 2$, 30.3%) similarly suggests graph fragmentation.

Direct vs. co-attention (λ) Accuracy peaks at $\lambda = 0.5$ (32.0%) and degrades toward pure direct attention ($\lambda = 1$, 28.5%) or pure co-attention ($\lambda = 0$, 28.8%), confirming that neither signal is sufficient alone.

Centroid budget By default, retrieval greedily fills the token budget with tokens from top-ranked communities. We test reserving part of the budget for additional community centroids as compressed summaries ($b_c \in \{0, 4, 16, 64, 256\}$). The greedy policy ($b_c = 0$, 32.0%) consistently outperforms centroid summaries (e.g., 28.6% at $b_c = 256$): full-token depth is more valuable than compressed breadth.

Graph aggregation In grouped-query attention (GQA) architectures, query heads share KV heads. We test three granularities for graph construction: per-query-head (default), query-group (one graph per KV group), and layer-wise (one graph per layer). Per-head (32.0%) and query-group (31.8%)

Table 5: Generation of 32,768 tokens on an H200 with batch size 1. Timing spans prefill through final decode and includes all model and CommunityKV operations. Auxiliary storage and peak memory are reported in GiB. Auxiliary storage comprises graphs and centroids, with percentages relative to KV-cache size; peak-memory percentages are increases over dense attention.

Model	Context	Graph Agg	Tok/s	Speedup	Aux	Aux/KV	Peak	Δ Peak
Qwen3-4B	64k	Dense	88.95	1.00×			32.80	
		Per-query head	99.73	1.12×	9.07	67.2%	39.69	21.0%
		Query group	133.86	1.50×	2.03	15.0%	33.83	3.1%
	128k	Dense	73.13	1.00×			47.19	
		Per-query head	90.77	1.24×	12.36	54.9%	60.65	28.5%
		Query group	125.37	1.71×	2.69	12.0%	51.22	8.5%
Qwen3-8B	64k	Dense	75.86	1.00×			40.81	
		Per-query head	83.97	1.10×	8.57	63.5%	48.14	18.0%
		Query group	107.33	1.41×	1.98	14.7%	42.72	4.7%
	128k	Dense	63.49	1.00×			61.95	
		Per-query head	79.08	1.25×	11.63	51.7%	72.74	17.4%
		Query group	103.02	1.62×	2.63	11.7%	63.86	3.1%
Qwen3-14B	64k	Dense	56.54	1.00×			58.69	
		Per-query head	60.94	1.08×	12.32	82.1%	69.52	18.5%
		Query group	78.95	1.39×	2.30	15.3%	58.97	0.5%
	128k	Dense	48.60	1.00×			79.95	
		Per-query head	56.90	1.17×	16.95	67.8%	97.16	21.5%
		Query group	75.50	1.55×	3.15	12.6%	84.49	5.7%

Table 6: Ablation studies on LongBench v2 (Qwen3-14B, token budget=4096). The default configuration (κ=8, λ=0.5, per-head graphs, sinks excluded) is marked with gray shading and achieves the best accuracy in every category.

Graph Sparsity (κ)	$\kappa = 2$	$\kappa = 4$	$\kappa = 8$	$\kappa = 16$	$\kappa = 32$
	30.3	31.1	**32.0**	28.7	27.8
Graph Weight (λ)	$\lambda = 0.0$	$\lambda = 0.25$	$\lambda = 0.50$	$\lambda = 0.75$	$\lambda = 1.0$
	28.8	29.0	**32.0**	29.0	28.5
Centroid Budget	0	4	16	64	256
	32.0	27.5	29.7	28.4	28.6
Graph Aggregation	No Agg.	Query Group	Layer-Wise	–	–
	32.0	31.8	21.5	–	–
Sink Tokens	Excluded	In-Graph	–	–	–
	32.0	31.3	–	–	–

perform comparably, while layer-wise collapses to 21.5%. Query-group aggregation is a viable memory-saving alternative.

Sink tokens Excluding sink tokens from the graph and always retrieving them (32.0%) outperforms including them in the partition (31.3%). Sinks attract attention from nearly all positions, creating high-degree hubs that bridge unrelated communities.

Isolating the contribution of Leiden We compare Leiden with two controls on the identical weighted graph: the connected-components control removes edges below a calibrated threshold and uses the surviving components as retrieval groups, while the top-k-neighborhood control seeds retrieval at the highest-scoring tokens and expands each seed by one graph hop to fill the budget. Both controls are calibrated to match CommunityKV's mean group size of 16. CommunityKV

Table 7: CommunityKV outperforms connected-components and top-k-neighborhood controls at every budget and model scale, highlighting the importance of Leiden partitioning.

Model	Method	128	256	512	1024	2048	4096
Qwen3-4B	Connected components	1.1	2.7	3.5	7.3	12.1	20.3
	Top-k neighborhoods	23.5	23.3	23.3	21.9	19.0	18.2
	CommunityKV	**25.4**	**26.1**	**25.8**	**26.6**	**26.2**	**27.4**
Qwen3-8B	Connected components	3.2	5.5	6.7	11.6	15.8	21.8
	Top-k neighborhoods	27.6	28.4	27.6	26.4	28.8	26.2
	CommunityKV	**28.5**	**28.8**	**29.5**	**30.1**	**29.9**	**31.1**
Qwen3-14B	Connected components	0.3	1.8	3.5	9.7	19.2	27.0
	Top-k neighborhoods	28.0	28.2	29.4	28.6	28.6	29.6
	CommunityKV	**30.2**	**30.0**	**30.5**	**30.7**	**29.7**	**32.0**

outperforms both at every budget and model scale (Table 7), isolating the benefit of Leiden from graph construction.

5 Conclusion

We presented CommunityKV, a training-free framework that formulates sparse attention as dynamic community detection on a token graph induced from $QK^\top$ scores. Partitioning this graph with the Leiden algorithm during prefill yields communities that serve as the retrieval unit during decoding, with sub-linear $\mathcal{O}(\sqrt{n}\,d)$ cost per step. A constant-time local update rule assigns generated tokens to communities without global re-partitioning, enabling streaming decoding at fixed update cost per step. End-to-end throughput reaches $1.25\times$ per query head and $1.71\times$ with query-group aggregation, which nearly matches per-head accuracy while reducing auxiliary storage from 52–82% to 12–15% of KV-cache size.

Limitations and future work Our greedy community assignment freezes prior assignments, which may cause partition drift over very long generation horizons; periodically re-partitioning every $\sqrt{n}$ steps mitigates this at amortized $\mathcal{O}(\sqrt{n}\log n)$ cost per step, but adds latency in practice. The edge approximation during decoding is restricted to the retrieved set, so true top-κ keys outside retrieved communities are missed, and retrieval recall affects the quality of incremental updates. Unstructured graphs and isolated-span tasks may also limit whole-community retrieval. These limitations suggest several natural extensions of the framework. First, incremental Leiden refinement [23], performing local moves in the neighborhood of newly inserted tokens rather than single-node greedy assignment, could better maintain partition quality during decoding over longer horizons. Second, the hierarchical structure that Leiden naturally produces via its aggregation phase could enable $\mathcal{O}(d\log n)$ retrieval at longer contexts where even $\mathcal{O}(\sqrt{n})$ communities become expensive to score. Finally, sharing community structure across adjacent layers, whose attention patterns are often correlated, could reduce both graph memory and pipelining depth.

Acknowledgments and Disclosure of Funding

Funding. Funding in direct support of this work was provided by Amazon. **Competing interests.** The authors' Amazon affiliations are disclosed above; the authors declare no additional competing interests.

References

[1] Yushi Bai, Shangqing Tu, Jiajie Zhang, Hao Peng, Xiaozhi Wang, Xin Lv, Shulin Cao, Jiazheng Xu, Lei Hou, Yuxiao Dong, et al. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3639–3664, 2025.

[2] Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, et al. Codeplan: Repository-level coding using llms and planning. *arXiv preprint arXiv:2309.12499*, 2023.

[3] Iz Beltagy, Matthew E Peters, and Arman Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[4] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[5] Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Yang Zhou, Jianyu Zhang, Niklas Nolte, Yuandong Tian, Matthijs Douze, Leon Bottou, Zhihao Jia, et al. Magicpig: Lsh sampling for efficient llm generation. *arXiv preprint arXiv:2410.16179*, 2024.

[6] Rewon Child. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[7] Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

[8] DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

[9] Jiayu Ding, Shuming Ma, Li Dong, Xingxing Zhang, Shaohan Huang, Wenhui Wang, Nanning Zheng, and Furu Wei. Longnet: Scaling transformers to 1,000,000,000 tokens. *arXiv preprint arXiv:2307.02486*, 2023.

[10] Lutfi Eren Erdogan et al. Plan-and-act: Improving planning of agents for long-horizon tasks. *arXiv preprint arXiv:2503.09572*, 2025.

[11] Alex Fender, Brad Rees, and Joe Eaton. Rapids cugraph. In *Massive Graph Analytics*, pages 483–493. Chapman and Hall/CRC, 2022.

[12] Santo Fortunato and Marc Barthelemy. Resolution limit in community detection. *Proceedings of the national academy of sciences*, 104(1):36–41, 2007.

[13] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive kv cache compression for llms. *arXiv preprint arXiv:2310.01801*, 2023.

[14] Insu Han, Rajesh Jayaram, Amin Karbasi, Vahab Mirrokni, David P Woodruff, and Amir Zandieh. Hyperattention: Long-context attention in near-linear time. *arXiv preprint arXiv:2310.05869*, 2023.

[15] Coleman Hooper, Sebastian Zhao, Luca Manolache, Sehoon Kim, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Multipole attention for efficient long context reasoning. *arXiv preprint arXiv:2506.13059*, 2025.

[16] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models? *arXiv preprint arXiv:2404.06654*, 2024.

[17] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547, 2021. doi: 10.1109/TBDATA.2019.2921572.

[18] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

[19] Yuri Kuratov, Aydar Bulatov, Petr Anokhin, Ivan Rodkin, Dmitry Sorokin, Artyom Sorokin, and Mikhail Burtsev. BABILong: Testing the limits of LLMs with long context reasoning-in-a-haystack. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[20] Renaud Lambiotte, Jean-Charles Delvenne, and Mauricio Barahona. Random walks, Markov processes and the multiscale modular organization of complex networks. *IEEE Transactions on Network Science and Engineering*, 1(2):76–90, 2014.

[21] Xuelin Li, Xiangqi Jin, and Linfeng Zhang. Graphkv: Breaking the static selection paradigm with graph-based kv cache eviction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 21910–21920, 2025.

[22] 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. *Advances in Neural Information Processing Systems*, 37:22947–22970, 2024.

[23] Chunxu Lin, Yumao Xie, Yixiang Fang, Yongmin Hu, Yingqian Hu, and Chen Cheng. Efficient maintenance of leiden communities in large dynamic graphs. *arXiv preprint arXiv:2601.08554*, 2026.

[24] Guangda Liu, Chengwei Li, Zhenyu Ning, Jing Lin, Yiwu Yao, Danning Ke, Minyi Guo, and Jieru Zhao. Freekv: Boosting kv cache retrieval for efficient llm inference. *arXiv preprint arXiv:2505.13109*, 2025.

[25] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

[26] Enzhe Lu, Zhejun Jiang, Jingyuan Liu, Yulun Du, Tao Jiang, Chao Hong, Shaowei Liu, Weiran He, Enming Yuan, Yuzhi Wang, et al. Moba: Mixture of block attention for long-context llms. *arXiv preprint arXiv:2502.13189*, 2025.

[27] Piotr Nawrot, Robert Li, Renjie Huang, Sebastian Ruder, Kelly Marchisio, and Edoardo M Ponti. The sparse frontier: Sparse attention trade-offs in transformer llms. *arXiv preprint arXiv:2504.17768*, 2025.

[28] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[29] Bowen Peng, Jeffrey Quesnelle, Honglu Fan, and Enrico Shippole. Yarn: Efficient context window extension of large language models. *arXiv preprint arXiv:2309.00071*, 2023.

[30] Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020.

[31] Stefano Rando, Luca Romani, Alessio Sampieri, Luca Franco, John Yang, Yuta Kyuragi, Fabio Galasso, and Tatsunori Hashimoto. Longcodebench: Evaluating coding llms at 1m context windows. *arXiv preprint arXiv:2505.07897*, 2025.

[32] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient llm inference. *Proceedings of the 41st International Conference on Machine Learning*, 2024.

[33] 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.

[34] Subhajit Sahu. Heuristic-based dynamic leiden algorithm for efficient tracking of communities on evolving graphs. *arXiv preprint arXiv:2410.15451*, 2024.

[35] Subhajit Sahu, Kishore Kothapalli, and Dip Sankar Banerjee. Fast leiden algorithm for community detection in shared memory setting. In *Proceedings of the 53rd International Conference on Parallel Processing*, pages 11–20, 2024.

[36] Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference. *arXiv preprint arXiv:2406.10774*, 2024.

[37] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *ACM Computing Surveys (CSUR)*, 55(6):1–28, 2022. doi: 10.1145/3530811.

[38] Vincent A Traag, Ludo Waltman, and Nees Jan Van Eck. From louvain to leiden: guaranteeing well-connected communities. *Scientific reports*, 9(1):1–12, 2019.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[40] Apoorv Vyas, Angelos Katharopoulos, and François Fleuret. Fast transformers with clustered attention. *Advances in Neural Information Processing Systems*, 33:21665–21674, 2020.

[41] Hengyi Wang, Haizhou Shi, Shiwei Tan, Weiyi Qin, Wenyuan Wang, Tunyu Zhang, Akshay Nambi, Tanuja Ganu, and Hao Wang. Multimodal needle in a haystack: Benchmarking long-context capability of multimodal large language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3221–3241, 2025.

[42] Shuohang Wang, Luowei Zhou, Zhe Gan, Yen-Chun Chen, Yuwei Fang, Siqi Sun, Yu Cheng, and Jingjing Liu. Cluster-former: Clustering-based sparse transformer for question answering. In *Findings of the association for computational linguistics: ACL-IJCNLP 2021*, pages 3958–3968, 2021.

[43] Wei Wu, Zhuoshi Pan, Chao Wang, Liyi Chen, Yunchu Bai, Tianfu Wang, Kun Fu, Zheng Wang, and Hui Xiong. Tokenselect: Efficient long-context inference and length extrapolation for llms via dynamic token-level kv cache selection. *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025.

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

[45] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024. URL `https://arxiv.org/abs/2309.17453`.

[46] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[47] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Yuxing Wei, Lean Wang, Zhiping Xiao, et al. 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, 2025.

[48] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33: 17283–17297, 2020.

[49] Amir Zandieh, Insu Han, Majid Daliri, and Amin Karbasi. Kdeformer: Accelerating transformers via kernel density estimation. In *International Conference on Machine Learning*, pages 40605–40623. PMLR, 2023.

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

Appendix

A Community scaling and resolution parameter

We empirically verify two properties of the graph partition on LongBench v2 using Qwen3-14B (Figure 2).

A.1 Resolution parameter γ

The left panel shows the relationship between the Leiden resolution parameter γ and the average community size. Increasing γ produces smaller, denser communities, providing a direct control over retrieval granularity. In our experiments, we tune γ to match a cluster size of 16 for fair comparison with each baseline (Section 4.2). On Qwen3-8B at a 4,096-token budget, scaling the selected γ over a $64\times$ range changes the predicted answer on at most 2 examples in a fixed set of 30, despite a $2.5\times$ change in mean community size.

A.2 Sub-linear community scaling

The right panel shows the number of communities as a function of sequence length. The empirical scaling closely follows $\mathcal{O}(\sqrt{n})$, consistent with the resolution limit of modularity optimization [12]. At 100k tokens, the graph partitions into approximately 50 communities. This sub-linear growth is what enables $\mathcal{O}(\sqrt{n}\,d)$ retrieval cost: scoring a query against $\mathcal{O}(\sqrt{n})$ centroids is substantially cheaper than the $\mathcal{O}(nd)$ cost of scoring against all tokens.

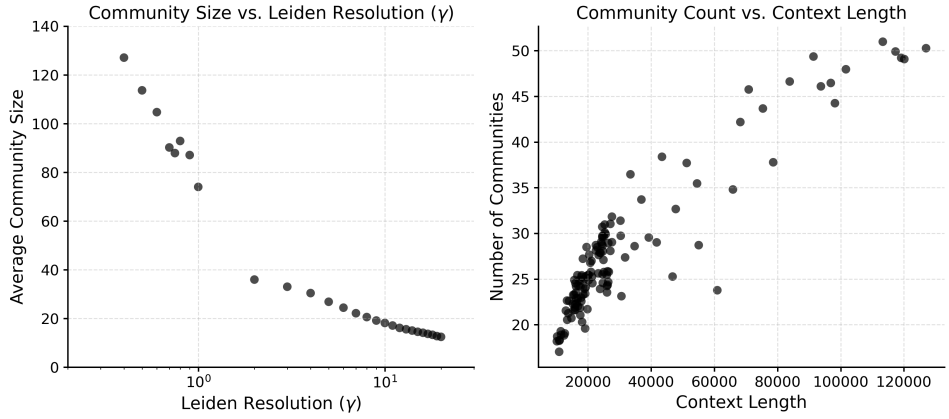

Figure 2: Empirical analysis of graph partition properties on LongBench v2 (Qwen3-14B). **Left:** Average community size as a function of the resolution parameter γ. **Right:** Number of communities vs. context length, confirming $\mathcal{O}(\sqrt{n})$ sub-linear scaling.

B Long-generation stability

B.1 Incremental partition drift

We stress-test the stability of our incremental update rule by tracking two metrics over a 4096-token generation horizon on LongBench v2 with Qwen3-14B, using the default configuration ($\kappa = 8$, $\lambda = 0.5$, 4096-token retrieval budget). No global re-partitioning is performed during generation.

We monitor (1) graph modularity Q (Eq. 5), which measures how well the current partition reflects the evolving graph structure, and (2) attention mass recall $\tilde{\mu}_\kappa = \sum_{j \in \mathcal{R}_i} \alpha_{ij}$, the fraction of the full attention distribution captured by the retrieved set at each decoding step.

Figure 3 shows both metrics as a function of generated tokens. Modularity remains in $[0.44, 0.50]$ throughout the 4096-token horizon, indicating that the greedy assignment rule continues to improve partition quality as new tokens reinforce existing community structure. Attention mass recall decays

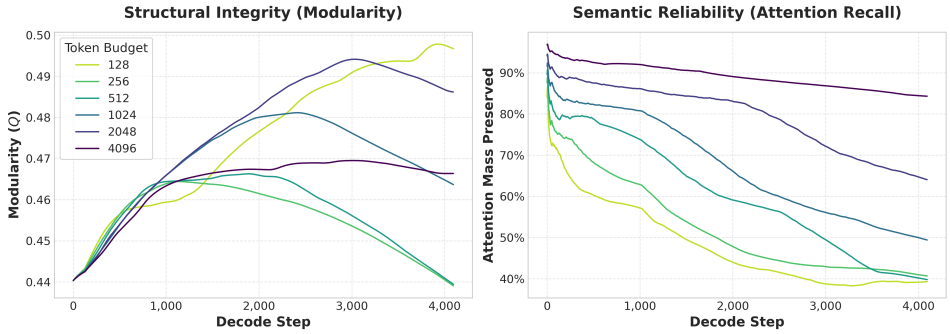

Figure 3: Partition stability over a 4096-token generation horizon (Qwen3-14B, LongBench v2). **Left:** Graph modularity Q remains stable and increases slightly as new tokens reinforce community structure. **Right:** Attention mass recall $\tilde{\mu}_\kappa$ at varying retrieval budgets. At the default 4096-token budget, recall stays above 85% throughout generation.

gracefully: at the 4096-token retrieval budget, recall remains above 85% after 4096 generated tokens. Smaller budgets (128–512 tokens) exhibit faster decay, as expected, but do not collapse catastrophically.

We evaluate Qwen3-8B on 65,536-token prompts from PG-19 and Python using the default per-query-head configuration, a 4,096-token retrieval budget, $\kappa = 8$, $\lambda = 0.5$, and 10 sink tokens. We compare purely incremental graph updates with periodic global repartitioning every 512 generated tokens (Refresh-512).

B.2 Fixed continuations

We teacher-force the same 16,384-token held-out continuation through every method, isolating accumulated approximation error from differences in generated histories. Results average five prompts per domain and are measured over the terminal 512-token window at each horizon. We report negative log-likelihood (NLL), KL divergence from dense, and dense top-1 agreement.

Table 8: Fixed-continuation stability over generation length. Lower NLL and KL are better; higher top-1 agreement is better.

		NLL			KL		Top-1 agreement	
Domain	**Horizon**	**Dense**	**Incr.**	**Refresh**	**Incr.**	**Refresh**	**Incr.**	**Refresh**
Python	2k	0.7045	0.7534	0.7375	0.0600	0.0311	94.34%	95.63%
	4k	0.4952	0.5478	0.5288	0.0694	0.0369	94.53%	95.59%
	8k	0.7264	0.8470	0.7729	0.1372	0.0418	92.30%	95.59%
	16k	0.7042	0.8297	0.7414	0.2510	0.0702	87.50%	94.30%
PG-19	2k	2.5129	2.6743	2.6158	0.1525	0.1144	82.81%	86.48%
	4k	2.3151	2.4675	2.3825	0.1991	0.0857	79.30%	86.80%
	8k	2.4834	2.7159	2.5853	0.2880	0.1109	75.23%	84.61%
	16k	2.6481	3.0052	2.7873	0.4401	0.1453	68.05%	82.81%

B.3 Sampled generation

We generate 8,192 tokens at temperature 0.7 and top-$p = 0.95$ using 10 prompts per domain and two sampling seeds. Because approximate methods produce different histories, we replay each generated history through dense attention and the generating method, then compare their distributions over the terminal 512-token window at 2k, 4k, and 8k tokens. NLL is reported only for the fixed-continuation experiment, where every method shares the same target tokens.

Table 9: Sampled-generation stability over generation length. Values compare dense attention with each method on the same generated history.

		KL		Top-1 agreement	
Domain	**Horizon**	**Incr.**	**Refresh**	**Incr.**	**Refresh**
PG-19	2k	0.0519	0.0521	93.96%	93.91%
	4k	0.0083	0.0196	99.34%	97.87%
	8k	0.0045	0.0097	99.75%	99.13%
Python	2k	0.0290	0.0150	98.13%	99.01%
	4k	0.0146	0.0105	99.21%	99.11%
	8k	0.0163	0.0053	99.03%	99.61%

C Implementation and systems analysis

This appendix provides the full complexity derivation for fused top-κ selection, the pipelining schedule that masks Leiden latency, and the memory accounting for per-head graphs.

C.1 Fused top-κ selection

We construct the initial sparse graph by identifying the top-κ keys for all n queries simultaneously, fusing this selection directly into the FlashAttention kernel with a WarpSelect-style register-resident selector [17]. As the kernel computes scores for each $b_q \times b_k$ tile of queries and keys, each GPU thread maintains a sorted queue of κ candidates using compare-and-insert operations. Each candidate contains a pre-softmax scaled QK logit and its key index. Block-level candidates are accumulated with warp shuffles to emit the row's top-κ indices and corresponding weights, which are normalized using the row's final softmax statistics. The total complexity aggregates two costs:

1. **Block-level sorting.** Every score in the causal attention matrix is tested against a queue threshold, requiring $\mathcal{O}(n^2)$ work. In the worst case, each score triggers an insertion requiring at most κ compare-and-insert operations, giving $\mathcal{O}(\kappa n^2)$ work.
2. **Global top-κ reduction.** Producing κ outputs requires a fixed-depth shuffle reduction for each selected candidate. Across all query rows, this costs $\mathcal{O}(\kappa n)$.

Combining these terms, the total graph construction worst case complexity is:

$$\mathcal{O}\big(\kappa n^2 + n\kappa\big) = \mathcal{O}(\kappa n^2). \tag{8}$$

Because κ is fixed in our experiments, this scales as $\mathcal{O}(n^2)$, matching the underlying causal prefill computation and adding no asymptotic overhead.

This bound is pessimistic because an insertion occurs only when a score exceeds the queue's current κ-th largest value. Total CommunityKV prefill overheads are 7.8% for Qwen3-4B, 7.7% for Qwen3-8B, and 7.2% for Qwen3-14B on 131k-token inputs (Table 10).

C.2 Pipelined graph partitioning

Naively, Leiden partitioning would extend the prefill critical path by the sum of per-layer partition costs. We instead dispatch each layer's partitioning to asynchronous workers immediately after its attention scores are computed, allowing it to run in parallel with the FFN of the same layer and the attention of subsequent layers (Figure 4). Because the FFN and following layers' attention together exceed the per-layer Leiden cost, all layers except the last have their partition cost fully masked.

The exposed latency is therefore the cost of partitioning a single layer's graphs, which we measure at 0.37–0.45 s across the Qwen3 family on an NVIDIA H100 (Table 10). This corresponds to 2–3% of total prefill wall time on a 130k-token input.

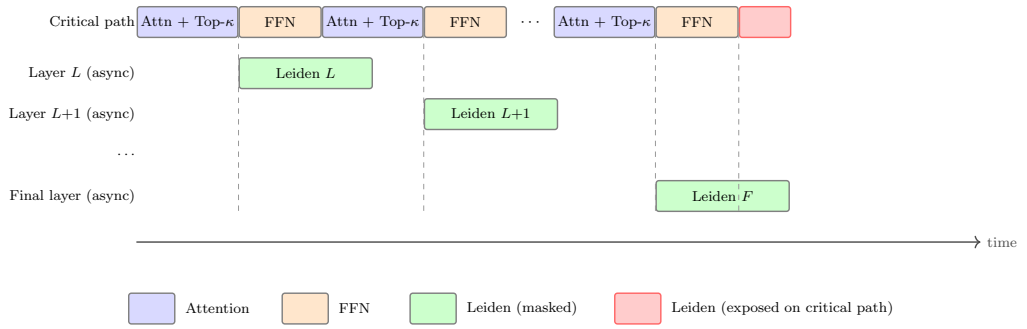

Figure 4: Pipeline schedule for graph partitioning during prefill. Leiden partitioning for each layer is dispatched asynchronously after its attention scores are computed, overlapping with subsequent layer execution. The red segment indicates the residual Leiden latency that remains exposed on the critical path when the final layer's partitioning has not completed by the time it is needed.

Table 10: Latency comparison between FlashAttention-2 (Dense) and CommunityKV using Hugging Face's `DynamicCache` on an NVIDIA H100 80GB at 131k context. Left: total prefill wall time with breakdown into fused top-κ selection and exposed Leiden partitioning latency. Right: average per-step decoding wall time (128 generated tokens) with breakdown into token retrieval and sparse attention. Only attention-related kernels are reported; the remaining per-step time is MLP and layer normalization.

Prefill – Total Wall Time (s)				**Decoding – Avg Per-Step Wall Time (ms)**			
	4B	**8B**	**14B**		**4B**	**8B**	**14B**
Dense	16.5	18.2	26.3	Dense	63	64	71
CommunityKV	17.8	19.6	28.2	CommunityKV	37	34	51
Top-κ Sel.	0.94	1.09	1.49	Retrieval	5.3	4.7	7.2
Leiden	0.37	0.45	0.42	Attention	6.1	5.5	8.2
Overhead	+7.8%	+7.7%	+7.2%	***Speedup***	**1.69**×	**1.86**×	**1.39**×

C.3 Component-level latency breakdown

C.4 Memory footprint

CommunityKV maintains an independent attention graph per query head (or per KV group or per layer under aggregation). For modern multi-layer, multi-head models, the aggregate footprint of these graphs warrants explicit accounting.

Per-graph footprint Each graph stores at most $|E| \leq 2n\kappa(1+\kappa)$ edges (Section 3.1). Storing edge weights in bfloat16 with 32-bit indices yields a per-graph cost of $8|E|$ bytes. At $n = 100$k tokens with $\kappa = 8$, we empirically observe $|E| \approx 1.15$M edges and a per-graph footprint of $\sim$9 MB; the observed footprint is $<$20 MB at $\kappa = 8$ across all sequence lengths in our evaluation.

Aggregate footprint Without aggregation, a model with L layers and H query heads maintains $L \cdot H$ independent graphs. For Qwen3-14B ($L = 40$, $H = 40$, with 8 KV heads per layer), this is 1600 per-head graphs. Under query-group aggregation (one graph per KV group, ablated in Section 4.4), this reduces to $L \cdot H_{\text{KV}} = 320$ graphs. At 100k tokens with $\kappa = 8$ in bfloat16, the aggregate footprint is $<$15 GB without aggregation and $<$3 GB with query-group aggregation, compared to $\sim$16 GB for the dense KV cache at the same context length and the 28 GB Qwen3-14B model weights in bfloat16. Graph memory under query-group aggregation is therefore a small fraction of total model state.

Centroid storage Community centroids are stored in d-dimensional bfloat16. With $\mathcal{O}(\sqrt{n})$ communities per graph (Appendix A), centroid storage is $\mathcal{O}(\sqrt{n}\,d)$ per graph and is dominated by the edge storage above at all evaluated context lengths.

Comparison to dense KV cache The dense KV cache scales as $2L \cdot H_{\text{KV}} \cdot n \cdot d_{\text{head}}$ in bfloat16. For Qwen3-14B at 100k context, this is approximately 16 GB, roughly an order of magnitude larger than the graph footprint under query-group aggregation. Because CommunityKV preserves the full KV cache (graphs serve as a routing index, not a replacement), graph memory is purely additive but small relative to the cache it indexes.

D Expanded baseline results

D.1 LongBench v2

Table 11: Accuracy on LongBench v2 across matched token budgets for direct retrieval, eviction/compression, and CommunityKV methods.

Method	**128**	**256**	**512**	**1024**	**2048**	**4096**
Qwen3-4B (Dense: 27.4)						
Quest	0.2	0.8	3.0	6.6	12.5	14.1
Multipole	25.8	26.2	25.6	25.8	25.6	25.8
TokenSelect	26.1	26.1	25.9	25.7	25.9	25.3
FreeKV	22.6	24.2	25.7	26.7	26.1	26.7
SparQ	25.6	25.0	24.8	26.4	26.6	27.2
StreamingLLM	25.0	25.0	25.6	25.6	26.2	26.2
H2O	24.8	24.8	25.2	25.8	26.2	27.0
SnapKV	25.2	25.2	25.4	25.6	25.8	27.4
CommunityKV	25.4	26.1	25.8	26.6	26.2	27.4
Qwen3-8B (Dense: 31.4)						
Quest	5.6	9.1	15.1	17.5	22.5	24.7
Multipole	29.4	30.0	28.4	29.8	28.2	28.8
TokenSelect	28.7	29.1	29.3	29.5	29.5	29.5
FreeKV	27.3	27.6	27.9	31.7	30.5	30.3
SparQ	28.2	27.8	28.8	29.8	29.4	29.2
StreamingLLM	24.6	26.2	28.0	28.0	27.4	28.0
H2O	26.2	26.6	27.8	28.0	28.2	29.2
SnapKV	26.2	27.8	28.2	28.2	28.4	30.0
CommunityKV	28.5	28.8	29.5	30.1	29.9	31.1
Qwen3-14B (Dense: 32.0)						
Quest	9.3	18.1	21.5	24.7	29.0	30.6
Multipole	30.4	29.2	29.2	28.0	28.8	28.8
TokenSelect	29.6	30.4	30.8	30.4	30.6	30.8
FreeKV	24.0	26.4	28.8	30.2	30.6	32.0
SparQ	29.8	30.2	30.4	30.0	30.6	30.8
StreamingLLM	27.0	29.4	30.8	31.2	31.2	31.4
H2O	28.6	30.4	31.2	31.4	31.8	31.2
SnapKV	27.8	30.2	30.6	31.4	31.8	32.0
CommunityKV	30.2	30.0	30.5	30.7	29.7	32.0

D.2 RULER

Table 12: RULER accuracy for direct retrieval methods on Qwen3-8B. MK-2, MQ, and VT denote Multikey-2, Multiquery, and variable tracking, respectively. Results use 64k contexts and a matched 4,096-token retrieval budget. The average covers the three reported tasks only.

Method	**MK-2**	**MQ**	**VT**	**Avg.**
CommunityKV	78.0	92.5	22.0	64.2
TokenSelect	81.0	91.8	17.4	63.4
FreeKV	64.0	93.5	46.4	68.0
SparQ	69.0	88.5	26.8	61.4

Table 13: RULER accuracy across the complete 13-task suite. Results use a matched 4,096-token active-token budget. Dense attention is included as a reference; reported values are unweighted averages over all 13 tasks.

Model	**Dense**	**StreamingLLM**	**H2O**	**SnapKV**	**CommunityKV**
Qwen3-8B	76	17	17	52	53

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state three contributions (formulation, complexity analysis, empirical results) that are directly supported by Section 3 (method and complexity bounds) and Section 4 (experiments on LongBench v2, BABILong, and RULER).

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 5 includes a "Limitations and future work" paragraph discussing partition drift from frozen greedy assignments and the lossy edge approximation that misses true top-κ keys outside retrieved communities.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The paper does not state formal numbered theorems but provides complete derivations for all complexity claims. Assumptions (constant κ, $\mathcal{O}(\sqrt{n})$ community scaling via the resolution limit [12]) are stated inline. The fused top-κ complexity derivation is provided in full in Appendix C.1.

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Section 4 specifies all hyperparameters ($\kappa = 8$, $\lambda = 0.5$, retrieval budget, sink tokens, decoding strategy, RoPE scaling), model identifiers, datasets, and hardware. Code is publicly available through the repository linked from the Introduction.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility.

In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: A public code repository is linked from the Introduction. All evaluation datasets (LongBench v2, BABILong, and RULER) and models (Qwen3 family) are publicly available.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 4 reports all hyperparameters (κ, λ, retrieval budget, sink token count, graph granularity), decoding strategy (greedy), context extension method (YaRN RoPE scaling), model identifiers, and the NVIDIA H200 used for end-to-end measurements. The implementation appendix identifies the NVIDIA H100 80GB used for component-level latency measurements.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

Justification: We use greedy decoding (deterministic) and evaluate on fixed benchmark datasets, so there is no randomness across method-comparison evaluation runs. Error bars are not reported for these comparisons because results are deterministic given the same model weights and inputs.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Table 5 reports batch size, context lengths, generation length, throughput, and peak GPU memory for the NVIDIA H200 end-to-end evaluation. Table 10 separately reports H100 80GB prefill and per-step decoding times at 131k context over 128 generated tokens.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This work proposes an inference-time efficiency method using publicly available models and benchmarks, with no human subjects, private data, or dual-use concerns.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: This work is a general-purpose inference efficiency method for Transformers with no direct path to specific negative societal applications beyond those inherent to the underlying language models themselves.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper releases an inference optimization method, not a pre-trained model or dataset. No new data is collected or generated.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The Qwen3 models and the LongBench v2, BABILong, and RULER benchmarks are cited to their original releases. Qwen3 is released under Apache 2.0, LongBench v2 under MIT, and BABILong and RULER under Apache 2.0.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We release code through the public repository linked from the Introduction. The repository includes instructions for reproducing experiments.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve human subjects research.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs are the subject of evaluation but are not used as a component of the core method development. No LLM was used in a non-standard way for the research methodology.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.