GraphKV: Breaking the Static Selection Paradigm with Graph-Based KV Cache Eviction

Xuelin Li[1] **Xiangqi Jin**[1] **Linfeng Zhang**[1,†]

[1]EPIC Lab, Shanghai Jiao Tong University
lxl.cooper@outlook.com, zhanglinfeng@sjtu.edu.cn

Abstract

Efficient Key-Value (KV) cache management is essential for processing long text sequences in large language models (LLMs), where memory constraints often limit performance. Conventional KV eviction strategies, such as top-k selection based on attention scores, depend on static heuristics that fail to capture the evolving implicit dependencies among tokens during inference. To overcome this, we propose GraphKV, a graph-based framework that redefines token selection for KV cache compression. In GraphKV, ***tokens*** are modeled as ***nodes*** with importance scores, and ***edges*** represent their ***similarity relationships***. Through a decay-signal-propagation mechanism, token importance is dynamically updated by propagating information across the graph, enabling adaptive retention of the most contextually significant tokens. GraphKV can be seamlessly utilized in existing KV cache eviction methods such as SnapKV and PyramidKV in a plug-and-play manner. *Our code is publicly available at* GitHub.

1 Introduction

Large language models (LLMs) have enhanced proficiency in processing long-text, improving performance in multi-turn dialogues (Chiang et al., 2023), document summarization (Zhang et al., 2024), question answering (Bai et al., 2023a), information retrieval (Zhu et al., 2023), and code generation (Li et al., 2025). New models such as GPT-4 (Achiam et al., 2023), Claude 3.5 (Anthropic, 2024), LLaMA 3.1 (Grattafiori et al., 2024), and Mistral Large 2 have extended token processing capacities beyond 128K. In the context of long-text processing, key value caching (KV cache) (Yang et al., 2024; Wu et al., 2024) is a crucial technique for improving the efficiency of LLMs. As the context length grows, the size of the KV cache increases linearly, leading to significant memory and computational overhead. Liu et al. (2024); Singhania et al. (2024) partially address this issue by leveraging low-rank decomposition to approximate the full-rank KV cache during the training phase. However, efficiently optimizing the key-value cache without extra training is vital for inference on long contexts under memory limitations, especially in standard deployment scenarios with a fixed model architecture.

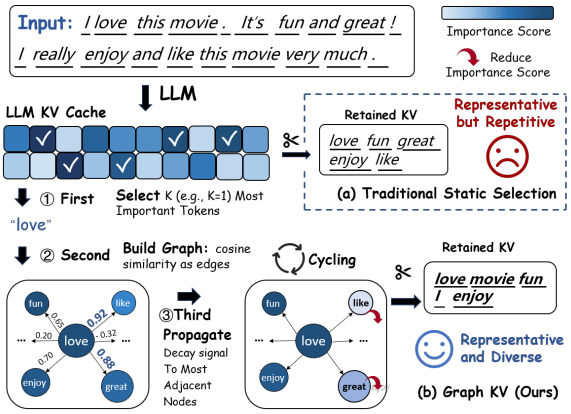

Figure 1: **Overview of GraphKV.** Tokens are defined as nodes with their initial importance scores. The cosine similarity between tokens is defined as the edges. GraphKV aims to firstly identify K most important nodes (*e.g., "love",* K=1), and then propagate **decay signal** to their most adjacent tokens (*e.g.,* "like" and "great") to obtain diverse retained KV. This propagation can be performed one or multiple times.

Building on training-free approaches to optimize the key-value cache, token eviction has emerged as a viable and effective method to compress the KV cache by selectively removing less important tokens. The core challenge in token eviction lies in accurately identifying and retaining the most critical tokens while discarding redundant ones, without compromising the model's performance.

Most existing token eviction strategies (Zhang et al., 2023; Li et al., 2024; Cai et al., 2024; Guo et al., 2024) reduce KV cache size via identifying

†Corresponding author

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 21899–21909
November 4-9, 2025

important tokens based on their importance scores (*e.g.,* attention scores). Then, as shown in Figure 1, a *static* selection is utilized to remove the KV cache of the redundant tokens and only retain the most important ones. Such a one-stage selection can identify the most representative tokens (*i.e.,* tokens which are most relevant to the following query), but always suffers from the problem of *duplication*, since the important tokens usually contain similar semantics, as shown in Figure 2 and will be discussed in Section 3. This observation highlights that retaining multiple tokens with high importance can lead to redundancy if they are highly similar, while the other less important but dissimilar tokens that contain more diverse semantic information can contribute uniquely to the overall context.

To bridge the gap, as shown in Figure 1, we propose GraphKV, which formulates KV cache as a graph, employing a decay-signal-propagation mechanism in the graph for importance propagation among tokens, enabling dynamic KV selection. Inspired by graph-based methods, where dynamic node updating mechanisms iteratively refine node states by aggregating information from neighbors, we construct a graph where tokens are defined as nodes, and the similarities among tokens are represented as edges. The importance scores defined by any previous KV cache eviction methods serve as the initial value for each node. Concretely, in the first step, we select only a few tokens (K) with the highest importance scores and compute their cosine similarity with other tokens to initialize the graph. In the second step, we perform a **decay-signal-propagation** over the graph, reducing the importance scores of the tokens that are most adjacent (*i.e.,* adjacent) to the previously selected important nodes. Such a negative propagation helps to reduce the possibility of retaining multiple highly similar tokens, and it can be performed by one or all multiple times. Finally, we remove the KV cache of tokens with lower scores after the propagation. Based on GraphKV, the retained KV cache can be both representative and diverse, minimizing the information loss from KV eviction.

Notably, GraphKV is not a new KV eviction method that introduces a new important score, it is a framework that can be directly applied to any previous KV cache eviction methods in a plug-and-play manner. The only additional computation in GraphKV is to compute the cosine similarity between the selected K most important tokens and other tokens. Since we define $K << N$, such a computation shows linear complexity as $O(N \times K) \approx O(N)$. which is ignorable. Experimental results validate GraphKV's effectiveness. For example, on the LongBench QA task, GraphKV outperforms the suboptimal Knorm (Devoto et al., 2024) method by 45.88% and achieves approximately 3% improvement over state-of-the-art models SnapKV (Li et al., 2024) and PyramidKV (Cai et al., 2024) with a KV cache size of 512 on the LLaMA-8B model. Additionally, GraphKV demonstrates superior performance on the Needle in a Haystack benchmark, further highlighting its ability to retain critical context details in long-context scenarios.

In summary, our contributions are as follows:

- Formulation of the KV cache as a graph, with tokens as nodes and semantic similarities as edges, enables dynamic modeling of token relationships for efficient eviction.
- A decay-signal-propagation mechanism in GraphKV iteratively propagates decay information to suppress redundant tokens, leveraging token similarity to prioritize diverse and representative context.
- Extensive experiments on LongBench and Needle-in-a-Haystack benchmarks demonstrate that GraphKV can be applied to any previous KV cache eviction methods in a plug-and-play manner with significant accuracy improvements under the same compression ratio.

2 Related works

KV Cache Eviction. With KV cache size scaling linearly with sequence length, efficient management of KV cache has garnered significant attention as an effective way to mitigate the memory constraints of LLMs when processing long contexts. Numerous studies have proposed token eviction strategies to reduce memory overhead while maintaining inference performance by selectively retaining only the most relevant tokens in the cache. Heavy Hitter Oracle (H2O) (Zhang et al., 2023) introduces a dynamic eviction policy that balances retention of recent and historically significant tokens, optimizing memory usage without sacrificing critical information. Similarly, SnapKV (Li et al., 2024) enhances efficiency by clustering significant KV positions based on attention scores of an observation window, while PyramidKV (Cai et al.,

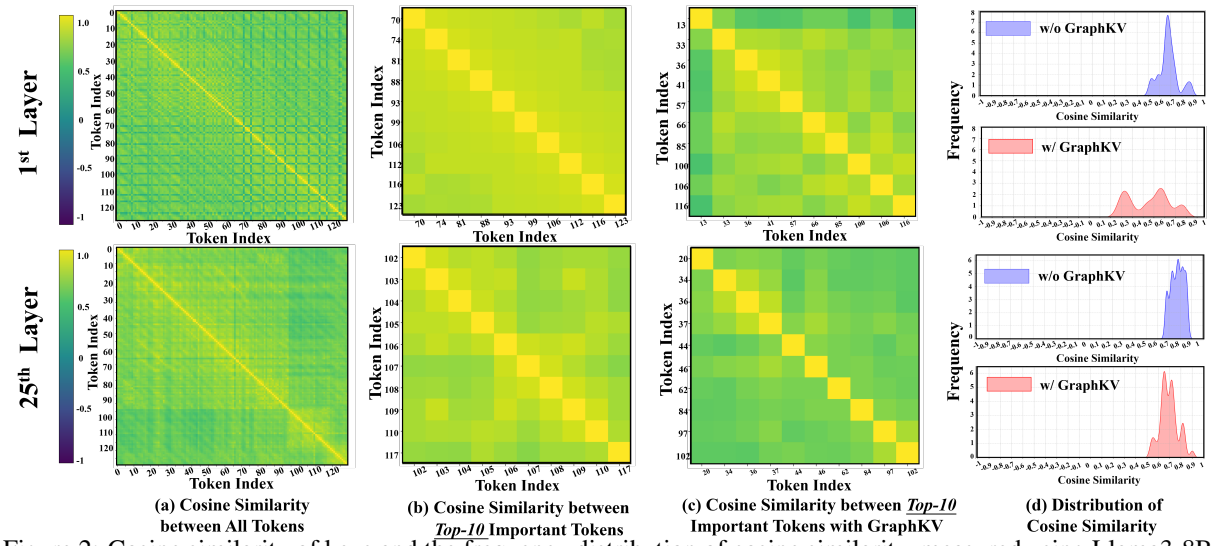

Figure 2: Cosine similarity of keys and the frequency distribution of cosine similarity, measured using Llama3-8B on a sample from the HotpotQA dataset in LongBench, with truncated 128 tokens for visualization. **(a)** illustrates the cosine similarity across all 128 tokens. **(b)** depicts the cosine similarity for the top-10 tokens with highest importance scores. **(c)** presents the cosine similarity for the top-10 tokens after GraphKV's decay signal propagation. **(d)** compares the frequency distribution of cosine similarity for the keys before (w/o GraphKV) and after (w/ GraphKV) GraphKV's decay signal propagation.

2024) adopts a layer-wise approach, allocating variable KV budgets according to each layer's attention demands. StreamingLLM (Xiao et al., 2023) enables models trained on finite attention windows to process infinite sequences without retraining by preserving initial attention sinks and recent local tokens. FastGen (Ge et al., 2023) employs an adaptive strategy, tailoring KV retention to the behavior of individual attention heads. Despite their success in reducing cache size, these methods predominantly rely on static importance scores, overlooking the dynamic, implicit relationships among tokens. This limitation can lead to suboptimal retention decisions, motivating the need for approaches that capture token interdependencies more effectively.

3 Observation

Most static-importance-score-driven methods utilize Top-K to select tokens based on their importance. However, Chen et al. (2024) notes that Top-K's reliance on attention score magnitude for importance estimation is biased and fails to capture the true distribution of important tokens. To better understand the limitations of static-importance-score-driven token eviction, we further analyzed the interplay between token importance and semantic similarity in the KV cache of LLaMA-8B, using a sample from the HotpotQA dataset in LongBench. Figure 2 visualizes the cosine similarity of key states and the frequency distribution of cosine similarity values for 128 truncated tokens, comparing the effects of GraphKV's decay signal propagation across two layers. Subfigures (a) and (b) show high cosine similarity among top-10 tokens by importance, with (b) having a brighter heatmap. Subfigure (d) reveals high mean and low variance in cosine similarity before GraphKV's decay signal propagation, indicating potential redundancy among high-importance tokens. This aligns with (Wang et al., 2024). Additionally, we identified several tokens with moderate importance scores but low key similarity, likely encoding critical semantic diversity. We extended our analysis across different model layers to verify the generalizability of this finding, and results show the phenomenon persists across layers.

This observation underscores a key insight: ***while high-scoring tokens are important, their similarity often leads to redundancy, whereas less similar tokens with moderate scores contribute unique semantic diversity.*** In token eviction for KV cache compression, prioritizing these diverse, non-redundant tokens is crucial for preserving critical context details under memory constraints. Therefore, our proposed GraphKV leverages this insight by dynamically refining importance scores through similarity-based decay signal propagation, as demonstrated in Subfigures (c) and (d). Specifically, (c) shows that after GraphKV's decay propagation, the heatmap exhibits significantly darker

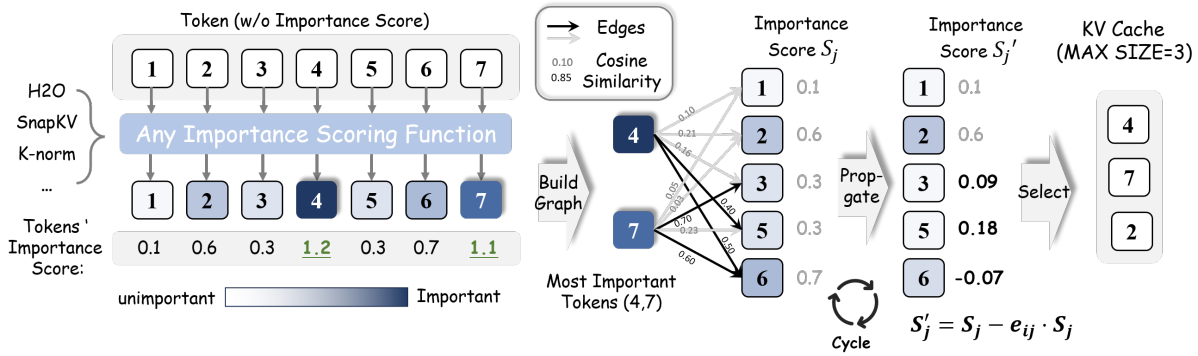

Figure 3: The pipeline of GraphKV. (I) **Initiate Importance Score**: We initialize importance scores for each token with any importance scoring function. (II) **Build Graph**: We select tokens with the highest importance scores as source nodes and compute the key cosine similarity between source nodes and other nodes, retaining the highest similarity values as edges in the graph. (III) **Decay Signal Propagation**: After constructing the graph, we perform multiple rounds of decay signal propagation on nodes adjacent to source nodes, attenuating their importance scores based on edge weights to reduce semantic redundancy. (IV) **Select Important Tokens**: Based on the updated scores, we reselect the most important tokens to balance significance and diversity for the compressed KV cache.

colors, indicating a notable reduction in the key similarity among the top-10 tokens. Moreover, (d) shows the post-propagation distribution exhibits a lower mean and reduced variance compared to the pre-propagation distribution, further confirming the decline in key similarity distribution.

4 Methodology

4.1 Problem Formulation

Token eviction aims to retain a minimal subset of the KV cache while preserving as much contextual information as possible. Formally, we consider an LLM with M transformer layers processing a sequence of prompt tokens $\mathbf{X}_{input} = [x_1, ..., x_n]$, where n represents the total number of tokens in the input sequence. For the l-th layer ($l \in 0, 1, ..., M-1$), the full key and value matrices are denoted as $\mathbf{K}^l, \mathbf{V}^l \in \mathbb{R}^{n \times d}$, where d is the hidden dimension of the model. The objective of token eviction is to identify compressed sub-matrices $\mathbf{K}_s^l, \mathbf{V}_s^l \in \mathbb{R}^{k_l \times d}$, where $k_l < n$ is the layer-specific cache budget, while minimizing performance degradation. Formally, given a dataset D and a performance metric $S_{core}(\cdot)$, the compressed model using $\mathbf{K}_s^l$ and $\mathbf{V}_s^l$ is expected to attain similar results with the full model: $\mathrm{S}_{\mathrm{core}}(\mathbf{K}_s^l, \mathbf{V}_s^l, \mathcal{D}) \approx \mathrm{S}_{\mathrm{core}}(\mathbf{K}^l, \mathbf{V}^l, \mathcal{D})$.

4.2 Sparse Graph Building

Note that GraphKV is a graph-based framework designed to dynamically manage the Key-Value(KV) cache by modeling token relationships using a graph structure. First of all, we define the KV cache as a weighted graph $G = (O, E)$. Specifically, each token x_i in the input sequence corresponds to a node $o_i \in O$, with an initial importance score s_i. Notably, this score can be initialized flexibly using important indicator of diverse established methods, such as attention-based scores from SnapKV, PyramidKV and CAKE, or KNorm, ensuring compatibility with prior high-performing frameworks. Inspired by the observation in 3, we model the cosine similarity between the keys of token pairs as an edge $e_{ij} \in E$ in the weighted graph G. This is formally expressed as followed:

$$e_{ij} = \frac{\langle k_i, k_j \rangle}{\|k_i\| \|k_j\|} \tag{1}$$

where k_i and k_j are key vectors of the token pairs x_i and x_j respectively.

It is worth noting that computing the similarity for all node pairs in the graph incurs significant computational overhead, particularly when processing long input sequences with a large number of tokens. To address this, we further sparsify the graph. Since the graph contains numerous nodes with low importance scores, which we consider to be tokens with minimal semantic relevance, we isolate these low-scoring nodes by removing their edges. Conversely, we prioritize the similarity relationships among the more significant tokens. Therefore, we adopt a top-k selection strategy to identify the most important source nodes, as follows:

$$O_{\text{source}} = \{o_i \mid s_i \text{ ranks in top-}k, \; i \in \{1, \ldots, n\}\} \tag{2}$$

where k is a hyperparameter tied to the layer-specific cache budget k_l. We compute the similarity only between these selected source nodes O_{source} and all other nodes, thereby completing the sparsification of the graph. Sparse graph structure building approach enables GraphKV to efficiently and effectively capture token relationships within the KV cache.

4.3 Decay Signal Propagation

Once the sparse graph $G = (O, E)$ is constructed, we introduce a dynamic decay-signal-propagation mechanism to refine token importance scores and eliminate semantic redundancy caused by token similarity. This process leverages the graph structure to propagate redundant signal across nodes, ensuring that the retained tokens are both significant and contextually diverse.

Initially, for each node $o_i \in O_{\text{source}}$, we define its neighborhood as the set of nodes o_j whose edge weights are among the top-m:

$$N(o_i) = \{o_j \mid e_{ij} \geq e_{i(m)}, j \neq i\}, \quad (3)$$

where $e_{i(m)}$ is the m-th largest edge weight among $\{e_{ij} \mid j \neq i\}$, m is a hyperparameter indicating the number of nodes in the neighborhood. This step ensures that decay score propagation targets tokens with high semantic overlap while preserving the diversity of retained tokens.

Furthermore, to prevent the over-representation of similar tokens, we apply a decay function to the importance scores of nodes in $N(O_i)$. The decay is proportional to their similarity to the source node, reducing the scores of redundant tokens. For a node $o_j \in N(o_i)$, the decayed score after one round of propagation as followed :

$$s'_j = s_j - \cdot e_{ij} \cdot s_j, \quad (4)$$

where s'_j represents the refined scores. To capture broader contextual dependencies, we extend this process over T rounds of propagation (e.g., $T = 3$), where in each round t, the updated score of a node o_j is computed as:

$$s_j^{(t)} = s_j^{(t-1)} \cdot \prod_{o_i \in O_{\text{source}}, o_j \in N(o_i)} (1 - e_{ij}) \quad (5)$$

with $s_j^{(0)} = s_j$, we aggregate cumulative decay from multiple source nodes to heavily suppress tokens similar to several retained ones. Finally, after T rounds, the updated scores $s_j^{(T)}$ determine the final k_l tokens for the compressed KV cache sub-matrices $\mathbf{K}_s^l$ and $\mathbf{V}_s^l$, retaining those with the highest $s_j^{(T)}$ to balance importance and diversity.

The decay-signal-propagation mechanism innovates over static-importance-score-driven methods by dynamically updating scores across multiple rounds, suppressing redundancy through similarity-based decay and offering flexibility by integrating initial scores from prior frameworks.

5 Experiment

5.1 Experimental Setup

5.1.1 Baseline Methods

We integrate GraphKV with five state-of-the-art methods: **CAKE** (Qin et al., 2025) considers the temporal and spatial aspects of attention, **SnapKV** (Li et al., 2024) clusters recent attention, **PyramidKV** (Cai et al., 2024) employs a budget allocation strategy, **H2O** (Zhang et al., 2023) uses cumulative attention and **KNorm** (Devoto et al., 2024) applies L_2 norm of keys. For more detailed information, please refer to Appendix A.1.

5.2 Evaluation on LongBench

The evaluation results for the LongBench dataset are presented in Table 1. We evaluated three state-of-the-art models Llama2-7B-Chat, Llama3-8B-Instruct, and Mistral-7B-Instruct-v0.2, each configured with a fixed KV cache size of 512. As shown in Table 1, a full KV cache yields the highest performance but is impractical for long-context applications due to its substantial GPU memory requirements. In contrast, our proposed GraphKV method achieves superior performance across all three models, surpassing other methods in the average score across 16 tasks. GraphKV shows a significant improvement on Retrieval-Based Passage (RB-P) dataset, evaluating the ability to understand cross-file dependencies, retrieve relevant code snippets, and generate accurate code completions in a multi-file programming context, while surpassing full-context model performance with only 10% of the full budget. This demonstrates its effectiveness for long-context inference. Furthermore, integrating GraphKV with three token eviction strategies enhances performance, highlighting its flexibility and compatibility. Overall, GraphKV offers an effective and flexible solution for long-context inference, significantly reducing GPU memory usage while maintaining high performance.

Method	Information Localization Task						Information Aggregation Task										Avg.
	Single-Document QA			Multi-Document QA			Summarization			Few-shot Learning			Synthetic		Code		
	NrtvQA	Qasper	MF-en	HotpotQA	2WikiMQA	Musique	GovReport	QMSum	MultiNews	TREC	TriviaQA	SAMSum	PCount	PRe	Lcc	RB-P	
	F1	F1	F1	F1	F1	F1	R-L	R-L	R-L	Acc (CLS)	F1	R-L	Acc (EM)	Acc (EM)	Edit Sim	Edit Sim	
Llama2-7B-Chat, KV Size = 512																	
Full	*18.39*	*20.10*	*35.67*	*31.25*	*25.73*	*10.64*	*25.67*	*20.89*	*26.34*	*64.00*	*83.38*	*40.90*	*5.50*	*10.00*	*58.67*	*53.00*	*33.13*
KNorm	6.72	8.10	7.94	8.38	7.06	2.72	16.60	15.69	19.69	19.50	27.26	11.86	4.50	1.67	28.60	22.13	13.03
+ GraphKV	12.57	15.55	20.01	26.15	20.87	6.52	18.86	18.82	20.61	31.50	79.21	33.43	5.00	8.00	44.35	41.67	**25.20**
SnapKV	16.01	19.11	32.40	32.25	24.18	10.47	19.96	20.97	23.50	62.00	82.70	39.11	6.00	11.00	58.06	53.50	31.95
+ GraphKV	16.39	20.12	33.58	33.44	25.19	10.66	20.23	21.09	23.38	62.00	83.21	39.56	6.00	11.00	57.65	54.15	**32.35**
PyramidKV	17.22	19.82	34.15	32.29	24.41	10.08	20.28	20.47	23.46	62.50	83.10	38.61	6.00	11.50	57.17	51.62	32.04
+ GraphKV	16.39	19.42	35.62	32.60	25.85	9.91	20.43	20.72	23.64	62.50	83.43	39.16	6.00	11.50	57.11	51.77	**32.25**
Llama3-8B-Instruct, KV Size = 512																	
Full	*25.56*	*32.21*	*39.65*	*43.56*	*35.29*	*21.14*	*28.73*	*23.36*	*26.63*	*74.00*	*90.48*	*42.36*	*4.80*	*69.25*	*57.03*	*52.38*	*41.65*
KNorm	10.00	7.98	11.05	11.90	8.92	4.83	21.24	18.39	23.43	40.00	51.87	17.14	5.42	47.89	28.22	21.92	20.64
+ GraphKV	19.51	16.02	24.05	25.45	14.71	11.01	21.29	20.29	23.47	43.50	85.50	31.52	4.05	61.60	38.11	36.00	**29.76**
SnapKV	23.45	23.37	37.14	42.60	34.60	20.08	22.08	22.61	24.06	70.50	90.52	39.88	5.81	69.50	59.50	53.87	39.97
+ GraphKV	23.68	25.37	37.55	43.21	35.72	20.94	22.57	22.86	24.02	70.50	90.33	39.48	5.59	69.25	58.42	56.10	**40.35**
PyramidKV	24.59	21.90	36.11	42.51	33.01	20.09	22.28	22.70	23.83	69.00	90.42	40.66	5.78	69.25	57.41	53.91	39.58
+ GraphKV	24.12	23.57	37.05	44.13	33.59	20.67	22.31	22.87	24.07	70.00	90.52	40.09	5.78	69.12	58.05	55.46	**40.09**
Mistral-7B-Instruct-v0.2, KV Size = 512																	
Full	*26.81*	*33.19*	*49.26*	*43.02*	*27.12*	*18.78*	*32.80*	*24.16*	*27.02*	*71.00*	*86.23*	*42.64*	*2.75*	*86.98*	*55.09*	*53.01*	*42.49*
KNorm	7.34	8.53	11.93	9.44	7.16	3.91	22.22	16.81	22.43	35.75	21.25	9.27	3.59	6.42	29.13	23.29	14.91
+ GraphKV	8.98	7.75	12.50	7.87	7.83	3.56	21.58	18.44	22.66	45.50	26.79	12.61	4.24	5.17	32.22	26.29	**16.50**
SnapKV	24.71	27.92	49.01	38.39	25.02	17.53	23.69	23.59	24.61	67.00	85.77	41.90	2.81	88.18	53.12	50.60	40.24
+ GraphKV	25.21	29.21	49.51	39.15	25.84	18.15	23.15	23.75	24.82	67.00	86.23	43.32	2.75	88.36	52.89	51.23	**40.60**
PyramidKV	24.25	26.40	48.48	40.22	24.97	17.62	23.18	23.33	24.40	67.50	85.73	41.64	3.03	87.64	53.90	50.26	40.16
+ GraphKV	24.62	28.59	49.36	42.08	25.83	18.03	23.16	22.82	24.07	67.50	85.20	41.93	2.57	87.56	54.18	50.57	**40.50**

Table 1: Performance comparison of GraphKV on LongBench.

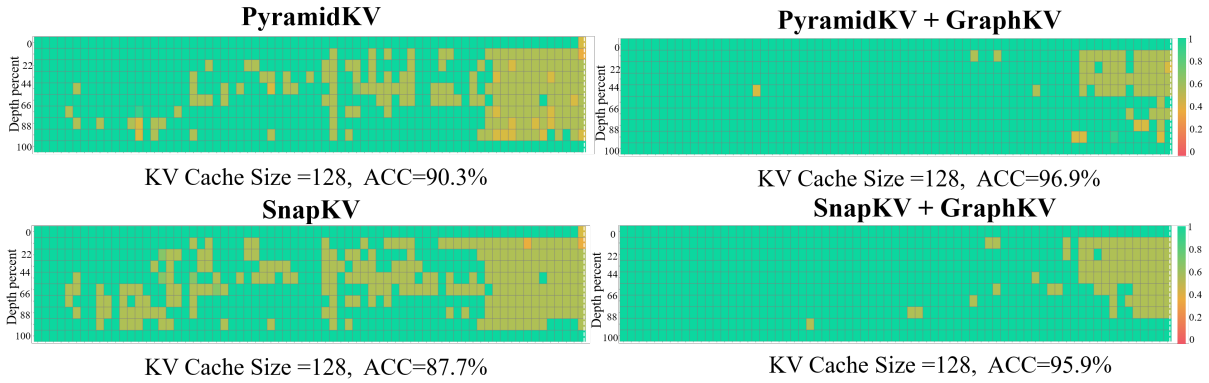

Figure 4: Results of the Fact Retrieval Across Context Lengths ("Needle In A HayStack") test in LlaMa-3-8B-Instruct with 8k context size in 128 KV cache size. The vertical axis of the table represents the depth percentage, and the horizontal axis represents the token length.

5.3 Evaluation on Needle In a Haystack

In this experiment, we assess the Llama3-8B model under resource constraints, with a context length of 8,000 tokens and a KV cache size of 128. Figure 4 shows retrieval accuracy on the Needle In a Haystack benchmark comparing PyramidKV, SnapKV, and Knorm with/without GraphKV integration. Notably, PyramidKV and SnapKV see notable improvements with GraphKV: PyramidKV's accuracy rises from 90.3% to 96.9% (+6.6%), and SnapKV's from 87.7% to 95.9% (+8.2%). These results highlight GraphKV's effectiveness in enhancing retrieval performance for long-context tasks under resource constraints.

6 Ablation

To further evaluate the design choices of GraphKV, we perform ablation studies to study the different components, including the choice of the number of source nodes, the number of adjacent nodes, and the number of propagation rounds, with experiments on some datasets in Longbench.

6.1 Effect of Adaptive Source Nodes

To evaluate the impact of the graph propagation range, we constructed sparse graphs with varying numbers of source nodes and conducted propagation tests. Specifically, we utilized a proportion of the KV cache budget as the number of source nodes. As shown in Figure 5 (a), a proportion of 0.3 $\times$ the KV cache budget achieved the best performance ($0.3 \times B$). As the number of source nodes increased, the performance declined. This is because the number of source nodes is positively correlated with the number of nodes affected by the score updates, and a greater number of influenced nodes tends to introduce noise into the token importance distribution.

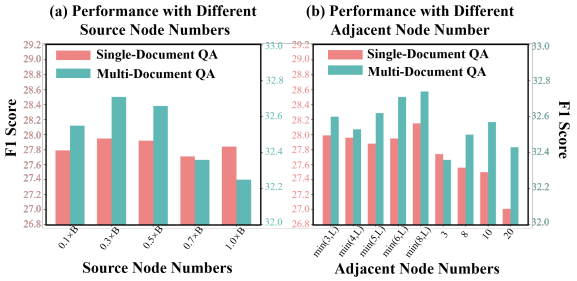

Figure 5: Impact of the number of adaptive source nodes and the number of adjacent nodes. "B" denotes the budget of the KV cache size. "L" denotes the length of input tokens $\times 10^{-3}$.

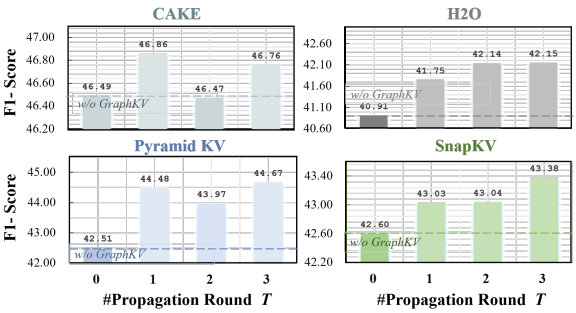

Figure 6: Impact on the number of propagation rounds of GraphKV with decay signal propagation.

6.2 Effect of Adjacent Nodes

In addition to test the impact of the number of source nodes, we also explored the effect of the number of adjacent nodes for each source node. To adapt to query inputs of varying lengths, we set the number of adjacent nodes as the minimum between a predefined value and the query length. As shown in Figure 5 (b), the adaptive approach to determining the number of adjacent nodes generally outperforms a fixed setting. Under a fixed setting, increasing the number of adjacent nodes leads to poorer performance, which further indicates that a greater number of influenced nodes introduces noise into the token importance distribution.

6.3 Effect of Propagation Round

To evaluate the impact of propagation rounds on GraphKV, we integrated it with state-of-the-art methods, including CAKE, PyramidKV, SnapKV, and H2O. We conducted experiments under four settings: no propagation (T=0) and propagation rounds T=1, 2, and 3.

As shown in Figure 6, comparing T=0 and T=1, all four methods exhibited the most significant performance improvements after the first round of decay signal propagation. Notably, PyramidKV exhibited a significant performance, jumping from 42.51 to 44.48, surpassing FullKV after just one round. However, CAKE and PyramidKV showed slight performance dips at T=2, though still outperforming the non-propagated baseline. These results indicate that while performance fluctuates slightly with increased propagation rounds, overall, propagation substantially enhances performance.

As Figure 7 shows, subfigures (a)–(d) show key vector distributions projected onto PCA components at steps T=0, 1, 2, and 3, respectively. Vectors are colored by normalized importance scores using a high-contrast colormap, with top5% high-score tokens highlighted as larger, opaque markers. At T=0, key vectors of tokens with high importance scores are concentrated in a single region. After the first round of propagation, the token distribution becomes noticeably sparser. This change leads to significant performance improvements, validating that propagation reduces token redundancy.

Table 2: Performance (f1 Scores or accuracy) comparison over three propagation signal types with 128 KV cache size across various datasets. The best result is highlighted in **bold**, the second best in underline.

Dataset	Decay (−)	Enhanced (+)	Evicted (−∞)	Baseline
NarrativeQA	**22.21**	21.71	5.09	21.35
Qasper	**13.59**	12.53	3.89	13.46
HotpotQA	**41.14**	39.75	11.84	40.68
2WikiMultihopQA	**29.43**	28.83	14.17	28.94
MusiQue	19.68	**19.78**	3.71	18.84
TREC	**55.00**	52.50	41.00	51.50
SAMSum	**39.01**	38.16	9.08	38.13
PassageCount	68.92	**69.50**	65.29	**69.50**
Lcc	**55.43**	55.04	8.33	54.49
RepoBench-P	**53.94**	52.84	9.89	52.78
Average	**39.84**	38.67	18.80	38.61

7 Discussion

Inspired by the relationship between key similarity and token's importance score, we propose a method for propagating decay signals based on graph structures. In the following sections, we further discuss this graph-based score updating paradigm.

7.1 Exploration of similarity for Graph Edge

In addition to key-to-key cosine similarity, we explore using query-to-key, query-to-query, key-to-value, and value-to-value similarities as decay signals for graph propagation. We assess their performance in multi-document QA and code generation tasks. Figure 8 shows that these additional measures improve performance across various subdatasets when used as edge-based decay signals.

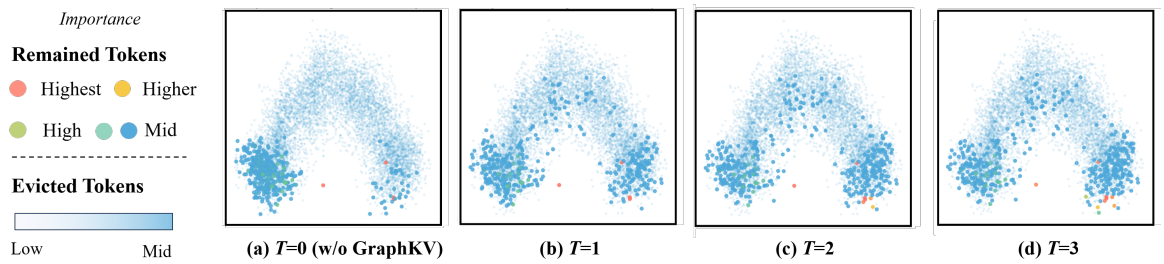

Figure 7: PCA visualizations in two dimensions of Keys with normalized importance scores for GraphKV across propagation rounds, where key vectors are colored by using a high-contrast colormap (sky blue to red), with retained tokens highlighted as larger, opaque markers.

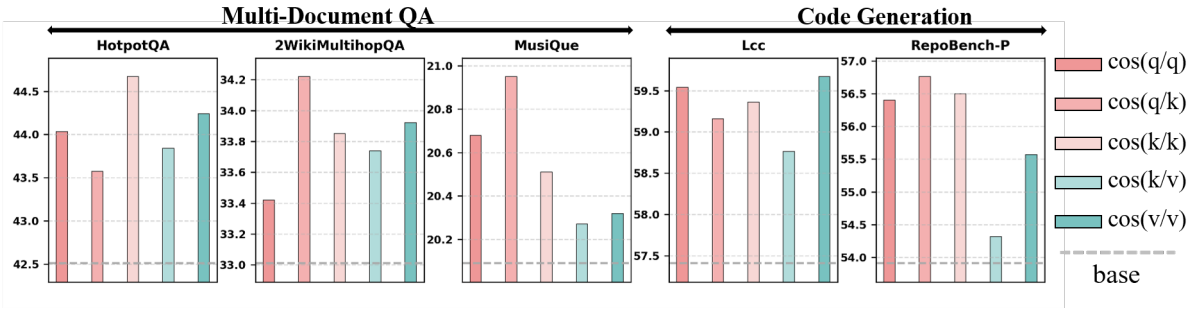

Figure 8: Performance comparison of GraphKV across datasets with different similarity graph edges.

Method	Decoding Latency (s)
PyramidKV	5.133
PyramidKV+ GraphKV	5.265 (+2.5%)
SnapKV	5.474
SnapKV+ GraphKV	4.912 (-10.2%)
Knorm	6.031
Knorm+ GraphKV	5.095 (-15.5%)

Table 3: The decoding latency of different methods on QMSum in LongBench at the KV budget of 512.

This underscores the effectiveness of avoiding storing too many highly similar tokens in GraphKV, supporting previous observation and motivation.

7.2 Diversity of Graph Signal

The experiments presented above validate the effectiveness of decay signal propagation in GraphKV. To comprehensively analyze graph signal propagation behavior, we further investigated different signal propagation modes. Specifically, we configured three signal types: decay signal ($-$), enhanced signal ($+$), and evicted signal ($-\infty$).

As presented in Table 2, the decay signal propagation yields the most significant improvement over the baseline method, followed by the enhanced signal, which provides a marginal improvement of only 0.06, rendering it negligible. This further validates the critical role of GraphKV in leveraging decay signals to eliminate token semantic redundancy. Importantly, propagating the evicted signal leads to a substantial performance decline. This is primarily due to the naive and coarse approach of evicting all nodes connected to the source node, which unintentionally removes too many important tokens. indicating a balance between importance and diversity should be achieved.

7.3 Efficiency Analysis

Table 3 gives the latency of GraphKV over three previous methods, demonstrating that GraphKV does not increase the inference latency, and even sometimes reduces it. This may be caused by that GraphKV increases model accuracy and hence prevents the model from over-long generation.

8 Conclusion

This paper investigates the critical limitations of static-importance-score-driven token eviction strategies in LLMs, which often fail to capture the relation between different tokens and suffer from duplicate KV cache. To address these challenges, we propose GraphKV, which models tokens as nodes and the similarity as edges. GraphKV dynamically refines token importance scores through an iterative decay-score-propagation mechanism, prioritizing contextually diverse and non-redundant tokens, leading to better performance in long-context scenarios.

9 Limitation

Although GraphKV substantially advances KV cache compression for long-context large language model inference through its innovative graph-based decay-signal-propagation framework for dynamic token relationship modeling, several limitations persist. The propagation mechanism depends on empirically determined hyperparameters, such as decay strength and propagation rounds, lacking a rigorous theoretical foundation for optimal configuration across diverse context complexities. Furthermore, due to computational constraints, evaluations are confined to 8B models, leaving the scalability and effectiveness of GraphKV on larger models unexplored. These gaps highlight the need for deeper theoretical analysis and broader empirical validation in future work.

Acknowledgements

This research was supported by the Shanghai Science and Technology Program (Grant No. 25ZR1402278). Besides, we thank Huawei Ascend Cloud Ecological Development Project for the support of Ascend 910 processors.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

AI Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku. *Claude-3 Model Card*, 1:1.

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

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

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, et al. 2024. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*.

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

Wei-Lin Chiang, Zhuohan Li, Ziqing Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6.

Alessio Devoto, Yu Zhao, Simone Scardapane, and Pasquale Minervini. 2024. A simple and effective l_2 norm-based strategy for kv cache compression. *arXiv preprint arXiv:2406.11430*.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *The world wide web conference*, pages 417–426.

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

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

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

Fengqing Jiang. 2024. Identifying and mitigating vulnerabilities in llm-integrated applications. Master's thesis, University of Washington.

Jia Li, Ge Li, Yongmin Li, and Zhi Jin. 2025. Structured chain-of-thought prompting for code generation. *ACM Transactions on Software Engineering and Methodology*, 34(2):1–23.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. 2024. Snapkv: Llm knows what you are looking for before generation. *Advances in Neural Information Processing Systems*, 37:22947–22970.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. 2024. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

Ziran Qin, Yuchen Cao, Mingbao Lin, Wen Hu, Shixuan Fan, Ke Cheng, Weiyao Lin, and Jianguo Li. 2025. Cake: Cascading and adaptive kv cache eviction with layer preferences. *arXiv preprint arXiv:2503.12491*.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2008. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. 2024. Loki: Low-rank keys for efficient sparse attention. *arXiv preprint arXiv:2406.02542*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Zheng Wang, Boxiao Jin, Zhongzhi Yu, and Minjia Zhang. 2024. Model tells you where to merge: Adaptive kv cache merging for llms on long-context tasks. *arXiv preprint arXiv:2407.08454*.

Jialong Wu, Zhenglin Wang, Linhai Zhang, Yilong Lai, Yulan He, and Deyu Zhou. 2024. Scope: Optimizing key-value cache compression in long-context generation. *arXiv preprint arXiv:2412.13649*.

Lingfei Wu, Yu Chen, Kai Shen, Xiaojie Guo, Hanning Gao, Shucheng Li, Jian Pei, Bo Long, et al. 2023. Graph neural networks for natural language processing: A survey. *Foundations and Trends® in Machine Learning*, 16(2):119–328.

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

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

Tianyi Zhang, Faisal Ladhak, Esin Durmus, Percy Liang, Kathleen McKeown, and Tatsunori B Hashimoto. 2024. Benchmarking large language models for news summarization. *Transactions of the Association for Computational Linguistics*, 12:39–57.

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

Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Haonan Chen, Zheng Liu, Zhicheng Dou, and Ji-Rong Wen. 2023. Large language models for information retrieval: A survey. *arXiv preprint arXiv:2308.07107*.

A Appendix

A.1 Implementation Details

Model Configuration. Our experiments include three state-of-the-art open-source LLMs. Specifically Llama2-7B-Chat (Touvron et al., 2023), Llama-3-8B-Instruct (Grattafiori et al., 2024) and Mistral-7B-Instruct-v0.2 (Jiang, 2024). All experiments are conducted on a single NVIDIA H20 GPU. We use beam search according to (Cai et al., 2024; Li et al., 2024).

Evaluation Tasks. To assess GraphKV's performance, we use two widely used benchmarks LongBench (Bai et al., 2023b) and Needle In a Haystack (Li et al., 2024). LongBench is a bilingual, multitask benchmark suite for LLMs, providing a comprehensive stress test for long prompt inputs. It consists of a diverse set of tasks, including question answering, summarization, reading comprehension, and code-related tasks, with input contexts ranging from a few thousand to over 100,000 tokens. The dataset spans multiple languages and domains, such as scientific literature, news, and dialogues, ensuring robust evaluation across varied scenarios. A detailed breakdown of the sub-tasks and their corresponding metrics is provided in Table 4 below. Needle In a Haystack assesses retrieval and reasoning via three components: Single-Needle Retrieval, Multi-Needle Retrieval, and Multi-Needle Reasoning, testing performance in complex contextual environments.

Dataset	Avg len	Metric	#data
NarrativeQA	18,409	F1	200
Qasper	3,619	F1	200
MultiFieldQA-en	4,559	F1	150
HotpotQA	9,151	F1	200
2WikiMultihopQA	4,887	F1	200
MuSiQue	11,214	F1	200
GovReport	8,734	Rouge-L	200
QMSum	10,614	Rouge-L	200
MultiNews	2,113	Rouge-L	200
TREC	5,177	Accuracy (CLS)	200
TriviaQA	8,209	F1	200
SAMSum	6,258	Rouge-L	200
PassageCount	11,141	Accuracy (EM)	200
PassageRetrieval-en	9,289	Accuracy (EM)	200
LCC	1,235	Edit Sim	500
RepoBench-P	4,206	Edit Sim	500

Table 4: The dataset statistics in LongBench include several key metrics. The 'Avg len' (average length) is measured by the number of words for datasets in English (or code). 'Accuracy (CLS)' represents classification accuracy, while 'Accuracy (EM)' denotes exact match accuracy.

KV Cache Size (Tokens)	KV Cache Memory (GB)
128	0.016
256	0.031
512	0.063
1024	0.125
2048	0.250
16K	1.953
32K	3.906
64K	7.813
128K	15.625

Table 5: Memory consumption of KV cache at different context lengths for a 7B-parameter model with 32 attention heads and 128-dimensional key/value vectors. Memory calculations assume BF16 precision.

A.2 More Related Works

Graph-Based Methods. Graph-based modeling (Scarselli et al., 2008) has emerged as a powerful paradigm for capturing complex relationships in structured data, with applications ranging from social networks (Fan et al., 2019) to natural language processing (Wu et al., 2023). Graph Neural Networks (GNNs), leverage message-passing mechanisms to dynamically update node representations by aggregating information from adjacent nodes. This iterative process effectively captures dependencies that evolve over time or context, making it well-suited for tasks requiring relational reasoning. In the context of language modeling, graph structures have been used to represent syntactic dependencies or semantic relationships in text. Inspired by these capabilities, our GraphKV method formulates the KV cache as a graph with tokens as nodes and semantic similarities as edges. To the best of our knowledge, our approach is the first method to explore the application of graph-based methods to KV cache eviction in LLMs.

Algorithm 1 Sparse Graph Construction

Input: Token sequence $X = \{x_1, \ldots, x_n\}$, scores $S = \{s_1, \ldots, s_n\}$, k
Initialize nodes $O = \{o_1, \ldots, o_n\}$
Initialize empty edge set $E = \emptyset$
$O_{\text{source}} \leftarrow \{o_i \mid s_i \text{ in top-}k\}$
for $o_i \in O_{\text{source}}$ **do**
  for $o_j \in O \setminus \{o_i\}$ **do**
    $E \leftarrow E \cup \left\{ \frac{\langle k_i, k_j \rangle}{\|k_i\| \|k_j\|} \right\}$
  end for
end for
Output: Sparse graph $G = (O, E)$

Task	# Source Nodes				
	10%	30%	50%	70%	100%
KV Cache Size = 512					
Qasper	24.86	24.47	24.45	**25.20**	24.53
NarrQA	**23.06**	23.03	23.01	22.65	22.18
MFQA-en	35.45	36.34	36.31	35.27	**36.82**
HotpotQA	**43.91**	43.75	43.70	43.10	42.66
2WikiQA	33.09	**34.05**	34.00	33.70	33.51
MusiQue	**20.65**	20.33	20.29	20.28	20.57

Table 6: Performance of GraphKV with varying numbers of source nodes (as a percentage of the KV cache size, 512) across multiple tasks. Bold values indicate the best performance for each task. Abbreviations: Qas (Qasper), NarrQA (NarrativeQA), MFQA-en (MultiFieldQA-en), 2WikiQA (2WikiMultihopQA).

Algorithm 2 Decay Signal Propagation

Input: Sparse graph $G = (O, E)$, scores $\{s_1, \ldots, s_n\}$, T, m
Initialize $s_j^{(0)} \leftarrow s_j$ for all $o_j \in O$
for round $t = 1$ **to** T **do**
  for each source node $o_i \in O_{\text{source}}$ **do**
    $N(o_i) \leftarrow \{o_j \mid e_{ij} \text{ is top-}m \text{ among } \{e_{ik}\}_{k \neq i}\}$
    for each $o_j \in N(o_i)$ **do**
      $s_j^{(t)} \leftarrow s_j^{(t-1)} \cdot (1 - \cdot e_{ij})$
    end for
  end for
end for
Output: Refined scores $\{s_1^{(T)}, \ldots, s_n^{(T)}\}$