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

Xuelin Li, Xiangqi Jin, Linfeng Zhang
EPIC Lab, Shanghai Jiao Tong University

Paper takeaway

Keep the original importance scores, then discount tokens similar to high-score representatives before choosing the cache.

arXiv:2509.00388v1 (30 August 2025)
Research group discussion · 25 minutes · 12 November 2026

Paper identity and current artifact status

- **Read version:** arXiv v1; 11 PDF pages, including appendix.
- **Published at** EMNLP 2025, main conference, pp. 21899–21909.
- **Official repository: partial code release.** Evaluation script is public; core propagation implementation is not visible in the inspected tree.
- All reported experiments below refer to arXiv v1.
  Status checked on 12 November 2026.

Three questions for the discussion

Why can top-score selection waste cache capacity?
What exactly does the graph update?
Where is the empirical benefit largest?

arXiv v1; official ACL proceedings; repository README and file tree.

A high-scoring set may repeat the same information

- KV eviction keeps a small subset of historical key/value entries.
- Existing scores estimate token relevance; top-budget selection treats entries independently.
- Several high-score tokens can have very similar keys, leaving less room for different information.

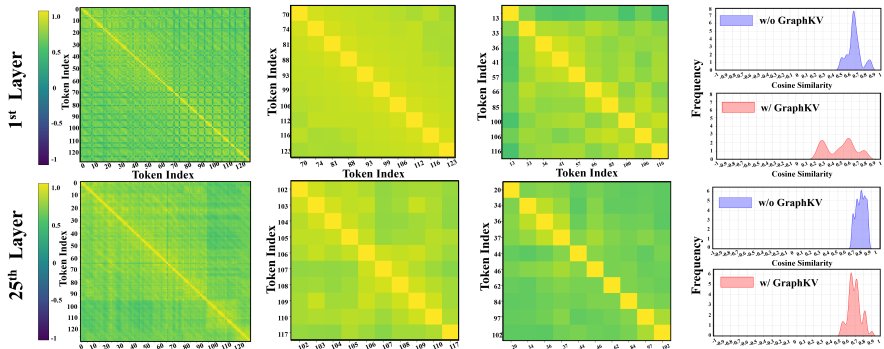
GraphKV intervention

Initial scores $\longrightarrow$ similarity-aware discounts $\longrightarrow$ final selection.

No additional model training; the KV vectors themselves are unchanged.

Secs. 1–3, PDF pp. 1–4.

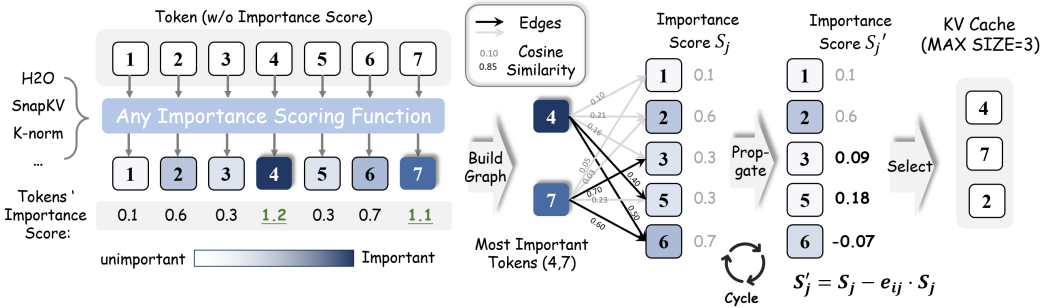
The motivating example shows clustered high-score keys



Rows: layers 1 and 25. Columns: all tokens, original top-10, GraphKV top-10, similarity distribution. Less yellow means less similarity.

Fig. 2, PDF p. 3. Llama3-8B, one HotpotQA sample, 128 tokens.

The graph sits between scoring and final retention



- Use an existing method for initial scores and cache budgets.
- Select source nodes; discount similar neighbors; select final entries.

Flow illustration only: the negative score shown in Fig. 3 is inconsistent with multiplicative Eq. (5); see backup.

Fig. 3, PDF p. 4; Eq. (5), PDF p. 5.

Source count and cache budget are different quantities

$$\mathbf{K}^l, \mathbf{V}^l \in \mathbb{R}^{n \times d} \quad \longrightarrow \quad \mathbf{K}^l_s, \mathbf{V}^l_s \in \mathbb{R}^{k_l \times d}$$

n	Candidate token count
k_l (or B)	Final cache budget for the layer
k	Number of source nodes
m	Neighbors per source node
T	Number of score-update rounds

The paper uses a simplified matrix notation here; head axes are omitted.

Sec. 4.1, PDF p. 4; Secs. 4.2–4.3, PDF pp. 4–5.

Only high-score sources compare against all candidates

$$O_{\text{source}} = \{o_i \mid s_i \text{ ranks in top-}k\}$$

$$e_{ij} = \frac{\langle k_i, k_j \rangle}{\|k_i\| \|k_j\|}, \qquad i \in O_{\text{source}}, \quad j \neq i$$

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

- s_i: an existing importance score; k_i: token i's key vector.
- Compare k sources to n candidates, then keep top-m neighbors.
- Cosine is symmetric, but only sources send score updates.

Eqs. (1)–(3), PDF pp. 4–5; Algorithm 1, PDF p. 11.

Multiple similar sources impose multiplicative discounts

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

- Start from $s_j^{(0)} = s_j$; choose the top-k_I final scores.
- Similarity 0.8 gives a factor of 0.2 on that edge.
- Multiple incoming edges compound their discounts.
- For nonnegative scores, positive similarities suppress scores. Negative similarities would increase them; no clipping rule is specified.

Sec. 4.3, Eq. (5), PDF p. 5. Explanation follows the explicit product formula.

A small example makes the selection change visible

Budget $k_l = 2$; source A; neighbors B, C with similarities $0.9, 0.1$.

Token	Initial score	Updated score	Calculation
A	1.0	1.00	No incoming discount
B	0.9	0.09	$0.9(1 - 0.9)$
C	0.7	0.63	$0.7(1 - 0.1)$
D	0.6	0.60	Outside the neighborhood

Selection changes from $\{A, B\}$ to $\{A, C\}$

B is highly relevant but mostly overlaps with representative A.

Teaching example constructed from Eq. (5); not paper experiment data.

With a fixed graph, rounds strengthen the same discount

$$c_j = \prod_{i:\, o_j \in N(o_i)} (1 - e_{ij}) \qquad \Longrightarrow \qquad s_j^{(T)} = s_j c_j^T$$

- This is a direct derivation from Eq. (5), for fixed sources and edges.
- The formal description does not reselect sources each round.
- T changes score suppression; it does not update key vectors or imply T-hop message passing.
- Sources are not formally forced into the final retained set.

Reading analysis of Sec. 4.3, Eq. (5), PDF p. 5.

Near-linear cost assumes a fixed, small source count

Operation	Cost with explicit key dimension d
Source-to-candidate similarities	$O(nkd)$
T rounds over selected edges	$O(Tkm)$
Stored candidate similarities	$O(nk)$
Stored pruned edges	$O(km)$

- Additional top-k/top-m selection work also applies.
- Paper: $O(nk) \approx O(n)$ for $k \ll n$ and fixed budget.
- If budget and k grow proportionally with n, similarity work grows quadratically in n.

Paper complexity claim: Sec. 1, PDF p. 2. Explicit dimensions and storage: reading analysis.

Evaluation compares the same scorer before and after

- **Models:** Llama2-7B-Chat, Llama3-8B-Instruct, Mistral-7B-Instruct-v0.2.
- **Main table:** 512-token cache setting; KNorm, SnapKV, PyramidKV, each with/without GraphKV; Full KV reference.
- **LongBench:** 16 tasks; F1, ROUGE-L, accuracy, edit similarity; all higher is better.
- **Needle:** Llama3-8B, input up to about 8K tokens, cache 128.
- Single NVIDIA H20; beam search reported in Appendix A.1. CAKE and H2O appear in the round ablation.

Secs. 5–6, PDF pp. 5–7; Appendix A.1 and Table 4, PDF p. 10.

LongBench gains are consistent in task averages

Model	Scorer	Base	+GraphKV	Δ
Llama2-7B	KNorm	13.03	25.20	+12.17
	SnapKV	31.95	32.35	+0.40
	PyramidKV	32.04	32.25	+0.21
Llama3-8B	KNorm	20.64	29.76	+9.12
	SnapKV	39.97	40.35	+0.38
	PyramidKV	39.58	40.09	+0.51
Mistral-7B	SnapKV	40.24	40.60	+0.36
	PyramidKV	40.16	40.50	+0.34

All entries: reported 16-task average, budget 512; Δ is an absolute score difference. Full averages remain higher.

Table 1, PDF p. 6; selected rows. Mistral KNorm: 14.91 to 16.50.

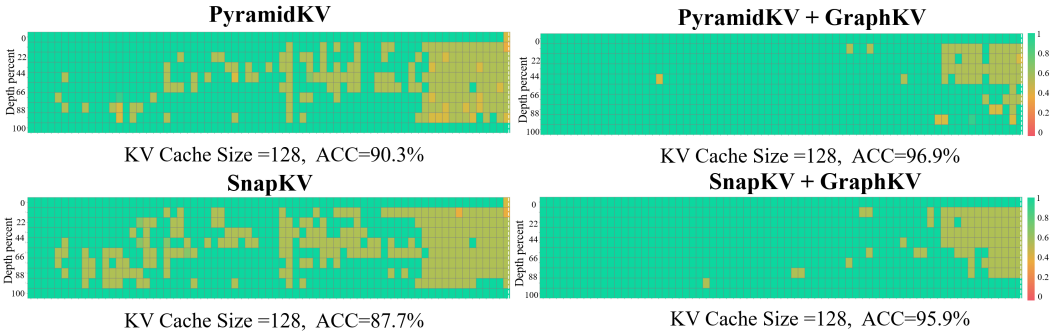
QA gains and all-task gains use different denominators

Llama3-8B, budget 512	SnapKV	PyramidKV
All 16 tasks: relative gain	+0.95%	+1.29%
Six QA tasks: relative gain	+2.89%	+2.76%

- The introduction's "about 3%" aligns with the six-QA-task calculation, rather than the all-task average.
- RepoBench-P: SnapKV 53.87 to 56.10; Full 52.38.
- LCC: the same scorer falls from 59.50 to 58.42.
- Individual tasks can exceed Full while the overall average stays below Full.

Table 1, PDF p. 6. Subset means and relative gains calculated from reported values.

At cache 128, Needle retrieval gains are larger

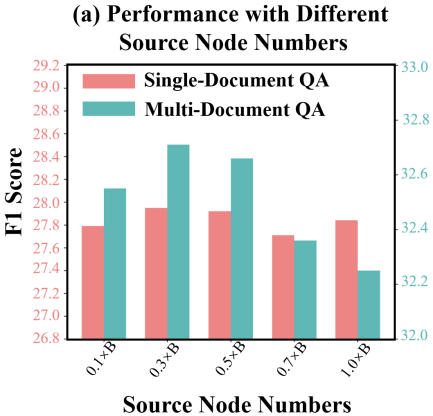


PyramidKV: 90.3% → 96.9% (**+6.6 percentage points**)

SnapKV: 87.7% → 95.9% (**+8.2 percentage points**)

Fig. 4 and Sec. 5.3, PDF p. 6. Llama3-8B; context up to about 8K.

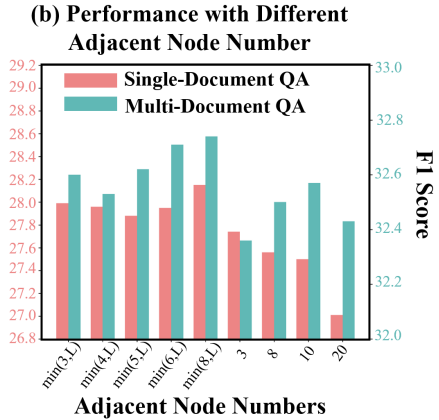
A moderate source set works best in the aggregate



- $k \approx 0.3B$ is best in the plotted QA aggregates.
- Larger source sets affect more tokens and may over-suppress useful scores.
- Task-specific optima differ: Qasper favors 70%, HotpotQA favors 10% in Table 6.

Fig. 5(a), PDF p. 7; Table 6, PDF p. 11. Pink and teal use different y-axes.

Neighbor count also controls the suppression range



- Adaptive $\min(c, L)$ choices generally outperform fixed large neighborhoods.
- More neighbors do not ensure better coverage after score decay.
- Figure: $L =$ input tokens $\times 10^{-3}$. Prose calls it query length; units need clarification for reproduction.

Fig. 5(b) and Sec. 6.2, PDF p. 7. Preserve the original figure labels.

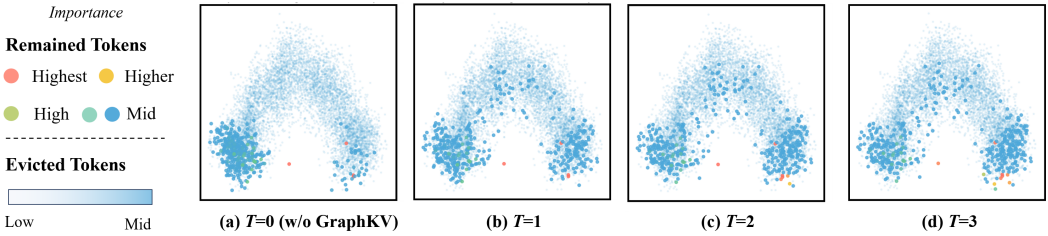
The first round helps; more rounds are not monotonic

Scorer	$T = 0$	$T = 1$	$T = 2$	$T = 3$
CAKE	46.49	46.86	46.47	46.76
H2O	40.91	41.75	42.14	42.15
PyramidKV	42.51	44.48	43.97	44.67
SnapKV	42.60	43.03	43.04	43.38

- Largest first-round jump: PyramidKV, $+1.97$ F1.
- CAKE at $T = 2$ is slightly below its no-propagation value, contrary to the prose's blanket claim.
- These are the plotted task F1 scores, not 16-task averages.

Fig. 6 and Sec. 6.3, PDF p. 7. Configuration is not fully specified in that paragraph.

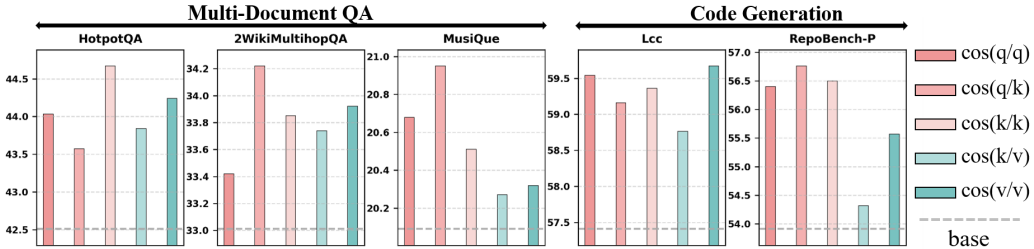
PCA shows which keys receive high scores



- Panels show $T = 0, 1, 2, 3$; color and larger markers highlight important/retained entries.
- The selection spreads across the key distribution. Key coordinates are not updated by GraphKV.

Fig. 7, PDF p. 8; mechanism from Eq. (5), PDF p. 5.

Several similarity definitions work; the best varies



- Five panels: three multi-document QA tasks and two code tasks.
- Bars: Q–Q, Q–K, K–K, K–V, V–V; dashed line: baseline.
- K–K is not uniformly best. Avoiding redundancy is the shared design idea.

Fig. 8 and Sec. 7.1, PDF pp. 7–8. Y-scales and task metrics differ between panels.

Soft decay preserves a useful importance–diversity balance

Signal	Reported average	Δ vs. baseline
No propagation	38.61	—
Decay ($-$)	**39.84**	$+1.23$
Enhancement ($+$)	38.67	$+0.06$
Evict neighbors ($-\infty$)	18.80	-19.81

- Cache 128; mean over the ten tasks listed in Table 2.
- Similar neighbors can still carry useful information: hard eviction fails badly.
- Decay is best on average, with task-level exceptions.

Table 2, PDF p. 7; Sec. 7.2, PDF p. 8. Mixed task metrics; higher is better.

Measured decoding latency can increase or decrease

Scorer	Base (s)	+GraphKV (s)	Change
PyramidKV	5.133	5.265	+2.5%
SnapKV	5.474	4.912	−10.2%
KNorm	6.031	5.095	−15.5%

- QMSum, cache 512; lower is better.
- A universal "no latency increase" claim conflicts with the PyramidKV row.
- Authors suggest shorter generations may explain speedups. No fixed-output-length timing isolates graph overhead.

Table 3 and Sec. 7.3, PDF p. 8.

The main open issues are precise update semantics and tuning

- **Empirical tuning:** source count, neighborhood size and rounds trade relevance against suppression.
- **Update semantics:** Eq. (5), Fig. 3 and Algorithm 2 do not fully agree; follow the product equation for this discussion.
- **Scale:** reported model evaluation is limited to 7B–8B.
- **Efficiency:** graph cost and variable generation length need separate timing.

Research question

Can a clear set-level objective explain when redundant-looking tokens should still be kept?

Sec. 9, PDF p. 9; Eq. (5), Fig. 3, Algorithm 2, Table 3. Last question: reading analysis.

Take home: discount overlap before spending cache capacity

1. **Mechanism:** keep a useful initial scorer, add similarity-aware score discounts, then select the cache.
2. **Evidence:** modest all-task gains on strong scorers; larger retrieval gains at a very small cache budget.
3. **Practical tradeoff:** soft suppression helps; excessive range, repeated discounts or hard deletion can remove useful information.

Token importance and contribution to an already represented set are different quantities.

Synthesis of Secs. 4–7, PDF pp. 4–8.

References and artifact links

- Xuelin Li, Xiangqi Jin, Linfeng Zhang.
  GraphKV: Breaking the Static Selection Paradigm with Graph-Based KV Cache Eviction.
  arXiv:2509.00388v1, 2025. This is the analyzed version.
- EMNLP 2025 official proceedings, pp. 21899–21909.
  DOI: 10.18653/v1/2025.emnlp-main.1112.
- Official repository: lxlcooper/GraphKV. Partial release as of 12 November 2026: evaluation script visible; core propagation code not found in the inspected tree.

Backup: the paper has conflicting numerical update examples

- Eq. (5): multiply all source discounts, $s_j^{(t)} = s_j^{(t-1)} \prod_i (1 - e_{ij})$.
- Fig. 3: score 0.7 becomes -0.07. With nonnegative initial scores and cosine edges in $[-1, 1]$, the product rule cannot produce this negative value.
- Algorithm 2: inner-loop assignments repeatedly use $s_j^{(t-1)}$; a literal implementation overwrites earlier source updates in the same round.
- Eq. (4) and Algorithm 2 also contain stray multiplication dots. No extra decay coefficient is defined there.

Interpretation used here: the explicit multi-source product in Eq. (5).

Fig. 3, PDF p. 4; Eqs. (4)–(5), PDF p. 5; Algorithm 2, PDF p. 11.

Backup: LongBench metrics must retain their task meanings

Task group	Datasets	Metric
Single-doc QA	NarrativeQA, Qasper, MF-en	F1
Multi-doc QA	HotpotQA, 2Wiki, MuSiQue	F1
Summary	GovReport, QMSum, MultiNews	ROUGE-L
Few-shot	TREC / TriviaQA / SAMSum	Acc. / F1 / R-L
Synthetic	PCount, PRetrieval	Exact match
Code	LCC, RepoBench-P	Edit similarity

Most datasets: 200 examples; MF-en: 150; each code task: 500.
Appendix average lengths are *words* for English/code, not tokenized model input lengths.
The mixed-task mean is not pooled accuracy.

Table 4, PDF p. 10; metric headings in Table 1, PDF p. 6.

Backup: the best source fraction differs by dataset

Task	10%	30%	50%	70%	100%
Qasper	24.86	24.47	24.45	**25.20**	24.53
NarrativeQA	**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

Budget 512; percentages refer to source count as a fraction of budget. The aggregate preference for 30% does not imply a per-task optimum.

Table 6, PDF p. 11.

Backup: memory estimates require KV heads and layer count

$$\text{KV bytes} = 2\, N_{\text{batch}}\, M\, n\, H_{\text{KV}}\, d_{\text{head}}\, b$$

- 2: keys and values; M: layer count; b: bytes per scalar.
- Use KV heads, not query heads; they differ with grouped-query attention.
- Table 5 reports 0.063 GB at cache 512 and 15.625 GB at 128K, but does not clearly specify layer/KV-head accounting behind these values.
- A fixed cache of 512 is not a universal 10% retention ratio: input lengths differ by sample and model.

Table 5, PDF p. 11; 10% statement in Sec. 5.2, PDF p. 5. Formula: reading analysis.