GRAPH-GUIDED AND ADAPTIVE KV-CACHE PREFETCHING FOR RETRIEVAL-AUGMENTED GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

The order and meaning of chunks in retrieval-augmented generation often contain relationships that semantic similarity alone does not capture. We introduce GraphKV, a graph-guided KV-cache prefetching policy that combines semantic neighbours with within-document structural links, performs two-hop candidate retrieval, and supports offline-learned and online-adaptive semantic–structural weighting. We evaluate GraphKV through archived simulation, real vLLM–LMCache experiments, component ablations, and answer-quality tests. On the controlled Hotpot access trace with Qwen2.5-1.5B at $K = 10$, offline adaptation raises target-prediction recall from 80.50% for cosine prefetching to 95.33% and reduces ready-cache time to first token from 56.68 to 51.36 ms. The same pattern is observed with Llama-3.2-1B and on controlled 2Wiki traces. Ablations show separate contributions from structural edges, two-hop traversal, and offline adaptation. With synchronous candidate preparation, the corresponding times are 902.67 and 883.27 ms, whereas they are 73.41 ms when there is no prefetching. Full 600-event asynchronous experiments show that improved cache readiness does not yet translate into lower application-level latency than cosine or no prefetch on a single T4, even when all candidates complete before arrival. In separate Gemini answer-quality experiments, offline GraphKV improves HotpotQA F1 from 47.94 to 53.83 but reduces 2WikiMultiHopQA F1 from 48.18 to 40.53 because graph pruning can exclude required evidence. These results position GraphKV as a workload-dependent selection mechanism: it improves next-chunk prediction and ready-cache latency, while end-to-end benefit depends on graph coverage, reuse, and the cost and overlap of cache preparation.

1 INTRODUCTION

Retrieval-augmented generation (RAG) gives a language model access to external passages that are relevant to a question (Lewis et al., 2020). These passages extend the input that the model has to process before it generates its first answer token. Reusing previously computed key and value tensors can reduce this cost, but only when the cached states match the context in which they are reused; once storage and preparation time are limited, this becomes a selection problem.

Cosine similarity provides a simple benchmark. Once a chunk has been accessed, the prefetcher is able to choose other chunks that have similar embeddings. But similarity does not capture all kinds of useful relationships. A paragraph can still depend on the one that comes right before it even if the two paragraphs deal with different topics. Moreover, a passage might also be connected to another passage via an intermediate explanation. We address this with a chunk graph that encodes semantic and structural relationships.

We study this problem through GraphKV, an external candidate-selection layer. GraphKV does not alter the language model's parameters or its attention mechanism; instead, it decides which chunks to prepare, while the serving system carries out the computation, storage, and reuse of their KV states. The graph includes semantic neighbours chosen on the basis of rank, a structural backbone that connects adjacent units in the document, and weighted paths with at most two edges. A static graph employs a semantic-first ratio of 0.70/0.30. An offline policy learns a global balance based on

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

training accesses. An online policy begins with that balance and then updates it following an instance where the target was not found.

We evaluate GraphKV in three settings. First, we report the earlier simulated-cache results across six model configurations. Second, we measure real cache reuse and latency with vLLM and LMCache using Qwen and Llama on Hotpot and 2Wiki traces. Third, we test whether the selected context improves answer quality with Gemini. We also run component ablations and an asynchronous experiment to determine whether better prediction is enough to improve end-to-end latency.

2 Related Work

KV-cache serving and reuse. PagedAttention manages KV-cache memory through a paging approach and underlies vLLM (Kwon et al., 2023). LMCache provides a reusable cache layer connected to an inference engine (LMCache Contributors, 2026). These systems supply the storage and execution machinery for our real experiment, while our candidate-selection rule operates outside it.

Other serving systems address KV reuse from different angles: SGLang's RadixAttention shares prefix cache across concurrent requests via a radix tree (Zheng et al., 2024), and Prompt Cache modularizes attention states so independently-encoded segments can be reused across prompts (Gim et al., 2024). GraphKV's candidate-selection problem is complementary to these as it decides what to prepare rather than how prepared states are shared or reused across requests.

A separate line of work reduces KV-cache cost after generation, by evicting low-value tokens (Zhang et al., 2023) or compressing and streaming cache contents (Liu et al., 2024b). GraphKV instead decides which chunks to prepare before they are requested; compression and eviction operate on states already computed and are orthogonal to this selection problem.

Cache reuse also depends on the arrangement of text in a prompt. CacheBlend studies how to combine independently computed chunk states when their texts occur together, including selective recomputation to account for their interaction (Yao et al., 2024). Our measured setup uses exact per-chunk prefix reuse in stock LMCache and evaluates advance preparation of an individual chunk request. It does not assume that a state computed for an independent chunk can be inserted without recomputation at a different position in a concatenated RAG prompt. That broader composition problem requires CacheBlend or an equivalent position- and attention-aware mechanism.

Recent work also explores KV-cache precomputation and speculative prefetching. TurboRAG precomputes chunk-level KV caches for RAG inference, whereas SpeCache predicts future KV accesses during decoding and prefetches them in advance (Lu et al., 2025; Jie et al., 2025). On the other hand, GraphKV carries out candidate selection before cache retrieval, using semantic and document-structural relationships to predict which chunks are likely to be needed.

Caching for RAG. RAGCache organizes intermediate states in a knowledge tree and combines cache management with retrieval-aware execution (Jin et al., 2024). Its design illustrates the importance of both retention and execution scheduling. HippoRAG builds a knowledge graph over the corpus to support associative, multi-hop retrieval (Jiménez Gutiérrez et al., 2024); its graph is built for retrieval quality over an entire corpus, whereas GraphKV builds a lightweight per-access graph for cache candidate selection and does not aim to replace the underlying retriever. KV-Runahead similarly generates KV states ahead of a causal request, using parallel decoding to overlap preparation with generation (Cho et al., 2024); GraphKV differs in using document structure and learned graph weighting to decide which chunks to prepare, rather than parallelizing preparation of a known sequence. GraphKV instead focuses on ranking candidate chunks from semantic and structural relationships after a current access. Cosine is our experimental baseline; we do not treat it as a default prefetcher built into vLLM or LMCache.

Embeddings and multi-hop evidence. Sentence embedding methods enable efficient similarity comparisons (Reimers & Gurevych, 2019). Dense retrieval methods learn query/passage embeddings for retrieval rather than lexical matching (Karpukhin et al., 2020); multi-hop variants such as MDR retrieve iteratively across hops (Xiong et al., 2021). GraphKV uses fixed sentence embeddings for local candidate ranking rather than learned end-to-end retrieval. We use MiniLM embeddings

to construct the semantic part of the graph. HotpotQA and 2WikiMultiHopQA supply questions with evidence distributed across passages (Yang et al., 2018; Ho et al., 2020). LongBench includes long-context tasks spanning these datasets and others (Bai et al., 2024). The presence of multi-hop questions motivates examining relationships between passages; it does not, by itself, imply that a two-hop similarity graph will outperform cosine selection.

3 Graph Selection and Adaptation

3.1 Problem and representation

Let $\mathcal{C} = \{c_1, \ldots, c_n\}$ contain text units, with embedding e_i, document identity d_i, and position p_i. Given a current unit c_i, a policy returns a set $P_K(i)$ of at most K other units. In the cache experiment, the observed next target is y_t, and a prediction succeeds when $y_t \in P_K(i_t)$. In the answer-quality experiment, the policy selects supporting context around a primary sentence retrieved from the question.

The text unit depends on the experiment. The real cache benchmark uses overlapping token chunks from source context records. The API experiment uses the original evidence sentences and document titles provided by the QA dataset. A unit's position is meaningful only within its corresponding source document or record.

3.2 Semantic and structural edges

For nonzero embeddings, semantic similarity is

$$s_{\text{sem}}(i,j) = \frac{e_i^\top e_j}{\|e_i\|_2 \|e_j\|_2}. \tag{1}$$

Let $S_M(i)$ be the M most similar other units. A semantic edge is admitted if $j \in S_M(i)$ or $i \in S_M(j)$. This symmetrization allows either endpoint to propose a connection. Unlike a fixed similarity threshold, top-M retains the best available neighbours even when all similarities are relatively low.

The structural score is

$$s_{\text{str}}(i,j) = \begin{cases} \dfrac{1}{1+|p_i - p_j|}, & d_i = d_j, \\ 0, & d_i \neq d_j. \end{cases} \tag{2}$$

Immediate same-document neighbours are admitted as edges regardless of semantic similarity. Other same-document pairs receive a structural contribution if admitted through the semantic rule. We do not create an all-pairs structural graph. The added constant avoids a singularity at zero distance, while inverse distance gives nearby units more influence without a learned positional model.

For an admitted edge, let $\tilde{s}_{\text{sem}}$ equal the cosine score when the semantic rule is satisfied, and zero otherwise. The weighted edge is

$$w_{ij} = \alpha \tilde{s}_{\text{sem}}(i,j) + \beta s_{\text{str}}(i,j), \qquad \alpha, \beta \geq 0, \quad \alpha + \beta = 1. \tag{3}$$

We set $(\alpha, \beta) = (0.70, 0.30)$ as a baseline to give priority in retrieval to semantic similarity while maintaining document adjacency. This keeps semantic similarity dominant while allowing document order to affect ranking. With $(\alpha, \beta) = (0.70, 0.30)$, an adjacent semantic neighbour outranks a cross-document neighbour with the same cosine similarity, because the structural term applies only within a document.

The real-cache experiment uses $M = 20$, while the API experiment uses the runner's default value of $M = 5$. In the archived simulation and the original real-cache implementation, the adaptive policy ranks paths from their stored semantic and structural features before the historical fixed-weight pruning step. Its candidate set can therefore differ from the candidate set used by the fixed graph, even when the two policies use the same weights.

We remove this difference in the standardized ablation. All graph arms use the same top-M graph, feature representation, candidate universe, and tie-breaking rule. The ablation changes only the component named by that arm: structural edges, two-hop traversal, offline weights, or online updates. Every policy still returns at most K candidates.

3.3 TWO-HOP RANKING

The fixed graph considers direct neighbours and neighbours of those neighbours. For a two-edge path with weights a, b, it uses

$$B(a, b) = 0.9 \min(a, b) + 0.1 \max(a, b). \tag{4}$$

The weaker edge carries most of the blended score, while the stronger edge contributes a smaller tie-breaking term. The 0.9/0.1 coefficients are fixed without tuning. Each candidate keeps its best one- or two-edge path before the top K are returned.

For adaptation, each edge retains a two-component feature $(\tilde{s}_{\text{sem}}, s_{\text{str}})$. Equation 4 is applied separately to each component along a two-hop path, producing $\phi_{ij}^{(r)}$. The adaptive candidate score is

$$q_\theta(i, j) = \max_{r \in \mathcal{P}_{ij}} \theta^\top \phi_{ij}^{(r)}, \qquad \theta = (\alpha, \beta), \tag{5}$$

where $\mathcal{P}_{ij}$ contains one- and two-edge paths. The standardized ablation uses this common feature-wise scoring path for all graph arms.

3.4 OFFLINE LEARNING

The offline policy searches $\alpha \in \{0, 0.05, \ldots, 1\}$ with $\beta = 1 - \alpha$ and maximizes training recall over the tested budgets:

$$\theta^* = \arg\max_\theta \frac{1}{|\mathcal{T}||\mathcal{K}|} \sum_{t \in \mathcal{T}} \sum_{K \in \mathcal{K}} \mathbf{1}[y_t \in P_K(i_t; \theta)]. \tag{6}$$

Ties favour a semantic weight closer to 0.70. The selected global pair is frozen for test evaluation and receives no query-type label. The API fit instead maximizes training supporting-sentence recall, with feedback-target hits breaking ties. The archived simulation used an earlier query-type-conditioned policy and is labelled accordingly.

3.5 ONLINE LEARNING

The online policy starts from θ^* and predicts before observing the target. If the target was selected, the weights stay unchanged. If it was absent from the graph's candidate universe, the policy records a coverage miss; changing weights cannot introduce a missing path. Otherwise, the event is a ranking miss. Let z_t be the highest-ranked selected candidate with stored features. The update uses

$$g_t = \phi(y_t) - \phi(z_t), \tag{7}$$

$$\hat{g}_t = \begin{cases} g_t / \|g_t\|_2, & \|g_t\|_2 > 10^{-12}, \\ g_t, & \text{otherwise}, \end{cases} \tag{8}$$

$$\theta_{t+1} = \Pi_\Delta(\theta_t + \eta \hat{g}_t), \tag{9}$$

where Π_Δ projects onto nonnegative weights summing to one; 10^{-12} prevents division by numerical zero. Only future events are affected. Development data select η from $\{0.01, 0.05, 0.1, 0.2, 0.5, 1.0\}$; the Qwen run selects 0.01 and starts from $(0.45, 0.55)$.

4 EXPERIMENTAL DESIGN

Archived simulation. The previous study contains hit-aware simulated-cache results for Qwen2.5-1.5B, Qwen2.5-3B, Qwen2.5-7B, Llama-3.2-1B, Llama-3.2-3B, and Gemma-3-1B over the available HotpotQA, 2WikiMultiHopQA, MuSiQue (Trivedi et al., 2022), and MultiFieldQA-en data. The archived query-type-conditioned policy and its analytical latency model differ from the policies and measured time intervals in the real-system experiments. We report the complete archived results in Appendix A.

Real cache benchmark. The model Qwen2.5-1.5B-Instruct is run using vLLM and LMCache on a Tesla T4 (with a 0.5-GB cache, occupying 0.82 of the GPU memory, having a maximum context length of 4096 tokens and using 16-token blocks); the MiniLM embeddings are processed on the CPU. From one hundred Hotpot context records, 1,223 chunks of 384 tokens are produced, with an overlap of 64 tokens. The splits of the records into disjoint parts include 600 training, development, and test accesses, with the accesses balanced among generated semantic, structural, and multi-hop patterns. We examine the following policies—no prefetch, cosine, fixed graph, offline, and online—for $K \in \{6, 10, 16\}$, using two arm orders: 30 arms and 18,000 observations. The same isolated-arm protocol is carried out again using Llama-3.2-1B on Hotpot and with both models on 2Wiki at $K = 10$.

Each candidate is populated as an exact one-token-completion prompt through vLLM's HTTP interface and stored by LMCache; the graph stays as an external selector. Before the benchmark, we test cold and repeated access, incorrect candidate preparation, oracle preparation of the known target, prompt identity, cache eviction, and execution-order effects. These controls verify reuse when the target request has the same prefix as the prepared request. They do not test whether independently prepared KV states can be inserted at a different position inside a concatenated prompt.

Ablation, sensitivity, and overlap. The standardized cumulative arms are semantic-only, +structure, +two-hop, +offline adaptation, and +online adaptation, all sharing topology and tie breaking. A CPU study sweeps $M \in \{5, 10, 15, 20, 25, 30\}$ and $K \in \{6, 10, 16\}$ on disjoint 600-event splits; it selects $M = 30$, but systems runs retain their pre-specified $M = 20$. The asynchronous runner makes use of separate foreground and background clients on a single T4. Full 600-event experiments use $M = 20$, $K = 10$, and lead times of 750 and 1500 ms; exploratory 60-event pilots test 1000 and 1250 ms. A new vLLM/LMCache process is started for each block. Native prefix-cache controls preserve the returned tokens and positive hit counts, with maximum absolute log-probability differences below 0.01. The stricter stock-LMCache gate, which uses a 0.001 tolerance, remains inconclusive; these measurements therefore characterize this exact-prefix test configuration rather than general cache composition.

Answer quality. We evaluate Gemini 3.5 Flash-Lite separately on HotpotQA and 2WikiMultiHopQA. Each dataset contributes 120 training questions, 40 development questions, and 60 held-out questions. At $K = 10$, all four policies answer the same held-out questions with an equal reference-context budget, capped at 768 tokens according to the Qwen tokenizer. Generation is limited to 64 tokens. We randomize policy order for each question. For the online policy, the labelled supporting sentence is revealed only after the context for the current question has been selected.

5 MEASUREMENTS

Prediction recall is the fraction of targets in the top K. Shadow residency/eviction comes from a byte-LRU audit model, whereas retrieved tokens are observed LMCache telemetry; estimated prefetch MiB is KV size rather than measured bus traffic. Ready-cache TTFT starts after population, while service time covers the target request. End-to-end accounting is

$$T_{\mathrm{E2E}} = T_{\mathrm{policy}} + T_{\mathrm{population,exposed}} + T_{\mathrm{ready}}. \tag{10}$$

The synchronous run exposes all population. In the asynchronous runner, arrival TTFT starts at scheduled target arrival; application TTFT begins before the current request and includes lead time and queueing. Answer metrics are normalized exact match (EM), token F1, supporting-fact recall after truncation, and delivery of the feedback target. Paired API intervals resample questions; sensitivity intervals resample source-document clusters; asynchronous intervals resample isolated blocks. Repeated traces measure execution variability but do not create new independent questions. FLOP figures from the archive are analytical, not GPU-profiled; we claim no measured-FLOP reduction.

6 RESULTS

Archived simulation. Across the five archived model configurations with adaptive results at $K = 10$, the historical adaptive policy lowers simulated-cache latency by 32.7–35.7% relative to

cosine; fixed graph is slightly slower. These model-specific means use different dataset pools and are not hardware scaling results (Appendix A).

Prediction and ready-cache latency. Offline fitting selects semantic/structural weights 0.45/0.55 (90.89% mean training recall). Table 1 shows that every graph-family policy has higher test recall and lower ready-cache TTFT than cosine at each budget.

Table 1: Real Qwen2.5-1.5B results. R is prediction recall (%); T is ready-cache TTFT (ms). Each policy–budget mean contains 1,200 repeated trace observations.

	$K = 6$		$K = 10$		$K = 16$	
Policy	R	T	R	T	R	T
No prefetch	0.00	73.19	0.00	73.41	0.00	72.16
Cosine	68.50	59.78	80.50	56.68	85.33	55.67
Fixed graph	74.83	56.58	89.50	53.22	92.50	53.37
Offline global	79.33	56.20	95.33	51.36	97.33	52.13
Online warm	75.83	55.85	95.17	52.03	97.33	50.69

At $K = 10$, offline gains 14.83 recall points and lowers ready TTFT by 5.32 ms (9.38%) relative to cosine. Online obtains 95.17% recall and 52.03 ms; its update frequency falls from 22.50% at $K = 6$ to 1.00% at $K = 16$, so it does not consistently beat offline. We observe the same ready-cache pattern in the additional runs shown in Table 2: on Llama/Hotpot, offline raises recall from 81.17% to 94.67% and lowers ready TTFT from 46.85 to 42.75 ms. On controlled 2Wiki traces it reaches 94.83% (Qwen) and 94.17% (Llama), versus cosine's 50.50% and 53.00%.

Table 2: Real vLLM/LMCache results at $K = 10$. R is target-prediction recall (%); T is ready-cache TTFT (ms). Each entry pools two repetitions of a 600-event trace.

Model and dataset	Policy	R	T
Qwen2.5-1.5B, Hotpot	Cosine	80.50	56.68
	Fixed graph	89.50	53.22
	Offline global	95.33	51.36
	Online warm	95.17	52.03
Llama-3.2-1B, Hotpot	Cosine	81.17	46.85
	Fixed graph	89.50	44.00
	Offline global	94.67	42.75
	Online warm	94.67	42.61
Qwen2.5-1.5B, 2Wiki	Cosine	50.50	70.56
	Fixed graph	69.67	64.47
	Offline global	94.83	58.17
	Online warm	94.50	58.37
Llama-3.2-1B, 2Wiki	Cosine	53.00	54.67
	Fixed graph	76.17	51.42
	Offline global	94.17	47.92
	Online warm	93.33	48.07

Preparation cost. At $K = 10$, offline lowers synchronous end-to-end TTFT from cosine's 902.67 to 883.27 ms, but no prefetch needs only 73.41 ms because it performs no speculative work (Table 3). Offline shadow residency is 96.17% versus 83.00%, and LMCache retrieves 369.05 versus 317.40 tokens. Increasing K from 10 to 16 adds only two recall points while population rises from 828.48 to 1354.62 ms.

Ablation and overlap. At $M = 20, K = 10$, cumulative recall is 80.50% for semantic-only, 88.00% with structure, 89.50% with two hops, 95.33% with offline adaptation, and 95.17% online; clustered-bootstrap intervals for every addition after semantic-only exclude zero. Development selects

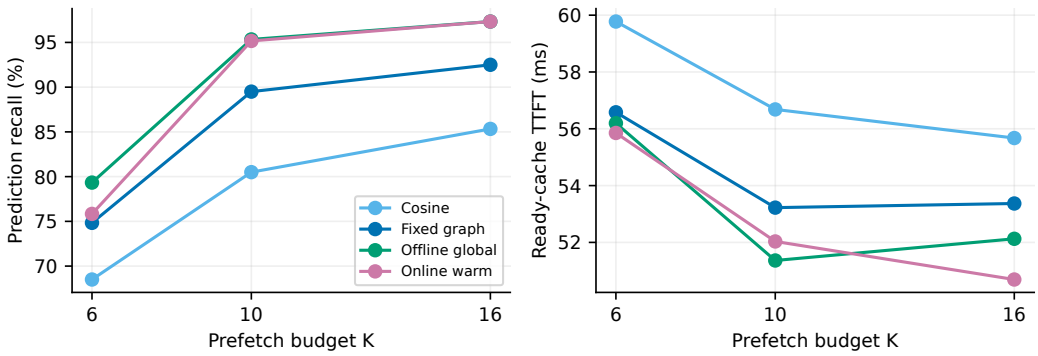


Figure 1: Real Qwen2.5-1.5B target prediction recall and ready-cache TTFT. Each point pools two repetitions of 600 trace events.

Table 3: End-to-end TTFT (ms), including policy time and all synchronous candidate-population time.

Policy	$K = 6$	$K = 10$	$K = 16$
No prefetch	73.20	73.41	72.17
Cosine	557.15	902.67	1428.38
Fixed graph	552.29	891.05	1411.78
Offline global	550.06	883.27	1410.24
Online warm	551.09	883.36	1406.41

$M = 30$, but its mean recall is only 0.17 points above $M = 20$ while mean degree rises from 25.72 to 38.35; we therefore retain the pre-specified $M = 20$ systems setting (Appendix E).

Table 4: CPU-only M sensitivity on the 600-event Hotpot development trace. Mean recall averages the offline policy over $K \in \{6, 10, 16\}$. Density is for the semantic-plus-structure graph.

M	Mean dev recall (%)	Directed edges	Mean degree	$K = 10$ test offline R (%)
5	90.89	8,442	6.90	93.67
10	92.22	15,574	12.73	94.50
15	92.72	23,482	19.20	95.17
20	92.83	31,458	25.72	95.33
25	92.89	39,252	32.09	96.50
30	**93.00**	46,900	38.35	**96.50**

At 750 ms, no prefetch has the lowest application TTFT (914.72 ms), while online is 6.06 ms faster than cosine in arrival TTFT (95% block-bootstrap CI $[-10.08, -2.50]$). Increasing the lead to 1500 ms lets every prefetching arm complete all ten candidates before arrival. Offline then raises recall from cosine's 81.67% to 94.33% and lowers arrival TTFT by 3.07 ms (CI $[-3.86, -2.22]$), but its application TTFT is 9.71 ms above cosine (CI $[7.07, 12.17]$) and 6.29 ms above no prefetch (CI $[4.35, 8.00]$). Cosine is 3.43 ms below no prefetch, but its CI $[-7.36, 0.78]$ crosses zero. Thus, additional lead improves cache readiness but does not produce a confirmed end-to-end GraphKV advantage on this single-T4 workload. The 60-event 1250-ms crossover is retained only as exploratory evidence; complete ablations are in Appendix E.

Answer quality. Table 6 gives the results of both completed API experiments. On HotpotQA, using a fixed graph raises F1 by 4.78 points as compared to cosine, and offline adaptation increases it by 5.89 points; offline EM is 45.00%, which amounts to 27 exact answers, as opposed to cosine's 23. The online warm approach achieves 25 exact answers and an F1 score of 48.51, this being only 0.57 points higher than that of cosine.

Table 5: Full asynchronous Hotpot runs ($M = 20$, $K = 10$, $N = 600$ per policy). Ready is arrival-to-first-token latency. ΔNP and ΔC are paired application-TTFT differences from no prefetch and cosine; negative is faster.

Lead	R (%)	Policy	Ready	App. TTFT	ΔNP	ΔC
750	0.00	No prefetch	79.09	914.72	0.00	–
	81.67	Cosine	98.14	974.94	60.22	0.00
	94.33	Offline	94.07	979.97	65.25	5.03
	93.83	Online	92.08	980.95	66.23	6.01
1500	0.00	No prefetch	90.13	1683.10	0.00	–
	81.67	Cosine	56.88	**1679.67**	-3.43	0.00
	94.33	Offline	**53.81**	1689.39	6.29	9.71
	93.83	Online	54.26	1689.95	6.85	10.28

Table 6: Gemini 3.5 Flash-Lite answer quality at $K = 10$, $M = 5$. All metrics are percentages; each policy answers the same 60 questions within a dataset. Support is fully delivered supporting-sentence recall.

Policy	EM	F1	Support	Target hit
HotpotQA				
Cosine	38.33	47.94	63.61	61.67
Fixed graph	43.33	52.72	64.17	61.67
Offline global	45.00	53.83	63.89	60.00
Online warm	41.67	48.51	63.06	58.33
2WikiMultiHopQA				
Cosine	43.33	48.18	68.33	76.67
Fixed graph	33.33	39.24	64.17	66.67
Offline global	35.00	40.53	66.25	66.67
Online warm	35.00	39.91	66.25	66.67

On 2Wiki, cosine comes out ahead in all the metrics since it provides 26 exact answers compared to 21 answered by offline, which has an F1 score 7.65 points lower. The Hotpot F1 confidence intervals cross zero; fixed graph's 2Wiki loss is resolved at 95%, while offline's interval crosses zero (Appendix E). In terms of mechanism, offline GraphKV produces a smaller candidate set on HotpotQA (22.85 instead of 41.37) while still maintaining support recall, which means that redundant text is removed before truncation. On 2Wiki it also considers fewer candidates (17.98 rather than 29.22), but eight questions that result in a loss of support account for 8.24 points lost in aggregate F1. This truncation-sensitivity is consistent with findings that LLMs use evidence unevenly across context position (Liu et al., 2024a); on 2WikiMultiHopQA, losing supporting evidence to truncation may matter more if that evidence would otherwise fall in a less-attended region of the reference context. The weights that it has learned are $(1.0, 0.0)$, indicating that it is relying on candidate reachability and pruning rather than on excessive structural weighting. Controlled next-access prediction and natural-question evidence selection are therefore distinct tasks.

Table 7: Paired F1 differences from cosine in the 60-question API study. Values and confidence intervals are percentage points from question-level bootstrap resampling.

Dataset	Policy	F1 difference	95% CI
HotpotQA	Fixed graph	+4.78	$[-5.82,\ 15.60]$
HotpotQA	Offline global	+5.89	$[-3.94,\ 15.91]$
2WikiMultiHopQA	Fixed graph	-8.93	$[-18.34,\ -0.15]$
2WikiMultiHopQA	Offline global	-7.65	$[-17.80,\ 2.10]$

7 DISCUSSION

The experiments separate candidate-selection quality from complete system latency. Structural edges, two-hop reachability, and offline adaptation each improve target prediction, and the resulting cache state reduces ready-cache TTFT across two models and two controlled traces. These gains are therefore not attributable to a single hidden difference between the graph policies. However, predicting a useful chunk does not make its preparation free. When candidate population runs synchronously, its cost dominates the time saved during the subsequent target request.

The asynchronous experiments clarify the operating condition needed to overcome this cost. A lead time of 750 ms hides part of candidate preparation but does not make any prefetching policy faster than no-prefetch in application TTFT. A 60-event pilot crosses over at 1250 ms, but the confirmatory 600-event run at 1500 ms does not reproduce an end-to-end GraphKV advantage, even though every selected candidate completes before arrival. The graph policies still reduce arrival TTFT relative to no prefetch and improve prediction over cosine, but single-GPU preparation and scheduling overhead remain larger than the service-time saving. Practical deployment therefore requires greater reuse, cheaper or batched population, or genuinely independent background resources.

The real-cache benchmark deliberately requests each target chunk using the same exact prefix under which it was prepared. It therefore validates advance computation, storage, and reuse for exact individual prompts. It does not assume that KV states independently calculated for several chunks can be inserted without correction into a newly concatenated RAG prompt. Different token positions and cross-chunk attention can invalidate that assumption. Extending GraphKV to full prompt composition consequently requires CacheBlend-style selective recomputation, or another position- and attention-aware composition mechanism.

The answer-quality experiments reveal a separate dataset-dependent boundary. On HotpotQA, offline graph selection reduces the mean candidate universe from 41.37 to 22.85 sentences while preserving approximately the same supporting-fact recall as cosine. Under the shared token limit, graph pruning therefore removes redundant context without removing much labelled evidence, producing a positive mean F1 difference. On 2WikiMultiHopQA, the offline candidate universe is also smaller, but some necessary evidence is missing before token-budget truncation. Eight questions on which offline selection loses supporting evidence account for an aggregate 8.24 F1 points of loss. The learned 2Wiki weights are $(1.0, 0.0)$, so this result cannot be explained simply by excessive structural weighting. It primarily reflects limited graph reachability and pruning.

Offline and online adaptation address different problems. Offline adaptation provides one interpretable semantic–structural balance learned from training data and frozen before testing. Online adaptation responds to ranking mistakes after deployment, but it cannot recover a target that is absent from the graph's candidate universe. Its usefulness therefore depends on encountering informative ranking misses rather than coverage misses. A practical deployment could monitor graph coverage and available overlap, using GraphKV when both are sufficient and falling back to global cosine selection otherwise. We leave such a learned routing policy to future work.

8 CONCLUSION

GraphKV incorporates both semantic and structural edges, uses two-hop traversal, involves offline fitting and allows for optional online updates. When tested on two small models and two controlled traces it achieves better predictions and a reduced ready-cache TTFT compared to cosine; the ablation studies attribute the improvements to the structure, the traversal, and the adaptation mechanism. Neither the 750-ms nor the 1500-ms full asynchronous run establishes an application-level GraphKV advantage over no prefetch, despite complete preparation at 1500 ms. Gemini quality increases on average for Hotpot but decreases on 2Wiki when graph pruning eliminates evidence. GraphKV is thus a workload-dependent selection layer whose deployment must account for graph coverage, reuse, overlap, and preparation cost.

REPRODUCIBILITY STATEMENT

We support reproducibility by documenting the graph construction procedure, offline/online adaptation algorithms, vLLM/LMCache experimental configuration, correctness gates, and bootstrap confidence-interval methodology in the appendix. Code, CSV outputs, and figure-generation scripts are provided in the accompanying repository.

REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of ACL*, 2024. URL https://aclanthology.org/2024.acl-long.172/.

Minsik Cho, Mohammad Rastegari, and Devang Naik. KV-runahead: Scalable causal LLM inference by parallel key-value cache generation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 8578–8592, 2024. URL https://proceedings.mlr.press/v235/cho24e.html.

In Gim, Guojun Chen, Seung-seob Lee, Nikhil Sarda, Anurag Khandelwal, and Lin Zhong. Prompt cache: Modular attention reuse for low-latency inference. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 325–338, 2024. URL https://arxiv.org/abs/2311.04934.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of COLING*, 2020. URL https://aclanthology.org/2020.coling-main.580/.

Shibo Jie, Yehui Tang, Kai Han, Zhi-Hong Deng, and Jing Han. SpeCache: Speculative key-value caching for efficient generation of LLMs. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 27917–27928, 2025. URL https://proceedings.mlr.press/v267/jie25a.html.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2405.14831.

Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Xin Liu, Xuanzhe Liu, and Xin Jin. RAGCache: Efficient knowledge caching for retrieval-augmented generation, 2024. URL https://arxiv.org/abs/2404.12457.

Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020. URL https://arxiv.org/abs/2004.04906.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023. doi: 10.1145/3600006.3613165. URL https://arxiv.org/abs/2309.06180.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, 2020. URL https://arxiv.org/abs/2005.11401.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024a. URL https://arxiv.org/abs/2307.03172.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. CacheGen: Kv cache compression and streaming for fast large language model serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024b. doi: 10.1145/3651890.3672274. URL https://arxiv.org/abs/2310.07240.

LMCache Contributors. LMCache: Documentation and source code, 2026. URL https://docs.lmcache.ai/. Accessed September 6, 2026.

Songshuo Lu, Hua Wang, Yutian Rong, Zhi Chen, and Yaohua Tang. TurboRAG: Accelerating retrieval-augmented generation with precomputed KV caches for chunked text. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6588–6601, 2025. doi: 10.18653/v1/2025.emnlp-main.334. URL https://aclanthology.org/2025.emnlp-main.334/.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of EMNLP-IJCNLP*, 2019. URL https://aclanthology.org/D19-1410/.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. URL https://aclanthology.org/2022.tacl-1.31/.

Wenhan Xiong, Xiang Lorraine Li, Srini Iyer, Jingfei Du, Patrick Lewis, William Yang Wang, Yashar Mehdad, Wen-tau Yih, Sebastian Riedel, Douwe Kiela, and Barlas Oğuz. Answering complex open-domain questions with multi-hop dense retrieval. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2009.12756.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of EMNLP*, 2018. URL https://aclanthology.org/D18-1259/.

Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. CacheBlend: Fast large language model serving for RAG with cached knowledge fusion, 2024. URL https://arxiv.org/abs/2405.16444.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2O: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2306.14048.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2312.07104.

AI Assistance Disclosure

We used generative AI tools to assist with manuscript editing, readability, LaTeX formatting, section organization, figure-caption refinement, and recovery or reformatting of LaTeX source files. We also used AI assistance to discuss how to present the benchmark, evaluation protocol, and results more clearly. The authors ultimately take the responsibility of all the content.

A Complete Archived Simulation Sweep

The following tables reproduce the available mean simulated-cache latencies from the archived summary. Each model retains its own dataset pool. C denotes cosine, G fixed graph, and A the historical query-type adaptive policy. Dashes mean no value was present. These values are not timings from the stock LMCache experiment.

Qwen2.5-1.5B

Dataset pool: 2WikiMQA, HotpotQA, MultiFieldQA-en, MuSiQue.

K	No prefetch	C	G	A
3	31.592	13.975	22.544	19.315
4	31.916	13.177	20.574	16.493
5	31.817	12.858	18.710	13.571
6	31.550	12.219	16.946	11.427
7	30.514	11.461	15.409	10.324
8	31.739	11.846	14.287	9.102
10	31.932	11.391	11.872	7.503
12	31.431	10.395	10.877	7.031
14	31.818	10.395	10.112	6.272
16	31.855	10.512	9.216	5.814

Qwen2.5-3B

Dataset pool: 2WikiMQA, HotpotQA, MultiFieldQA-en, MuSiQue.

K	No prefetch	C	G	A
3	37.283	17.173	28.632	24.470
4	37.584	16.033	25.227	20.242
5	37.028	15.601	22.534	16.854
6	37.668	15.439	20.428	13.683
7	37.180	14.789	18.958	12.413
8	37.232	14.387	17.246	10.849
10	37.120	13.360	14.157	8.992
12	37.189	12.522	12.891	7.990
14	37.463	12.351	11.928	6.840
16	37.167	12.356	11.336	6.272

Qwen2.5-7B

Dataset pool: 2WikiMQA, HotpotQA, MultiFieldQA-en, MuSiQue.

K	No prefetch	C	G	A
3	50.338	23.214	36.822	31.843
4	50.461	21.017	32.553	26.154
5	49.759	20.414	29.166	21.626
6	49.802	19.805	26.344	18.054
7	50.768	19.639	25.637	16.960
8	48.775	18.697	21.677	13.833
10	48.269	17.552	18.001	11.283
12	48.400	16.157	16.343	10.075
14	48.170	15.783	15.086	8.711
16	48.063	15.590	13.931	7.948

LLAMA-3.2-1B

Dataset pool: MultiFieldQA-en, MuSiQue.

K	No prefetch	C	G	A
3	23.724	10.656	17.219	14.612
4	22.652	9.733	14.817	11.550
5	23.033	9.412	13.575	10.006
6	23.201	9.339	12.254	8.525
8	22.739	8.924	10.156	6.735
10	23.178	8.270	8.549	5.542
12	22.505	7.838	7.448	5.050
14	22.704	7.459	7.034	4.326
16	23.011	7.253	6.524	3.933

LLAMA-3.2-3B

Dataset pool: HotpotQA.

K	No prefetch	C	G	A
3	54.003	26.870	45.441	–
4	55.396	25.291	40.555	–
5	55.384	25.104	35.538	–
6	55.343	24.699	32.689	–
8	55.290	23.788	25.671	–
10	55.425	22.406	22.555	–
12	55.480	21.124	19.942	–
14	55.511	21.042	18.399	–
16	55.004	21.278	17.596	–

GEMMA-3-1B

Dataset pool: 2WikiMQA, HotpotQA, MultiFieldQA-en, MuSiQue.

K	No prefetch	C	G	A
3	41.250	19.195	30.620	25.906
4	41.790	18.015	27.897	21.926
5	40.328	16.925	24.164	17.796
6	39.578	16.456	21.514	14.818
8	39.811	15.685	17.967	11.458
10	40.575	14.891	15.855	9.803
12	39.140	13.455	13.765	8.006
14	40.418	13.311	13.063	7.561
16	39.934	13.627	12.235	6.868

B CONFIGURATION AND REPRODUCTION

The source bundle includes the exact pooled Qwen CSV, archived simulation summary, API table, cross-model summaries, standardized sensitivity outputs, and asynchronous aggregate files used by the added tables. Tables and vector plots are generated from these files by `build_results.py`; the additional revision tables retain their source CSVs alongside the manuscript. The main vLLM configuration uses seed 43, $M = 20$, $K = 6, 10, 16$, 600 accesses per split, and two repetitions. The 2Wiki extension uses the same ranking formula at $K = 10$. The asynchronous revision uses seed 44 and fresh processes for every sequence block; the 750- and 1500-ms runs each contain six 100-event blocks per policy. The API commands use seed 42, $M = 5$, $K = 10$, 120/40/60 training/development/test questions, the MiniLM embedding model, and the Qwen reference tokenizer. All model, dataset, code fingerprint, and output identities should be retained when resuming a run.

C Complete Real-Run Diagnostics

The tables below reproduce diagnostic fields from the pooled headline output; the bundled CSV retains every column, including current-prefetch hit rate. Shadow residency and eviction counts originate from the byte-LRU audit model; retrieved tokens originate from LMCache counters. MiB values are estimated KV size of completed preparation. Service is the complete target request, and E2E request includes policy and exposed population time.

Budget $K = 6$

Policy	Service ms	Population ms	E2E request ms	Policy ms
No prefetch	73.32	0.00	73.32	0.01
Cosine	59.91	497.13	557.29	0.24
Fixed graph	56.68	492.29	552.39	3.41
Offline global	56.30	490.46	550.16	3.40
Online warm	55.96	491.82	551.19	3.41

Policy	KV MiB	Shadow res. %	Reused tokens	Evictions	Update %
No prefetch	0.00	24.00	83.85	0.00	0.00
Cosine	62.24	72.50	278.14	4.88	0.00
Fixed graph	62.35	79.50	302.60	4.68	0.00
Offline global	62.45	82.67	318.32	4.62	0.00
Online warm	62.37	80.33	319.37	4.64	22.50

Budget $K = 10$

Policy	Service ms	Population ms	E2E request ms	Policy ms
No prefetch	73.53	0.00	73.53	0.01
Cosine	56.90	845.73	902.89	0.26
Fixed graph	53.33	834.37	891.15	3.45
Offline global	51.46	828.48	883.37	3.42
Online warm	52.13	827.88	883.46	3.45

Policy	KV MiB	Shadow res. %	Reused tokens	Evictions	Update %
No prefetch	0.00	24.00	79.68	0.00	0.00
Cosine	102.73	83.00	317.40	8.62	0.00
Fixed graph	102.72	91.17	353.96	8.26	0.00
Offline global	102.76	96.17	369.05	8.09	0.00
Online warm	102.79	96.00	365.13	8.08	3.17

Budget $K = 16$

Policy	Service ms	Population ms	E2E request ms	Policy ms
No prefetch	72.28	0.00	72.29	0.01
Cosine	55.85	1372.45	1428.55	0.26
Fixed graph	53.47	1354.90	1411.89	3.51
Offline global	52.23	1354.62	1410.34	3.50
Online warm	50.79	1352.22	1406.51	3.49

Policy	KV MiB	Shadow res. %	Reused tokens	Evictions	Update %
No prefetch	0.00	24.00	82.22	0.00	0.00
Cosine	163.21	86.17	334.05	14.48	0.00
Fixed graph	161.90	93.17	345.43	14.13	0.00
Offline global	162.15	97.67	365.58	13.98	0.00
Online warm	162.15	97.67	377.12	13.98	1.00

D CROSS-MODEL REAL-CACHE RESULTS

Table 8 gives every tested Llama/Hotpot budget. Table 9 gives the complete 2Wiki extension for both models. The E column includes all synchronous candidate preparation and is therefore expected to remain above no prefetch.

Table 8: Complete Llama-3.2-1B Hotpot real-cache means. R is prediction recall (%), T is ready-cache TTFT (ms), and E is end-to-end TTFT (ms) with synchronous population. Each entry pools two 600-event repetitions.

	$K = 6$			$K = 10$			$K = 16$		
Policy	R	T	E	R	T	E	R	T	E
No prefetch	0.00	53.51	53.52	0.00	55.24	55.25	0.00	54.30	54.30
Cosine	70.83	45.51	409.31	81.17	46.85	684.45	84.17	46.33	1090.01
Fixed graph	76.33	45.65	424.46	89.50	44.00	677.36	92.50	43.30	1074.11
Offline global	78.33	43.85	418.69	94.67	42.75	664.29	97.67	44.01	1077.14
Online warm	78.00	44.75	421.96	94.67	42.61	670.89	97.67	43.42	1069.89

Table 9: Complete $K = 10$ real-cache extension on 2WikiMultiHopQA. R is prediction recall (%), T is ready-cache TTFT (ms), and E is synchronous end-to-end TTFT (ms). Each entry pools two 600-event repetitions.

Model	Policy	R	T	E
Qwen2.5-1.5B	No prefetch	0.00	73.79	73.80
	Cosine	50.50	70.56	937.80
	Fixed graph	69.67	64.47	922.99
	Offline global	94.83	58.17	905.98
	Online warm	94.50	58.37	905.83
Llama-3.2-1B	No prefetch	0.00	56.03	56.03
	Cosine	53.00	54.67	731.38
	Fixed graph	76.17	51.42	723.04
	Offline global	94.17	47.92	704.02
	Online warm	93.33	48.07	705.44

E Ablation and Overlap Details

Tables 10 and 11 report the complete cumulative ablations under 750 and 1500 ms of asynchronous lead time. All policies use the same test trace, graph construction, tie-breaking rules, and prefetch budget. Consequently, differences between adjacent graph rows isolate the addition of structural edges, two-hop traversal, offline adaptation, and online adaptation.

Table 10: Full asynchronous component ablation on Hotpot ($M = 20$, $K = 10$, lead $= 750$ ms, $N = 600$). R is target-prediction recall. Ready is arrival-to-first-token latency; application TTFT begins before the current request and includes the scheduled lead.

Policy	R (%)	Ready ms	App. TTFT ms	Target ready (%)
No prefetch	0.00	79.09	914.72	0.00
Cosine	81.67	98.14	974.94	79.67
Semantic only	81.67	100.96	990.33	79.50
+ structure	87.00	97.38	986.07	86.17
+ two hop	87.83	98.93	987.99	84.83
+ offline adaptation	**94.33**	94.07	979.97	**89.00**
+ online adaptation	93.83	**92.08**	980.95	88.50

Table 11: Full asynchronous component ablation on Hotpot ($M = 20$, $K = 10$, lead $= 1500$ ms, $N = 600$). R is target-prediction recall; all prefetching arms completed ten candidates before arrival.

Policy	R (%)	Ready ms	App. TTFT ms	Target ready (%)
No prefetch	0.00	90.13	1683.10	0.00
Cosine	81.67	56.88	**1679.67**	81.67
Semantic only	81.67	58.05	1695.73	81.67
+ structure	87.00	55.50	1690.50	87.00
+ two hop	87.83	56.17	1691.87	87.83
+ offline adaptation	**94.33**	**53.81**	1689.39	**94.33**
+ online adaptation	93.83	54.26	1689.95	93.83