

LivingRAG: Augmenting Graph RAG with Experience

Yuzhuo Cui^{1,2}, Zongye Zhang^{1,2}, Qingjie Liu^{1,2,*}

¹State Key Laboratory of Virtual Reality Technology and Systems,
Beihang University, Beijing, China

²Hangzhou Innovation Institute, Beihang University,
Hangzhou, Zhejiang, China

{cyz020403, zhangzongye, qingjie.liu}@buaa.edu.cn

*Corresponding author

Abstract

Graph-based RAG improves multi-hop question answering by organizing evidence as a knowledge graph. However, most existing RAG systems process each query in isolation and discard useful reasoning from the LLM’s response after inference. As a result, later related queries need to retrieve evidence and reason from scratch. We propose LivingRAG, a Graph RAG framework with writable and reusable reasoning experience. LivingRAG adds a writable experience store to a graph-based retrieval backbone, enabling verified experiences to be reused during inference in two ways. Stored graph signals help retrieval find entities and passages that were useful in earlier related queries. Stored summaries provide a reference reasoning pattern for answer generation. We analyze online QA streams and find reusable signals from shared entities, graph neighborhoods, and question templates. Experiments on multi-hop QA benchmarks show that LivingRAG improves accuracy over strong RAG baselines and reduces completion-token use when relevant prior experience is reused.

1 Introduction

Large language models perform well on many language understanding and reasoning tasks. However, they still face limits in knowledge-intensive tasks. They may hallucinate facts. They may also lack access to domain-specific or up-to-date knowledge. Retrieval-augmented generation addresses these limits by grounding model outputs in external documents (Lewis et al., 2020; Karpukhin et al., 2020; Gao et al., 2024).

Graph-based RAG further improves retrieval for complex questions. It represents passages, entities, and their relations in a graph. The graph structure helps the retriever find multi-hop evidence that dense retrieval may miss. Recent systems show that graph structure is useful for global context and entity-level reasoning (Edge et al., 2024; Guo et al.,

2024; Gutiérrez et al., 2025; Zhuang et al., 2025; Luo et al., 2026b). However, online QA streams raise a further question. When related queries arrive over time, should the system reuse useful reasoning from earlier queries?

Despite this progress, most RAG and Graph RAG systems still answer each query independently at inference time (Edge et al., 2024; Guo et al., 2024; Gutiérrez et al., 2025; Zhuang et al., 2025; Luo et al., 2026b). These methods retrieve evidence for the current query and then generate an answer, but they discard most query-level reasoning signals after inference. As a result, later related queries often need to retrieve and reason again from scratch. This read-only inference process limits reuse in online QA streams.

This limitation matters because useful reuse is not limited to repeated entities. Two questions may visit nearby regions of the retrieval graph. They may also follow the same reasoning pattern with different entities. For example, two comparison questions may both ask which person was born earlier, even when the people and evidence passages differ. Therefore, a useful online Graph RAG system should preserve more than final answers. It should preserve graph-neighborhood priors and reusable reasoning patterns.

To address this problem, we introduce LivingRAG, a Graph RAG framework with writable and reusable reasoning experience. LivingRAG adds a writable experience store to a graph-based retrieval backbone. After each query, it can write a compact experience if the experience is grounded and novel. Each stored experience keeps the source query, activated graph entities, a short reasoning summary, and validation metadata. During later inference, LivingRAG reuses stored experiences in two ways. Activation maps guide graph retrieval toward useful evidence neighborhoods. Reasoning summaries provide scaffolds for answer generation.

A writable experience store also introduces a

risk. If unsupported reasoning is written into the store, later queries may reuse it and amplify the error. LivingRAG reduces this risk with quality gates. A candidate is stored only when its claims are supported by retrieved evidence and when it is not redundant with stored experiences. Thus, LivingRAG can return an answer without storing low-quality experience.

We evaluate LivingRAG on multi-hop QA benchmarks. The experience store is initialized at the beginning of each run and updated online during that run. Across these benchmarks, LivingRAG improves accuracy over strong RAG baselines. It also reduces completion-token use and estimated API cost when later queries retrieve useful prior experience.

Our contributions are: (1) We introduce LivingRAG, a Graph RAG framework with writable and reusable reasoning experience. It stores grounded and novel experiences, and reuses them through activation maps for retrieval and reasoning summaries for generation. (2) We analyze online QA streams and find reusable signals from shared entities, graph neighborhoods, and question templates. We show that LivingRAG can reuse these signals in both multi-hop QA and support-style QA settings. (3) We show that LivingRAG improves accuracy over strong RAG baselines. It also reduces completion-token use and estimated API cost when relevant prior experience is reused.

2 Related Work

Retrieval-augmented generation grounds language models in external documents and reduces reliance on parametric knowledge (Lewis et al., 2020; Karpukhin et al., 2020; Gao et al., 2024). Early systems usually retrieve independent text chunks. Recent Graph RAG methods add graph structure to support complex retrieval. GraphRAG builds entity graphs and community summaries (Edge et al., 2024). LightRAG combines graph indexing with vector retrieval (Guo et al., 2024). HippoRAG and HippoRAG2 use graph-based associative retrieval (Gutiérrez et al., 2024; Gutiérrez et al., 2025). LinearRAG uses a lightweight passage-sentence-entity graph for efficient retrieval (Zhuang et al., 2025). These systems show that graph structure helps multi-hop evidence search. LivingRAG builds on this direction, but studies a different problem. It asks how a Graph RAG system can write and reuse verified reasoning experience across an

online query stream.

Several methods also improve RAG with reasoning or memory. Reasoning-enhanced RAG methods use decomposition, intermediate notes, self-reflection, or tool use to improve single-query reasoning (Trivedi et al., 2023; Jiang et al., 2023; Asai et al., 2024; Yu et al., 2024; Kabir et al., 2025). Memory-based agents store past observations or feedback for later decisions (Shinn et al., 2023; Madaan et al., 2023). Recent memory-augmented RAG systems further show that reusable memory can support retrieval. REMem builds a hybrid episodic memory graph for language-agent histories and uses an agentic retriever for episodic recollection and reasoning (Shu et al., 2026). MemoRAG builds a global memory for long-context processing and uses it to produce clues for retrieval (Qian et al., 2025). LivingRAG is complementary to these works. It focuses on online Graph RAG for QA streams. Its written unit is a verified reasoning experience, not a raw interaction history or a global long-context memory. Each experience stores graph activation information and a compact reasoning scaffold. It is written only after grounding and novelty checks. The experience is not only retrieved as text. It also changes the initial graph activation used by the retriever.

3 Method

3.1 Motivating Analysis

LivingRAG is designed for online QA streams. In this setting, the system answers queries one by one. For a query q_t , only earlier queries $q_1, \dots, q_{t-1}$ can be used as reuse sources. This causal setup matches an online experience store and avoids using future information.

We measure three reuse signals. Entity reuse occurs when the current query shares at least one seed entity with an earlier query. Graph-neighborhood reuse occurs when the current query and an earlier query visit nearby regions of the retrieval graph. We measure this signal with a fixed LinearRAG retriever. Two queries are counted as graph-neighborhood reuse when their retrieved passages or filtered context entities have Jaccard similarity of at least 0.05. Template reuse occurs when two queries have no seed-entity overlap but have similar masked questions. We mask entity mentions, quoted spans, and numeric as Section 3.4. We then compute TF-IDF cosine similarity between the masked questions. Two queries are counted as

Table 1: Reuse signals in online QA streams. Only earlier queries are used as reuse sources. All values are percentages.

Dataset	Entity reuse	Graph-neigh. reuse	Template reuse
2Wiki	4.30	99.70	96.30
HotpotQA	26.53	99.10	21.82
MuSiQue	40.34	95.60	39.14
MuSiQue-full	54.10	96.32	54.30
WixQA	76.69	80.45	16.04

template reuse when the score is at least 0.35.

Table 1 shows that reusable signals exist across online QA streams, but they appear in different forms. Direct entity reuse is limited in most datasets, so an entity-only cache would miss many opportunities. In contrast, graph-neighborhood reuse is common across datasets. This suggests that later queries often need evidence from regions that are close to earlier retrieval paths.

Template reuse is also useful, but it is more dataset-dependent. It is stronger in formulaic multi-hop streams and weaker in more diverse support-style questions. These results motivate LivingRAG to reuse experience through two paths.

3.2 Overview

LivingRAG extends a graph-based RAG system with a writable experience store. The base retriever follows LinearRAG (Zhuang et al., 2025). Given a document corpus $\mathcal{D}$, it builds a retrieval graph with passage nodes, sentence nodes, and entity nodes. The graph is constructed with lightweight entity extraction and does not require LLM calls during indexing. For each query, LinearRAG first activates query-related entities, propagates relevance over the sentence-entity graph, and then ranks passages with Personalized PageRank over the passage-entity graph. LivingRAG keeps this retrieval backbone unchanged. It adds a writable experience store during an online QA stream. LivingRAG does not reuse previous answers as a response cache. It reuses verified activation maps and reasoning scaffolds during retrieval and generation. Figure 1 gives an overview of the full workflow.

At time step t , the system receives a query q_t and has access to a static retrieval graph $\mathcal{G}$ and an experience store $\mathcal{E}_t$. A standard graph-based RAG system answers each query independently:

$$a_t = \text{LLM}(q_t, \mathcal{R}(q_t, \mathcal{G})), \quad (1)$$

where $\mathcal{R}$ retrieves passages from the graph. LivingRAG instead uses stored experiences during both

retrieval and generation:

$$a_t = \text{LLM}(q_t, \mathcal{R}_{\text{exp}}(q_t, \mathcal{G}, \mathcal{E}_t), \mathcal{S}_{\text{exp}}(q_t, \mathcal{E}_t)). \quad (2)$$

Here $\mathcal{R}_{\text{exp}}$ is experience-augmented graph retrieval, and $\mathcal{S}_{\text{exp}}$ is an optional reasoning scaffold selected from the experience store. If no useful stored experience is found, LivingRAG reduces to the base graph RAG system.

After answering query q_t , LivingRAG forms a *candidate experience* C_t . The candidate contains the query, retrieved passages, generated explanation, and final answer. LivingRAG then decides whether the candidate should be written to the store. If the candidate passes the grounding and novelty checks, it becomes a *verified experience* e_t . We use C_t for the candidate before validation and e_t for the verified experience after validation. The experience-store update is:

$$\mathcal{E}_{t+1} = \begin{cases} \mathcal{E}_t \cup \{e_t\}, & \text{if Store}(C_t, \mathcal{E}_t) = 1, \\ \mathcal{E}_t, & \text{otherwise.} \end{cases} \quad (3)$$

The function Store checks whether the candidate is grounded in retrieved evidence and sufficiently different from stored experiences. This write-back mechanism lets LivingRAG accumulate reusable experiences without changing model parameters.

3.3 Experience Store

The basic unit of the store is a verified experience. After this definition, we use experience as a short form when the meaning is clear. Each verified experience is a compact form of one candidate experience that passed validation:

$$e_t = (q_t, \mathbf{v}_t, \tilde{\mathbf{a}}_t, \text{sum}_t, a_t, ts_t, \text{conf}_t). \quad (4)$$

Here q_t is the source query, $\mathbf{v}_t = \text{Emb}(q_t)$ is its query embedding, $\tilde{\mathbf{a}}_t$ is a sparse activation map over graph entities, sum_t is a compact reasoning summary, a_t is the final answer, ts_t is the timestamp, and conf_t is the grounding confidence.

The activation map is the main graph-space representation of a verified experience. After the base retriever finishes entity propagation for query q_t , it produces a final entity activation vector $\mathbf{a}_{q_t}^{\text{final}}$. LivingRAG sparsifies this vector:

$$\tilde{\mathbf{a}}_t = \text{sparsify}(\mathbf{a}_{q_t}^{\text{final}}, \epsilon), \quad (5)$$

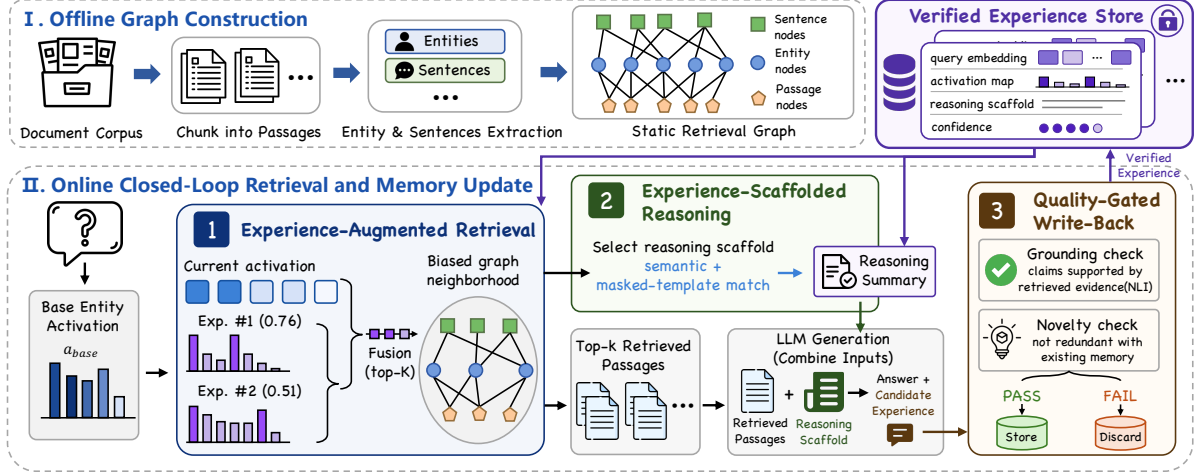


Figure 1: Overview of LivingRAG. The framework first constructs a static retrieval graph offline. It then performs experience-augmented retrieval, scaffolded generation, and write-back through quality gates.

where entries below threshold ϵ are set to zero. The sparse activation map stores which graph entities were useful for the candidate and how strongly they were activated. Unlike a plain entity list, the map preserves graded graph-neighborhood information and can be reused directly in later graph retrieval.

The compact reasoning summary becomes a reasoning scaffold during later generation. It is not the full raw reasoning trace. To keep the prompt short, LivingRAG selects a small number of key sentences from the generated explanation. The selection is anchored by the most activated entities in $\mathbf{a}_{q_t}^{\text{final}}$. Sentences that mention more activated entities are ranked higher, and the selected sentences are kept in their original order. The stored summary is:

$$\text{sum}_t = (q_t, \text{KeySent}(r_t, \mathbf{a}_{q_t}^{\text{final}}), a_t), \quad (6)$$

where r_t is the generated explanation. The compact summary preserves the main reasoning pattern while adding few tokens to future prompts.

3.4 Experience-Augmented Retrieval and Scaffolded Generation

LivingRAG uses the experience store in two places. First, it uses activation maps to guide graph retrieval. Second, it uses compact summaries as reasoning scaffolds for answer generation. The two paths are related but separate. The retrieval path transfers graph-neighborhood information. The generation path transfers a reusable reasoning pattern.

For graph retrieval, LivingRAG first matches the current query with stored experiences. Each

stored experience is scored by combining semantic similarity and activation-map similarity:

$$s_k^{\text{ret}} = \alpha \cos(\text{Emb}(q_t), \mathbf{v}_k) + (1 - \alpha) \cos(\mathbf{a}_{q_t}^{\text{base}}, \tilde{\mathbf{a}}_k). \quad (7)$$

Here $\mathbf{a}_{q_t}^{\text{base}}$ is the initial entity activation vector produced from the current query before stored experiences are used. The first term captures surface-level query similarity. The second term captures whether the current query starts near an evidence neighborhood that was useful before. LivingRAG retrieves the top- K experiences according to s_k^{ret} .

The retrieved activation maps are then fused into the initial entity activation vector:

$$\mathbf{a}_{q_t}^0 = \text{normalize} \left(\mathbf{a}_{q_t}^{\text{base}} + \beta \sum_{e_k \in \mathcal{N}_t^{\text{ret}}} w_k \tilde{\mathbf{a}}_k \right). \quad (8)$$

We first compute a softmax weight over the selected experiences:

$$\bar{w}_k = \frac{\exp(s_k^{\text{ret}}/\tau_s)}{\sum_{j \in \mathcal{N}_t^{\text{ret}}} \exp(s_j^{\text{ret}}/\tau_s)}. \quad (9)$$

We then scale this weight by the confidence of the stored experience:

$$w_k = \bar{w}_k \cdot \text{conf}_k. \quad (10)$$

The scaled weights are not renormalized. Thus, low-confidence experiences have less influence on the fused activation vector. The resulting vector $\mathbf{a}_{q_t}^0$ replaces the original initial activation vector of the base retriever. All later retrieval steps are

unchanged. When the experience store is empty, or when the current query has no seed entities for graph fusion, LivingRAG falls back to the base retriever.

For generation, LivingRAG may also select a reasoning scaffold. Scaffold selection can use a template signal because the selected scaffold is added only to the LLM prompt and does not inject historical entities into graph retrieval. For a query q , we define a masked template:

$$T(q) = \text{Mask}(q), \quad (11)$$

where entity mentions, quoted spans, and numeric expressions are replaced with placeholders. The scaffold score is:

$$s_k^{\text{scaf}} = (1 - \lambda)s_k^{\text{ret}} + \lambda \cos(\text{Emb}(T(q_t)), \text{Emb}(T(q_k))). \quad (12)$$

When stored experiences are available, LivingRAG selects the highest-scoring experience as the scaffold:

$$e_t^{\text{scaf}} = \arg \max_{e_k \in \mathcal{E}_t} s_k^{\text{scaf}}. \quad (13)$$

If the experience store is empty, or if no useful scaffold is selected, no scaffold is inserted. In the special case $\lambda = 0$, scaffold selection reduces to the highest-scoring experience from experience-augmented retrieval.

The LLM receives the retrieved passages and, when available, the selected scaffold:

$$\text{Context}_t = \{p_1, \dots, p_k\} \cup \{\text{sum}(e_t^{\text{scaf}})\}. \quad (14)$$

The scaffold shows how a related problem was solved. The model can still use the current passages to produce a different answer when the new query requires different evidence.

3.5 Experience Write-Back

After generation, LivingRAG checks whether the new candidate experience should be stored. LivingRAG separates write-back validation from answer generation. The system can return an answer even when the candidate is not stored. This separation prevents unsupported or redundant reasoning from entering the experience store.

The first criterion is factual grounding. LivingRAG extracts atomic claims from the available reasoning trace and the final answer. Each claim is checked against the retrieved passages with an NLI model:

$$g_i = \max_{p_j \in \mathcal{D}_t^{\text{used}}} P_{\text{NLI}}(\text{entail} \mid p_j, cl_i). \quad (15)$$

The grounding score is:

$$\text{GS}(C_t) = \frac{1}{M} \sum_{i=1}^M \mathbb{I}[g_i \geq \tau_{\text{nli}}]. \quad (16)$$

If $\text{GS}(C_t) < \tau_{\text{gs}}$, the candidate is not stored. The grounding score measures evidential support. It does not explicitly measure whether a candidate will be useful for future questions.

The second criterion is novelty. LivingRAG avoids storing candidates that are near duplicates of stored experiences. Novelty is measured before experience fusion, using the current query embedding and its base activation vector:

$$\eta_k = \alpha \cos(\text{Emb}(q_t), \mathbf{v}_k) + (1 - \alpha) \cos(\mathbf{a}_{q_t}^{\text{base}}, \tilde{\mathbf{a}}_k), \quad (17)$$

$$\text{NS}(C_t, \mathcal{E}_t) = 1 - \max_{e_k \in \mathcal{E}_t} \eta_k. \quad (18)$$

Using the base activation vector avoids circularity. A new candidate should not look redundant only because a previous stored experience influenced retrieval. If $\text{NS}(C_t, \mathcal{E}_t) < \tau_{\text{novel}}$, LivingRAG returns the answer but does not store the candidate.

The write-back rule is:

$$\text{Store}(C_t, \mathcal{E}_t) = 1 \quad \text{iff} \quad \text{GS}(C_t) \geq \tau_{\text{gs}} \quad \wedge \quad \text{NS}(C_t, \mathcal{E}_t) \geq \tau_{\text{novel}}. \quad (19)$$

The rule defines the storage condition. In implementation, LivingRAG checks novelty before grounding to avoid unnecessary NLI calls. Appendix D reports gate dynamics and examples of accepted and rejected candidates. For accepted candidates, the confidence score is set to the grounding score $\text{conf}_t = \text{GS}(C_t)$. The new verified experience is then appended to the store and can be used by later queries.

3.6 Complexity

LivingRAG preserves the static graph construction and graph retrieval backbone of LinearRAG. Therefore, the corpus-side complexity remains unchanged. For each query, the added retrieval cost comes from matching the query against stored experiences. If the experience store contains $|\mathcal{E}_t|$ experiences and the average number of nonzero entries in an activation map is $\bar{z}$, the added retrieval cost is $O(|\mathcal{E}_t|d + |\mathcal{E}_t|\bar{z})$, where d is the embedding dimension. Since only the top- K experiences are fused, and K is fixed, the experience-augmented retrieval overhead is linear in the store size and independent of the corpus graph size.

Table 2: Overall accuracy (%) on QA benchmarks. Cont.-Acc. denotes Contain-Match Accuracy. WixQA is reported only for LinearRAG and LivingRAG; unavailable entries are marked with –. The best result is marked in **bold**, and the second-best result is underlined.

Method	2Wiki		HotpotQA		MuSiQue		MuSiQue-full		WixQA
	Cont.-Acc.	LLM-Acc.	Cont.-Acc.	LLM-Acc.	Cont.-Acc.	LLM-Acc.	Cont.-Acc.	LLM-Acc.	LLM-Acc.
Vanilla RAG (Top-5) (Karpukhin et al., 2020)	58.80	59.40	65.20	77.10	34.00	43.70	27.35	41.33	–
LightRAG (Guo et al., 2024)	60.60	48.20	62.20	71.70	39.20	49.10	29.83	44.89	–
GFM-RAG (Luo et al., 2026b)	75.50	77.80	72.50	86.10	42.70	52.50	37.28	47.75	–
HippoRAG2 (Gutiérrez et al., 2025)	76.50	68.00	71.20	84.60	45.40	52.90	40.92	44.06	–
LinearRAG (Zhuang et al., 2025)	<u>79.60</u>	<u>84.40</u>	<u>72.90</u>	<u>86.80</u>	<u>48.50</u>	<u>60.70</u>	<u>41.00</u>	<u>52.71</u>	<u>66.00</u>
LivingRAG (Ours)	82.70	87.00	73.20	88.00	50.80	64.40	44.39	58.42	70.25

Write-back validation is also selective. Novelty scoring uses the same lightweight store scan. Grounding verification is executed only for candidates that pass the novelty gate. If the candidate contains M claims and the validator keeps K_g candidate evidence sentences per claim, the write-back overhead is:

$$T_{\text{write}}(q_t) = O(|\mathcal{E}_t|(d + \bar{z})) + \mathbb{K}_{\text{novel}} O(MK_g). \quad (20)$$

Thus, the NLI grounding cost is paid only for candidates that pass novelty filtering. The NLI cost is used for write-back validation. It does not change the base graph index and does not require additional LLM calls.

The additional storage is compact. Each stored experience contains a dense embedding, a sparse activation map, a short reasoning scaffold, and metadata. The experience-store overhead is $O(|\mathcal{E}_t|(d + \bar{z} + \bar{\ell}))$, where $\bar{\ell}$ is the average scaffold length. Appendix C provides the full time and space analysis.

When useful stored experiences are retrieved, LivingRAG can reduce generation cost in two ways. Activation maps can move retrieval toward relevant evidence earlier, and scaffolds can reduce the reasoning the LLM must reconstruct from scratch. We write the expected token cost as:

$$\mathbb{E}[\text{Cost}(t)] = (1 - f_{\text{exp}}(t))C_{\text{base}} + f_{\text{exp}}(t)C_{\text{exp}}, \quad (21)$$

where $f_{\text{exp}}(t)$ is the fraction of queries with useful support from stored experiences. Since C_{exp} depends on the quality of retrieved experiences, LivingRAG does not assume that cost always decreases. It predicts lower cost when later queries can reuse relevant stored experiences, which we test in the experiments.

4 Experiments

4.1 Overall Accuracy on QA Benchmarks

We first evaluate whether LivingRAG improves end-to-end QA accuracy over representative RAG and GraphRAG baselines. Later sections isolate experience reuse, token use, and online efficiency.

Setup. We evaluate all methods on four multi-hop QA streams: 2WikiMultiHopQA, HotpotQA, MuSiQue, and MuSiQue-full. MuSiQue-full uses the 2,417-question answerable stream described in Appendix E. We additionally report WixQA only for LinearRAG and LivingRAG, since its LLM-based entity extraction is not suitable for the other baselines. Appendix F describes these choices. We compare LivingRAG with Vanilla RAG (Karpukhin et al., 2020), LightRAG (Guo et al., 2024), GFM-RAG (Luo et al., 2026b), HippoRAG2 (Gutiérrez et al., 2025), and LinearRAG (Zhuang et al., 2025). All methods use Qwen3.6 Plus for answer generation. We report Contain-Match Accuracy and LLM-Evaluation Accuracy; WixQA uses only LLM-Evaluation Accuracy.

For LivingRAG, the experience store is initialized as empty at the beginning of each dataset and updated online during evaluation. Only earlier queries can contribute experience to later queries, following the online setting defined in Section 3.1.

Table 2 shows that LivingRAG achieves the best overall accuracy on the four multi-hop benchmarks and in the WixQA comparison against LinearRAG. Under our evaluation setting, the writable experience store does not weaken the base QA ability of the graph-RAG backbone. Since LivingRAG builds retrieval framework based on LinearRAG, later experiments use LinearRAG as the main baseline to better isolate the effect of experience reuse. Appendix K.3 reports a retrieval-level transfer test with HippoRAG2’s entity-passage PPR.

Table 3: Token consumption and cost of LinearRAG and LivingRAG. Prompt and completion tokens are per-query averages. Rel. Δ columns report $(\text{LivingRAG} - \text{LinearRAG}) / \text{LinearRAG}$. Negative values indicate reductions. Run cost is computed with Qwen3.6 Plus pricing: \$0.50 per million prompt tokens and \$3.00 per million completion tokens.

Dataset	# Queries	Prompt tokens			Completion tokens			Run cost (USD)		
		Linear	Living	Rel. Δ	Linear	Living	Rel. Δ	Linear	Living	Rel. Δ
2WikiMultiHopQA	1,000	6,038.2	6,241.6	+3.4%	1,067.6	906.4	-15.1%	\$6.22	\$5.84	-6.1%
HotpotQA	1,000	5,779.8	5,985.8	+3.6%	941.9	767.0	-18.6%	\$5.72	\$5.29	-7.4%
MuSiQue	1,000	5,888.4	6,092.4	+3.5%	2,098.8	1,641.7	-21.8%	\$9.24	\$7.97	-13.7%
MuSiQue-full	2,417	8,639.3	8,861.0	+2.6%	2,223.8	1,643.7	-26.1%	\$26.57	\$22.63	-14.8%
WixQA	400	6,633.0	7,412.8	+11.8%	1,653.8	1,378.1	-16.7%	\$3.31	\$3.14	-5.3%
Overall (weighted)	5,817	7,089.7	7,340.9	+3.5%	1,744.0	1,347.6	-22.7%	\$51.05	\$44.87	-12.1%

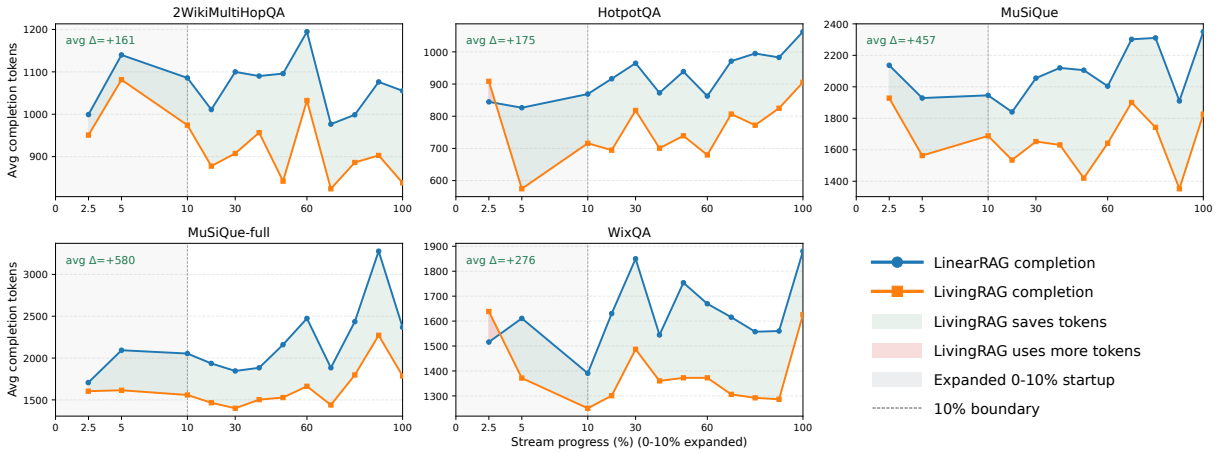


Figure 2: Completion-token trends over online QA streams. Each panel corresponds to one dataset. The blue and orange curves show the average completion tokens of LinearRAG and LivingRAG in each stream segment. Green shading between the two curves marks segments where LivingRAG uses fewer completion tokens, while red shading marks segments where it uses more. The first 10% of each stream is expanded to show the startup stage.

4.2 Online Token Efficiency

We next study whether online experience reuse reduces token use and estimated cost. We use the same datasets and online setting as in Section 4. Since LivingRAG uses LinearRAG as its retrieval backbone, we compare only these two systems. This comparison isolates the effect of the writable experience store.

Setup. We record prompt tokens, completion tokens, and total run cost for each run. Prompt tokens measure the input context. Completion tokens measure the generated output. Cost is estimated with the Qwen3.6 Plus prices used in Table 3.

We report prompt tokens, completion tokens, and total run cost. Prompt tokens correspond to the input context, while completion tokens correspond to generated output. We report them separately because they reflect different parts of inference and are priced differently by many LLM APIs. In our estimate, prompt tokens are charged at \$0.5 per

million tokens, and completion tokens are charged at \$3.0 per million tokens.

Table 3 shows a consistent trade-off. LivingRAG increases prompt tokens on all datasets because it may add compact reasoning scaffolds or experience-related context. The weighted prompt-token increase is 3.5%. However, LivingRAG reduces completion tokens on all five datasets, with a weighted reduction of 22.7%. The reduction is largest on MuSiQue and MuSiQue-full, where queries often require longer multi-hop reasoning.

The total cost reduction is smaller than the completion-token reduction because the added prompt tokens partly offset the savings. Even so, LivingRAG reduces total run cost on every dataset. Overall, the estimated cost decreases from \$51.05 to \$44.87, a 12.1% reduction. Section 4.3 further analyzes which reuse paths are used during inference.

Figure 2 shows the online trend. In the earliest segments, LivingRAG has few stored experiences

Table 4: Reuse opportunities and realized reuse in LivingRAG. “Opp.” denotes the template-reuse opportunity from Section 3.1. “Reuse” denotes realized scaffold reuse during inference. Graph transfer is measured from activation-map fusion. The opportunity and realized values use different measurements, so the table should be read as a trend comparison rather than a direct numerical comparison. All values are percentages.

Dataset	Graph signal	Template signal	
	Graph trans.	Opp.	Reuse
2Wiki	0.00	96.30	84.15
HotpotQA	0.00	21.82	9.04
MuSiQue	91.20	39.14	40.69
MuSiQue-full	82.42	54.30	50.44
WixQA	91.50	16.04	7.03

and may temporarily use more completion tokens. After experience accumulates, the area between the two curves is green in most segments, indicating that LivingRAG usually uses fewer completion tokens. The segment-level trend suggests that the savings are not caused by a single late-stage average and become more stable after the startup stage.

4.3 Realized Reuse Signals

We next examine whether LivingRAG uses the reuse signals that motivate our design. This analysis is not an accuracy evaluation. It is also not a direct conversion from reuse opportunity to realized reuse. The opportunity values in Section 3.1 describe whether a stream contains possible reuse signals. The realized values here are measured from LivingRAG inference traces.

We measure two realized signals. *Graph-neighborhood transfer* occurs when activation-map fusion introduces new above-threshold graph entities. *Template reuse* occurs when the selected scaffold has masked-template similarity of at least 0.35 with the current query.

Table 4 shows that LivingRAG uses different reuse paths on different datasets. On 2Wiki, template reuse is high. This matches the strong template signal observed in Section 3.1. Graph transfer is 0.00% under our strict definition. This does not mean that graph signals are absent. It only means that activation-map fusion does not introduce new above-threshold entities in this setting.

On MuSiQue, MuSiQue-full, and WixQA, graph transfer is much stronger. This shows that stored activation maps often change the graph retrieval state. Template reuse is also visible on MuSiQue and MuSiQue-full. These results support the two

Table 5: Ablation results. 2W, MQ-F, and Wix report LLM accuracy (%) on 2WikiMultiHopQA, MuSiQue-full, and WixQA. Red. is average completion-token reduction against LinearRAG.

Variant	2W	MQ-F	Wix	Red.
LivingRAG	87.00	58.42	70.25	23.6%
w/o Act. fusion	86.70	56.32	67.45	15.1%
w/o Scaffold	83.80	56.92	69.55	8.3%
w/o Quality Gate	84.60	55.52	68.25	18.0%

reuse paths in LivingRAG. Activation maps help retrieval reuse graph neighborhoods. Scaffolds help generation reuse reasoning patterns.

HotpotQA has low values under both strict trace metrics. This suggests that these two measurements may not capture all experience effects on this dataset. Overall, the table shows that the realized behavior of LivingRAG is consistent with the reuse signals that motivated the method. Appendix K further analyzes performance by realized reuse signal and robustness across query orders.

4.4 Ablation Study

We ablate three key components on 2WikiMultiHopQA, MuSiQue-full, and WixQA: activation-map fusion, reasoning scaffolds, and the write-back quality gate. Each run starts with an empty experience store and updates it online.

Table 5 shows that the two experience-use paths play different roles. Removing activation-map fusion hurts MuSiQue-full and WixQA more, where activation-map transfer is frequently realized during inference. Removing scaffolds hurts 2WikiMultiHopQA most and also reduces completion-token savings, suggesting that scaffolds help avoid repeated reasoning. Across the five streams, the quality gates store only 27.4% of candidate experiences (Appendix D). Removing the quality gate lowers accuracy on all datasets, confirming that writable experience must pass checks before reuse. Overall, the ablation supports the design of LivingRAG. Activation maps help retrieval, scaffolds help generation, and quality gates keep online experience reliable.

5 Conclusion

We presented LivingRAG, a Graph RAG framework with writable and reusable reasoning experience. LivingRAG stores verified experiences. It reuses them through activation-map fusion and reasoning scaffolds. Experiments on multi-hop QA

benchmarks show that LivingRAG improves accuracy and reduces completion tokens over strong RAG baselines. The results suggest that verified experience reuse is a simple way to augment Graph RAG.

6 Limitations

One limitation of LivingRAG is long-term experience growth. As shown in our complexity analysis, LivingRAG keeps the LinearRAG graph unchanged and adds experience-store costs that are linear in the number of accepted experiences. The runtime breakdown also shows that this overhead is acceptable in our experiments, and that post-answer validation does not need to increase user-visible latency. However, in much larger deployments, experiences may accumulate over a long period. Too many repeated or weakly useful experiences may increase matching cost and make reuse less stable. Future systems may need stronger store maintenance, such as pruning, compression, or indexed experience search.

LivingRAG also assumes that stored experiences remain valid for later queries. The current store does not update or delete accepted experiences. The assumption can fail in dynamic domains. For example, product policies, event information, or other time-sensitive facts may change after an experience is written. Although each experience stores a timestamp, the current system does not explicitly reduce the influence of old experience. The fixed-corpus benchmarks contain no chronological updates or fact-validity intervals, so they cannot directly evaluate expiration under changing knowledge. Future work should explore time-aware refresh and adaptive pruning, so that older experiences can be downweighted, updated, or removed when new evidence appears.

Our evaluation also treats controlled QA benchmarks as proxies for online query streams. MuSiQue is constructed by composing single-hop questions and may therefore contain denser reusable structure than naturally occurring traffic. WixQA is more deployment-oriented, but it is not timestamped and its questions share an enterprise knowledge base. In natural streams, reusable experiences may be more scattered, while topic drift and noise may reduce their value. Evaluation on naturally timestamped streams with topic drift and controlled noise remains future work.

Acknowledgments

This work was supported by the Zhejiang Provincial Natural Science Foundation of China (Youth Program) under Grant No. LQN26F020018 and by the “Pioneer” and “Leading Goose” R&D Program of Zhejiang under Grant No. 2024C01020.

References

- Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. [Self-RAG: Learning to retrieve, generate, and critique through self-reflection](#). In *The Twelfth International Conference on Learning Representations*.
- Yuzheng Cai, Zhenyue Guo, YiWen Pei, WanRui Bian, and Weiguo Zheng. 2025. [SimGRAG: Leveraging similar subgraphs for knowledge graphs driven retrieval-augmented generation](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 3139–3158, Vienna, Austria. Association for Computational Linguistics.
- Junnan Dong, Siyu An, Yifei Yu, Qian-Wen Zhang, Linhao Luo, Xiao Huang, Yunsheng Wu, Di Yin, and Xing Sun. 2025. [Youtu-graphrag: Vertically unified agents for graph retrieval-augmented complex reasoning](#). *Preprint*, arXiv:2508.19855.
- Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2024. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*.
- Jinyuan Fang, Zaiqiao Meng, and Craig MacDonald. 2025. [KiRAG: Knowledge-driven iterative retriever for enhancing retrieval-augmented generation](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18969–18985, Vienna, Austria. Association for Computational Linguistics.
- Guangze Gao, Zixuan Li, Chunfeng Yuan, Jiawei Li, Wu Jianzhuo, Yuehao Zhang, Xiaolong Jin, Bing Li, and Weiming Hu. 2025a. [D-RAG: Differentiable retrieval-augmented generation for knowledge graph question answering](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 35398–35417, Suzhou, China. Association for Computational Linguistics.
- Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. [Retrieval-augmented generation for large language models: A survey](#). *Preprint*, arXiv:2312.10997.
- Zengyi Gao, Yukun Cao, Hairu Wang, Ao Ke, Yuan Feng, S Kevin Zhou, and Xike Xie. 2025b. [FRAG: A flexible modular framework for retrieval-augmented](#)

- generation based on knowledge graphs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6178–6192, Vienna, Austria. Association for Computational Linguistics.
- Ziyi Guan, Jason Chun Lok Li, Zhijian Hou, Pingping Zhang, Donglai Xu, Yuzhi Zhao, Mengyang Wu, Jinpeng Chen, Thanh-Toan Nguyen, Pengfei Xian, Wenao Ma, Shengchao Qin, Graziano Chesi, and Ngai Wong. 2025. [KG-RAG: Enhancing GUI agent decision-making via knowledge graph-driven retrieval-augmented generation](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 5396–5405, Suzhou, China. Association for Computational Linguistics.
- Kai Guo, Harry Shomer, Shenglai Zeng, Haoyu Han, Yu Wang, and Jiliang Tang. 2025. [Empowering GraphRAG with knowledge filtering and integration](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 25439–25453, Suzhou, China. Association for Computational Linguistics.
- Zirui Guo, Lianghao Xia, Yanhua Yu, Tian Ao, and Chao Huang. 2024. [Lightrag: Simple and fast retrieval-augmented generation](#). *arXiv preprint arXiv:2410.05779*, 2(3).
- Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. 2025. [From RAG to memory: Non-parametric continual learning for large language models](#). In *Forty-second International Conference on Machine Learning*.
- Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. [Hipporag: Neurobiologically inspired long-term memory for large language models](#). In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Xiaoxin He, Yijun Tian, Yifei Sun, Nitesh V Chawla, Thomas Laurent, Yann LeCun, Xavier Bresson, and Bryan Hooi. 2024. [G-retriever: Retrieval-augmented generation for textual graph understanding and question answering](#). In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.
- Mossad Helali, Yutai Luo, Tae Jun Ham, Jim Plotts, Ashwin Chaugule, Jichuan Chang, Parthasarathy Ranganathan, and Essam Mansour. 2025. [Reliable and cost-effective exploratory data analysis via graph-guided RAG](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 16536–16553, Suzhou, China. Association for Computational Linguistics.
- Xinke Jiang, Ruizhe Zhang, Yongxin Xu, Rihong Qiu, Yue Fang, Zhiyuan Wang, Jinyi Tang, Hongxin Ding, Xu Chu, Junfeng Zhao, and Yasha Wang. 2025. [HyKGE: A hypothesis knowledge graph enhanced RAG framework for accurate and reliable medical LLMs responses](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11836–11856, Vienna, Austria. Association for Computational Linguistics.
- Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. [Active retrieval augmented generation](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992, Singapore. Association for Computational Linguistics.
- Imran Kabir, Md Alimoor Reza, and Syed Billah. 2025. [Logic-rag: Augmenting large multimodal models with visual-spatial knowledge for road scene understanding](#). In *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.
- Jiazheng Kang, Mingming Ji, Zhe Zhao, and Ting Bai. 2025. [Memory OS of AI agent](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 25961–25970, Suzhou, China. Association for Computational Linguistics.
- Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. [Dense passage retrieval for open-domain question answering](#). In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.
- Meng-Chieh Lee, Qi Zhu, Costas Mavromatis, Zhen Han, Soji Adeshina, Vassilis N. Ioannidis, Huzefa Rangwala, and Christos Faloutsos. 2025. [HyBGRAG: Hybrid retrieval-augmented generation on textual and relational knowledge bases](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 879–893, Vienna, Austria. Association for Computational Linguistics.
- 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. 2020. [Retrieval-augmented generation for knowledge-intensive nlp tasks](#). In *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS ’20*, Red Hook, NY, USA. Curran Associates Inc.
- Mufei Li, Siqi Miao, and Pan Li. 2025. [Simple is effective: The roles of graphs and large language models in knowledge-graph-based retrieval-augmented generation](#). In *ICLR 2025 Workshop on Foundation Models in the Wild*.
- Hao Liu, Zhengren Wang, Xi Chen, Zhiyu Li, Feiyu Xiong, Qinhan Yu, and Wentao Zhang. 2025. [HopRAG: Multi-hop reasoning for logic-aware retrieval-augmented generation](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 1897–1913, Vienna, Austria. Association for Computational Linguistics.

- Yayu Long, Kewei Chen, Long Jin, and Mingsheng Shang. 2025. [DRAE: Dynamic retrieval-augmented expert networks for lifelong learning and task adaptation in robotics](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23098–23141, Vienna, Austria. Association for Computational Linguistics.
- Haoran Luo, Haihong E, Guanting Chen, Yandan Zheng, Xiaobao Wu, Yikai Guo, Qika Lin, Yu Feng, Zemin Kuang, Meina Song, Yifan Zhu, and Anh Tuan Luu. 2026a. [HypergraphRAG: Retrieval-augmented generation via hypergraph-structured knowledge representation](#). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- LINHAO LUO, Yuan-Fang Li, Gholamreza Haffari, and Shirui Pan. 2024. [Reasoning on graphs: Faithful and interpretable large language model reasoning](#). In *The Twelfth International Conference on Learning Representations*.
- Lin hao Luo, Zicheng Zhao, Gholamreza Haffari, Dinh Phung, Chen Gong, and Shirui Pan. 2026b. [GFM-RAG: Graph foundation model for retrieval augmented generation](#). In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Shengjie Ma, Chengjin Xu, Xuhui Jiang, Muzhi Li, Huaren Qu, Cehao Yang, Jiaxin Mao, and Jian Guo. 2025. [Think-on-graph 2.0: Deep and faithful large language model reasoning with knowledge-guided retrieval augmented generation](#). In *The Thirteenth International Conference on Learning Representations*.
- Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegrefe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. [Self-refine: Iterative refinement with self-feedback](#). Preprint, arXiv:2303.17651.
- Costas Mavromatis, Soji Adeshina, Vassilis N. Ioannidis, Zhen Han, Qi Zhu, Ian Robinson, Bryan Thompson, Huzefa Rangwala, and George Karypis. 2025. [BYOKG-RAG: Multi-strategy graph retrieval for knowledge graph question answering](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 27881–27898, Suzhou, China. Association for Computational Linguistics.
- Hongjin Qian, Zheng Liu, Peitian Zhang, Kelong Mao, Defu Lian, Zhicheng Dou, and Tiejun Huang. 2025. [Memorag: Boosting long context processing with global memory-enhanced retrieval augmentation](#). In *Proceedings of the ACM on Web Conference 2025*, WWW '25, page 2366–2377, New York, NY, USA. Association for Computing Machinery.
- Rana Salama, Jason Cai, Michelle Yuan, Anna Currey, Monica Sunkara, Yi Zhang, and Yassine Benajiba. 2025. [MemInsight: Autonomous memory augmentation for LLM agents](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 33136–33152, Suzhou, China. Association for Computational Linguistics.
- Zhili Shen, Chenxin Diao, Pavlos Vougiouklis, Pascual Merita, Shriram Piramanayagam, Enting Chen, Damien Graux, Andre Melo, Ruofei Lai, Zeren Jiang, Zhongyang Li, Ye Qi, Yang Ren, Dandan Tu, and Jeff Z. Pan. 2025. [GeAR: Graph-enhanced agent for retrieval-augmented generation](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 12049–12072, Vienna, Austria. Association for Computational Linguistics.
- Haizhou Shi, Zihao Xu, Hengyi Wang, Weiyi Qin, Wenyuan Wang, Yibin Wang, Zifeng Wang, Sayna Ebrahimi, and Hao Wang. 2024. Continual learning of large language models: A comprehensive survey. *arXiv preprint arXiv:2404.16789*.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. [Reflexion: language agents with verbal reinforcement learning](#). In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Yiheng Shu, Saisri Padmaja Jonnalagedda, Xiang Gao, Bernal Jiménez Gutiérrez, Weijian Qi, Kamalika Das, Huan Sun, and Yu Su. 2026. [REMem: Reasoning with episodic memory in language agent](#). In *The Fourteenth International Conference on Learning Representations*.
- Xingyu Tan, Xiaoyang Wang, Qing Liu, Xiwei Xu, Xin Yuan, Liming Zhu, and Wenjie Zhang. 2025. [HydraRAG: Structured cross-source enhanced large language model reasoning](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 14431–14459, Suzhou, China. Association for Computational Linguistics.
- Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. [Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.
- Jingjin Wang and Jiawei Han. 2025. [PropRAG: Guiding retrieval with beam search over proposition paths](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 6212–6227, Suzhou, China. Association for Computational Linguistics.
- Nengbo Wang, Xiaotian Han, Jagdip Singh, Jing Ma, and Vipin Chaudhary. 2025a. [CausalRAG: Integrating causal graphs into retrieval-augmented generation](#). In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 22680–22693, Vienna, Austria. Association for Computational Linguistics.

- Zihan Wang, Zihan Liang, Zhou Shao, Yufei Ma, Huangyu Dai, Ben Chen, Lingtao Mao, Chenyi Lei, Yuqing Ding, and Han Li. 2025b. [InfoGain-RAG: Boosting retrieval-augmented generation through document information gain-based reranking and filtering](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7190–7204, Suzhou, China. Association for Computational Linguistics.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. [Chain of thought prompting elicits reasoning in large language models](#). In *Advances in Neural Information Processing Systems*.
- Zhishang Xiang, Chuanjie Wu, Qinggang Zhang, Shengyuan Chen, Zijin Hong, Xiao Huang, and Jinsong Su. 2026. [When to use graphs in RAG: A comprehensive analysis for graph retrieval-augmented generation](#). In *The Fourteenth International Conference on Learning Representations*.
- Yilong Xu, Jinhua Gao, Xiaoming Yu, Yuanhai Xue, Baolong Bi, Huawei Shen, and Xueqi Cheng. 2025. [Training a utility-based retriever through shared context attribution for retrieval-augmented language models](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 629–648, Suzhou, China. Association for Computational Linguistics.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.
- Junchi Yu, Yujie Liu, Jindong Gu, Philip Torr, and Dongzhan Zhou. 2026. [Can knowledge-graph-based retrieval augmented generation really retrieve what you need?](#) In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.
- Wenhao Yu, Hongming Zhang, Xiaoman Pan, Peixin Cao, Kaixin Ma, Jian Li, Hongwei Wang, and Dong Yu. 2024. [Chain-of-note: Enhancing robustness in retrieval-augmented language models](#). In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 14672–14685, Miami, Florida, USA. Association for Computational Linguistics.
- Haozhen Zhang, Tao Feng, and Jiaxuan You. 2025a. [Graph of records: Boosting retrieval augmented generation for long-context summarization with graphs](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23780–23799, Vienna, Austria. Association for Computational Linguistics.
- Hengrui Zhang, Pin-Siang Huang, Zhen Zhang, Peican Lin, Yao-Ching Yu, Bo Hu, and Yulu Du. 2025b. [PathwiseRAG: Multi-dimensional exploration and integration framework](#). In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 22904–22925, Suzhou, China. Association for Computational Linguistics.
- Zhuoping Zhou, Davoud Ataee Tarzanagh, Sima Didari, Wenjun Hu, Baruch Gutow, Oxana Verkholyak, Masoud Faraki, Heng Hao, Hankyu Moon, and Seungjai Min. 2026. [Query-aware flow diffusion for graph-based RAG with retrieval guarantees](#). In *The Fourteenth International Conference on Learning Representations*.
- Luyao Zhuang, Shengyuan Chen, Yilin Xiao, Huachi Zhou, Yujing Zhang, Hao Chen, Qinggang Zhang, and Xiao Huang. 2025. Linearrag: Linear graph retrieval augmented generation on large-scale corpora. *arXiv preprint arXiv:2510.10114*.

A Additional Related Work

Graph-based RAG. Graph-based RAG organizes external knowledge with explicit graph structures. GraphRAG builds an entity graph and community summaries to support global sense-making over a corpus (Edge et al., 2024). LightRAG combines graph indexing with vector retrieval to retrieve entities, relations, and chunks efficiently (Guo et al., 2024). HippoRAG and HippoRAG2 use Personalized PageRank over an automatically constructed graph to support associative and multi-hop retrieval (Gutiérrez et al., 2024; Gutiérrez et al., 2025). LinearRAG builds a lightweight passage-sentence-entity graph and performs sparse graph propagation for efficient retrieval (Zhuang et al., 2025). GFM-RAG trains a graph foundation model to capture query-knowledge relations over graph structure (Luo et al., 2026b). GraphFlow learns a retrieval policy over text-rich knowledge graphs (Yu et al., 2026). These methods show that graph structure is useful for complex retrieval. LivingRAG builds on this line of work. It adds verified experience write-back to reuse prior reasoning in later graph retrieval and generation.

Reasoning-Enhanced RAG. Another line of work improves RAG with stronger reasoning. Iterative retrieval methods decompose a query and retrieve evidence step by step (Trivedi et al., 2023; Jiang et al., 2023). Self-RAG uses self-reflection to decide when and how to retrieve (Asai et al., 2024). Chain-of-Note generates intermediate notes over retrieved evidence (Yu et al., 2024). LogicRAG decomposes complex questions into subqueries according to logical dependencies (Kabir et al., 2025). These methods mainly improve reasoning for the

current query. LivingRAG instead studies how useful reasoning from earlier queries can be stored and reused in later queries.

Memory-augmented LLMs and RAG.

Memory-based language agents store past observations, feedback, or plans to improve later decisions (Shinn et al., 2023; Madaan et al., 2023). Recent systems also study memory for retrieval and reasoning. REMem builds a hybrid episodic memory graph from interaction histories and uses an agentic retriever for episodic recollection and reasoning (Shu et al., 2026). MemoRAG builds a global memory for long-context processing and uses it to generate retrieval clues (Qian et al., 2025). LivingRAG is complementary to these methods. It targets online Graph RAG over QA streams. Its stored unit is a verified reasoning experience. Each experience contains a sparse activation map, a compact reasoning summary, and validation metadata. The activation map guides later graph retrieval. The summary provides a reasoning scaffold for generation. A candidate is written only when it passes grounding and novelty checks.

B Experimental Details

Entity extraction model. For experiments that use spaCy-based entity extraction, we use the English transformer pipeline `en_core_web_trf`.

NLI grounding model. For the grounding check in experience write-back, we use the cross-encoder/`nli-deberta-v3-large`. The model is used as the NLI cross-encoder to score entailment between retrieved evidence and extracted claims.

Parameter setting. For fair comparison, LinearRAG and LivingRAG use the same LinearRAG retrieval settings. LivingRAG keeps the original LinearRAG parameters unchanged and only adds the experience-store components described in Section 3.

C Detailed Complexity Analysis

This appendix summarizes the extra cost introduced by LivingRAG over LinearRAG. Let $N_t = |\mathcal{E}_t|$ be the number of stored experiences before query q_t , d the embedding dimension, b_t the number of nonzero entities in the current base activation vector, and $\bar{z}$ the average number of nonzero entries in a stored activation map. LivingRAG does

not rebuild or modify the LinearRAG corpus graph. It only changes the initial entity activation vector and then calls the same graph retrieval procedure. Thus, the corpus-side construction and retrieval complexity remains the LinearRAG complexity.

The main added online cost is the exact scan over the experience store. For each stored experience, LivingRAG computes one dense query-embedding similarity and one sparse activation-map similarity:

$$O(N_t(d + b_t + \bar{z})) \approx O(N_t(d + \bar{z})). \quad (22)$$

After matching, only the top- K experiences are fused into the seed activation vector. If each selected activation map is optionally truncated to its top H entities, the fusion cost is

$$O(K \min(\bar{z}, H)), \quad (23)$$

where K and H are fixed hyperparameters. Template-aware scaffold selection, when enabled, adds at most one additional store scan. Template embeddings may be computed lazily from stored scaffold or source question text and cached transiently, but they are not part of persistent experience storage.

Write-back validation is also store-side and selective. Novelty Score (NS) uses the same dual-channel scan, $O(N_t(d + \bar{z}))$. Grounding Score (GS) is computed only after NS passes. If c_t claims are extracted and the evidence pre-filter keeps K_g candidate evidence sentences per claim, the NLI stage costs

$$O(c_t K_g), \quad (24)$$

where K_g is fixed and the evidence comes only from the small set of retrieved passages. The combined per-query cost can therefore be written as

$$\begin{aligned} T_{\text{Living}}(q_t) = & T_{\text{Linear}}(q_t) \\ & + O(N_t(d + \bar{z})) \\ & + O(K \min(\bar{z}, H)) \\ & + \mathbb{1}_{\text{GS}} O(c_t K_g). \end{aligned} \quad (25)$$

The expression separates the unchanged corpus-side LinearRAG cost from the added experience-store cost.

Over a stream of Q queries, exact store scanning contributes $\sum_{t=1}^Q N_t$. The worst case is quadratic in Q if every query is stored, but LivingRAG does not store every candidate. Across our five streams, only 1806 of 5817 candidates pass NS and reach GS. The other 4011 candidates skip the NLI check.

Finally, 1592 candidates are written to the store, giving a final write-back ratio of 27.4%. Thus, store growth is tied to verified and novel experiences rather than to the raw number of queries.

The additional storage is similarly compact. Each stored experience contains one query embedding, a sparse activation map, a short scaffold, and metadata. Let $\bar{\ell}$ denote the average scaffold length. The experience-store overhead is

$$S_{\text{Living}}(t) = S_{\text{Linear}} + O(N_t(d + \bar{z} + \bar{\ell})). \quad (26)$$

Template embeddings, when used for scaffold selection, are transient cached values and are not counted in this persistent storage term. Crucially, LivingRAG stores sparse activation maps rather than dense graph-entity vectors, and stores compact scaffolds rather than full reasoning traces. The extra space is therefore linear in the number of accepted experiences and remains separate from the static corpus graph. Overall, LivingRAG preserves LinearRAG’s corpus-side scalability while adding a controlled experience-store overhead that is bounded by sparsification, top- K fusion, and selective write-back. Appendix G reports the measured runtime of these components. Appendix D explains why grounding checks are run only for a subset of candidates.

D Quality Gates for Experience Write-Back

Figure 3 and Table 6 analyze the two gates used before a candidate is written as a verified experience. For implementation efficiency, LivingRAG evaluates NS before GS. The change affects only the order of computation. A new experience is stored only when both gates pass. When NS fails, the system skips GS and avoids the NLI check.

The NS curve is high at the beginning because there are few stored experiences. As more experiences are written, the NS pass rate decreases. The write-back curve follows the same trend. Across the five streams, only 1806 of 5817 candidates pass NS. The number of GS checks is also 1806, which confirms that GS is skipped when NS fails. The final number of stored experiences is 1592. The count shows that LivingRAG does not store every query. It mainly stores candidates that remain novel relative to earlier experiences. GS is then applied to the smaller candidate set. In total, 4011 candidates fail NS and skip GS. Among the 1806 candidates checked by GS, 1592 pass and 214 are

rejected. The GS pass rate among checked candidates is 88.15%. The results show that NS controls store growth. GS removes ungrounded candidates before write-back.

WixQA is a special case in Table 6. Its GS pass rate is 100% among the candidates that reach the GS check. The value does not mean that all GS scores are saturated. The average GS on the checked WixQA candidates is 0.886, and the lowest score is 0.653. Thus, the result means that all checked candidates are above the grounding threshold of 0.6. The likely reason is that the WixQA corpus consists of official help documents. Many answers contain procedural information that is directly supported by the retrieved help documents. For this dataset, we therefore use GS as a light grounding constraint rather than as the main selective gate. In parameter tuning, this corresponds to keeping the grounding threshold permissive and relying on the novelty gate and WixQA-specific retrieval settings to control write-back. The effective selectivity is shifted toward NS, which filters repeated support patterns before GS is run. The setting prevents WixQA from being dominated by grounding rejection, while still requiring each stored experience to have explicit passage support.

Table 7 gives concrete examples of these decisions. The two stored cases show when a candidate is written to the store. It must be new enough and supported by retrieved passages. The WixQA grounding case also explains why this dataset can have high GS pass rates. Its answer is stated directly in the help articles. The NS rejection case shows the opposite behavior. A later domain-refund query retrieves an almost identical stored experience. NS assigns a very low score and prevents another copy from being written. Finally, the GS rejection case shows why grounding is still needed after NS. The model predicts the correct answer, but the retrieved passages do not contain the key entities needed to support it. LivingRAG therefore returns the answer but refuses to store the candidate. The grounding gate prevents unsupported reasoning from becoming reusable experience. The selectivity also explains the validation term in the complexity analysis in Appendix C.

E Dataset Details

We evaluate LivingRAG on five online QA streams. Three of them are standard multi-hop QA benchmarks: 2WikiMultiHopQA, HotpotQA,

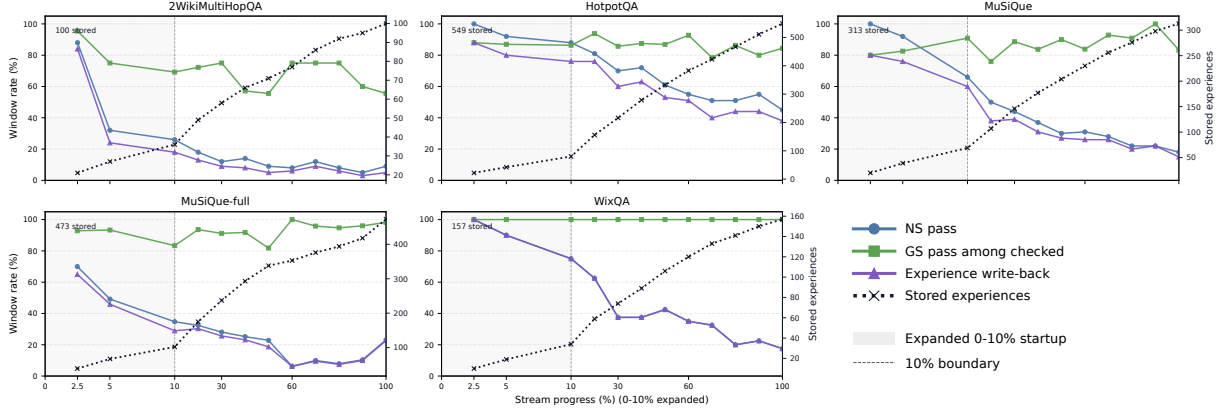


Figure 3: Dynamics of the NS and GS gates during online inference. NS pass and experience write-back are computed over each window. GS pass is computed only over candidates that reach the GS check. The black curve shows the cumulative number of stored experiences. The first 10% of each stream is expanded to show the startup stage.

Table 6: Effect of the NS and GS gates on experience write-back. All runs use $\tau_{\text{novel}} = 0.7$ and $\tau_{\text{gs}} = 0.6$. NS pass is measured over all questions. GS pass is measured only on candidates that reached the GS check. GS reject counts the checked candidates that failed the grounding gate.

Dataset	Questions	NS pass (%)	GS checked	GS pass (%)	GS reject	Stored experiences
2Wiki	1000	13.80	138	72.46	38	100
HotpotQA	1000	63.30	633	86.73	84	549
MuSiQue	1000	36.30	363	86.23	50	313
MuSiQue-full	2417	21.31	515	91.84	42	473
WixQA	400	39.25	157	100.00	0	157
All	5817	31.05	1806	88.15	214	1592

and MuSiQue. They mainly test whether a system can retrieve and combine evidence across multiple facts. Most answers in these datasets are short entities, dates, locations, or yes/no labels.

We further include MuSiQue-full and WixQA. These two datasets test different aspects of online experience reuse. MuSiQue-full provides a larger multi-hop stream with denser reuse structure. WixQA provides a customer-support stream over a shared enterprise knowledge base. Together, they make the evaluation less dependent on a single dataset style.

MuSiQue-full. MuSiQue is built by composing single-hop questions into multi-hop questions. It provides decomposition annotations that expose which single-hop questions are reused across different multi-hop questions. This makes it useful for studying whether an experience-reusing RAG system can reuse prior reasoning steps.

Prior RAG evaluations often use a 1,000-question MuSiQue subset. We additionally evaluate on a larger answerable split, which we call MuSiQue-full. It contains 2,417 answerable ques-

tions. This name is used only to distinguish it from the smaller subset in our experiments. It does not refer to the official unanswerable MuSiQue-Full setting.

We define an experience cluster using the official MuSiQue decomposition. Two questions are linked if they share at least one decomposed single-hop question. We then compute connected components over this question graph. A component with at least two questions is treated as an experience cluster. This definition is based on the dataset structure. It does not use text similarity, embedding similarity, or model predictions.

Table 8 shows that MuSiQue-full contains a stronger reuse structure than the smaller MuSiQue subset. It has more clusters, higher cluster coverage, and larger average cluster size. This makes it a useful stress test for online experience reuse. A system that stores verified experiences should benefit more when later questions reuse earlier reasoning components.

WixQA. WixQA is different from the Wikipedia-style benchmarks. It is built from product-support

Table 7: Representative NS and GS case studies. The cases are selected from the final LivingRAG runs without re-running retrieval or validation.

Case	Dataset	Idx.	Query	Answer	NS	GS	Decision	Interpretation
Stored multi-hop candidate	HotpotQA	33	What subsidiary of the largest airline of the Republic of China (Taiwan) has a main hub at Taichung Airport?	Mandarin Airlines	0.793	1.000	Stored	Both gates pass. The retrieved airline passage directly supports the answer, so the candidate becomes a verified experience.
Direct WixQA grounding	WixQA	57	I would like to know how many sites I can have.	There is no limit to the number of sites you can create under a single Wix account; you can have as many as you want.	0.829	1.000	Stored	The answer contains directly supported procedural information from the retrieved help articles. This illustrates why WixQA GS scores are often high under the permissive grounding threshold.
Novelty rejection	WixQA	391	can I get a refund on the domain	No. Wix domain purchases are non-refundable and do not include a 14-day money-back guarantee or trial period. If you no longer...	0.213	–	Rejected by NS	The query repeats a stored domain-refund experience. NS assigns a low score, skips GS, and avoids storing another copy of the same support pattern.
Grounding rejection	HotpotQA	858	Spawn, a fictional character who Mustafa Shakir has expressed interest in playing, was created by?	Todd McFarlane	0.701	0.368	Rejected by GS	NS passes, but the retrieved passages do not mention the key entities. The answer matches the gold label, yet GS blocks write-back because the candidate is not grounded in retrieved evidence.

Table 8: Reuse structure in the MuSiQue streams. Experience clusters are connected components of questions that share at least one decomposed single-hop question.

Dataset	#Q	#Chunks	#Clusters	Coverage	Avg. Size	Max Size
MuSiQue	1,000	1,354	178	79.2	4.4	21
MuSiQue-full	2,417	2,379	300	89.9	7.2	58

questions over the Wix Help Center. The questions are about product features, account settings, payments, domains, site editing, email, bookings, and related user tasks. The answers are usually support-style solutions. They often describe what the user should do, which product should be used, or whether a feature is supported.

This setting is important for LivingRAG. All questions are grounded in the same enterprise knowledge base. Many user issues refer to related products, settings, and help articles. This creates repeated evidence neighborhoods and recurring support patterns. At the same time, the language is less formulaic than standard multi-hop QA. This makes WixQA a useful test for experience reuse in a more deployment-like setting.

We use 400 WixQA questions in the main evaluation. The sample is balanced between ExpertWritten and Simulated questions. We exclude the Synthetic subset because it is generated and more tightly tied to individual articles. The retrieval corpus is the Wix Help Center corpus. Following the LinearRAG setting, articles are segmented into chunks of roughly 600 tokens.

F WixQA-specific Processing and Evaluation

WixQA requires several dataset-specific choices. These choices are not changes to the LivingRAG framework. They are needed because WixQA differs from the other datasets in entity type, passage length distribution, and answer format. In the experiments that compare LinearRAG and LivingRAG on WixQA, both systems use the same WixQA-specific processing.

F.1 LLM-based Entity Extraction

The graph retriever relies on query entities to initialize graph activation. For Wikipedia-style QA, a general-purpose NER model is usually sufficient. Most questions contain people, locations, organizations, works, dates, or named events. These entities are also the main anchors of the reasoning chain. For example, in 2WikiMultiHopQA and HotpotQA, spaCy often extracts useful entities such as person names, film titles, places, and organizations.

WixQA has a different language style. Its important query cues are often product names, UI labels, plan names, feature names, settings, and user actions. These cues are not ordinary named entities.

They are domain terms from the Wix Help Center. If they are missed, graph retrieval starts from weak or empty query seeds.

Table 9 compares representative cases. For standard multi-hop QA, spaCy usually extracts useful named entities. For WixQA, spaCy often returns no entity or only a broad brand-level entity. LLM NER instead extracts product-domain terms that are more useful for support retrieval.

These examples show that the issue is not only low recall. It is also a mismatch between general NER labels and product-support terminology. For example, the last WixQA question cannot be answered by recognizing only the word *Wix*. The retrieval system must distinguish whether the user should use *Wix Stores* or *Wix Bookings*. This distinction is central to the answer.

We therefore use LLM-based entity extraction for WixQA. The extractor is implemented with Qwen3.5-9B. This is a relatively small LLM, but we find it sufficient for this task in practice. The task only requires extracting short product-domain seed terms, not solving the question. Thus, a smaller LLM can provide useful domain entities without changing the retrieval framework. For fairness, LinearRAG and LivingRAG use the same passage-level and query-level extractor in all WixQA runs.

The goal is not to add reasoning to the retriever. The goal is to produce better product-domain seed terms. The downstream retrieval pipeline remains unchanged. Extracted query terms are mapped to graph entities and then used by the same graph propagation and passage ranking steps.

Figure 4 shows the passage-level prompt used during indexing. Figure 5 shows the question-level prompt used for query-side entity extraction.

Table 10 shows that the examples in Table 9 are not isolated cases. On 2WikiMultiHopQA, spaCy almost always extracts at least one query entity. On WixQA, it extracts no entity for 52.5% of the questions. This weakens graph retrieval because many queries have no useful starting nodes. LLM NER removes this failure case.

LLM NER is not noise-free. Some extracted terms can be mapped to imperfect graph entities. We therefore only keep the top mapped candidates for each query term and remove high-degree hub entities. This keeps query activation focused while still giving the retriever product-specific seeds.

We also apply the LLM extractor to WixQA passages during indexing. This keeps the entity space

consistent between queries and passages. The resulting graph remains sparse enough for propagation. Almost every chunk has extracted entities, and the average number of entities per entity-bearing sentence is below two.

F.2 Passage Length Normalization

WixQA also differs in passage length distribution. The Wikipedia-style datasets are mostly segmented into near-fixed long chunks. In contrast, WixQA keeps the natural granularity of help-center articles. Some chunks are short feature notes or known-issue pages. Others are long tutorials with detailed steps.

This creates a different retrieval problem. Long chunks naturally contain more words and more entity mentions. They can therefore receive higher retrieval scores because they cover more topics. Short chunks may be more precise, but they provide fewer matching signals. Without correction, a long generic article may dominate a short article that directly answers the user issue.

Table 12 shows this difference. In WixQA, 44.4% of chunks contain fewer than 1,000 characters, and 59.6% contain fewer than 2,000 characters. The corresponding ratios are almost zero in the other datasets. This means that WixQA retrieval must handle a mix of very short and long passages.

We apply a weak BM25-style length normalization to WixQA passage scores:

$$\text{score}'(p) = \frac{\text{score}(p)}{(1 - b) + b \cdot |p|/\overline{|p|}}. \quad (27)$$

Here $|p|$ is the passage length, and $\overline{|p|}$ is the average passage length in the corpus. We set $b = 0.1$. This value is intentionally small. It mildly corrects length bias, but it does not strongly favor short passages. This is important because some long Wix articles contain necessary step-by-step evidence.

F.3 Evaluation Metric for WixQA

We do not use contain-match accuracy as the main metric for WixQA. Contain-match accuracy is suitable for short-answer QA. In 2WikiMultiHopQA, HotpotQA, and MuSiQue, the gold answer is usually a short entity, date, place, or yes/no label. A correct generated answer often contains the exact gold answer span.

WixQA is different. Its gold answers are support-style solutions. They often include conditions,

Table 9: Examples of query-side entity extraction. spaCy works well for standard multi-hop QA because the key retrieval cues are ordinary named entities. On WixQA, the key cues are product-domain terms, so spaCy often fails or extracts only broad entities.

Dataset	Question	spaCy NER	LLM NER or useful seeds
2Wiki	When did Lothair II’s mother die?	<i>Lothair II</i>	Not needed
2Wiki	Which film was released first, Aas Ka Panchhi or Phoolwari?	<i>Aas Ka Panchhi; Phoolwari</i>	Not needed
HotpotQA	In which county is the town in which Raymond Robertsen was born?	<i>Raymond Robertsen</i>	Not needed
MuSiQue	When was the person who Messi’s goals in Copa del Rey compared to get signed by Barcelona?	<i>Messi; Copa del Rey; Barcelona</i>	Not needed
WixQA	I want to know how much it would cost to upgrade my email plan.	[]	<i>Email Marketing Plan; upgrade Email Marketing Plan; Email Marketing Plan pricing</i>
WixQA	How do I sync the hotel app with my calendar link to allow visitors to book available dates on my site?	[]	<i>Wix Bookings; iCal URL; syncing calendars; booking available dates</i>
WixQA	I have not received my payment in my bank account yet.	[]	<i>Wix Payments; payout delay; failed payout; Payouts on Hold</i>
WixQA	I want to know if the Wix store function work for selling services instead of just physical goods.	<i>Wix</i>	<i>Wix Stores; selling services; service bookings; Wix Bookings</i>

Table 10: Query-side entity extraction statistics. LLM NER addresses the low-recall problem of standard NER on WixQA while keeping the number of query seeds small.

Method	Query Set	0 Ent.	Avg.	Med.	Range
spaCy NER	2Wiki, first 400	1.2	1.68	2	0–4
spaCy NER	WixQA, all 400	52.5	0.61	0	0–4
LLM NER + mapping	WixQA, all 400	0.0	3.77	4	1–5

Table 11: Index-side entity extraction statistics for WixQA with LLM NER.

Statistic	Value
Chunks	7,132
Chunks with no entity	1
Avg. entities per chunk	11.23
Median entities per chunk	10
Sentences with entities	68,798
Avg. entities per entity-bearing sentence	1.79
Median entities per entity-bearing sentence	1

product-specific guidance, operation steps, caveats, and official wording. A generated answer may solve the user’s problem without copying the full gold answer. Thus, requiring the full gold answer to appear as a substring is too strict.

Table 13 gives representative examples. The standard multi-hop QA answers are short spans. In contrast, WixQA answers describe how to solve a user-support problem. The answer may be 60, 120, or more than 300 words.

Table 12: Passage length distribution by character count. WixQA contains many short chunks, while the Wikipedia-style datasets are close to fixed-length chunks.

Dataset	#Chk.	Median	< 1000	< 2000
WixQA	7,132	1,321	44.4	59.6
2Wiki	658	4,449	0.0	0.0
HotpotQA	1,311	4,651	0.0	0.0
MuSiQue	1,354	4,701	0.0	0.0

Table 14 shows the scale of this mismatch. The median answer length in WixQA is 59 words. For the other datasets, the median answer length is only 2 words. For WixQA, 85.0% of answers are longer than 30 words. For the other datasets, this ratio is zero.

The following example illustrates why containment is misleading. A generated answer can be useful and correct even if it does not contain the full official support answer.

Passage-level LLM NER prompt

```
System:
You are an entity extraction assistant for Wix product documentation. Extract ALL of the following from the given text:

1. Named product / feature / UI names (e.g. 'Wix Events', 'Event Details Page', 'Dashboard', 'DNS records', 'CMS collection')
2. The main topic or action concept of this article as 1-3 short phrases that would match a user's question about this
   content (e.g. 'Wix Payments account verification', 'adding funds to cover refund', 'publishing pages to live site', '
   staff working hours', 'CMS collection item order')

Return ONLY a JSON array combining all extracted terms. No explanation, no markdown fences.

Example -- article about verifying a Wix Payments account:
["Wix Payments", "Wix Payments account verification", "account verification status", "Accept Payments", "Adyen", "Stripe"]

User:
{passage}

Response:
```

Figure 4: Passage-level prompt used for WixQA LLM-based entity extraction during indexing. The extractor identifies Wix product-documentation entities and article-level action-topic terms from each passage.

Why contain-match fails

Question. How can I add discounts to my service prices when customers pay for a plan?

Gold answer type. A long official support answer. It explains how to create a Pricing Plans coupon. It also covers discount type, applicable plans, billing-cycle behavior, validity dates, usage limits, and final coupon creation.

Possible correct model answer. Go to Pricing Plans and create a coupon. Choose a fixed or percentage discount. Select the plan it applies to. Then configure billing behavior, validity dates, and usage limits.

Why contain-match fails. The model answer gives the correct operation. However, it will not contain the full official answer as a substring.

This is why contain-match accuracy severely underestimates answer quality on WixQA. In our 400-question WixQA evaluation, LinearRAG has 0 contain-match hits. LivingRAG has only 1 contain-match hit. However, LLM-based evaluation gives 66.00% accuracy for LinearRAG and 70.25% for LivingRAG.

We use LLM accuracy for all datasets, but the evaluation prompt differs by dataset type. For standard multi-hop QA, the evaluator checks whether the generated answer contains the key gold information and does not contradict it. For WixQA, the evaluator checks whether the answer correctly resolves the support issue. This requires a task-oriented prompt. The answer must recommend the right Wix product or feature. It must describe the core operation correctly. It must not claim that a supported feature is unsupported, or the reverse. It does not need to reproduce the full official wording.

This difference is important in product-support QA. For example, if a user asks whether Wix Stores

can be used to sell services, the key issue is not whether the answer mentions the words *Wix Stores*. The key issue is whether it recommends the right product. An answer that tells the user to use Wix Stores for service booking would be misleading. A correct answer should identify *Wix Bookings* as the suitable product for booking and paying for services.

G Runtime Breakdown

We further measure the runtime of the main online components. We use the same concurrent setting as the main experiments. The system answers up to 16 questions in parallel. We exclude one-time indexing, model loading, and evaluation-only LLM calls. The reported values are per-question averages. They are computed by summing the measured time of each component and dividing it by the number of questions. Because the system runs concurrently, these values are module-level averages. They should not be read as serialized wall-clock latency. The corresponding asymptotic cost is analyzed in Appendix C. Gate selectivity is analyzed in Appendix D.

The implementation keeps the LinearRAG retrieval path unchanged. For each query, entity matching and graph retrieval are run on CPU in the same way as LinearRAG. LLM calls are issued concurrently. For LivingRAG, answer generation and experience write-back are separated. After an answer is generated, the system forms a candidate experience. The candidate is then checked by the novelty and grounding gates described in Section 3.5. Grounding uses batched NLI cross-

Question-level LLM NER prompt

System:
You are a Wix documentation entity extractor. Extract 3-5 key terms from the user's question that best identify the relevant help article.

CRITICAL: Your terms must match the vocabulary used in Wix help article entity labels -- phrase them as they appear in article titles or article-level tags, NOT as the user phrased them.

Priority order:

1. Exact canonical Wix product / feature name -- use the official Wix name, not user paraphrase ('Google Workspace' not 'Gmail business'; 'Wix Bookings' not 'booking system'; 'Email Marketing Plan' not 'email plan')
2. A specific action-topic phrase describing the article's main subject
Good: 'resetting Workspace user password', 'failed payout', 'blocking off time in Booking Calendar', 'Publishing Issues', 'Premium Vouchers page', 'upgrade Email Marketing Plan'
Bad: 'password reset', 'payment', 'block time', 'page issue', 'voucher', 'upgrade'
3. Precise technical terms only if clearly stated (e.g. 'TXT record', 'iCal URL', 'DNS settings')

Rules:

- Match article vocabulary: 'Google Workspace' -> 'Google Workspace'; 'Gmail' in an admin/business context -> 'Google Workspace'
- Use full noun phrases, not single generic nouns: 'Email Marketing Plan upgrade' > 'email'; 'payout delay' > 'payment'
- Do NOT include ultra-generic terms: 'site', 'page', 'account', 'email', 'help', 'settings', 'Payments' (alone), 'Gmail' (alone for business email questions)
- Do NOT invent irrelevant terms not directly related to the user's problem
- Return ONLY a JSON array of strings, max 5 items. No explanation, no markdown fences.

Examples:

Q: 'I forgot the password for my second Gmail business account and I have an admin account.'
Output: ["Google Workspace", "resetting Workspace user password", "Google Workspace admin console"]

Q: 'I have not received my payment in my bank account yet.'
Output: ["Wix Payments", "payout delay", "failed payout", "Payouts On Hold"]

User:
{question}

Response:

Figure 5: Question-level prompt used for WixQA LLM-based entity extraction. The extractor maps user questions to canonical Wix product names and action-topic phrases for query-side graph activation.

Prompt for standard multi-hop QA

System:
You are an expert evaluator.

User:
Please evaluate if the generated answer is correct by comparing it with the gold answer.
Generated answer: {pred}
Gold answer: {gold}

The generated answer should be considered correct if it:

1. Contains the key information from the gold answer
2. Is factually accurate and consistent with the gold answer
3. Does not contain any contradicting information

Respond with ONLY 'correct' or 'incorrect'.

Response:

Figure 6: LLM-based accuracy evaluation prompt used for standard multi-hop QA datasets.

encoder calls. Accepted candidates are written to the store in order. The asynchronous design keeps online answering separate from experience-store maintenance.

Table 15 shows that retrieval is much cheaper

than LLM inference. Across the measured datasets, retrieval takes about 1.7% to 5.1% of the LLM time. The retrieval time of LivingRAG is close to that of LinearRAG. The result supports the complexity analysis. The added experience-augmented retrieval step is not the main runtime bottleneck.

Write-back validation is more expensive than retrieval. The cost comes from local NLI validation and experience write-back. All NLI checks are run on a single NVIDIA GeForce RTX 3060 GPU. The RTX 3060 is a modest GPU for cross-encoder validation. More importantly, write-back validation is a post-answer maintenance step. The answer has already been generated before this step starts. In a serving system, the answer can be returned before this experience update finishes. Thus, the validation column describes the cost of maintaining a reliable experience store. It does not need to increase user-visible answer latency.

H Prompt Token Trends

Figure 8 complements the completion token analysis in Figure 2. It shows the input cost of Linear-

Table 13: Gold answer format differs sharply between standard multi-hop QA and WixQA. Standard QA usually expects a short span. WixQA expects a support solution.

Dataset	Question	Gold answer	Words
2Wiki	When did Lothair II’s mother die?	20 March 851	3
2Wiki	Which film was released first, Aas Ka Panchhi or Phoolwari?	Phoolwari	1
HotpotQA	In which county is the town in which Raymond Robertsen was born?	Finnmark county	2
MuSiQue	When was the person who Messi’s goals in Copa del Rey compared to get signed by Barcelona?	June 1982	2
WixQA	Can I start accepting payments on my site while my Wix Payments account is still under verification?	You can start accepting payments almost immediately, but Wix must verify your identity before the account is fully activated.	27
WixQA	I want to know if the Wix store function work for selling services instead of just physical goods.	The answer distinguishes Wix Stores from Wix Bookings . For selling services, it recommends Wix Bookings because it supports service booking and online payment.	66
WixQA	I am inquiring about purchasing the yearly premium plan with a free domain for 1 year. The voucher does not show up at checkout. Do I need to purchase the plan first?	The answer explains that the voucher is not shown at checkout. It becomes available after purchasing a yearly Premium plan and can be claimed from the Premium Vouchers page.	120
WixQA	How can I add discounts to my service prices when customers pay for a plan?	The answer explains how to create a Pricing Plans coupon, including discount type, applicable plans, billing-cycle behavior, dates, usage limits, and coupon creation.	328

Table 14: Answer length statistics. WixQA answers are long support solutions rather than short answer spans.

Dataset	Median words	≤ 5 words	> 30 words
2WikiMultiHopQA	2	95.5	0.0
HotpotQA	2	96.1	0.0
MuSiQue	2	91.2	0.0
WixQA	59	0.2	85.0

RAG and LivingRAG over the online stream. At the very beginning, LivingRAG has not yet stored useful experiences. It also has no reasoning scaffold to add to the prompt. Thus, its prompt is the same as the LinearRAG prompt for the earliest queries. After experiences are written, LivingRAG can select a reasoning scaffold and place it in the input context. The selected scaffold makes the LivingRAG prompt longer than the LinearRAG prompt in later segments. The trend is therefore consistent with Table 3. LivingRAG pays a small input token increase, but this increase is much smaller than the reduction in completion tokens. Appendix I further shows that scaffold use increases after the startup stage.

I Experience Use During Online Inference

Figure 9 and Table 16 describe how LivingRAG uses experiences after they are written. The figure shows that the use of stored experiences rises quickly after the startup stage. Activation-map retrieval and reasoning scaffolds are used for most later queries in all datasets. At the same time, experience write-back is more selective. The pattern matches the design of LivingRAG. The system can reuse earlier experiences often, but it only writes a new experience when the candidate is grounded and sufficiently novel. The table further shows that reuse is spread across many stored experiences. For activation-map retrieval, the coverage is above

Prompt for WixQA

System:
You are an expert evaluator for Wix customer support answers. Your task is to judge whether a generated answer correctly resolves the customer's issue, compared to an official gold answer extracted from a Wix help article.

User:
Evaluate whether the generated answer is correct for this Wix customer support question.

Generated answer:
{pred}

Gold answer (from official Wix help article):
{gold}

Mark as CORRECT only if ALL of the following hold:

1. The same Wix product / feature is recommended (e.g. "Wix Bookings" vs "Wix Stores" is a meaningful difference).
2. The core action described would guide the customer to actually solve their problem.
3. It does not claim something is impossible when the gold shows it is possible (or vice versa).
4. It does not contain factually incorrect steps that would mislead the customer.

Mark as INCORRECT if ANY of the following hold:

- It recommends the wrong Wix product or feature.
- It describes the wrong operation (e.g. "delete a collection" vs "delete items from a collection").
- It contradicts the gold on a key fact (e.g. a feature exists vs does not exist).
- It provides outdated or clearly wrong information.

Leniency: the generated answer does NOT need to reproduce every step from the gold. Partial coverage of the correct solution is acceptable as long as the main direction is right and no incorrect information is given.

Respond with ONLY 'correct' or 'incorrect'.

Figure 7: LLM-based accuracy evaluation prompt used for WixQA. The prompt evaluates whether a generated support answer resolves the user issue without recommending the wrong Wix product, operation, or unsupported behavior.

Table 15: LLM, retrieval, and validation times are module-level averages. Indexing, model loading, and evaluation-only LLM calls are excluded. The Retrieval / LLM column reports the ratio between retrieval time and LLM time. Validation refers to the post-answer novelty and grounding checks used for experience write-back.

Dataset	Method	LLM s/q	Retrieval s/q	Retrieval / LLM	Validation s/q
2Wiki	LinearRAG	30.07	0.50	1.7%	N/A
2Wiki	LivingRAG	24.04	0.54	2.2%	2.13
HotpotQA	LinearRAG	15.42	0.72	4.7%	N/A
HotpotQA	LivingRAG	14.45	0.74	5.1%	2.35
MuSiQue	LinearRAG	37.23	1.49	4.0%	N/A
MuSiQue	LivingRAG	33.88	1.50	4.4%	6.64

79% on every dataset. For reasoning-scaffold reuse, the coverage is also broad, ranging from 58.11% on HotpotQA to 85.00% on 2Wiki. These results indicate that the stored experiences are not only accumulated. They are repeatedly selected by the retrieval and generation paths during later inference. Appendix J gives concrete examples of these reuse patterns.

J Direct Case Study: LinearRAG vs. LivingRAG

We next give aligned case studies from the final runs. Each case compares the same stream index under LinearRAG and LivingRAG. The first two cases show how activation-map fusion changes retrieval toward the answer-bearing evidence. The

last two cases show how a selected scaffold gives the LLM a reusable reasoning form and sharply shortens generation.

E1: Experience-augmented retrieval recovers a missing Cabot chain

Dataset / index: MuSiQue-full / 2231

Question: Whose navigator father explored the east coast of the region where Ignacio Esparza was later born?

LinearRAG output: Francisco de Orellana. **Result:** incorrect. **Completion tokens:** 8,739.

LivingRAG output: Sebastian Cabot. **Result:** correct. **Completion tokens:** 1,264.

Experience retrieved by LivingRAG: Source question: Who is the child of the Italian navigator who sailed for England and explored the eastern coast of the continent Francisco Bojado's birthplace is located? Stored answer: Sebastian Cabot. Activation signal: 20 experience entities are added, and 16 new entities cross the retrieval threshold. Important activated entities include Sebastian Cabot, John Cabot, Giovanni Caboto, Vinland, and continental North America.

Retrieved articles and answer evidence: Ignacio Esparza: Esparza was born in Guadalajara, Jalisco, Mexico. John Cabot: John Cabot was a Venetian navigator and explorer whose 1497 voyage under Henry

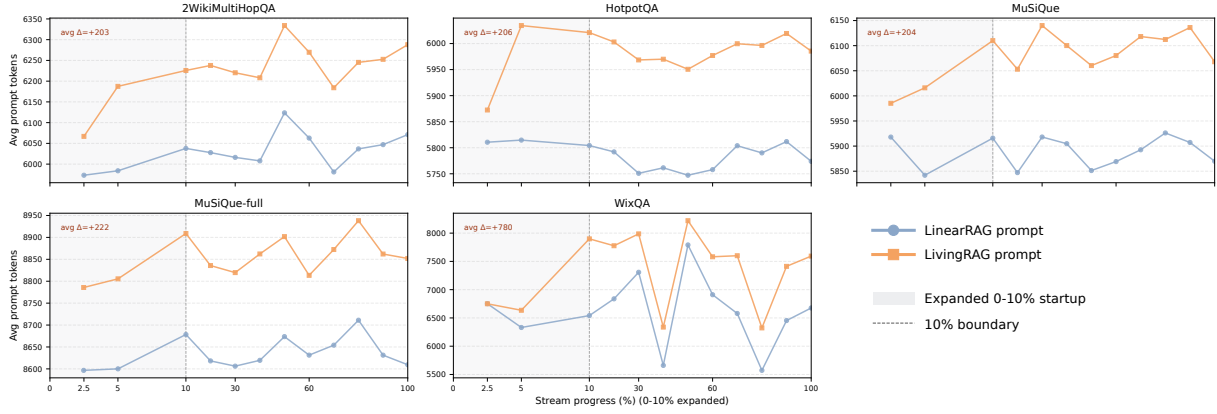


Figure 8: Prompt token trends over online QA streams. Each panel corresponds to one dataset. The blue curve shows LinearRAG and the orange curve shows LivingRAG. The first 10% of each stream is expanded to show the startup stage.

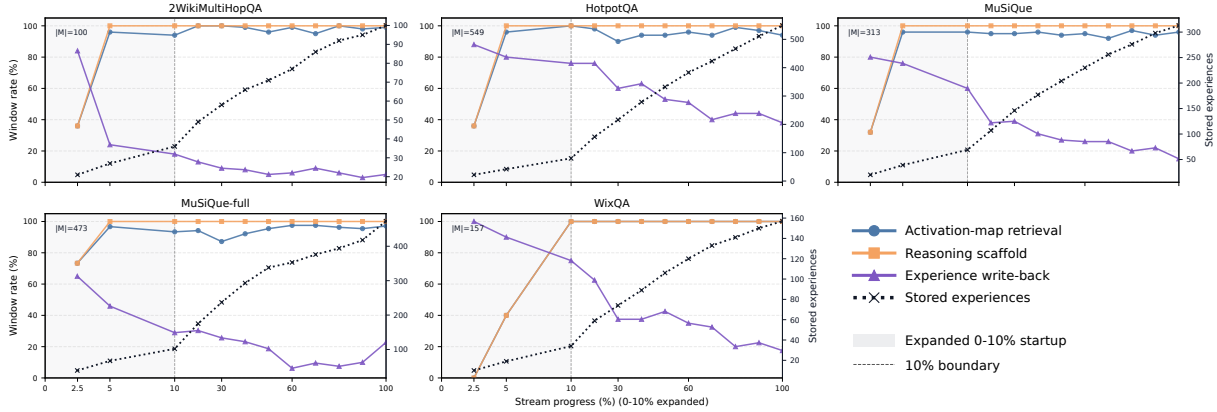


Figure 9: Online use of stored experiences in LivingRAG. The curves show activation-map retrieval, reasoning scaffold use, experience write-back, and the cumulative number of stored experiences. The first 10% of each stream is expanded to show the startup stage.

VII explored the coast of North America. Sebastian Cabot (explorer): Sebastian Cabot was the son of John Cabot.

Summary of the reasoning chain: The current question asks for the child of a navigator. Ignacio Esparza gives Guadalajara, and the Guadalajara evidence links the location to North America. The retrieved Cabot evidence identifies the relevant navigator as John Cabot, not Francisco de Orellana. Since the Sebastian Cabot article states that Sebastian Cabot was John Cabot's son, the answer is Sebastian Cabot.

In E1, the useful stored experience is not only a previous answer. It injects a coherent Cabot-centered entity neighborhood, which moves retrieval from a generic explorer branch to the evidence chain that contains the gold answer.

E2: Experience transfers the Central Europe–406 event

Dataset / index: MuSiQue-full / 798

Question: At the end of which year did tribes from the area where Linear Pottery culture was found invade the Roman Empire?

LinearRAG output: 113 BC. **Result:** incorrect. **Completion tokens:** 4,360.

LivingRAG output: 406. **Result:** correct. **Completion tokens:** 847.

Experience retrieved by LivingRAG: Source question: At the end of which year did tribes from the place where the Celts are said to have migrated from invade the Roman empire? Stored answer: 406. Activation signal: 11 experience entities are added, and 10 new entities cross the retrieval threshold. Important activated entities include Suebi, Vandals,

Alani, Central Europe, and 31 December 406.

Retrieved articles and answer evidence: Linear Pottery (LBK) context: the relevant Neolithic evidence places the cultural area in Central Europe. Galicia (Spain): in the early fifth century, tribes of Central Europe, including the Suebi, Vandals, and Alani, crossed the Rhine on 31 December 406.

Summary of the reasoning chain: The question first resolves the Linear Pottery area to Central Europe. The reused experience points to the same historical event pattern: Central European tribes entering Roman territory. The Galicia passage gives the date explicitly as 31 December 406. Therefore the requested year is 406.

E2 shows the same mechanism in a different domain. The earlier experience supplies event-specific entities, so LivingRAG retrieves the passage that directly states the answer year instead of following a distracting ancient-date association.

S1: Scaffold reuse for the Moon River–Ondine template

Dataset / index: MuSiQue-full / 627

Question: For what did the singer of Moon River in Breakfast at Tiffany's win a Tony?

LinearRAG output: Best Actress in a Play for Ondine. **Result:** correct. **Completion tokens:** 3,628.

LivingRAG output: Ondine. **Result:** correct. **Completion tokens:** 325.

Selected scaffold: Source question: For what did the singer of the original version of Moon River win a Tony award? Stored answer: Ondine.

Table 16: Coverage of stored experiences during LivingRAG inference. Activation-map retrieval counts how often stored experiences are selected in the top- K experience match set used for retrieval guidance. Reasoning scaffold reuse counts how often stored experiences are selected as generation scaffolds. Coverage is the percentage of stored experiences selected at least once.

Dataset	Stored experiences	Activation-map retrieval		Reasoning scaffold reuse		
		Top- K selections	Coverage (%)	Uses	Coverage (%)	Max uses
2Wiki	100	4820	97.00	984	85.00	103
HotpotQA	549	2814	79.60	984	58.11	20
MuSiQue	313	2799	86.90	983	70.29	30
MuSiQue-full	473	6828	93.02	2401	81.61	68
WixQA	157	1910	88.54	384	67.52	20

Scaffold score: 0.598. Template score: 0.877. Semantic score: 0.798.
Retrieved articles and answer evidence: Moon River: the song was originally performed by Audrey Hepburn in Breakfast at Tiffany’s. List of awards and honours received by Audrey Hepburn: Hepburn won the 1954 Tony Award for Best Performance by a Leading Actress in a Play in Ondine.
Scaffold shown to the LLM: Resolve the Moon River performer as Audrey Hepburn. Then use the Audrey Hepburn awards passage. The Tony Award evidence points to Ondine. Answer with the award object, not the whole intermediate chain.
LLM response after scaffold reuse: The current question uses the same template as the stored case. Moon River in Breakfast at Tiffany’s identifies Audrey Hepburn. Her awards passage states that the Tony Award was for Ondine. The answer is Ondine.

S1 illustrates scaffold reuse without an accuracy change. LivingRAG still uses the same evidence chain, but the scaffold makes the decomposition explicit before generation, reducing completion tokens by an order of magnitude.

S2: Scaffold reuse for a repeated Clarksville–Nashville distance hop

Dataset / index: MuSiQue / 948
Question: How many miles are there between Nashville and the place in Tennessee where Mark Day was born?
LinearRAG output: 45. **Result:** correct. **Completion tokens:** 7,054.
LivingRAG output: 45 miles. **Result:** correct. **Completion tokens:** 1,042.
Selected scaffold: Source question: How far is Hod Lisenbee’s place of death in TN from Nashville? Stored answer: 45 miles. Scaffold score: 0.475. Template score: 0.692. Semantic score: 0.732.
Retrieved articles and answer evidence: Mark Day (racing driver): Mark Day was born in Clarksville, Tennessee. Tennessee: Clarksville is about 45 miles northwest of Nashville.
Scaffold shown to the LLM: Resolve the person-specific Tennessee location first. Then reuse the Clarksville-to-Nashville distance passage. If the current person also maps to Clarksville, answer 45 miles.
LLM response after scaffold reuse: Mark Day’s birthplace is Clarksville, Tennessee. The Tennessee passage gives Clarksville as about 45 miles northwest of Nashville. The answer is 45 miles.

S2 shows that the scaffold can be reused even when the surface entity changes. The old case binds Hod Lisenbee to Clarksville, while the new case binds Mark Day to Clarksville; the second hop and the answer form remain the same.

K Additional Robustness and Transfer Analyses

K.1 Benefits by Realized Reuse Signal

We further conduct a query-level analysis of realized graph and template reuse. Table 17 reports

Table 17: Performance changes conditioned on realized graph and template reuse.

Dataset	Graph	Template	Δ LLM-Acc.	Δ Completion tokens
2Wiki	–	–	+0.59	-12.8
	–	✓	+2.97	-189.1
MuSiQue-full	–	–	+0.56	+23.6
	✓	–	+2.12	-297.2
	–	✓	+4.73	-1352.9
	✓	✓	+9.81	-964.7
WixQA	–	–	+1.03	-155.6
	✓	–	+4.23	-272.9
	✓	✓	+8.39	-454.8

paired changes; Δ = LivingRAG – LinearRAG, and negative completion values indicate savings.

Either signal yields greater benefits than neither, while their combination can further strengthen the gains. Missing rows contain no samples.

K.2 Query-Order Robustness

We evaluate the same complete 2Wiki, MuSiQue-full, and WixQA streams under two additional random query permutations that change only arrival order. In Table 18, “3 orders” reports the mean and sample standard deviation over the paper order and two permutations.

Accuracy and token cells report LinearRAG / LivingRAG. Accuracies and reuse rates are percentages; tokens are per-query averages. LivingRAG achieves higher accuracy than LinearRAG on every reported metric and uses fewer completion tokens in every tested order. The realized reuse patterns also remain stable after reordering. These consistent results provide evidence that LivingRAG’s benefits are not only driven by one favorable stream arrangement.

Table 18: Robustness across different online query orders.

Dataset	Setting	Cont.-Acc.		LLM-Acc.	Completion tokens	Graph trans.	Template reuse
2Wiki	Paper	79.60 / 82.70		84.40 / 87.00	1067.6 / 906.4	0.00	84.15
	3 orders	79.13±0.42 / 83.33±2.03		83.80±1.22 / 89.40±5.14	1038.7±25.4 / 806.4±92.7	0.00±0.00	81.54±2.69
MuSiQue-full	Paper	41.00 / 44.39		52.71 / 58.42	2223.8 / 1643.7	82.42	50.44
	3 orders	41.36±1.12 / 44.21±0.90		53.10±0.44 / 58.09±0.88	2231.8±26.5 / 1744.8±94.4	81.84±0.55	50.15±0.64
WixQA	Paper	–		66.00 / 70.25	1653.8 / 1378.1	91.50	7.03
	3 orders	–		66.25±1.39 / 70.25±1.00	1689.4±38.6 / 1403.6±22.2	91.92±0.38	7.29±0.30

Table 19: Retrieval-level transfer on WixQA.

Setting	Article R@5	Hit@5	MRR@5
Native PPR	62.42	68.00	42.74
+ Experience fusion	63.42	69.00	44.82

K.3 Transfer to an Entity-Passage PPR Retriever

We chose LinearRAG because it is a strong, lightweight retriever with an explicit entity state before PPR. This allows us to isolate experience reuse without changing the corpus graph or ranking procedure. To test compatibility beyond LinearRAG, we run a preliminary retrieval-level test on a separate entity-passage PPR retriever. As the test isolates interface compatibility rather than end-to-end integration, we report retrieval rather than generation metrics.

The adapter treats verified states as sparse priors over target graph nodes, an interface also exposed by HippoRAG2 (Gutiérrez et al., 2025). For native reset $\mathbf{r}_q = [\mathbf{r}_q^E; \mathbf{r}_q^P]$, our adapter uses

$$\mathbf{r}'_q = [(1 - \lambda_q)\mathbf{r}_q^E + \lambda_q\tilde{\mathbf{h}}_q; \mathbf{r}_q^P], \quad (28)$$

where $\tilde{\mathbf{h}}_q$ mixes earlier verified states. PPR and ranking remain unchanged.

We use all 400 WixQA queries and 7,132 corpus chunks. The graph uses cached entity annotations and maps only earlier quality-gated states.

R@5 measures top-five gold-article coverage, Hit@5 any gold hit, and MRR@5 the first-hit rank. The gains provide preliminary evidence of transfer through this interface.