SEARCH-MEM: REINFORCEMENT LEARNING FOR MEMORY-AUGMENTED SEARCH AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Reinforcement learning (RL) has recently been used to train search agents that iteratively retrieve evidence for large language models (LLMs). However, two problems remain. First, a search agent can stop before gathering enough evidence to produce the correct answer. Once the search stops, the system has no way to recover from insufficient or incorrect evidence. Second, deeper search increases the amount of retrieved information carried across search steps, which can eventually exceed the available context and remove evidence needed for later decisions. We introduce SEARCH-MEM, a memory-augmented multi-agent search system trained with RL. The system consists of five cooperating agents: a planner, compressor, retriever, generator, and memory agent. The planner controls the search by deciding what to query, which retrieved documents to keep, when to use the compressor, and when to stop. The compressor is trained jointly with the planner to condense retrieved evidence into short statements that can be carried into later search steps. Both agents are optimized using the improvement that the gathered context brings to the answer of a frozen generator, while the retriever and generator remain fixed. After RL training, the system constructs the memory agent by replaying multi-hop questions and storing how each question was decomposed without using reference answers. At inference, the trained search agents first gather evidence, and the generator produces several answers. If these answers are inconsistent or empty, the memory agent retrieves similar past decompositions and uses them to guide an additional search. The additional search gives the system another chance to find missing evidence, while the compressor keeps the planner's context short as the search continues. Across six QA benchmarks and three frozen generators, SEARCH-MEM achieves the best average accuracy under both an open and a proprietary judge, with the best result in 34 of 36 settings.

1 INTRODUCTION

Pretrained large language models (LLMs) perform a broad range of language tasks (Brown et al., 2020; Chowdhery et al., 2023), but the knowledge they carry is fixed when training ends. They cannot answer questions about facts they never saw (Mallen et al., 2023b) or facts that have since changed (Lazaridou et al., 2021), and retraining is far too slow to keep up with either. Retrieval-augmented generation (RAG) avoids retraining by supplying evidence from an external corpus at inference (Lewis et al., 2020; Gao et al., 2023). A single retrieval may be enough for a direct factual question, but some questions require finding an intermediate fact before the model knows what to search for next. A recent line of work therefore moves retrieval out of the pre-processing step and into the model's own decision-making: a search agent issues a query, reads what comes back, and decides what to ask next and when it has gathered enough for a generator to answer (Li et al., 2025; Jin et al., 2025; Jiang et al., 2025).

Recent work casts search-augmented question answering as an agentic problem (Yao et al., 2022; Li et al., 2025): the model is given a retriever and decides what to query, whether to query again, and whether to stop the search loop. Those decisions are obtained either by prompting a frozen LLM to make them in context or by training a search policy to make them with reinforcement learning (RL). Prompt-based methods interleave retrieval with reasoning: IRCoT (Trivedi et al., 2023) retrieves at each chain-of-thought step, Self-Ask (Press et al., 2023) asks explicit follow-up questions, and decompose-then-retrieve pipelines (Zhu et al., 2025; Bacellar, 2026) resolve sub-

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

questions in sequence. These pipelines decompose every question, whether or not the first retrieval sufficed, and the examples they show the model are often curated. Training-based methods reward the planner for the answers it enables: Search-R1 (Jin et al., 2025) optimizes answer reward over full rollouts; s3 (Jiang et al., 2025) freezes the generator and credits only the gain the retrieved context yields, matching its accuracy on a fraction of the data. These agents learn effective search policies, yet two problems remain. First, the planner decides when to stop before the final answer is produced. If the gathered evidence turns out to be insufficient, the generator may answer incorrectly, but the system has no mechanism to search again. Second, deeper search increases the amount of information carried by the planner. Every retrieval adds new evidence to its context, and once the context exceeds the available window, useful evidence may be truncated even when it is needed for later search decisions. Existing work addresses these problems separately: one line learns when to stop searching (Jin et al., 2025; Jiang et al., 2025), while another compresses retrieved context (Yoon et al., 2024; Xu et al., 2024). However, they lack a mechanism to restart search when the evidence is insufficient, and separately trained compressors may discard information needed in later steps.

We present SEARCH-MEM, a memory-augmented search system with five cooperating agents: a planner, compressor, retriever, generator, and memory agent. The planner controls queries, document selection, compression, and stopping. The compressor shortens retrieved text in the planner's context, while the generator answers from the retained documents. The system runs in three stages. First, the planner and compressor are trained jointly with RL while the retriever and generator remain frozen. Next, the trained agents build memory by replaying multi-hop training questions and storing decompositions that pass verification checks without reference answers. At inference, the system performs an initial search and samples answers from the generator. If these answers disagree or are empty, memory retrieves similar decompositions to guide an additional search, and the newly retrieved documents are appended to the generator's context. In summary, our contributions are:

- We introduce an agentic search system with a memory: the trained agents are run on a training set of multi-hop questions, a verifier admits the episodes without reading a reference answer, and a gate reads the memory only where the generator's samples are not settled. The memory supplies a decomposition format that serves at inference time.
- We place compression inside the search loop and train the compressor jointly with the planner on a shared trajectory reward, which helps the planner maintain much less retrieved text during search, while accuracy stays approximately the same compared to no compression.
- We show that on six QA benchmarks and multiple generator sizes, SEARCH-MEM attains the best average accuracy under an open and a proprietary judge, consumes 2.5% of the questions Search-R1 trains on, and serves every generator, including a proprietary one.

2 RELATED WORK

Reinforcement learning trained search agents. Search-R1 (Jin et al., 2025) optimizes answer reward over full rollouts, while s3 (Jiang et al., 2025) rewards context gains for a frozen generator with less training data. ReSum (Wu et al., 2025) pairs the search policy with a frozen summarizer, MEM1 (Zhou et al., 2025) trains one policy to consolidate its own context into a constant-size state, and MMOA-RAG (Chen et al., 2025) trains the modules of a single-pass pipeline jointly. These approaches leave a gap between managing context during search and recovering missing evidence after the generator answers. SEARCH-MEM jointly trains a planner and compressor using an answer-based reward from a frozen generator. It also stores question decompositions in memory to guide additional search when the generator's answers disagree or are empty.

Decomposition and iterative retrieval. Interleaving retrieval with reasoning is well explored: IRCoT (Trivedi et al., 2023) retrieves at each chain-of-thought step, Self-Ask (Press et al., 2023) elicits explicit follow-up questions, and ChainRAG (Zhu et al., 2025) and BridgeRAG (Bacellar, 2026) use decomposition and bridge-conditioned retrieval. These approaches structure initial retrieval around intermediate reasoning but leave open when decomposition is needed after an initial answer. SEARCH-MEM retrieves relevant past decompositions from memory to guide additional search when the generator's answers disagree or are empty. Newly retrieved documents are appended to the existing context, preserving all the evidence already gathered.

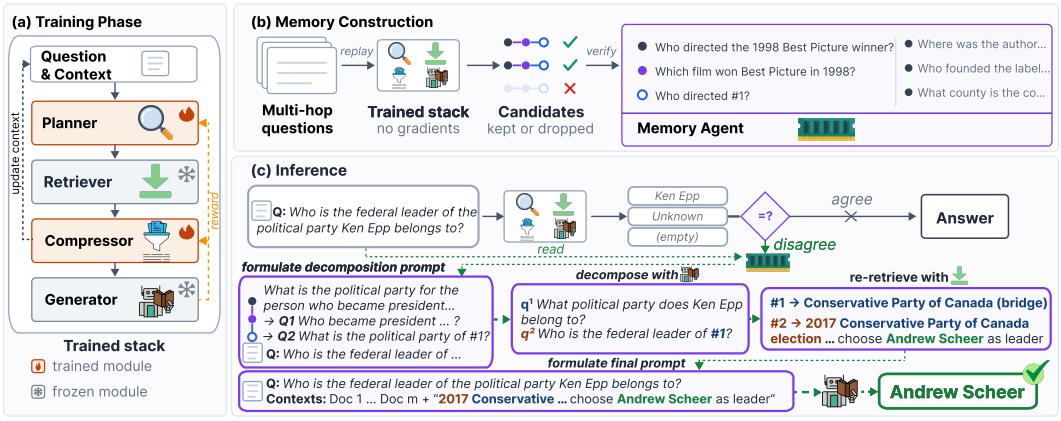


Figure 1: Overview of SEARCH-MEM. **Training Phase.** The planner and compressor are trained jointly with RL. **Memory Construction.** The trained agents replay multi-hop questions to construct the memory agent. **Inference.** The generator answers several times from the gathered context. If the answers agree, one is returned. Otherwise the memory agent is read: its nearest entries are placed in the decomposition prompt as examples, the generator splits the question into q^1 and q^2, each is retrieved in turn, and the new documents are appended to the existing context for a final answer. In the example, the answers disagree, q^1 resolves the bridge, and q^2 with the bridge filled in retrieves the document with the correct final answer.

Memory for LLM agents. A-MEM (Xu et al., 2025) maintains evolving interconnected notes, while MemAgent (Yu et al., 2025) learns a memory policy for long-context reading. Agent Workflow Memory (Wang et al., 2025) induces reusable workflows from past trajectories, and Buffer of Thoughts (Yang et al., 2024) retrieves thought templates to instantiate on new problems. These approaches do not directly address constructing and selectively reusing question decompositions to recover missing retrieval evidence. SEARCH-MEM writes its own memory without reference answers, storing questions, decompositions, and bridge flags without answers or documents. It consults this memory when the system's generator answers disagree or are empty.

3 METHODOLOGY

SEARCH-MEM is a search system of five cooperating agents: a planner and a compressor, trained jointly over a frozen retriever; a frozen generator; and a memory the system writes for itself. The planner issues each query, keeps the documents worth showing the generator, and decides when to compress and when to stop; the compressor condenses a retrieval into a statement the planner carries instead; both are credited by the answer the generator gives from the kept documents. The method runs in three stages. First, the training phase trains the planner and the compressor. Then, in the Memory Construction phase, the trained agents then write the memory $\mathcal{M}$ over multi-hop training questions, keeping each split whose bridge entity appears in a retrieved document and whose answer is stable, with no reference answer read. At inference, where the generator's k samples disagree, the question is decomposed in the form of the nearest episodes, re-retrieved, and the new documents are appended to $c(q)$ for a final answer. We describe the agents, then the loop, the stages in our pipeline and then the results and analysis in the sections below. The overall diagram of our framework is shown in Figure 1.

3.1 SEARCH-MEM AGENTS

Planner π_p. The planner observes the question q and its current state s_{t-1}, which contains the search history and available evidence. Its actions are to issue a query, select documents, request compression, and stop:

$$o_{p,t} = (q, s_{t-1}), \qquad u_t \sim \pi_p(\cdot \mid o_{p,t}), \qquad u_t \in \mathcal{U}_p(s_{t-1}). \tag{1}$$

These actions may be combined within a turn, with at most one query and with selection and compression restricted to available document blocks. A turn need not issue a query. A turn may select or compress documents without searching. The planner receives the trajectory reward r_p in Equation 7, which combines answer correctness, improvement over naive RAG, and a shaping bonus.

Retriever R. The retriever is a fixed index over the corpus. Given a search query q_t, it returns the nearest retrieval documents, $D_t = R(q_t)$. Because it is fixed, the planner can only improve what reaches the generator by asking better queries and keeping better documents, never by changing what the index returns.

Compressor π_c. When requested by the planner, the compressor observes the original question, the query associated with a retrieved block, and its documents. Its action is to generate a short evidence statement:

$$o_{c,j} = (q, q_j, D_j), \qquad e_j \sim \pi_c(\cdot \mid o_{c,j}), \qquad e_j \in \mathcal{X}, \tag{2}$$

where $\mathcal{X}$ is the space of text sequences. The statement replaces the block only in the planner's context, keeping it short for later search decisions. The generator receives the documents explicitly selected by the planner. Each compressor call receives the return of its trajectory, $r_c = r_p$, training it through its effect on later search decisions and the final answer.

Generator G. The generator is a frozen LLM used for three tasks: answering the question from the context $c(q)$ the planner assembled, splitting a question into two sub-questions when asked, and naming the bridge entity from the documents retrieved for the first sub-question.

Memory $\mathcal{M}$. The memory contains episodes constructed by other agents in our system. We define:

$$z = \langle q, q^{(1)}, q^{(2)}, \tau \rangle \tag{3}$$

z is an entry of $\mathcal{M}$, which includes the question q, a first sub-question $q^{(1)}$, a second sub-question $q^{(2)}$ whose answer depends on the answer to the first, and a flag τ for whether that first answer is an intermediate entity, which we call the bridge. In $q^{(2)}$ the bridge is written as a placeholder, so a stored split reads the same for every question of its shape. Questions are embedded with a frozen sentence encoder and indexed. When the generator is asked to split a new question, the n_e questions nearest to it are shown as worked examples of a split, three with a bridge and one without, and the new split is always written fresh. The memory agent is written after the training of planner and compressor is completed. How the memory is written and read is given in Section 3.4 and Section 3.5, respectively.

3.2 The Search Loop

The system first retrieves documents using the original question. The planner then has T_{max} turns in which it may search, followed by a final turn without search access. A turn may select or compress documents without issuing a query, so the initial search uses at most $T_{max} + 1$ retrievals. Let s_t denote the planner's state and $\mathcal{B}_t$ the collected document blocks. Starting with $s_0 = D_0$ and $\mathcal{B}_0 = (D_0)$, each turn updates them as

$$u_t \sim \pi_p(\cdot \mid q, s_{t-1}), \qquad s_t = \mathrm{Update}(s_{t-1}, u_t, D_t), \qquad \mathcal{B}_t = \mathcal{B}_{t-1} \oplus D_t, \tag{4}$$

where u_t is the planner's output and D_t contains newly retrieved documents, or is empty if no query is issued. The operator $\oplus$ appends non-empty retrieval blocks. The function Update adds any newly retrieved documents to the planner's state, records its document-selection marks, and replaces any block chosen for compression with the compressor's evidence statement. All other content remains unchanged. Compression replaces retrieved text only in the planner's view; the underlying documents remain available to the generator.

After the final turn T, the generator receives the selected documents from each marked block and all documents from each unmarked block. The same rule applies to the initial retrieval. Together, these documents form c_T, the context used to answer the question:

$$c(q) = c_T, \qquad \hat{a} = G\big(q, c(q)\big). \tag{5}$$

3.3 TRAINING PHASE

First, the search loop is trained on the answer G produces from $c(q)$. In this phase, $\mathcal{M}$ does not yet exist. Let $\mathcal{A}(\hat{a}, a^\star) \in \{0, 1\}$ score a predicted answer $\hat{a}$ against the reference $a^\star$, using the generation accuracy of Section 4.1, and let

$$V(c) = \mathcal{A}\big(G_0(q, c),\, a^\star\big), \tag{6}$$

denote whether G's greedy answer under context c is correct, where G_0 is greedy decoding. Under G_0, the generator is deterministic, so V depends on the context alone and adds no sampling noise to the advantage, and it costs one call per trajectory. Both policies are trained against V and differ only in what they are credited for; neither differentiates through G.

Planner Objective. The planner's return combines its gain over a single retrieval with the quality of the context it assembled:

$$r_p = \underbrace{V\big(c(q)\big) - V\big(c_{\text{naive}}\big)}_{\text{gain over naive RAG}} + \underbrace{V\big(c(q)\big)}_{\text{answer correctness}} + \underbrace{\beta\, r_{\text{shaping}}}_{\text{shaping bonus}} = 2V\big(c(q)\big) - V\big(c_{\text{naive}}\big) + \beta\, r_{\text{shaping}}. \tag{7}$$

where $c_{\text{naive}} = R(q)$ is a single retrieval with the original question, scored once per training question and cached. Before shaping, a correct trajectory receives a reward of 2 if naive RAG fails and 1 if it succeeds. An incorrect trajectory receives 0 if naive RAG also fails, or -1 if naive RAG succeeds. The shaping term r_{shaping} is 1 only if the trajectory meets all three conditions: (1) the planner uses the compressor at least once, (2) it stops voluntarily before exhausting the turn budget, and (3) its context improves answer correctness over naive RAG, so $V(c(q)) > V(c_{\text{naive}})$. Otherwise, $r_{\text{shaping}} = 0$. The bonus rewards trajectories that improve on naive RAG while using compression and stopping voluntarily. Requiring all three conditions ties the bonus to this combination of behaviors, rather than rewarding compression or early stopping alone. The coefficient β scales this bonus and is small, so shaping steers behaviour without competing with the answer terms. We optimize π_p with GRPO (Shao et al., 2024), which estimates each trajectory's advantage relative to a group sampled for the same question.

Compressor Objective. The compressor affects the final answer through the planner's later search decisions. We therefore train it using the same trajectory return as the planner, rewarding summaries that help the search produce a correct answer rather than those that merely shorten the text. Each compressor call receives the return of the trajectory it belongs to:

$$r_c = r_p, \qquad \nabla_{\theta_c} J = \mathbb{E}\big[(r_c - b)\, \nabla_{\theta_c} \log \pi_c(e_t \mid q, q_t, D_t)\big], \qquad b = \frac{1}{N} \sum_{i=1}^{N} r_p^{(i)}, \tag{8}$$

optimized with REINFORCE (Williams, 1992). We use a batch-mean baseline because, unlike the planner, a compressor input (q, q_t, D_t) is specific to one hop of one trajectory and therefore has no natural group of comparable samples. Prior compressors are typically trained with a separate compression objective or reward (Yoon et al., 2024; Gu et al., 2026; Xu et al., 2024), while search methods with a summarizer often keep it frozen (Wu et al., 2025). Instead, our compressor is optimized directly by the downstream answer reward. Since its output is only read by the planner, it affects the answer indirectly through later queries and document selections.

3.4 MEMORY CONSTRUCTION

We build memory by running the trained agents once over multi-hop questions from a dataset separate from those used to train the planner and compressor. No model parameters are updated at any point during this stage. Each question produces a candidate episode z for $\mathcal{M}$ (Equation 3). Given a question q, the planner and compressor first assemble $c(q)$ using the same procedure as at inference. We then prompt G to split q into $q^{(1)}$ and $q^{(2)}$. The retriever R searches for $q^{(1)}$, and G extracts its answer from the returned documents as the bridge entity b. We substitute b into $q^{(2)}$ and retrieve again. This query was not issued during the original search loop. m of the returned documents are appended to $c(q)$, and G answers q from this expanded context k times. We determine the flag τ during replay, without using dataset annotations. If $q^{(2)}$ has no placeholder or the extracted bridge fails a simple entity check, we mark the split as having no bridge and issue $q^{(2)}$ unchanged.

We apply two checks before storing a split. First, at least one document retrieved for the second sub-question must contain the bridge entity. For splits without a bridge, a retrieved document must instead contain the non-empty majority answer. Second, a majority of the k generated answers must agree on a determinate answer. Neither check uses the reference answer, so they assess whether the decomposition is well formed without establishing that the answer is correct. Both rely on string matching and were fixed before memory construction (Appendix H). Splits that pass both checks are stored in the format defined in Equation 3. We discard the bridge b, retrieved documents, and generated answers. The completed memory remains fixed across every generator and test set in our evaluation.

3.5 Inference Loop

For a test question q, the search agents assemble $c(q)$, and the generator produces k answers. If all are non-empty and agree after normalization, the agreed answer is returned. Otherwise, the system retrieves four decomposition examples from memory agent to guide one additional search. The intervention retrieves documents using the original question and asks the generator to split it into two sub-questions. If the second contains a bridge placeholder, the system retrieves evidence for the first and extracts the bridge. After checking the entity and rejecting redundant queries, it substitutes the bridge into the second sub-question and retrieves again. The returned set is accepted if at least one document mentions the bridge. Accepted second-hop documents take priority over the original-question results. The combined list is deduplicated, capped at four documents, and appended to $c(q)$. If decomposition or verification fails, the original-question results provide the additional evidence. The existing context is preserved. Finally, the generator answers again and gives out the final answer.

4 Experimental Results

Our experiments answer four questions. RQ1: does Search-Mem answer more accurately than trained and training-free search agents, and does this hold across generators? RQ2: what is the contribution of the memory agent? RQ3: what is the contribution of the compressor? RQ4: what does searching again cost, and does the gate keep that cost down? Section 4.2 answers RQ1, and Sections 5.1, 5.2, and 5.3 answer RQ2, RQ3, and RQ4 respectively.

4.1 Experimental Setup

Datasets The training set is drawn from the Natural Questions and HotpotQA training splits, following prior work (Jin et al., 2025; Jiang et al., 2025). As we observe, training stabilize within about 15 GRPO steps (shown below in Figure 2 Right). Based on this observation, we use the checkpoint at step 20 throughout. With batch size of 60 questions, this corresponds to 1.2k training questions for our Training phase. The memory is written from 3k multi-hop questions from the MuSiQue training split, disjoint from every evaluation set, with no gradient updates. In total the method consumes 4.2k questions, 1.2k with gradients and 3k without. We evaluate on the full test splits of six open-domain QA datasets over a Wikipedia corpus: three single-hop, Natural Questions (Kwiatkowski et al., 2019), TriviaQA (Joshi et al., 2017), and PopQA (Mallen et al., 2023a), and three multi-hop, HotpotQA (Yang et al., 2018), 2Wiki (Ho et al., 2020), and MuSiQue (Trivedi et al., 2022). A seventh, MultiHop-RAG, uses a news corpus and is described in Section 4.2.

Baselines We compare against three groups. **End-to-end fine-tuning**, where the generator itself is trained: Search-R1-FT, which includes finetuning the whole pipeline (Jin et al., 2025), SFT, and R1, reported as released with their own 3B or 7B generators. **Training-free search with a frozen generator**: Direct Inference, CoT, IRCoT (Trivedi et al., 2023), and ChainRAG (Zhu et al., 2025), prompted to retrieve as they reason. **Trained search with a frozen generator**: s3 (Jiang et al., 2025), which trains only the searcher on the gain its context yields; MEM1 (Zhou et al., 2025); and Search-R1 used as a searcher only, its released policy retrieving the context that our frozen generator then answers from. The implementation details for all baselines are shown in Appendix I.

Evaluation Metric Following prior work (Jiang et al., 2025), we measure performance using *Generation Accuracy*, which combines a span-matching test (Ma et al., 2021; Lin et al., 2021) with an

Table 1: **Main results.** GenAcc over 6 datasets with Qwen2.5-14B (Q) and Claude-Haiku (C) as judges. #Train = training examples used by the baselines. Avg. is the mean over the six datasets. The side bar marks rows whose generator is fine-tuned (red) and panels whose generator is frozen (blue). **Bold** = best, underlined = second-best per column within each panel.

			NQ		HotpotQA		TriviaQA		PopQA		2Wiki		MuSiQue		**Avg.**	
	Method	#Train	Q	C	Q	C	Q	C	Q	C	Q	C	Q	C	Q	C
	$\text{SFT}_{\text{Qwen2.5-3B-Inst}}$	170k	24.0	22.7	18.9	18.5	42.8	41.5	17.6	17.1	20.6	20.0	5.1	5.1	21.5	20.8
	$\text{R1}_{\text{Qwen2.5-7B-Inst}}$	170k	33.4	33.0	27.8	27.0	61.2	60.8	21.5	21.4	28.8	27.5	9.7	9.5	30.4	29.9
	Search-R1-FT-7B	170k	50.8	49.8	49.7	48.9	70.6	69.9	49.7	49.2	50.9	50.5	23.0	22.0	49.1	48.4
Qwen2.5-7B	Direct Inference	0	34.6	32.9	27.4	25.7	53.7	53.3	20.3	20.4	36.2	36.5	9.8	8.8	30.3	29.6
	CoT	0	36.3	34.3	29.8	27.4	59.2	59.0	20.3	20.4	29.7	29.8	11.0	10.1	31.1	30.2
	IRCoT	0	56.0	53.1	48.3	41.7	73.1	71.4	50.9	50.0	37.2	37.2	13.5	12.3	46.5	44.3
	ChainRAG	0	52.5	49.9	43.4	39.5	70.7	68.3	47.8	47.1	35.1	35.1	11.3	10.2	43.5	41.7
	Search-R1	170k	57.6	54.8	52.3	47.4	73.1	71.4	50.0	49.0	45.8	45.5	23.3	22.1	50.3	48.4
	MEM1	170k	53.9	51.4	52.9	48.9	67.6	66.3	46.8	45.9	23.6	23.5	16.2	15.4	43.5	41.9
	s3	2.4k	58.3	55.1	52.3	47.4	74.5	72.8	52.2	51.2	43.3	43.2	17.7	16.8	49.7	47.8
	SEARCH-MEM	4.2k	**58.9**	**55.9**	**53.0**	**50.8**	**76.3**	**74.6**	**53.7**	**52.6**	**48.1**	**47.7**	**24.8**	**23.3**	**52.5**	**50.8**
Qwen2.5-14B	Direct Inference	0	42.0	40.1	33.3	31.1	65.8	65.3	26.4	26.1	40.3	40.4	13.0	12.8	36.8	36.0
	CoT	0	44.4	41.5	35.8	32.2	69.6	68.7	27.6	26.9	36.5	36.6	13.8	13.3	37.9	36.5
	IRCoT	0	57.8	55.1	48.6	44.0	75.1	73.4	53.1	52.1	30.9	30.9	16.8	15.5	47.1	45.2
	ChainRAG	0	58.1	54.2	47.2	42.3	75.8	74.3	54.8	53.9	36.4	36.4	15.4	14.2	47.9	45.9
	Search-R1	170k	59.2	56.2	54.5	49.8	75.7	74.2	52.2	51.2	41.0	40.9	**29.4**	**27.8**	52.0	50.0
	MEM1	170k	56.2	53.4	42.0	37.8	70.5	69.1	48.9	47.8	21.8	21.7	19.5	18.0	43.1	41.3
	s3	2.4k	59.8	56.9	54.1	49.0	76.8	75.2	54.4	53.4	40.7	40.7	20.9	19.3	51.1	49.1
	SEARCH-MEM	4.2k	**60.4**	**57.2**	**55.5**	**50.5**	**78.1**	**76.5**	**56.0**	**55.1**	**43.0**	**43.0**	28.4	26.7	**53.6**	**51.5**
Claude-Haiku	Direct Inference	0	53.6	51.8	37.9	34.8	79.1	78.0	35.8	35.3	38.8	39.1	15.1	14.2	43.4	42.2
	CoT	0	59.1	58.8	46.5	43.9	79.9	79.8	43.1	42.6	47.7	47.8	23.4	22.2	49.9	49.2
	IRCoT	0	62.5	61.4	47.3	43.8	75.8	74.4	56.6	55.7	45.0	46.1	13.5	13.0	50.1	49.1
	ChainRAG	0	63.7	62.2	47.8	44.5	76.7	75.3	56.4	55.8	39.2	39.2	13.5	12.4	49.6	48.2
	Search-R1	170k	64.0	63.1	58.8	54.7	76.3	75.1	55.8	54.9	52.0	52.5	27.6	26.9	55.8	54.5
	MEM1	170k	59.6	58.8	38.8	34.7	70.1	69.0	51.0	49.9	31.1	32.1	16.3	15.8	44.5	43.4
	s3	2.4k	64.5	63.8	55.2	51.4	78.4	77.2	58.1	57.2	51.4	52.2	19.2	18.5	54.5	53.4
	SEARCH-MEM	4.2k	**66.5**	**65.7**	**60.5**	**56.6**	**83.3**	**83.2**	**59.2**	**59.2**	**53.7**	**54.3**	**27.8**	**27.1**	**58.5**	**57.7**

LLM-based accuracy check. Let p be the model prediction and $\mathcal{A}$ the set of gold answers. After standard normalization, `span_check` returns 1 if any $a \in \mathcal{A}$ is a substring of p; `judge_check` returns 1 if an LLM judge, shown the question, p, and $\mathcal{A}$, answers *yes* to a fixed correctness prompt. We compute: $\text{GenAcc} = \texttt{span_check} \vee \texttt{judge_check}$, querying the judge only when the span test fails, so it recovers correct answers whose wording differs from the reference. We report GenAcc under two judges, Qwen2.5-14B-Instruct and Claude-Haiku. The full protocol is in Appendix F.

4.2 BENCHMARKING RESULTS

Main Results. We report the main results in Table 1, which answers RQ1. Across the three frozen generators, SEARCH-MEM has the best average accuracy under both judges and the best performance in 34 out of 36 cases: with the 14B generator, Search-R1 keeps a lead on MuSiQue, at 40× the training data. SEARCH-MEM has the highest average GenAcc under both judges. The panels also separate the two contributions. Under the Qwen judge, adding memory improves accuracy by 0.5–2.6% on the single-hop datasets and 3.6–8.9% on the multi-hop datasets, averaging each dataset's gain across the three generators. MuSiQue shows the largest improvement for every generator under both judges. Notably, with Claude-Haiku, the memory adds 4.8 points on TriviaQA. Claude-Haiku generator also benefits from the same planner checkpoint and memory, despite not being used during training. The fine-tuned rows above the panels show what training the generator itself buys. Search-R1-FT-7B remains competitive on 2Wiki, but frozen 7B, 14B and Claude generators with SEARCH-MEM match or exceed it elsewhere, without any generator training.

Table 2: **Results on MultiHop-RAG** with Qwen2.5-14B-Instruct as generator. Q = GenAcc under the Qwen2.5-14B judge, C = under the Claude-Haiku judge. Avg. is the unweighted mean over the three question types. **Bold** = best, underlined = second-best per column.

	Infer.		Comp.		Temp.		**Avg.**	
Method	Q	C	Q	C	Q	C	Q	C
Direct Inference	60.5	60.3	56.1	58.2	56.6	57.5	57.7	58.7
CoT	70.0	69.4	50.1	51.9	46.3	50.8	55.5	57.4
IRCoT	94.0	92.3	38.9	48.5	47.5	52.5	60.1	64.4
Search-R1	**95.5**	**94.5**	56.3	67.3	55.8	64.5	69.2	75.4
s3	94.4	93.5	52.1	63.6	56.8	62.5	67.8	73.2
SEARCH-MEM	**95.5**	**94.5**	**58.3**	**67.8**	**61.2**	**66.1**	**71.7**	**76.1**

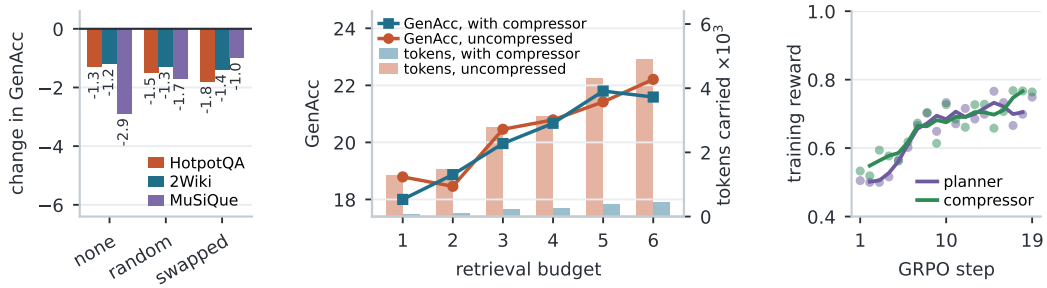


Figure 2: **Left:** GenAcc changes relative to the full method when the decomposition examples are removed (none), chosen at random, or swapped between questions. **Middle:** GenAcc on MuSiQue dataset against the planner's retrieval budget, with and without the compressor and the retrieved tokens carried per trajectory. **Right:** Training rewards per GRPO step for planner and compressor. Dots are the per-step values, and the line is a three-step moving average.

Table 3: **Memory agent ablation.** GenAcc with Claude as judge (without memory → with memory). Results with both judges are given in Table 6.

Dataset	Qwen-7B	Qwen-14B	Haiku
NQ	55.1→55.9	56.9→57.2	65.0→65.7
HotpotQA	46.2→50.8	48.5→50.5	50.1→56.6
TriviaQA	73.3→74.6	75.1→76.5	77.3→83.2
PopQA	51.3→52.6	53.6→55.1	57.2→59.2
2Wiki	41.3→47.7	40.7→43.0	50.4→54.3
MuSiQue	16.2→23.3	17.6→26.7	17.2→27.1
Avg.	47.2→50.8	48.7→51.5	52.9→57.7

Table 4: **Effect of example count and mix on MuSiQue**. Columns report GenAcc under each judge.

n_e (bridge : no-bridge)	Qwen	Claude
0 examples	21.50	20.29
1 (1 : 0)	23.59	21.80
4, deployed (3 : 1)	**24.84**	**23.34**
4 (4 : 0)	24.80	23.34
6 (4 : 2)	24.76	22.88
8 (6 : 2)	24.63	22.92
4 (0 : 4)	19.62	18.46

Zero-shot Transfer Results to a News Corpus. The six datasets above share a Wikipedia corpus with the training and memory-writing questions. To test whether the trained pipeline and its memory depend on that corpus, we evaluate on MultiHop-RAG (Tang & Yang, 2024), whose knowledge base is a collection of news articles and whose queries require linking two to four of them by inference, comparison, or temporal order. We keep the planner checkpoint, the compressor, and the memory exactly as above, and only the retrieval index changes. The null-query type, which tests refusal rather than retrieval, is excluded. Table 2 reports the result with the 14B generator. SEARCH-MEM has the best average under both judges. Since the memory was written from Wikipedia questions, its episodes transfer to a corpus and question style they never saw, which is the strongest evidence that what they carry is a way of splitting a question rather than anything about its content.

5 ANALYSIS

5.1 MEMORY AGENT CONTRIBUTIONS

Table 3 shows that the memory-guided intervention improves accuracy on all six datasets for all three generators under the Claude judge. The largest gains are on MuSiQue, where GenAcc rises from 16.2% to 23.3% for Qwen2.5-7B, from 17.6% to 26.7% for Qwen2.5-14B, and from 17.2% to 27.1% for Claude-Haiku. We provide the full results under both judges in Table 6. We then keep the intervention enabled and change only its decomposition examples. Removing the examples, selecting them randomly, or using examples retrieved for another question reduces accuracy on all three multi-hop datasets (Figure 2, left). Relevant examples therefore contribute beyond the additional search itself. Table 4 varies the number and mix of examples on MuSiQue. The deployed mix achieves the highest accuracy under both judges, but four no-bridge examples reduce Claude-judged GenAcc from 23.34% to 20.29%. Using no examples at all further reduces performance to 18.46%. This suggests that examples showing how to connect sub-questions through a bridge are particularly useful for repair.

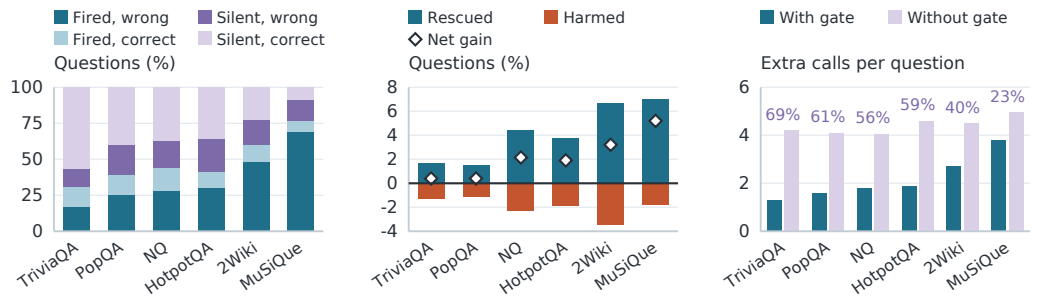


Figure 3: **Left:** share of questions by whether the gate fired or stayed silent and whether the initial majority vote was wrong or correct. **Middle:** share of questions rescued or harmed by repair, with the net gain (diamond). **Right:** extra generator calls per question with the gate and without it; percentages show the calls saved by gating.

5.2 Compressor Contributions

Figure 2 (middle) compares MuSiQue accuracy and planner context size as the retrieval budget increases. Accuracy generally improves with a larger budget, and the compressed and uncompressed variants remain within roughly one percentage point of each other. Their context sizes differ substantially, however. Without compression, the planner accumulates more retrieved text with each search; with compression, its context stays nearly flat and about an order of magnitude smaller. The compressor therefore supports deeper search with a much smaller planner context while maintaining similar accuracy, at a cost of about 1.3 additional model calls per question (details in Table 8, Appendix E).

Figure 2 (right) shows the planner and compressor rewards during training. Both increase over the run and they largely stabilizing after about fifteen GRPO steps. Based on that observation, we train for twenty steps and use the resulting checkpoint throughout evaluation.

5.3 Accuracy and Cost Tradeoffs

Figure 3 (left) shows that the gate fires more often on harder datasets; on MuSiQue, about nine in ten triggered questions have an incorrect initial vote. Repair corrects more answers than it harms across all six datasets as shown in Figure 3 (middle). Applying it to every question yields little additional accuracy, suggesting that the gate mainly avoids unnecessary computation. Each intervention requires one to three retrievals and four or five generator calls, depending on whether bridge extraction and second-hop retrieval are performed. Gating reduces these additional generator calls by 23% to 69% across datasets (Figure 3, right).

Because of space limitations, we place Table 8 in Appendix E, where it compares the costs across different methods. Search-Mem averages about four retrievals and ten model calls per question, more than the other search methods. Its search depth is similar to s3, with additional calls for compression and repair. Averaged over all questions, repair adds about one retrieval and two model calls, concentrating this cost on questions with inconsistent initial answers.

6 Conclusion

We presented Search-Mem, a memory-augmented search system that jointly trains a planner and compressor with a shared reward from a frozen generator. The system builds a memory of question decompositions without reference answers and uses it to guide additional search when generator answers disagree or are empty. Compression reduces the planner's context while preserving accuracy. Using only 2.5% of Search-R1's training questions, Search-Mem achieve the best average accuracy across six QA benchmarks and open and proprietary generators under both judges. Search-Mem also achieves the best average accuracy in zero-shot transfer to the MultiHop-RAG news corpus, supporting the reuse of decomposition formats across domains.

AI USE STATEMENT

Large language models were not involved in the design of our approach. We used them solely to polish the writing of this paper (grammar and style). All methods, experiments, and analyses were designed and run by the authors, who take responsibility for the content.

ETHICS STATEMENT

This work uses public question-answering benchmarks and a public Wikipedia and news corpus; no human subjects or personal data are involved. The memory stores how training questions were split, not their answers or retrieved text, so it carries no content from the corpus. As with any retrieval agent, answers depend on the corpus searched, and a corpus containing false or biased text can be reproduced in an answer; the system does not verify retrieved claims beyond the corpus itself.

REPRODUCIBILITY STATEMENT

The method is specified in Section 3 and given as pseudocode in Appendix G. Table 9 lists every deployed setting: checkpoints, optimizers, hop budgets, sampling temperatures, and the constants used when writing and reading the memory. Appendix 5 gives the memory build and both verifier checks, Appendix I the configuration of every baseline, and Appendices F and J the evaluation protocol and judges. All results use one frozen retriever and publicly available checkpoints; the training phase is a single run at seed 42, and the memory is written once, before evaluation, and then fixed for every generator and test set reported. Code and checkpoints will be released.

REFERENCES

Anthropic. Claude Haiku 4.5 system card, 2025.

Andre Bacellar. Bridgerag: Training-free bridge-conditioned retrieval for multi-hop question answering, 2026. URL https://arxiv.org/abs/2604.03384.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Yiqun Chen, Lingyong Yan, Weiwei Sun, Xinyu Ma, Yi Zhang, Shuaiqiang Wang, Dawei Yin, Yiming Yang, and Jiaxin Mao. Improving retrieval-augmented generation through multi-agent reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=9Ia0KiVAut.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240): 1–113, 2023.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

Jia-Chen Gu, Junyi Zhang, Di Wu, Yuankai Li, Kai-Wei Chang, and Nanyun Peng. BRIEF-Pro: Universal context compression with short-to-long synthesis for fast and accurate multi-hop reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics (COLING)*, 2020.

Pengcheng Jiang, Xueqiang Xu, Jiacheng Lin, Jinfeng Xiao, Zifeng Wang, Jimeng Sun, and Jiawei Han. s3: You don't need that much data to train a search agent via RL. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21599–21617,

Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1095. URL https://aclanthology.org/2025.emnlp-main.1095/.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550/.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. Natural questions: A benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:452–466, 2019.

Angeliki Lazaridou, Adhiguna Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Tomas Kocisky, Sebastian Ruder, Dani Yogatama, Kris Cao, Susannah Young, and Phil Blunsom. Mind the gap: Assessing temporal generalization in neural language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*, 2025.

Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, pp. 2356–2362, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380379. doi: 10.1145/3404835.3463238. URL https://doi.org/10.1145/3404835.3463238.

Xueguang Ma, Kai Sun, Ronak Pradeep, and Jimmy Lin. A replication study of dense passage retriever, 2021. URL https://arxiv.org/abs/2104.05740.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, 2023a.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023b.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL https://arxiv.org/abs/1908.10084.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Yixuan Tang and Yi Yang. Multihop-rag: Benchmarking retrieval-augmented generation for multi-hop queries, 2024.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 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 (ACL)*, 2023.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Multilingual e5 text embeddings: A technical report. *arXiv preprint arXiv:2402.05672*, 2024.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=NTAhi2JEEE.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

Xixi Wu, Kuan Li, Yida Zhao, Liwen Zhang, Litu Ou, Huifeng Yin, Zhongwang Zhang, Xinmiao Yu, Dingchu Zhang, Yong Jiang, Pengjun Xie, Fei Huang, Minhao Cheng, Shuai Wang, Hong Cheng, and Jingren Zhou. ReSum: Unlocking long-horizon search intelligence via context summarization. *arXiv preprint arXiv:2509.13313*, 2025.

Fangyuan Xu, Weijia Shi, and Eunsol Choi. RECOMP: Improving retrieval-augmented LMs with context compression and selective augmentation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=mlJLVigNHp.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. In *Advances in Neural Information Processing Systems*, 2025.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*, 2024.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Chanwoong Yoon, Taewhoo Lee, Hyeon Hwang, Minbyul Jeong, and Jaewoo Kang. CompAct: Compressing retrieved documents actively for question answering. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 21424–21439, 2024.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, et al. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. *arXiv preprint arXiv:2507.02259*, 2025.

Zijian Zhou, Ao Qu, Zhaoxuan Wu, Sunghwan Kim, Alok Prakash, Daniela Rus, Jinhua Zhao, Bryan Kian Hsiang Low, and Paul Pu Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents, 2025. URL https://arxiv.org/abs/2506.15841.

Rongzhi Zhu, Xiangyu Liu, Zequn Sun, Yiwei Wang, and Wei Hu. Mitigating lost-in-retrieval problems in retrieval augmented multi-hop question answering. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22362–22375, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1089. URL https://aclanthology.org/2025.acl-long.1089/.

APPENDIX CONTENTS FOR
Search-Mem: Reinforcement Learning For Memory-Augmented Search Agents

A Notations **14**

B Detailed Memory Ablation **14**

C Hand-Written Exemplars as a Baseline **14**

C.1 Baseline Settings 14

C.2 Results 15

D Qualitative Comparison with ChainRAG **16**

E Detailed Cost Analysis **17**

F GenAcc Algorithm **18**

G The SEARCH-MEM Procedure **19**

H Memory Construction Details **19**

I Baselines Details **20**

J Judge Details **23**

J.1 Judge Information 23

J.2 Relation to The Numbers Reported For s3 Baseline 24

K Illustrative Examples **24**

A NOTATIONS

We present the notations used throughout the paper in Table 5.

B DETAILED MEMORY ABLATION

Table 6 reports the effect of removing the memory agent from our pipeline across six datasets and three generators under both judges. The intervention improves accuracy in every setting, with the largest gains on MuSiQue.

C HAND-WRITTEN EXEMPLARS AS A BASELINE

C.1 BASELINE SETTINGS

The controls of Figure 2 (left) all draw their exemplars from the memory, so they show that exemplars matter and that which ones are shown matters. The decisive control replaces the memory with exemplars a person writes by hand and shows them to every fired question in place of the retrieved episodes, with the rest of the intervention unchanged. We report three such sets in Table 7: a generic set, a set written in MuSiQue's long nested style, and a terse set. The full text of the three sets is in the released code.

Table 5: Notation used throughout the paper.

Symbol	Meaning
$q,\ a^\star$	question and its reference answer
$R,\ n_d$	frozen retriever; documents returned per query
$\pi_p,\ \theta_p$	planner policy and its parameters
$\pi_c,\ \theta_c$	compressor policy and its parameters
G	frozen generator
$\mathcal{M}$	memory: the bank of episodes
$q_t,\ D_t$	query issued at hop t; documents retrieved for it
$S_t,\ e_t$	documents kept at hop t; compressor's statement for hop t
x_t	what hop t adds to the planner's state: e_t if compressed, S_t otherwise
$s_t,\ c_t$	planner's state and generator context after hop t
T	hops taken before the planner stops
$c(q)$	finished generator context, c_T
$\mathcal{A},\ V(c)$	answer scorer; expected score of G under context c
c_{naive}	context of a single retrieval with the original question, $R(q)$
$r_p,\ r_c,\ r_{\text{shaping}},\ \beta$	planner return; compressor return; shaping indicator and its weight
$b,\ N$	compressor baseline; trajectories in the rollout batch
k	answers sampled per question; the gate fires when they disagree
$z = \langle q, q^{(1)}, q^{(2)}, \tau \rangle$	episode: question, two sub-questions, bridge flag
b	bridge entity, the answer to $q^{(1)}$ (discarded after the checks)
$D^+,\ m$	documents retrieved for the resolved $q^{(2)}$; at most m are appended
n_e	episodes shown as examples when splitting

Table 6: Full memory ablation results. Each cell shows GenAcc without $\rightarrow$ with the memory-guided intervention. Q and C denote the Qwen and Claude judges. Qwen-7B and Qwen-14B denote Qwen2.5-7B and Qwen2.5-14B.

Dataset	Judge	Qwen-7B	Qwen-14B	Claude-Haiku
NQ	Q	$58.3 \rightarrow 58.9$	$59.9 \rightarrow 60.4$	$66.0 \rightarrow 66.5$
	C	$55.1 \rightarrow 55.9$	$56.9 \rightarrow 57.2$	$65.0 \rightarrow 65.7$
HotpotQA	Q	$50.9 \rightarrow 53.0$	$53.4 \rightarrow 55.5$	$53.9 \rightarrow 60.5$
	C	$46.2 \rightarrow 50.8$	$48.5 \rightarrow 50.5$	$50.1 \rightarrow 56.6$
TriviaQA	Q	$74.8 \rightarrow 76.3$	$76.7 \rightarrow 78.1$	$78.5 \rightarrow 83.3$
	C	$73.3 \rightarrow 74.6$	$75.1 \rightarrow 76.5$	$77.3 \rightarrow 83.2$
PopQA	Q	$52.4 \rightarrow 53.7$	$54.6 \rightarrow 56.0$	$58.1 \rightarrow 59.2$
	C	$51.3 \rightarrow 52.6$	$53.6 \rightarrow 55.1$	$57.2 \rightarrow 59.2$
2Wiki	Q	$41.5 \rightarrow 48.1$	$40.8 \rightarrow 43.0$	$49.7 \rightarrow 53.7$
	C	$41.3 \rightarrow 47.7$	$40.7 \rightarrow 43.0$	$50.4 \rightarrow 54.3$
MuSiQue	Q	$17.5 \rightarrow 24.8$	$19.0 \rightarrow 28.4$	$17.8 \rightarrow 27.8$
	C	$16.2 \rightarrow 23.3$	$17.6 \rightarrow 26.7$	$17.2 \rightarrow 27.1$
Avg.	Q	$49.2 \rightarrow 52.5$	$50.7 \rightarrow 53.6$	$54.0 \rightarrow 58.5$
	C	$47.2 \rightarrow 50.8$	$48.7 \rightarrow 51.5$	$52.9 \rightarrow 57.7$

C.2 Results

All sets show worse performance compared to the memory. They are 1,2 to 1.4 % lower than our approach. In hand-written example set 2, our gains significantly come from 2Wiki ($+3.0$), whose questions are short and formulaic. A fixed set written in one dataset's style costs accuracy on a dataset with another; the memory, whose exemplars are selected by similarity to the question, does not. We read these findings: selection with the answers in hand does not help (Table 7). What the memory also brings to the table is that no one has to write those exemplars or know which kind to write, and that a mismatch between a fixed set and a dataset's question style, which costs a hand-written set up to three points on 2Wiki, does not arise.

Table 7: **Hand-written exemplars in place of the memory.** Each set replaces the retrieved episodes with four fixed exemplars shown to every fired question; everything else in the intervention is unchanged. GenAcc under the Qwen2.5-14B judge on the paired n=1,000 subsample of each dataset, the pipeline of Figure 2 (left); Δ is the pooled paired difference from the deployed memory.

Exemplar source	Size	HotpotQA	2Wiki	MuSiQue	Pooled	Δ (p)
SEARCH-MEM (ours), n_e=4	1,128	54.6	37.4	24.4	38.8	—
Hand-written set 1: generic	4	52.1	36.5	23.9	37.5	−1.3
Hand-written set 2: MuSiQue-style	4	53.6	34.4	24.8	37.6	−1.2
Hand-written set 3: terse	4	52.4	36.5	23.4	37.4	−1.4

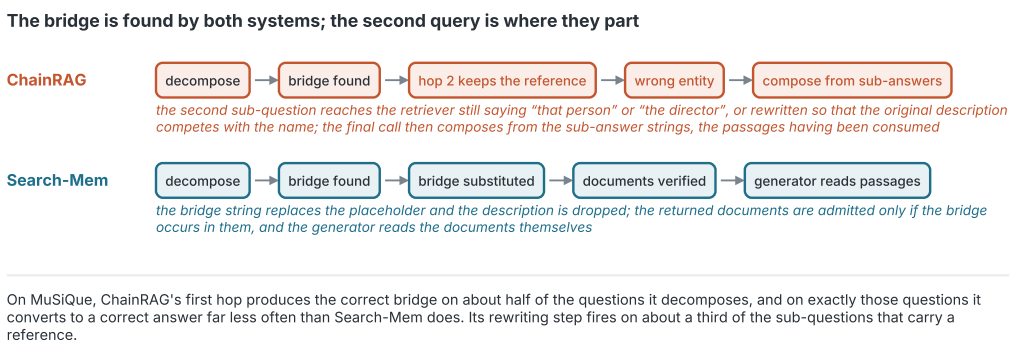


Figure 4: Where the two chains part. Both decompose and both resolve the bridge at the first hop; ChainRAG then issues a second query that still refers to the entity by description, and composes its answer from sub-answer strings, while SEARCH-MEM substitutes the bridge, verifies it in the returned documents, and lets the generator read the documents.

D QUALITATIVE COMPARISON WITH CHAINRAG

We provide qualitative comparison between SEARCH-MEM and ChainRAG (Zhu et al., 2025), which also utilizes the bridge extraction, to clearer illustrate the effectiveness of our method. ChainRAG is slightly losses to SEARCH-MEM on the single-hop sets and loses more where more than one hop is required, thus, we reckon **the gap localizes to how a resolved entity is carried from the first hop into the second** rather than to the retriever or the reader.

It is not that ChainRAG fails to find the bridge. On MuSiQue, where the intermediate answer is annotated, its first hop produces the correct bridge on about half of the questions it decomposes, and on exactly those questions it still converts to a correct answer far less often than we do. **What differs is the second query**. Its reference-resolution step fires on about a third of the sub-questions that carry a reference, so most reach the retriever with the reference intact, and when it does fire it restates the original description alongside or instead of the name.

Figure 4 places the two paths side by side and Figure 5 shows five questions in which both systems resolve the same bridge and only the second query differs: three where ChainRAG asks about "that person" or "the director", and two where the rewrite keeps the description that competes with the name. In each case its second retrieval lands on a different entity and the composed answer inherits the error. **Our repair substitutes the bridge string into the second sub-question, drops the description, and admits the returned documents only if the bridge occurs in them.** A second difference compounds the first: when ChainRAG decomposes, its final call composes an answer from the sub-question and sub-answer strings alone, the retrieved passages having been consumed by the intermediate steps, so anything lost in an intermediate extraction cannot be recovered, whereas **our generator always reads the passages themselves**.

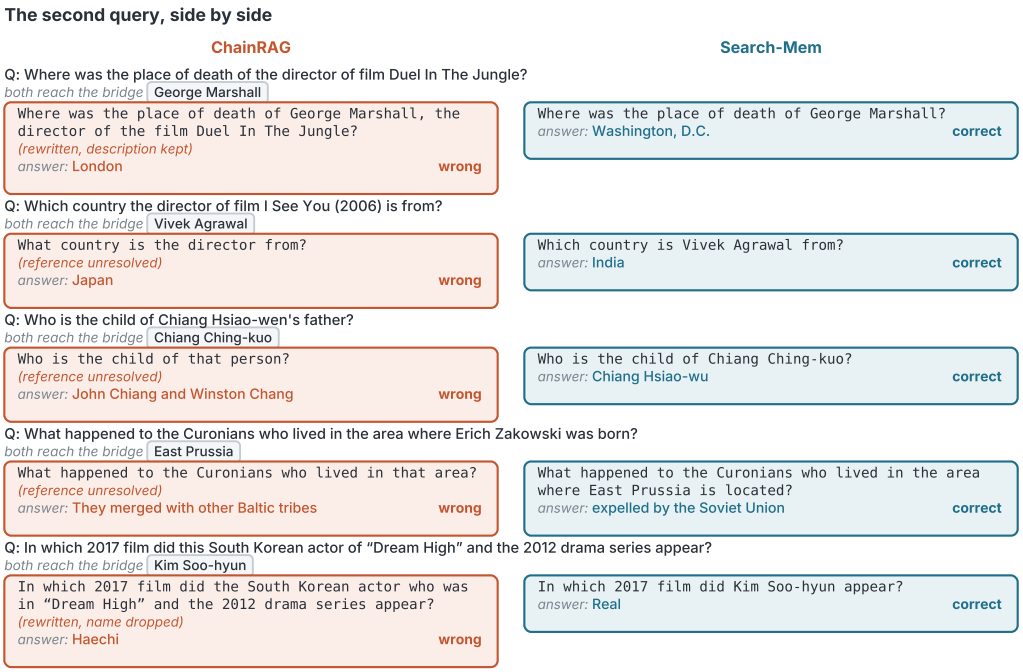


Figure 5: Five questions on which both systems resolve the same bridge entity and only the second query differs. ChainRAG's query is shown with the reason it fails, either an unresolved reference or a rewrite that keeps the description; SEARCH-MEM's is the same sub-question with the bridge substituted. Both columns are the logged runs behind Table 1, restricted to questions on which our gate fired and the bridge check admitted a bridge.

E DETAILED COST ANALYSIS

In this section we give a detailed cost analysis of our method against the baselines. Table 8 gives retrievals and model calls per question for every method of Table 1 on each dataset, with the Qwen2.5-14B generator, and Figure 6 plots the pooled values against accuracy.

Retrievals count every retriever call, including the initial one; calls count every model call, including the three votes of the answer stage, the compressor calls, and the repair calls. Four things stand out.

The two trained agents that reach a similar depth, s3 and SEARCH-MEM (WS) search to the same depth, 3.1 and 3.2 retrievals, so the planner adds nothing over s3's search; the compressor adds about 1.3 model calls per question, which is what buys the reduction in carried tokens in Figure 2 (middle). The memory adds about one retrieval and two calls per question on average, because the gate fires on between a third and a half of single-hop questions (31 to 45%) and on up to three quarters of multi-hop ones (38 to 77%), and each intervention uses one to three retrievals and four or five generator calls, at most $k+2$; without the gate every question would pay that, 4.0 to 4.9 extra calls each instead of the 1.3 to 3.8 the gate leaves. IRCoT retrieves as deeply as the trained agents, 3.3 retrievals against 3.1 for s3 and 3.2 for our training phase, and still trails them by ten points on 2Wiki; the multi-hop gap is in which queries the policy issues, not in how many. ChainRAG, at 2.4, trails on both counts. Search-R1 is the exception that proves the point about training data rather than about search: it reaches 29.4 on MuSiQue at 3.1 retrievals, above our 28.4, on forty times the training questions. Over the six datasets, even though SEARCH-MEM is the most expensive in Table 1, our method is also the most accurate with 53.6 average GenAcc. This could be understood as the tradeoff cost for test-time methods like ours, as we only used a fraction of training data compared to training-based baselines like Search-R1.

Table 8: Retrievals / model calls per question, Qwen2.5-14B generator, on the full test splits. A retrieval is one response from the retriever, including the initial retrieval on the original question, so a hop budget of T bounds a trajectory at $T + 1$; the deployed setting is $T = 3$. Model calls count every generation: the planner turns, the compressor calls, and the three answer-stage votes.

Method	NQ	TriviaQA	PopQA	HotpotQA	2Wiki	MuSiQue	Pooled
Direct Inference	0.0 / 1.0	0.0 / 1.0	0.0 / 1.0	0.0 / 1.0	0.0 / 1.0	0.0 / 1.0	0.0 / 1.0
CoT	0.0 / 1.0	0.0 / 1.0	0.0 / 1.0	0.0 / 1.0	0.0 / 1.0	0.0 / 1.0	0.0 / 1.0
IRCoT	2.7 / 4.7	2.6 / 4.6	2.9 / 4.9	3.4 / 5.4	4.1 / 6.1	4.2 / 6.2	3.3 / 5.3
ChainRAG	2.0 / 3.9	2.0 / 3.9	2.0 / 4.0	2.6 / 7.1	2.7 / 8.6	3.1 / 11.1	2.4 / 6.4
Search-R1	2.0 / 6.0	1.8 / 5.8	1.9 / 5.9	2.6 / 6.6	3.1 / 7.2	3.1 / 7.2	2.4 / 6.5
s3	3.1 / 7.0	3.1 / 7.0	3.1 / 7.0	3.1 / 7.0	3.2 / 7.0	3.2 / 7.0	3.1 / 7.0
SEARCH-MEM (WS)	3.2 / 8.4	3.2 / 8.4	3.2 / 8.3	3.2 / 8.2	3.2 / 8.4	3.2 / 8.2	3.2 / 8.3
SEARCH-MEM	4.1 / 10.2	3.8 / 9.7	3.9 / 9.9	4.1 / 10.0	4.5 / 10.9	5.2 / 12.0	4.3 / 10.4

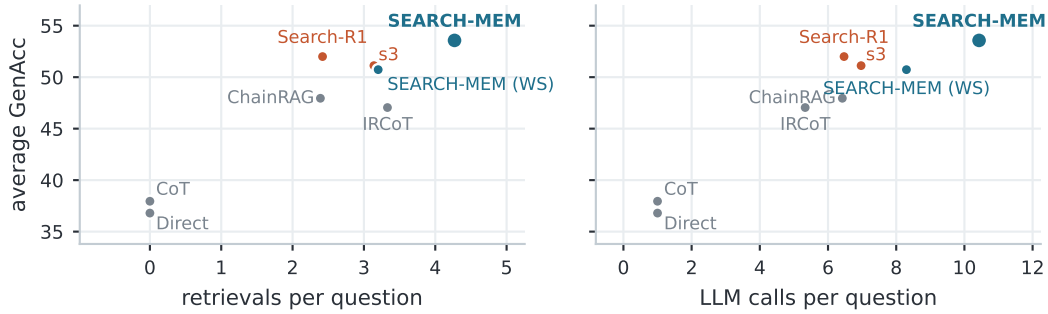


Figure 6: Cost against accuracy for every method in Table 1 with Qwen2.5-14B as the generator. The vertical axis is the average GenAcc over the six test splits under the Qwen judge; **Left:** retrievals per question; **Right:** model calls per question. Grey, training-free; orange, trained search agents; blue, ours.

F GENACC ALGORITHM

In this section, we present the algorithm to collect the generation accuracy in Algorithm 1.

Algorithm 1 Two-stage evaluation of generation accuracy (GENACC).

Require: Predicted answer string p; gold answer set $\mathcal{A} = \{a_1, \ldots, a_n\}$
Ensure: GENACC $\in \{0, 1\}$

$\tilde{p} \leftarrow$ NORMALIZE(p) ▷ lowercase, drop punctuation and articles
$\tilde{\mathcal{A}} \leftarrow \{$ NORMALIZE$(a) : a \in \mathcal{A} \}$
for $\tilde{a} \in \tilde{\mathcal{A}}$ **do** ▷ Stage 1: exact span lookup
    if $\tilde{a}$ appears as a contiguous token span of $\tilde{p}$ **then**
        return 1
    end if
end for
$v \leftarrow$ LLMJUDGE$(\tilde{p}, \tilde{\mathcal{A}})$ ▷ Stage 2: query "Is any gold answer expressed in $\tilde{p}$?"
if $v =$ YES **then**
    return 1
end if
return 0

G THE SEARCH-MEM PROCEDURE

Algorithms 2 to 6 give the method in the order it runs. Algorithm 2 is the search loop that every phase begins with. Algorithm 3 trains the planner and compressor. Algorithm 4 describes decomposition during memory construction, while Algorithm 6 includes the verification and fallback steps used at inference. Algorithm 5 writes the memory, and Algorithm 6 answers a test question with it.

Algorithm 2 SEARCH: initial retrieval and planner turns

Require: question q; initial retrieved documents D_0; planner π_p; compressor π_c; retriever R; search-turn budget $T_{\max} = 3$
Ensure: generator context $c(q)$

$s \leftarrow D_0$ ▷ initial documents are already in the prompt
$\mathcal{B} \leftarrow (D_0)$ ▷ retain the underlying document blocks
for $t = 1, \ldots, T_{\max} + 1$ **do**
  $u_t \sim \pi_p(\cdot \mid q, s)$ ▷ search is disabled on the final turn
  Record the document-selection marks in u_t
  Add the planner's output to s
  if u_t requests compression **then**
    Compress the requested blocks using π_c
    Replace their text in s with the resulting statements ▷ the documents in $\mathcal{B}$ remain unchanged
  end if
  if $t \leq T_{\max}$ and u_t contains a search query q_t **then**
    $D_t \leftarrow R(q_t)$
    Append D_t to $\mathcal{B}$ and s
  end if
  if u_t signals that the search is complete **then**
    break
  end if
end for
$c \leftarrow \varnothing$
for each document block in $\mathcal{B}$ **do**
  if the block has document-selection marks **then**
    Append its selected documents to c
  else
    Append all its documents to c
  end if
end for
return $c(q) \leftarrow c$

H MEMORY CONSTRUCTION DETAILS

The questions used for building the memory agent are the 3,000 two- to four-hop answerable questions from the MuSiQue training split; none appears in the training data or in any evaluation set. For each question the frozen generator is prompted once, with no examples and greedy decoding, to write a sub-question that finds an intermediate entity and a sub-question that uses it. The entity is read off the passages retrieved for the first sub-question, substituted into the second, and a second retrieval is made; episodes whose second sub-question has no placeholder, or whose entity fails an entity check or merely restates the question, are labelled as having no bridge. The generator then answers from the combined passages with the same three-sample vote used at deployment. The verifier admits an episode when (i) the intermediate entity, or the final answer for episodes without one, appears verbatim in at least one passage retrieved at the second hop, and (ii) at least two of the three sampled answers agree after normalization. Both tests are string operations; no model is called and no gold answer is read. Of 3,000 candidates, 1,128 are admitted (774 with a bridge, 354 without), of which 418 carry a correct answer.

Algorithm 3 Training phase: training the planner and compressor (Section 3.3)

Require: training questions with reference answers $(q, a^\star)$; frozen generator G; scorer $\mathcal{A}$; group size N_g; shaping weight β

cache $V(c_{\text{naive}})$ for every q, with $c_{\text{naive}} = R(q)$ ▷ single retrieval with the original question
for each training step **do**
  for each question q in the batch **do**
    for $i = 1, \ldots, N_g$ **do** ▷ a group of trajectories for the same question
      Run SEARCH(q), sampling from π_p and π_c
      $c^{(i)} \leftarrow$ context
      $\hat{a}^{(i)} \leftarrow G(q, c^{(i)})$ ▷ one greedy answer
      $V^{(i)} \leftarrow \mathcal{A}(\hat{a}^{(i)}, a^\star)$
      $r^{(i)}_{\text{shaping}} \leftarrow \mathbb{1}\big[\pi_c \text{ was called } \wedge\ \pi_p \text{ stopped on its own } \wedge\ V^{(i)} > V(c_{\text{naive}})\big]$
      $r^{(i)}_p \leftarrow \big(V^{(i)} - V(c_{\text{naive}})\big) + V^{(i)} + \beta\, r^{(i)}_{\text{shaping}}$ ▷ Equation 7
      $r^{(i)}_c \leftarrow r^{(i)}_p$ ▷ every compressor call in the trajectory inherits the return
    end for
  end for
  update π_p by GRPO, each trajectory's advantage taken relative to its group
  $b \leftarrow$ mean of $r^{(i)}_p$ over all trajectories in the batch
  update π_c by REINFORCE with return $r^{(i)}_c - b$ broadcast over the tokens of every call ▷ Equation 8
end for

Algorithm 4 DECOMPOSE: decomposition for memory construction (Section 3.4)

Require: question q; generator context c; example episodes $\mathcal{E}$ (may be empty); append cap m; vote size k

Ensure: sub-questions $q^{(1)}, q^{(2)}$; bridge b; second-hop documents D^+; k answers

$(q^{(1)}, q^{(2)}) \leftarrow G$ splits q, shown $\mathcal{E}$ as worked examples ▷ $q^{(2)}$ carries a placeholder for the bridge
$D^{(1)} \leftarrow R(q^{(1)})$
$b \leftarrow G$ names the answer to $q^{(1)}$ from $D^{(1)}$ ▷ the bridge entity
$\tilde{q}^{(2)} \leftarrow q^{(2)}$ ▷ preserve the original template
if $q^{(2)}$ contains a bridge placeholder **then**
  $\tilde{q}^{(2)} \leftarrow \text{Substitute}(q^{(2)}, b)$
end if
$D^+ \leftarrow R(\tilde{q}^{(2)})$
$c' \leftarrow c \oplus D^+_{1:m}$ ▷ additive: c is kept in full, at most m documents appended
$\hat{a}'^{(1)}, \ldots, \hat{a}'^{(k)} \sim G(q, c')$
return $q^{(1)}, q^{(2)}, b, D^+, \{\hat{a}'^{(i)}\}_{i=1}^k$

I BASELINES DETAILS

What is held fixed. Every row of Table 1 is scored on the same questions, retrieved from the same index, and answered by the same frozen generator within a panel; the only thing that varies is how the context reaching that generator is built. The retriever is E5 (`intfloat/e5-base-v2`) (Wang et al., 2024) over a flat index of the 2018 English Wikipedia (Karpukhin et al., 2020), served as a single endpoint that every method queries with its own top-k.

SEARCH-MEM. Table 9 lists the deployed configuration in full. The planner is Qwen2.5-7B-Instruct after 20 GRPO steps and the compressor Qwen2.5-0.5B-Instruct after 20 REINFORCE steps, both from a single training run at seed 42. At inference the planner searches under a hop budget of three, requesting $n_d = 8$ documents per query, decoded at temperature 0.6 with one trajectory per question. The generator then samples $k = 3$ answers over the curated context at temperatures 0, 0.7 and 0.7; the gate fires when they disagree. On a firing, the four nearest episodes

Algorithm 5 Writing the memory (Section 3.4)

Require: multi-hop training questions Q_{mem}; trained π_p, π_c (fixed); G; R
Ensure: memory $\mathcal{M}$

$\mathcal{M} \leftarrow \varnothing$ ▷ empty while it is written, so every split is elicited zero-shot
for each $q \in Q_{\text{mem}}$ **do**
  $c \leftarrow \textsc{Search}(q)$ ▷ exactly as at inference, no gradients
  $(q^{(1)}, q^{(2)}, b, D^+, \{\hat{a}'^{(i)}\}) \leftarrow \textsc{Decompose}(q, c, \varnothing)$
  $\tau \leftarrow \mathbb{1}\big[q^{(2)}$ carried a placeholder and b passed the entity check$\big]$ ▷ shape of the split
  if $\tau = 1$ **then**
    grounded $\leftarrow$ bridge entity b appears in D^+
  else
    grounded $\leftarrow$ a non-empty majority answer exists and appears in D^+
  end if
  stable $\leftarrow \mathbb{1}\big[$a majority of $\{\hat{a}'^{(i)}\}$ agree$\big]$ ▷ check 2: the split led to a determinate answer
  if grounded $\wedge$ stable **then**
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{\langle q, q^{(1)}, q^{(2)}, \tau\rangle\}$ ▷ b, D^+, and the answers are discarded
  end if
end for
index every stored q with a frozen sentence encoder
return $\mathcal{M}$

are selected from the memory by mean-pooled MiniLM question embeddings under a leave-one-out guard, three of the bridge kind and one without; the decomposer writes two sub-questions; the first is retrieved at top-5 and the bridge entity extracted; the resolved second question is retrieved at $m = 4$ of its documents are appended to the generator's context, which is then re-read with the same three-sample vote. The memory is written once, before evaluation, from 3,000 MuSiQue training questions that appear in no test split.

We note that we use a group of two rollouts per question, the smallest group under which GRPO has a within-question contrast; with 60 questions per step this is 120 trajectories of up to 8k tokens with compressor calls, which is what fits the 7B actor and its optimizer state on our node setting. With a binary reward the standardized advantage is $\pm\frac{1}{\sqrt{2}}$ or 0. We verified that this is not what limits the training phase: an estimator that injects gradient into tied groups raised the gradient norm by 36% without changing accuracy, tied rollouts are near-duplicates, and the planner before any RL step reaches the accuracy of the training phase

Direct Inference and CoT. Both are the frozen generator alone, with no retrieval, using s3's verbatim prompts. Direct carries the instruction that the model must answer directly without any other text or thinking; CoT uses their step-by-step prompt and takes the contents of the answer tags, falling back to the full response as their code does. One call per question at temperature 0 with at most 500 new tokens. These two rows measure what the generator already knows.

IRCoT. (Trivedi et al., 2023) We run the s3 authors' own implementation with their configuration rather than a reimplementation: the reasoning model is Qwen2.5-7B-Instruct at temperature 0.7, top-p 0.95 and 2,048 new tokens, retrieval is top-3 per step, and the loop performs an initial retrieval followed by at most four reasoning-and-retrieval steps. The documents it collects, at most fifteen, are passed to the panel's answer stage untruncated.

ChainRAG. (Zhu et al., 2025) A port of the published algorithm: two-stage decomposition, a sentence graph whose entity edges come from spaCy NER and are kept above the 40th percentile of BM25 importance, similarity edges at top-10 and adjacency edges at ± 3, seed retrieval taking the top-7 of 100 candidates, an LLM sufficiency check, one to three hops of graph expansion under a 3,000-word cap, a forced answer, and their AnsInt final step. Its prompts are verbatim from the authors' repository where published. Four settings differ from the paper because our setting is open-corpus: the per-question pool is retrieved with our E5 index rather than taken from provided contexts, ranking is by embedding similarity rather than a cross-encoder reranker, and the LLM is our frozen Qwen2.5-14B at temperature 0.2.

Algorithm 6 Inference

Require: question q; trained search agents; generator G; retriever R; memory $\mathcal{M}$; $k = 3$; $m = 4$
Ensure: final answer

$c(q) \leftarrow \textsc{Search}(q)$
Sample $\hat{a}^{(1)}, \ldots, \hat{a}^{(k)}$ from $G(q, c(q))$
if all normalized answers agree and are non-empty **then**
    return $\hat{a}^{(1)}$
end if
Retrieve three bridge examples and one no-bridge example from $\mathcal{M}$ by question similarity
Retrieve five documents using the original question q
$(q^{(1)}, q^{(2)}) \leftarrow G$ decomposes q using the retrieved examples
$D^{+} \leftarrow \varnothing$
if $q^{(2)}$ contains a bridge placeholder **then**
    Retrieve five documents for $q^{(1)}$
    $b \leftarrow G$ extracts the bridge from these documents
    if b passes the entity check, does not appear in q, and the resolved query passes the redundancy check **then**
        $\tilde{q}^{(2)} \leftarrow q^{(2)}$ with the placeholder replaced by b
        Retrieve five documents for $\tilde{q}^{(2)}$
        if at least one returned document contains b **then**
            $D^{+} \leftarrow$ the first four returned documents
        end if
    end if
end if
Place D^{+} before the original-question retrieval results
Deduplicate this list and retain at most m documents
$c' \leftarrow c(q)$ with these documents appended
Sample $\hat{a}'^{(1)}, \ldots, \hat{a}'^{(k)}$ from $G(q, c')$
if the initial majority answer is `yes` or `no` **then**
    return the initial majority answer
else
    return the majority answer from the new samples
end if

Search-R1. (Jin et al., 2025) The released checkpoint: PeterJinGo/SearchR1-nq_hotpotqa_train-qwen2.5-7b-em-ppo, run in its own rollout harness with top-8 retrieval, at most four turns, a 6,144-token prompt budget, 500 new tokens and one rollout per question. The documents its searches return are then read by each panel's answer stage, exactly as ours are.

MEM1. (Zhou et al., 2025) The released checkpoint: Mem-Lab/Qwen2.5-7B-RL-RAG-Q2-EM-Release, ported from the authors' inference code: their single-question prompt verbatim, a constant-size context holding the prompt, the previous response and the last information block, at most six turns with their turn-count hints, top-3 retrieval, and decoding at temperature 0.01, top-p 0.95 and 1,024 new tokens, stopping on the search and answer tags. Like Search-R1 it is reported as a search planner, with each panel's generator answering from the documents its searches returned. Three settings are forced by our stack: our E5 retriever over the same corpus in place of theirs, our question order so that every row scores identical questions, and our scoring in place of its in-loop substring match.

s3. (Jiang et al., 2025) The released checkpoint: pat-jj/s3-8-3-3-20steps run through the same rollout harness as our own planner: top-8 retrieval under a hop budget of three.

Rows whose generator is fine-tuned. SFT, R1 and Search-R1-FT are trained and reported. Their generators are trained rather than frozen, so they answer with their own stage and are separated from the three frozen-generator panels by the side bar in Table 1.

Table 9: Configuration of the deployed system.

Agents	
Planner π_p	Qwen2.5-7B-Instruct (Qwen Team, 2024), trained
Compressor π_c	Qwen2.5-0.5B-Instruct (Qwen Team, 2024), trained
Retriever R	E5 (Wang et al., 2024) over the 2018 English Wikipedia (Karpukhin et al., 2020)
Generator G	Qwen2.5-7B-Instruct (Qwen Team, 2024), Qwen2.5-14B-Instruct (Qwen Team, 2024), or Claude-4.5-Haiku (Anthropic, 2025), frozen
Episode encoder	all-MiniLM-L6-v2 (Reimers & Gurevych, 2019)
Training Phase	
Training questions	NQ and HotpotQA training splits, yes/no and true/false questions removed; 1.2k questions
Optimizer	GRPO for π_p, group size 2; REINFORCE for π_c with the batch-mean baseline
Steps, batch	20 steps of 60 questions; checkpoint at step 20; seed 42
Learning rate, KL	$\pi_p = 10^{-6}$, $\pi_c = 10^{-5}$, KL coefficient = 0.001
Shaping weight	$\beta = 0.3$
Reward scorer	GenAcc with the Qwen2.5-14B-Instruct judge in 4-bit precision, one greedy sample of G
Naive-RAG reference	Top-3 documents retrieved using the original question; the reference score is computed once per training question and cached.
Hop budget, window	3 hops; 8k-token planner window
Memory writing	
Questions replayed	3,000 two- to four-hop questions from the MuSiQue training split, disjoint from every test set
Split prompt	zero-shot, greedy decoding
Verifier	bridge entity appears as a case-insensitive substring; at least 2 of 3 sampled answers agree after normalization
Memory	1,128 episodes admitted of 3,000 (774 with a bridge, 354 without); written once.
Inference	
Answer samples	$k = 3$, temperatures 0, 0.7, 0.7; majority vote; gate fires if any answer is empty or the normalized answers disagree
Episodes shown	$n_e = 4$ nearest by question embedding, three with a bridge and one without
Documents appended	$m = 4$ from the second-hop retrieval, appended to $c(q)$
Initial retrieval	Top-12 during training; top-3 at inference, using the original question.
Planner retrieval	Top-8 for each query issued during the search loop.
Evaluation	
Test splits	NQ 3,610; TriviaQA 11,299; PopQA 13,052; HotpotQA 7,400; 2Wiki 12,550; MuSiQue 2,395; MultiHop-RAG 2,255 (excluding questions without answer in the corpus)
Judges	Qwen2.5-14B-Instruct at full precision; Claude-4.5-Haiku; span check first, judge only where it fails

J JUDGE DETAILS

J.1 JUDGE INFORMATION

GenAcc is computed by Algorithm 1: the span check first, and a judge only where it fails, with the prompt "Please check if any of the golden answers is contained in the following response" followed by the response and the golden answers, answered yes or no. The open judge is Qwen2.5-14B-Instruct (Qwen Team, 2024) at full precision and the proprietary judge is Claude-Haiku (Anthropic, 2025), version 20251001_v1; during training the reward uses the same rule with the open judge. Because a judge is consulted only where the span check fails, it sees about half of the questions on the single-hop datasets and about two thirds on the multi-hop ones, and it flips a few per cent of them. Each judge evaluates outputs from every generator, including its own model family. The two judges agree on the best-performing method in each frozen-generator dataset setting in Table 1, but

some lower-ranked methods change order. Because both evaluations also share the span-matching stage, this agreement does not rule out scoring bias.

J.2 Relation to The Numbers Reported For s3 Baseline

In this section, we give explanations why our reproduced s3 row is lower than the values Jiang et al. (2025) report for the same generators. Several differences of protocol account for this, and each applies to every row of Table 1 alike, so comparisons within our tables are unaffected while absolute values are not comparable across the two papers.

J.2.1 The judge version and precision

s3 paper scores with **Claude-3-Haiku**; we score with Qwen2.5-14B-Instruct at full precision and with **Claude-4.5-Haiku, version 20251001-v1**, which are different.

J.2.2 The generator version

Again, we note their Claude panel is **Claude-3-Haiku**, ours is the **Claude-4.5-Haiku**, which under s3's own answer prompt declines to answer when it finds the context insufficient.

K Illustrative Examples

In this section, we provide examples that shows the effectiveness of Search-Mem. We show examples from HotpotQA and MuSiQue datasets, with detailed analysis on each step of our method.

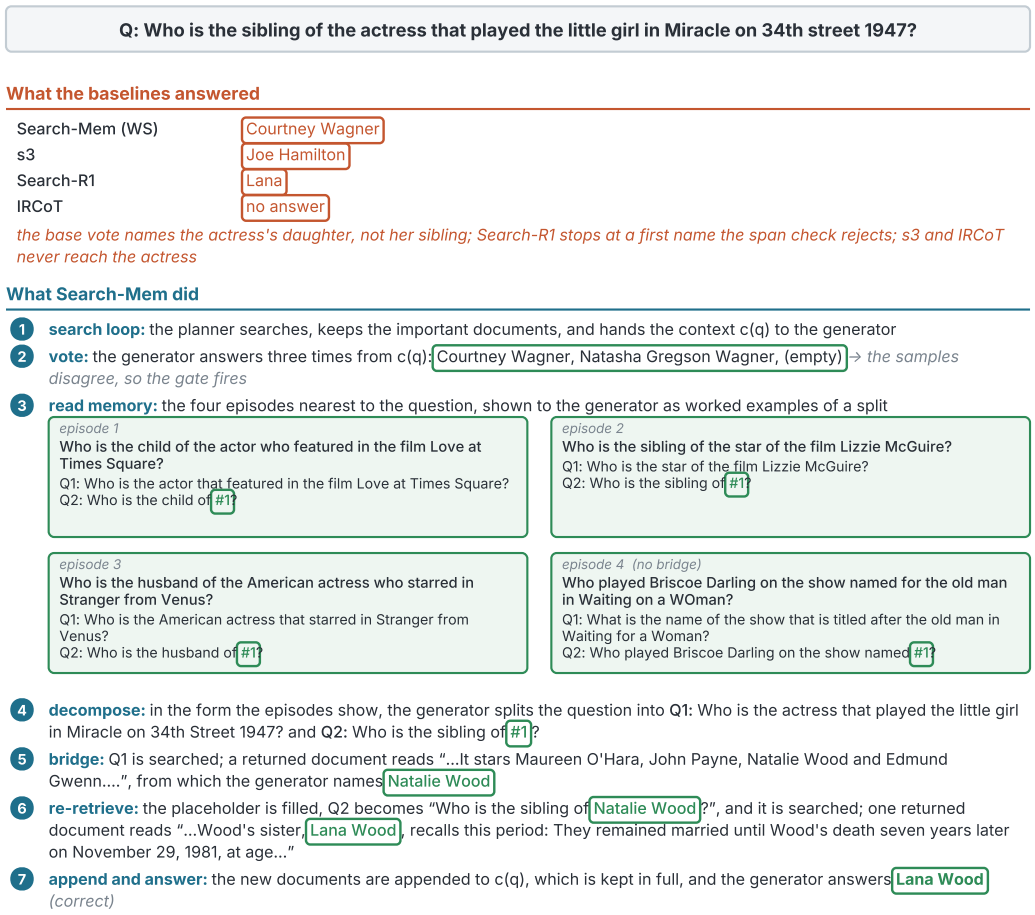


Figure 7: MuSiQue. The base vote names the actress's daughter rather than her sibling, and the samples disagree; the memory's episodes are all of the "relative of the person in film X" form, the split resolves the actress, and the second search reaches her sister.

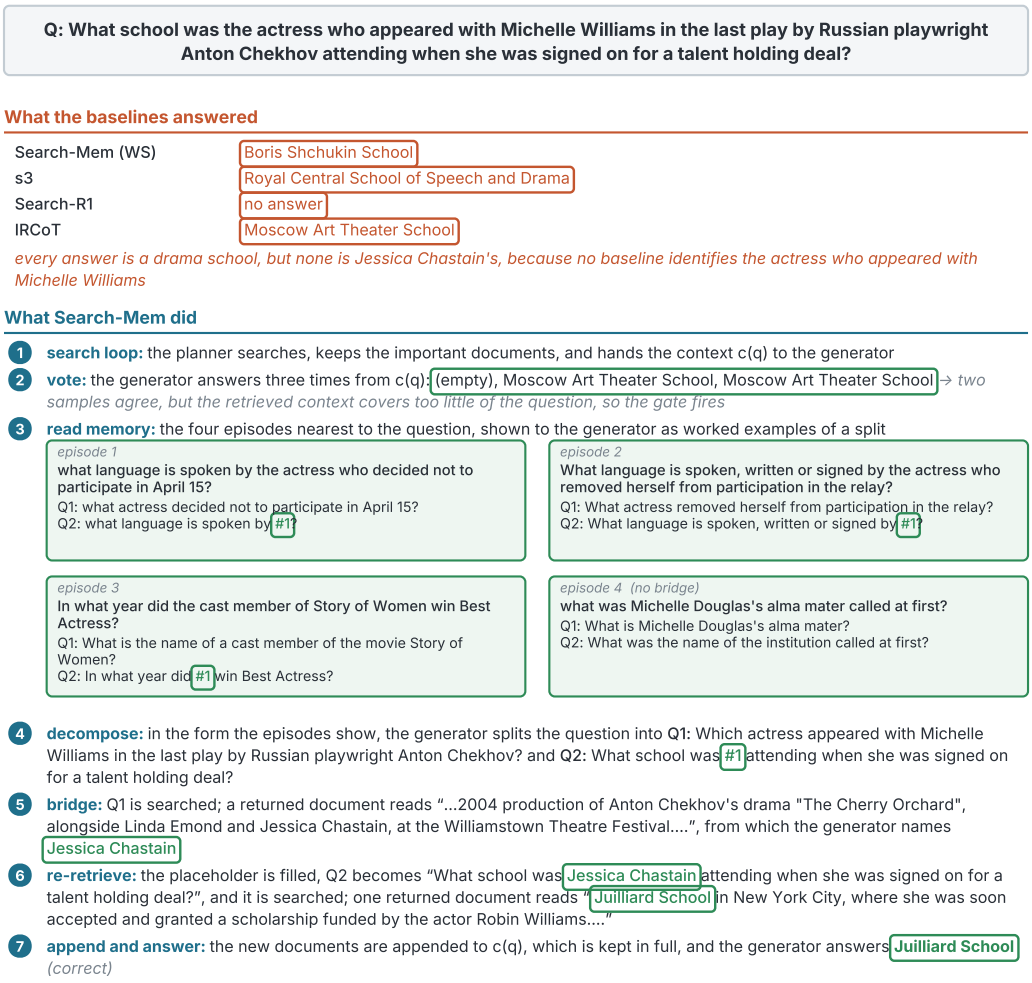


Figure 8: HotpotQA. Every baseline answers with a drama school, none of them Jessica Chastain's, because none identifies the actress; two of our samples agree on a wrong school and the gate fires on the coverage signal, after which the split resolves the actress and the second search reaches her school.

Q: Who produced the 1998 British comedy starring the director of the feature film "Eddie the Eagle"?

What the baselines answered

Search-Mem (WS)	no answer
s3	Geoff Posner
Search-R1	the BBC
IRCoT	Jim Abrahams

none resolves the director, Dexter Fletcher, so his 1998 film is never searched; the answers are unrelated producers or nothing

What Search-Mem did

1. **search loop:** the planner searches, keeps the important documents, and hands the context c(q) to the generator
2. **vote:** the generator answers three times from c(q): (empty), (empty), (empty) → *all three samples are empty, so the gate fires*
3. **read memory:** the four episodes nearest to the question, shown to the generator as worked examples of a split

episode 1
Where was the opening scene of the show named for the feature actor in Onionhead filmed?
Q1: Who is the feature actor in the show named Onionhead?
Q2: Where was the opening scene of the show #1 filmed?

episode 2
Who is the father of the person who directed Unhook the Stars?
Q1: Who directed the film Unhook the Stars?
Q2: Who is the father of #1?

episode 3
Who's the father of the director of From the Sky Down?
Q1: Who is the director of the film "From the Sky Down"?
Q2: Who is the father of #1?

episode 4 (no bridge)
Who played Floyd the Barber in the show named for the actor featured in Onionhead?
Q1: What is the name of the show that features the actor from Onionhead?
Q2: Who played Floyd the Barber in the show named #1?

4. **decompose:** in the form the episodes show, the generator splits the question into **Q1:** Who is the director of the feature film "Eddie the Eagle"? and **Q2:** Who produced the 1998 British comedy starring #1?
5. **bridge:** Q1 is searched; a returned document reads "...Eagle (film) Eddie the Eagle is a 2016 biographical sports comedy-drama film directed by Dexter Fletcher....", from which the generator names Dexter Fletcher
6. **re-retrieve:** the placeholder is filled, Q2 becomes "Who produced the 1998 British comedy starring Dexter Fletcher?", and it is searched; one returned document reads "...1998 British-American crime comedy film written and directed by Guy Ritchie, produced by Matthew Vaughn and starring an ensemble cast featuring Jason Flemyng, Dexter Fletcher, Nick Moran, Jason Statham, Steven Mac..."
7. **append and answer:** the new documents are appended to c(q), which is kept in full, and the generator answers **Matthew Vaughn** *(correct)*

Figure 9: HotpotQA. The baselines name unrelated producers or nothing, because none resolves the director; all three of our samples are empty and the gate fires, the split resolves Dexter Fletcher, and the second search reaches the 1998 film.

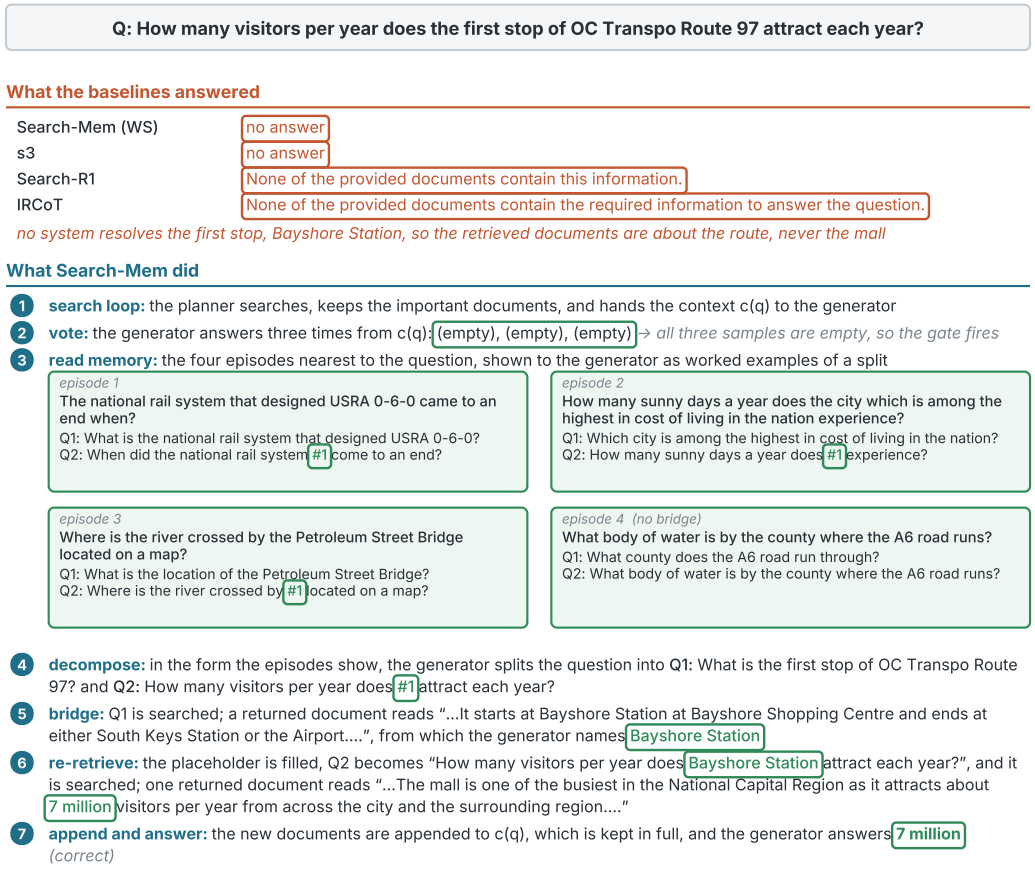


Figure 10: HotpotQA. No system resolves the first stop of the route, so the retrieved documents describe the route and never the mall; all three of our samples are empty, the split resolves Bayshore Station, and the second search reaches the visitor count.