LIBRARIAN AGENTS: ANTICIPATORY FILESYSTEM CONSTRUCTION VIA REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Reasoning over long document contexts remains a challenge for large language models (LLMs). Standard retrieval methods are inherently reactive, bottlenecking agents with complex, cross-document reasoning at inference time. Existing approaches that pre-compute corpus structure, such as knowledge graphs, often fall short due to their reliance on rigid heuristics such as identifying interdocument relationships via named entity recognition. We propose an *anticipatory* approach to memory: proactively reorganizing unstructured document corpora to efficiently serve diverse future queries. This pre-computation synthesizes and restructures information within documents in advance, significantly improving an answering agent's downstream performance. While agentic scaffolds (e.g., Claude Code) are heavily trained to navigate and extend filesystems while solving the task currently in front of them, they do not pre-construct one in anticipation of future queries that our approach aims to do. We introduce `Librarian Agents`, a novel framework that restructures unstructured corpora into file-based memories arranged around the operations that LLM agents already perform well, such as reading summaries and following links. `Librarian Agents` is trained to maximize downstream success rate via end-to-end reinforcement learning. Empirically, `Librarian Agents` improves accuracy over the best performing baseline by 33 points on BrowseComp+ and additionally generalizes to out-of-distribution corpora such as BRIGHT with a 31 point increase on average.

1 INTRODUCTION

Large language models (LLMs) are increasingly tasked with reasoning over extensive corpora of documents, such as enterprise knowledge bases, complex software repositories, and user chat histories, that exceed today's LLM context windows. In these domains, the information required to complete a given task is rarely centralized; instead, it is dispersed and disconnected across many disparate sources. Thus, for each incoming request, agents repeatedly process and revisit the corpus from scratch. Can we instead enable systems to efficiently navigate these massive context spaces without incurring high, redundant computational costs each time a user poses a new query?

A standard approach to this challenge is retrieval-augmented generation (RAG) (Gao et al., 2023), in which a system responds to a query by searching raw documents, retrieving relevant passages, and synthesizing an answer. Recent work has strengthened this pipeline with multi-agent frameworks (Jiang et al., 2023; Asai et al., 2023; Yan et al., 2024; Edge et al., 2024; Singh et al., 2025). However, these methods remain largely reactive: each query triggers a new round of search, requiring the model to repeatedly rediscover useful cross-document relationships at inference time. Other approaches pre-structure the search space before inference, for example as a knowledge graph (Peng et al., 2025). These methods can improve retrieval, but they typically commit to a rigid set of abstractions involving entities, relationships, and communities (Keraghel et al., 2024). Such a structure is useful when future queries precisely align with the index, but it is less natural when queries require new task-specific intermediate artifacts. This raises a critical question: can we train an agent to *reorganize a corpus in advance* so future LLM agents can efficiently find, inspect, and verify the right evidence?

Motivated by approaches that shift computation across different aspects of the model training and deployment process (Lin et al., 2025; Wu et al., 2026; Suzgun et al., 2025; Zhang et al., 2025a), we study *anticipatory memory construction*: the problem of augmenting and reorganizing a corpus *before* future queries arrive. The goal is to shift part of the search, synthesis, and organization

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.

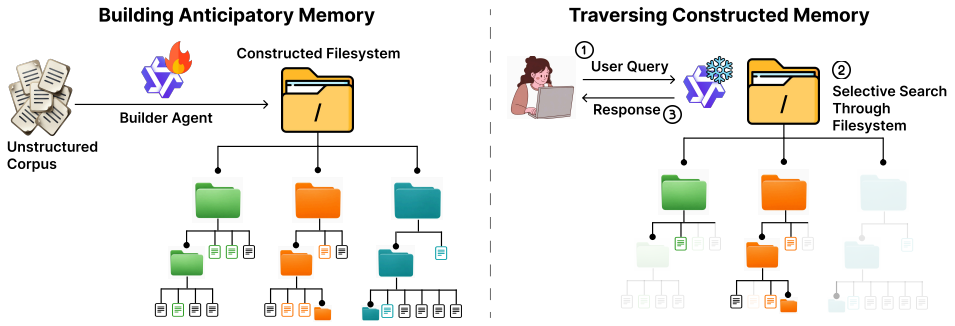


Figure 1: **Overview of `Librarian Agents`.** (Left) We train a builder agent to construct an augmented filesystem consisting of synthetic files and directories that *anticipates* future queries for a downstream answering agent. (Right) This filesystem is built once and repeatedly used to answer different queries.

costs from query time to a precomputation phase, thereby amortizing these costs across many downstream queries. To instantiate this setting, we introduce `Librarian Agents`, a framework built on a key asymmetry in current agentic systems: LLM agents are effective at navigating existing hierarchical filesystems, but are rarely trained to *proactively construct such workspaces for future use*. `Librarian Agents` addresses this gap by tasking a **builder agent** with transforming source documents into an augmented, multi-level filesystem with provenance to the original corpus. An **answering agent** then navigates this filesystem to answer an individual user query. We formalize the builder's operations as a Markov Decision Process (MDP) and directly optimize the memory construction policy for downstream answering performance by posing it as a reinforcement learning (RL) problem on this MDP.

We demonstrate the efficacy of `Librarian Agents` on the BrowseComp-Plus domain, where compared to the strongest baseline for each answering agent, `Librarian Agents` improves accuracy by an average of 33 percentage points over the best performing baseline. Furthermore, for BRIGHT, a domain which our builder agent is untrained on, gains persist over baselines with an average 31 point improvement over baselines.

In summary, our key contributions are:

- We formulate memory anticipation as a desirable model behavior: given long corpora, agents should materialize useful context *before* receiving user queries.
- We propose `Librarian Agents`, a filesystem-based framework, where a builder agent synthesizes and organizes memory artifacts.
- We formulate filesystem-based memory construction as an MDP and optimize the builder agent via reinforcement learning to improve downstream query-answering performance.
- We evaluate `Librarian Agents` against retrieval and pre-computation baselines on BrowseComp-Plus and test zero-shot transfer on question-answering tasks derived from BRIGHT.

2 LIBRARIAN AGENTS: AGENT-AUGMENTED FILESYSTEMS AS ANTICIPATORY MEMORIES

We introduce `Librarian Agents`, a framework for training an agent to construct *anticipatory memories*: corpus representations built before future queries arrive. Document-grounded agents waste compute when each new query forces them to rediscover the same information across the raw corpus. Anticipatory memory shifts part of this work into a reusable construction phase. We instantiate this memory as an augmented filesystem, which exposes a long-context structure through operations LLM agents already use well: scanning filenames, opening concise artifacts, traversing directories, and checking claims against source files. The builder converts a flat collection of documents into a multi-level workspace whose synthetic files reduce search friction while preserving fallback paths to the original corpus. We refer to this builder as a **Librarian Model**: an agent trained to organize a corpus for future use by an answering agent.

We first define the document corpus, filesystem, and augmentation operations that transform a flat collection of raw documents into an answer-facing memory structure. We then articulate desiderata of an effective anticipatory memory, which includes coverage with efficient navigation, fallback to raw evidence, and reuse across queries. Finally, we cast memory construction as a finite-horizon Markov decision process in which the librarian model edits the filesystem to maximize the answering agent's expected performance, subject to a construction-cost penalty.

2.1 Problem Setup.

We study long-context reasoning over a corpus of raw documents, $\mathcal{D} = \{d_1, d_2, \ldots, d_n\}$, where the system must answer queries $q \sim \mathcal{Q}$. This setting matches retrieval-augmented generation (RAG) systems, which typically answer each query by searching a retrieval index over $\mathcal{D}$ and then synthesizing a response.

We represent the agent's external memory as a *filesystem* $\mathcal{F}$: a set of files and directories together with metadata specifying each node's type and parent. The initial filesystem $\mathcal{F}_0$ contains the raw documents in $\mathcal{D}$ as files under the root. A **filesystem augmentation** $\mathcal{A}$ is an edit operation that modifies a filesystem, for example by creating, rewriting, deleting, or moving files and folders. At iteration i, an augmentation $\mathcal{A}_i$ transforms the previous filesystem $\mathcal{F}_{i-1}$ into $\mathcal{F}_i$. Applying a sequence of augmentations for horizon H produces a final memory $\mathcal{F}_H$.

System performance is measured by a scoring function $\mathrm{Acc}(\tilde{y}, y^*)$, which evaluates the correctness of the generated response $\tilde{y}$ against a ground-truth target y^*. Memory construction may also incur a cost, denoted $\mathrm{Cost}(A_i)$, such as the number of tokens generated or modified by $\mathcal{A}_i$.

Overview of Problem Setup

Anticipatory memory construction is the problem of transforming a large corpus into a structured filesystem memory before downstream queries are observed.

2.2 Desiderata for Anticipatory Memories.

A useful anticipatory memory should do more than "remember" a corpus. It should reshape the corpus into an *answer-facing workspace* that helps future agents quickly find likely evidence, while preserving a path back to raw source documents. As a running example, consider an enterprise corpus containing design docs, issue threads, launch notes, and customer reports for the same product. One future query may ask for a low-level implementation detail; another may ask how a user-visible behavior changed across releases; a third may ask for the exact source that supports a disputed claim. We use these examples to motivate three key desiderata.

Coverage with Efficient Navigation. The memory should cover many plausible queries while reducing the need to search the raw corpus. In the enterprise example, the builder might create a release timeline, a component-level design index, and customer-facing behavior summaries, each linking the same underlying documents from a different angle. These artifacts help only if the answering agent can reach them through short, legible paths, such as directories named by abstraction level. Thus, the builder should synthesize cross-document artifacts that cover likely regions of the query distribution $\mathcal{Q}$ and place them where an agent can discover them with shallow traversal.

Faithful Fallback. No finite memory can anticipate *every* query, and synthesized artifacts can omit details that are critical in hindsight. The filesystem must therefore preserve provenance links and raw source paths so the answering agent can verify claims or bypass the synthesized layer when needed. In the enterprise example, a rare query about a single customer may not appear in the high-level summary, but the agent should still be able to read any existing issue thread. As an extreme example, such a fallback is important for needle-in-a-haystack queries where the answer depends on a localized, very specific fact that is unlikely to survive compression (Wang et al., 2024).

Amortized Reuse. The builder's construction cost should pay off across many downstream queries. For example, a compact release timeline written by the builder agent may help answer many questions about regressions, roadmap changes, customer messaging, and dependency updates. In contrast, a one-off answer draft may surface similar information but help only a single request. This creates a tradeoff between the construction cost $\sum_{i=1}^{H} \mathrm{Cost}(\mathcal{A}_i)$ and the downstream utility.

Overview of Memory Desiderata

An effective anticipatory memory should make likely queries easy to answer, preserve fallback paths to raw documents, and pay off across many downstream queries.

2.3 MEMORY MDP FORMALIZATION.

In `Librarian Agents`, anticipatory memory construction proceeds in two phases: an upfront synthesis phase and a downstream answering phase. The synthesis phase is analogous to the offline construction of a retrieval index or knowledge graph, but instead produces a flexible filesystem memory. To instantiate this idea, we define a **builder agent** $\pi_{\text{build}}(\mathcal{A} \mid \mathcal{F})$, which proposes filesystem augmentations given the current filesystem, and an **answering agent** $\pi_{\text{ans}}(\tilde{y} \mid q, \mathcal{F})$, which generates an answer $\tilde{y}$ for a new query q using the constructed filesystem.

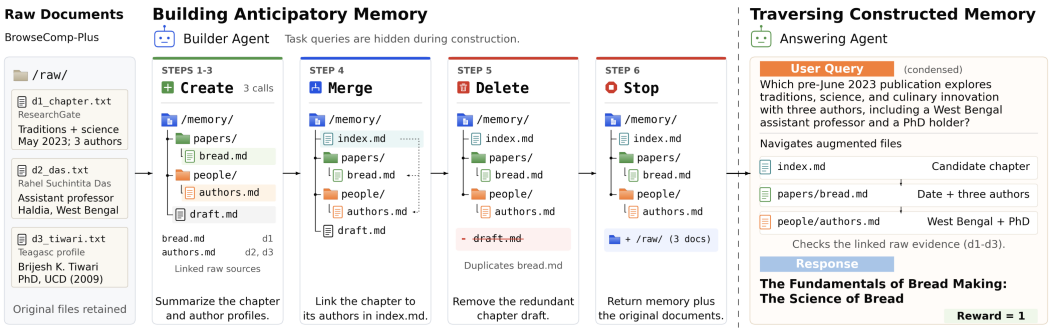


Figure 2: **MDP Definition of `Librarian Agents`.** Starting from a raw list of documents, a builder agent can create a filesystem with four actions: `Create`, `Merge`, `Delete`, and `Stop`. An answering agent navigates the resulting filesystem through file-listing and file-reading actions, produces a response, and receives a task-specific correctness score. Here, we depict an augmented filesystem from the BrowseComp-Plus dataset.

We cast the builder agent's sequence of filesystem operations as a finite-horizon Markov Decision Process (MDP), allowing memory construction to be optimized as a policy-learning problem.

State, Action, and Transition Dynamics. At each timestep t, the state is the current filesystem $\mathcal{F}_t$, consisting of the original raw documents $\mathcal{D}$ together with the synthetic files, folders, and provenance links created so far. The builder agent selects an augmentation $\mathcal{A}_t \sim \pi_{\text{build}}(\cdot \mid \mathcal{F}_t)$, which is applied to the filesystem to produce $\mathcal{F}_{t+1}$. We constrain the builder to the following discrete action types that can each be applied to different files:

- CREATEFILE: Creates a new synthetic file by consolidating information from a subset of raw documents and/or existing synthetic files. The file records provenance links to the sources used to construct it.
- MERGEFILES: Aggregates a set of related synthetic files into a higher-level summary or directory-like artifact. This action introduces hierarchy, linking to child files.
- DELETE: Removes redundant/unhelpful synthetic files from the active answer-facing filesystem. This operation never deletes raw documents in $\mathcal{D}$, preserving fallback access.
- STOP: Terminates memory construction, returning the current filesystem $\mathcal{F}_t$ as the final memory artifact.

The action types are intentionally constrained to simplify optimization. In principle, this interface can be generalized to a broader set of atomic filesystem commands (e.g., `touch`, `mkdir`, `mv`, `rm`, etc). We describe the instantiation of these actions in the filesystem in Appendix A.8.

Reward Function. The reward should incentivize the builder to construct a memory artifact that improves both downstream answer accuracy and navigation efficiency over the query distribution $\mathcal{Q}$. We first define the expected answering success of a filesystem $\mathcal{F}$ as

$$S(\mathcal{F}) := \mathbb{E}_{q\sim\mathcal{Q}}\,\mathbb{E}_{\tilde{y}\sim\pi_{\text{ans}}(\cdot|q,\mathcal{F})}\big[\text{Acc}\big(\tilde{y}, y^*(q)\big)\big],$$

where $y^*(q)$ is the ground-truth answer to query q and $\mathrm{Acc}(\tilde{y}, y^*(q)) \in \{0, 1\}$ indicates whether the answer $\tilde{y}$ is correct. To account for navigation efficiency, we define the downstream utility

$$U(\mathcal{F}) \coloneqq S(\mathcal{F}) - \lambda_{\mathrm{tok}}\, C_{\mathrm{tok}}(\mathcal{F}) - \lambda_{\mathrm{read}}\, C_{\mathrm{read}}(\mathcal{F}), \tag{1}$$

where C_{tok} and C_{read} denote the expected normalized synthetic-memory token cost and file-read cost, respectively. Before averaging over queries and answering rollouts, each rollout's cost is multiplied by its correctness indicator. Navigation penalties therefore apply only to correct answers: an incorrect answer receives zero utility regardless of its navigation cost. Setting $\lambda_{\mathrm{tok}} = \lambda_{\mathrm{read}} = 0$ recovers the accuracy-only utility $U(\mathcal{F}) = S(\mathcal{F})$.

The sparse construction objective scores the final filesystem while penalizing the total cost of building it:

$$r^{\mathrm{sparse}} \coloneqq U(\mathcal{F}_H) - \alpha \sum_{t=1}^{H} \mathrm{Cost}_{\mathrm{build}}(\mathcal{A}_t), \tag{2}$$

where $\alpha \geq 0$ weights construction-time expenditure. Although this objective captures the desired outcome, it provides only coarse credit assignment over a potentially long sequence of edits. An ideal dense reward instead credits each augmentation with its marginal contribution to downstream utility, net of its construction cost:

$$r_t^{\mathrm{dense}} \coloneqq U(\mathcal{F}_t) - U(\mathcal{F}_{t-1}) - \alpha\, \mathrm{Cost}_{\mathrm{build}}(\mathcal{A}_t). \tag{3}$$

Under undiscounted returns ($\gamma = 1$), these rewards telescope:

$$\sum_{t=1}^{H} r_t^{\mathrm{dense}} = r^{\mathrm{sparse}} - U(\mathcal{F}_0). \tag{4}$$

Because $U(\mathcal{F}_0)$ is fixed for a given initial filesystem, maximizing the cumulative dense reward is equivalent to maximizing the sparse objective, following potential based reward shaping (Ng et al., 1999). Advantages are computed at each step, following multi-turn GRPO implementations Baronio et al. (2025), allowing for earlier builder turns to get denser learning signal.

Evaluating downstream utility after every edit can be expensive, so we train with a practical surrogate objective. We probe answerability every eight turns, with at most four probes per construction episode, and combine changes between consecutive probes with a terminal downstream-answer reward. In the primary experiment, we use $\gamma = 1$ and optimize answerability and terminal correctness without explicit navigation-cost, filesystem-structure, or construction-cost shaping; equivalently, $\lambda_{\mathrm{tok}} = \lambda_{\mathrm{read}} = \alpha = 0$. During training, all downstream quantities are estimated on minibatches of training queries that are hidden from the builder; evaluation queries are never used to construct a filesystem or select the reported checkpoint. Appendix A.7 provides the complete configuration.

Overview of MDP Formulation

We formulate memory construction as an MDP in which periodic answerability changes and terminal answer correctness provide credit for useful filesystem edits; the general objective can additionally incorporate navigation and construction costs.

2.4 Memory Augmentations via RL.

The MDP formulation above enables direct optimization of the memory-construction process with reinforcement learning. In each rollout, the builder iteratively edits the filesystem for up to H steps, producing a trajectory

$$\tau = (\mathcal{F}_0, \mathcal{A}_1, \mathcal{F}_1, \ldots, \mathcal{A}_T, \mathcal{F}_T), \tag{5}$$

where $T \leq H$ is determined either by the horizon or the Stop action.

The resulting filesystem is then evaluated by the answering agent on a minibatch of queries sampled from $\mathcal{Q}$. These queries are used only for reward estimation and are **not** provided to the builder during construction. In the primary configuration, the builder receives periodic answerability-delta rewards together with the terminal answer score. This hybrid signal approximates the dense objective while avoiding an expensive answering evaluation after every filesystem edit. This training signal encourages the builder to discover filesystem edits that are not merely locally coherent but also useful

for improving downstream question answering. When $\alpha > 0$, the objective additionally trades these improvements against construction cost. We optimize the builder policy with GRPO (Shao et al., 2024), a policy-gradient method used for RL-based fine-tuning of language models.

By default, we treat the answering agent π_{ans} as fixed during builder optimization. This isolates the learning problem: the builder learns to construct memories useful to the answering agent, which may be a language model or another long-context reasoning system such as RLM (Zhang et al., 2025a). We can also consider an alternative joint-training variant in which the answering agent is adapted using the same downstream correctness signal. In this setting, optimization can alternate between updating the builder and updating the answerer via cooperative multi-agent RL framework, following methods such as RLAD (Qu et al., 2025a). The full optimization process is outlined in Algorithm 1.

Overview of RL Optimization

We optimize the builder as an LLM policy over filesystem edits. Each rollout constructs a memory, evaluates it with a fixed answering agent, and updates the builder with GRPO using periodic answerability changes and terminal answer correctness.

3 Experiments

Having formulated anticipatory memory construction as an MDP in Section 2, we evaluate whether `Librarian Agents` improves the reasoning performance and inference efficiency of downstream answering agents on long-context document reasoning tasks. We compare `Librarian Agents` with prior reactive retrieval and pre-computation methods to address four questions: **(1)** Does `Librarian Agents` improve in-distribution long-context reasoning performance? **(2)** What amortization benefits do `Librarian Agents` provide when serving multiple queries over a shared document set? **(3)** Does the builder agent's anticipatory construction capability generalize zero-shot to out-of-distribution retrieval settings? **(4)** Qualitatively, how do the builder and answering agents behave during the construction and answering phases, and do these behaviors reflect the intended anticipatory capability? We examine each question in turn below.

Task Definition. The main domain we consider for long-document reasoning is BrowseComp-Plus (Chen et al., 2025), a curated corpus of deep-research web tasks. We split this dataset into two sets for training and evaluation: 680 tasks for training and 50 held-out tasks for all evaluations (sampled at random). For training of π_{build}, each task consists of an independent set of raw documents and a downstream query that needs to be answered about the set of documents. We additionally evaluate the performance of `Librarian Agents` on the BRIGHT Benchmark (Su et al., 2025), which comprises a set of tasks from diverse domains (StackExchange, LeetCode, and math competitions), to assess the builder agent's efficacy on unseen distributions.

Comparisons and baselines. We evaluate our approach against several representative baselines. First, we compare with standard reactive retrieval methods: an *Embedding* with optional *Reranker* baseline (Zhang et al., 2025c) and *Late Interaction Embedding* (Chaffin & Sourty, 2025), which represent conventional retrieve-and-read RAG systems that process each query from scratch. Second, we compare with *Graph RAG* (Peng et al., 2025), a knowledge graph baseline that uses precomputation. Third, we include *Agentic RAG* (Singh et al., 2025), which represents multi-agent approaches that dynamically interleave retrieval and reasoning at query time. Next, we compare autocompaction method such as those that periodically summarize the computation performed, such as RC (Wu et al., 2026). These baselines help quantify the trade-offs of deferring structural reasoning to inference time to a summarization agent. We also include an ablation of our own framework: a *builder agent without RL fine-tuning*. This isolates the contribution of our MDP-based policy optimization. For all methods, the downstream answering agents, Qwen3.5-4B and Gemini 3.1 Flash lite, are kept frozen so that performance differences reflect only the quality of the resulting memory artifacts. More details about the baselines and hyperparameters can be found in Appendix A.7.

(1) In Distribution Performance of `Librarian Agents` on BrowseComp-Plus. We evaluate the performance of `Librarian Agents` on the BrowseComp-Plus domain in Figure 3. Here we find that `Librarian Agents` with RL fine-tuning significantly outperforms all reactive and pre-computation baselines. When using the frozen Qwen3.5-4B answering agent, `Librarian Agents` achieves an accuracy of 0.75, representing a substantial improvement over the Embedding (0.42) and Reranker (0.44) baselines. A similar trend is observed with Gemini 3.1 Flash Lite, where `Librarian Agents` reaches 0.77 compared to the baseline scores of 0.30 and 0.42. Crucially, the

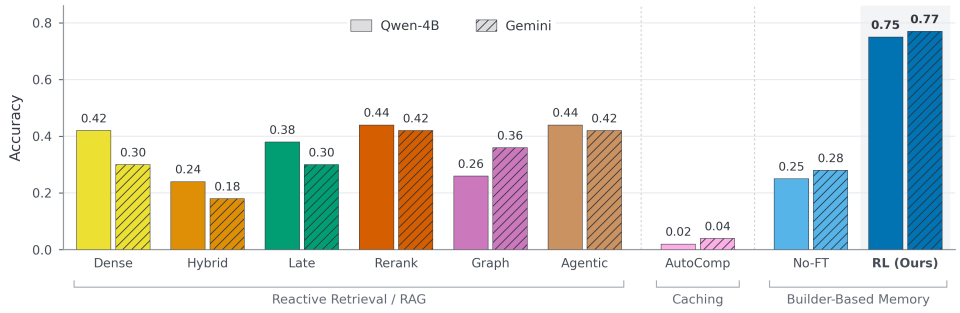


Figure 3: **Results on BrowseComp-Plus.** We compare `Librarian Agents` against RAG, Caching and Builder-Based Memory baselines using two answering agent models (Qwen 3.5 4B and Gemini 3.1 Flash Lite). We find that `Librarian Agents` significantly outperforms all methods for both answering agents.

Table 1: **Construction Cost, BrowseComp-Plus.** We compare the accuracy and cost of `Librarian Agents` with indexing based baselines in tokens.

Memory / Retrieval	Acc.	Construction Cost (tokens)
GraphRAG	0.36	11,923
AutoCompaction	0.04	501,637
Builder (No FT)	0.28	127,683
`Librarian Agents` (ours)	**0.77**	**178,926**

Table 2: **Performance on amortized BrowseComp+ (base vs. fine-tuned).** The fine-tuned builder improves accuracy from 0.24 to 0.64 on the amortized variant of browsecomp. The resulting answering agent reads 26.9 files and 9,346 tokens per query on average.

Model	Accuracy	Answer tokens per query	Files per query
Qwen 3.5 4B Builder	0.24	1,238	2.72
Ours	**0.64**	9,346	26.90

ablation without RL fine-tuning (No FT) additionally underperforms below `Librarian Agents`, highlights that policy optimization with RL is essential for teaching the builder agent to structure memories effectively for downstream consumption, a task the base model was not trained for.

(2) Amortization of `Librarian Agents`. One motivation for decoupling memory construction from query execution is that construction costs can be amortized across multiple queries over the same document set. As shown in Table 1, `Librarian Agents` incurs an upfront construction cost of 179K tokens, while reaching an accuracy of 0.77 compared with 0.36 and 0.04, respectively. To examine how this cost is distributed when a filesystem is reused, we evaluate 20 synthetic queries per document set, generated by Gemini 3.1 Flash Lite few-shot prompted with sample DeepDive (Lu et al., 2025) questions. On these queries, the same evaluation filesystems yield an accuracy of 0.64 versus 0.24 for the base builder at an answer-time cost of 9.3K tokens per query. When including construction cost spread over the 20 queries, the total is roughly 18K tokens per query (Table 2). Because the answering agent reads a richer filesystem, raw per-query token counts remain higher than with the base builder. After normalizing for accuracy, `Librarian Agents` uses 33.8% fewer recorded tokens per expected correct answer at ten questions per corpus (RL/Base ratio 0.662, 95% CI [0.504, 0.836]) as seen in Appendix A.4.

Table 3: **Zero-Shot Transfer to BRIGHT.** The builder agent, trained only on DeepResearch tasks (BrowseComp) is evaluated on BRIGHT without task-specific training, testing whether `Librarian Agents` learns a transferable capability of anticipation across heterogeneous query and document sources. `Librarian Agents` outperforms baselines on all tasks within BRIGHT (AOPS, Biology, and Stack Overflow) achieving an average a pass@1 accuracy of 52% and pass@4 accuracy of 0.69 with the `Qwen 3.5 4b` model.

Domain	BM25	Dense	Hybrid	Rerank	Agentic	Late-inter.	Graph	AutoComp.	**Librarian**
AOPS	0.31 / 0.38	0.32 / 0.38	0.33 / 0.38	0.32 / 0.38	0.29 / 0.38	0.29 / 0.38	0.13 / 0.16	0.06 / 0.06	**0.36 / 0.52**
Biology	0.18 / 0.28	0.20 / 0.32	0.14 / 0.22	0.14 / 0.26	0.17 / 0.28	0.18 / 0.26	0.05 / 0.06	0.06 / 0.06	**0.35 / 0.62**
Stack Overflow	0.09 / 0.18	0.12 / 0.18	0.07 / 0.18	0.06 / 0.12	0.07 / 0.18	0.07 / 0.16	0.02 / 0.04	0.02 / 0.02	**0.86 / 0.94**
Average	0.19 / 0.28	0.21 / 0.29	0.18 / 0.26	0.17 / 0.25	0.17 / 0.28	0.18 / 0.27	0.06 / 0.09	0.05 / 0.05	**0.52 / 0.69**

(3) Transfer of Builder Agent to novel long-context document reasoning setting. To assess whether the builder agent learns a generalized anticipatory capability rather than simply overfitting to the DeepResearch training distribution, we evaluate `Librarian Agents` zero-shot on the BRIGHT benchmark. Here, we study 150 questions spanning 3 domains with 50 questions each, with one filesystem per domain (additionally studying amortization). As detailed in Table 3, the RL-

finetuned builder agent successfully transfers to highly out-of-distribution domains such as biology and math problem solving. `Librarian Agents` achieves an average pass@1 accuracy of 0.52 and pass@4 accuracy of 0.69 across the 3 BRIGHT datasets, outperforming baselines. These results confirm that framing memory construction as an MDP enables the builder to learn fundamental, transferable heuristics for organizing information, which remain robust even when encountering novel, unseen documents.

The structural statistics on BRIGHT further show that RL substantially changes construction behavior. Across the 3 sub-domains, the trained builder produces richer hierarchical memories, averaging 53.4 active synthetic files organized into 45.4 clusters and 8.0 merged artifacts, compared with 4.8 files, 4.2 clusters, and 0.6 merges for the base builder.

(4) Analysis of `Librarian Agents` and capability of anticipation. We next examine the filesystems that `Librarian Agents` constructs and how the answering agent uses them. Figure 2 illustrates that the trained builder combines broad topical coverage with hierarchical organization. Tables 4 and 5 complement this view by distinguishing answering-agent retrieval (the synthetic files and tokens read) from builder structure (the active files, clusters, and merges it produces). Broad coverage expands the range of future queries the memory can support, while hierarchical organization makes fine-grained evidence easier to locate.

The first qualitative case examines robustness when anticipatory coverage is incomplete. In Figure 5, a BrowseComp+ query requires a detail omitted from the synthetic layer. The answering agent follows a provenance link to the relevant raw file and recovers the answer without enumerating the full unstructured corpus.

The second case isolates what reinforcement learning changes in the constructed memory. Figure 4 compares the trained builder with the base policy on the same BrowseComp+ document set. The trained builder creates a broader, more coherent set of query-relevant artifacts, while the base policy leaves a fragmented organization that is harder for the answering agent to navigate.

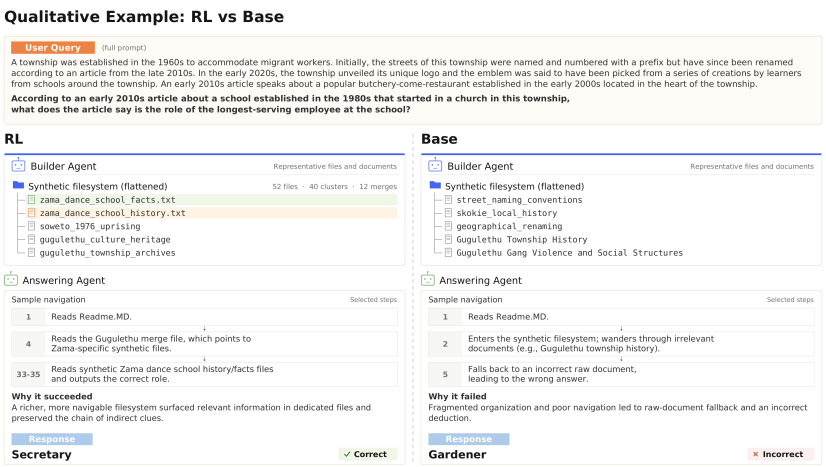


Figure 4: **Qualitative comparison of RL and base building.** On BrowseComp+, the base model creates a fragmented filesystem that is difficult for the answering agent to navigate, whereas the RL-trained builder surfaces relevant information in dedicated files and produces a richer, more ergonomic organization.

Overview of Experimental Evaluation

Our evaluation shows that `Librarian Agents` improves long-context document reasoning in-domain, transfers to BRIGHT, amortizes construction cost across repeated queries, and learns memory structures that remain navigable when synthetic coverage is incomplete.

4 Related Work

Retrieval-Augmented Generation. RAG grounds language model outputs in external evidence, mitigating the limitations of purely parametric memory (Lewis et al., 2020; Gao et al., 2023; Li et al.,

2025; Oche et al., 2025). Recent innovations have evolved this pipeline into agentic systems capable of multi-step reasoning (Xu et al., 2024; Zhang et al., 2025b; Singh et al., 2025). For instance, models can now predict upcoming content to trigger searches (Jiang et al., 2023), use reflection tokens for self-critique (Asai et al., 2023; Yan et al., 2024), or execute planned searches to navigate multi-hop questions (Li et al., 2026). However, these approaches share a reactive bottleneck: they initiate complex, iterative retrieval only *after* receiving a user query. In contrast, `Librarian Agents` shifts this computational burden by proactively synthesizing a structured, query-agnostic filesystem *prior* to inference, amortizing the cost of corpus-level reasoning.

Structured Indexes and Corpus-Level Organization. Prior systems precompute corpus structure: RAPTOR builds hierarchical summaries (Sarthi et al., 2024); GraphRAG and related methods construct entity graphs and community summaries (Edge et al., 2024; Li et al., 2024; Guo et al., 2025); and HippoRAG builds graph-structured memory (Jiménez Gutiérrez et al., 2024). Although valuable, these preprocessing methods rely heavily on rigid, rule-based extraction pipelines or predefined schemas. In contrast, `Librarian Agents` make anticipatory memory construction fully agentic and optimizable. Instead of static graph heuristics, our builder agent uses RL to generate synthetic filesystem augmentations: dynamically synthesizing, merging, or pruning documents to strictly optimize the corpus for downstream utility.

Long-Term Memory for Language Agents. Prior work explores endowing language models with persistent, stateful memory to support continuous operation. For instance, recent works propose to cache conversational histories (Wang et al., 2023; Zhong et al., 2023; Uddin et al., 2026), manage virtual context windows via memory tiers (Packer et al., 2023), and unify short- and long-term memory directly within the agent's policy (Yu et al., 2026b; Xu et al., 2025). However, these approaches primarily focus on an agent retaining its own historical context. In contrast, `Librarian Agents` studies anticipatory memory construction over external corpora. Rather than simply logging past interactions, a builder agent proactively synthesizes, organizes, and prunes an answer-facing filesystem before queries arrive. This generates a reusable memory artifact for a separate answering agent, explicitly optimizing the trade-off between downstream utility and construction cost.

Compute Reallocation. Our approach aligns with emerging literature on reallocating computation across time to improve test-time scaling (Lin et al., 2025; Wu et al., 2026; Yan et al., 2026; Yu et al., 2026a). Other approaches formalize test-time compute optimization as meta-reinforcement learning optimization to adaptively allocate tokens (Qu et al., 2025b) or utilize self-curated test-time curricula to reinforce successful strategies (Hübotter et al., 2025). While these methods share the goal of using compute to prevent reactivity, `Librarian Agents` uniquely formulates this shift as a synthetic filesystem construction problem. We model the memory building of a filesystem as an MDP, which is optimized using RL. Furthermore, `Librarian Agents` makes the compute reallocation problem an explicit target of policy optimization rather than a byproduct of inference.

5 Discussion, Limitations, and Future Work

We study the capability of *anticipatory memory construction*, where agents can proactively reorganizing their context into a memory artifact optimized for future queries the agent encounters. To instantiate this capability, we rely on an asymmetry for agents today, where agents are trained to navigate a filesystem for a given task, yet do not pre-construct filesystems for future queries that they might encounter. Our novel framework, `Librarian Agents`, introduces builder agent to create reusable memory artifact in the form a synthetic filesystem that an answering agent can navigate. `Librarian Agents` yields consistent improvements across several long-context document reasoning benchmarks such as BrowseComp+ and BRIGHT, outperforming existing methods for retrieval on both in-domain and out-of-distribution evals.

Our evaluation is limited to question answering over static corpora, and maintaining anticipatory memories as documents change remains an open problem. The builder agent also operates over a large file context, which may limit scalability, though pairing Librarian Models with clustering or retrieval could extend the framework to much larger domains. Since no finite set of synthesized artifacts can anticipate every query, compression may discard details needed for needle-in-a-haystack tasks; we preserve provenance links for fallback access, but deciding when to bypass the synthetic filesystem remains a challenge. Future work could co-adapt the builder and answering agents using the shared correctness signal, and generalize the builder's discrete actions to atomic filesystem commands (e.g., bash) for richer memory structures.

6 AI USE STATEMENT

Large language models were used to assist with manuscript preparation, specifically for proofreading, suggesting phrasing improvements for clarity and conciseness, and generating boilerplate code for data formatting and baseline implementations. All conceptual contribution, and the experimental designs, were done by human authors.

7 ETHICS STATEMENT

Anticipatory memory construction via Librarian Models may introduce some ethical risks. Firstly because the builder agent's synthesis operations (such as creating and merging files) rely on compression, information is necessarily discarded. There is a risk that the builder may synthesize hallucinated facts or inadvertently misrepresent the original text during this phase. If this occurs, downstream answering agents navigating the flawed filesystem may confidently produce incorrect answers. Although our framework mitigates this by preserving the provenance of the original documents, giving the answering agent the option to fall back to raw evidence, there is a risk that the answering agent might over-rely on the synthetic artifacts rather than rigorously verifying claims against the set of source documents.

8 REPRODUCIBILITY STATEMENT

We describe the memory-construction algorithm and GRPO objective in Section 2 and Algorithm 1. Appendix A.7 reports the builder model, optimization settings, reward and probe schedule, generation budgets, and evaluation decoding parameters. Appendix A.8 specifies the filesystem action space, while Appendix A.9 provides the builder, answering-agent, judge, and baseline prompts. We will release the code, training scripts, and evaluation framework under the MIT License, and the synthesized filesystem artifacts, logs, and evaluation results under CC BY 4.0, subject to the underlying datasets' licensing terms.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*, 2023. URL https://arxiv.org/abs/2310.11511.

Carlo Baronio, Pietro Marsella, Ben Pan, Simon Guo, and Silas Alberti. Kevin: Multi-turn rl for generating cuda kernels, 2025. URL https://arxiv.org/abs/2507.11948.

Antoine Chaffin and Raphaël Sourty. Pylate: Flexible training and retrieval for late interaction models, 2025. URL https://arxiv.org/abs/2508.03555.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Zijian Chen, Xueguang Ma, Shengyao Zhuang, Ping Nie, Kai Zou, Andrew Liu, Joshua Green, Kshama Patel, Ruoxi Meng, Mingyi Su, et al. Browsecomp-plus: A more fair and transparent evaluation benchmark of deep-research agent. *arXiv preprint arXiv:2508.06600*, 2025.

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024. URL https://arxiv.org/abs/2404.16130.

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. URL https://arxiv.org/abs/2312.10997.

Kai Guo, Harry Shomer, Shenglai Zeng, Haoyu Han, Yu Wang, and Jiliang Tang. Empowering GraphRAG with Knowledge Filtering and Integration, 2025. URL https://arxiv.org/abs/2503.13804.

Jonas Hübotter, Leander Diaz-Bone, Ido Hakimi, Andreas Krause, and Moritz Hardt. Learning on the Job: Test-Time Curricula for Targeted Reinforcement Learning, 2025. URL https://arxiv.org/abs/2510.04786.

Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. *arXiv preprint arXiv:2305.06983*, 2023. URL https://arxiv.org/abs/2305.06983.

Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. HippoRAG: Neurobiologically inspired long-term memory for large language models. *arXiv preprint arXiv:2405.14831*, 2024. URL https://arxiv.org/abs/2405.14831.

Imed Keraghel, Stanislas Morbieu, and Mohamed Nadif. Recent advances in named entity recognition: A comprehensive survey and comparative study, 2024. URL https://arxiv.org/abs/2401.10825.

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. *arXiv preprint arXiv:2005.11401*, 2020. URL https://arxiv.org/abs/2005.11401.

Mufei Li, Siqi Miao, and Pan Li. Simple Is Effective: The Roles of Graphs and Large Language Models in Knowledge-Graph-Based Retrieval-Augmented Generation, 2024. URL https://arxiv.org/abs/2410.20724.

Siran Li, Linus Stenzel, Carsten Eickhoff, and Seyed Ali Bahrainian. Enhancing Retrieval-Augmented Generation: A Study of Best Practices, 2025. URL https://arxiv.org/abs/2501.07391.

Xingyu Li, Rongguang Wang, Yuying Wang, Mengqing Guo, Chenyang Li, Tao Sheng, Sujith Ravi, and Dan Roth. PAR^2-RAG: Planned Active Retrieval and Reasoning for Multi-Hop Question Answering, 2026. URL https://arxiv.org/abs/2603.29085.

Kevin Lin, Charlie Snell, Yu Wang, Charles Packer, Sarah Wooders, Ion Stoica, and Joseph E. Gonzalez. Sleep-time compute: Beyond inference scaling at test-time. *arXiv preprint arXiv:2504.13171*, 2025. URL https://arxiv.org/abs/2504.13171.

Rui Lu, Zhenyu Hou, Zihan Wang, Hanchen Zhang, Xiao Liu, Yujiang Li, Shi Feng, Jie Tang, and Yuxiao Dong. Deepdive: Advancing deep search agents with knowledge graphs and multi-turn rl. *arXiv preprint arXiv:2509.10446*, 2025.

Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287. Citeseer, 1999.

Agada Joseph Oche, Ademola Glory Folashade, Tirthankar Ghosal, and Arpan Biswas. A Systematic Review of Key Retrieval-Augmented Generation (RAG) Systems: Progress, Gaps, and Future Directions, 2025. URL https://arxiv.org/abs/2507.18910.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Boci Peng, Yun Zhu, Yongchao Liu, Xiaohe Bo, Haizhou Shi, Chuntao Hong, Yan Zhang, and Siliang Tang. Graph retrieval-augmented generation: A survey. *ACM Transactions on Information Systems*, 44(2):1–52, 2025.

Yuxiao Qu, Anikait Singh, Yoonho Lee, Amrith Setlur, Ruslan Salakhutdinov, Chelsea Finn, and Aviral Kumar. Rlad: Training llms to discover abstractions for solving reasoning problems, 2025a. URL https://arxiv.org/abs/2510.02263.

Yuxiao Qu, Matthew Y. R. Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. Optimizing Test-Time Compute via Meta Reinforcement Fine-Tuning, 2025b. URL https://arxiv.org/abs/2503.07572.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D. Manning. RAPTOR: Recursive abstractive processing for tree-organized retrieval. *arXiv preprint arXiv:2401.18059*, 2024. URL https://arxiv.org/abs/2401.18059.

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

Aditi Singh, Abul Ehtesham, Saket Kumar, Tala Talaei Khoei, and Athanasios V. Vasilakos. Agentic retrieval-augmented generation: A survey on agentic RAG. *arXiv preprint arXiv:2501.09136*, 2025. URL https://arxiv.org/abs/2501.09136.

Hongjin Su, Howard Yen, Mengzhou Xia, Weijia Shi, Niklas Muennighoff, Han yu Wang, Haisu Liu, Quan Shi, Zachary S. Siegel, Michael Tang, Ruoxi Sun, Jinsung Yoon, Sercan O. Arik, Danqi Chen, and Tao Yu. Bright: A realistic and challenging benchmark for reasoning-intensive retrieval, 2025. URL https://arxiv.org/abs/2407.12883.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory, 2025. URL https://arxiv.org/abs/2504.07952.

Md Nayem Uddin, Kumar Shubham, Eduardo Blanco, Chitta Baral, and Gengyu Wang. From Recall to Forgetting: Benchmarking Long-Term Memory for Personalized Agents, 2026. URL https://arxiv.org/abs/2604.20006.

Weiyun Wang, Shuibo Zhang, Yiming Ren, Yuchen Duan, Tiantong Li, Shuo Liu, Mengkang Hu, Zhe Chen, Kaipeng Zhang, Lewei Lu, Xizhou Zhu, Ping Luo, Yu Qiao, Jifeng Dai, Wenqi Shao, and Wenhai Wang. Needle in a multimodal haystack, 2024. URL https://arxiv.org/abs/2406.07230.

Weizhi Wang, Li Dong, Hao Cheng, Xiaodong Liu, Xifeng Yan, Jianfeng Gao, and Furu Wei. Augmenting language models with long-term memory. *arXiv preprint arXiv:2306.07174*, 2023. URL https://arxiv.org/abs/2306.07174.

Ian Wu, Yuxiao Qu, Amrith Setlur, and Aviral Kumar. Reasoning cache: Continual improvement over long horizons via short-horizon RL. *arXiv preprint arXiv:2602.03773*, 2026. URL https://arxiv.org/abs/2602.03773.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-MEM: Agentic Memory for LLM Agents, 2025. URL https://arxiv.org/abs/2502.12110.

Zhipeng Xu, Zhenghao Liu, Yibin Liu, Chenyan Xiong, Yukun Yan, Shuo Wang, Shi Yu, Zhiyuan Liu, and Ge Yu. ActiveRAG: Revealing the Treasures of Knowledge via Active Learning, 2024. URL https://arxiv.org/abs/2402.13547v1.

Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling. Corrective retrieval augmented generation. *arXiv preprint arXiv:2401.15884*, 2024. URL https://arxiv.org/abs/2401.15884.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schuetze, Volker Tresp, and Yunpu Ma. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12805–12825, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.583. URL https://aclanthology.org/2026.acl-long.583/.

Hongli Yu, Tinghong Chen, Jiangtao Feng, Jiangjie Chen, Weinan Dai, Qiying Yu, Ya-Qin Zhang, Wei-Ying Ma, Jingjing Liu, Mingxuan Wang, and Hao Zhou. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent, 2026a. URL https://arxiv.org/abs/2507.02259.

Yi Yu, Liuyi Yao, Yuexiang Xie, Qingquan Tan, Jiaqi Feng, Yaliang Li, and Libing Wu. Agentic Memory: Learning Unified Long-Term and Short-Term Memory Management for Large Language Model Agents, 2026b. URL https://arxiv.org/abs/2601.01885.

Alex L. Zhang, Tim Kraska, and Omar Khattab. Recursive language models. *arXiv preprint arXiv:2512.24601*, 2025a. URL https://arxiv.org/abs/2512.24601.

Chao Zhang, Yuhao Wang, Derong Xu, Haoxin Zhang, Yuanjie Lyu, Yuhao Chen, Shuochen Liu, Tong Xu, Xiangyu Zhao, Yan Gao, Yao Hu, and Enhong Chen. TeaRAG: A Token-Efficient Agentic Retrieval-Augmented Generation Framework, 2025b. URL https://arxiv.org/abs/2511.05385.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176*, 2025c.

Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. *arXiv preprint arXiv:2305.10250*, 2023. URL https://arxiv.org/abs/2305.10250.

A Appendix

A.1 More details on experimental setup

BrowseComp-Plus. BrowseComp-Plus (Chen et al., 2025) pairs document collections with questions that require locating and combining information distributed across those collections. We train the filesystem builder on 780 corpus-level tasks and reserve 50 previously unseen corpus-level tasks, for evaluation. We use a mixture of evidence, gold, and negative docs for each representative task, following the composition in the original dataset. During construction, the builder receives only the documents and never observes the evaluation questions. The completed filesystem is then reused for every question associated with its corpus. We report failure-inclusive binary semantic answer correctness against the fixed reference answers, so unsuccessful constructions and failed generations remain in the denominator. In addition, for the Base-versus-RL comparison, both builders receive identical construction limits, and we report realized construction tokens separately because equal limits do not imply equal consumption.

BRIGHT-derived question answering. BRIGHT spans technical, scientific, mathematical, and programming domains with substantially different document and query distributions (Su et al., 2025). We convert its retrieval instances into a generation task by associating each query with a fixed reference answer and scoring generated responses for semantic correctness. For the primary matched-baseline evaluation, we use 150 questions spanning three domains, with 50 questions each from AOPS, Biology, and Stack Overflow. To test cross-benchmark transfer, we apply the BrowseComp-trained Librarian zero-shot: it constructs one reusable filesystem per domain without observing any evaluation question, and this single shared filesystem supports all questions in that domain. We compare the Librarian against matched retrieval baselines, generating four independent responses per question and reporting both mean per-response accuracy, which estimates pass@1, and pass@4, which we compute using the numerically stable estimator from Codex (Chen et al., 2021).

A.2 Additional Results

First, we list statistics on the number of synthetic files and tokens created by the builder agent. We also include details about the breadth and depth of the constructed filesystems from the builder.

Table 4: **BRIGHT statistics by domain.** We report answerer retrieval behavior and final filesystem-structure statistics for BRIGHT. Syn. Files, Syn. Tok., and Tot. Tok. measure files and tokens read by the answering agent, while Act. Files, Clusters, and Merges summarize the active synthetic filesystem produced by the builder.

Domain	Model	Syn. Files	Syn. Tok.	Tot. Tok.	Act. Files	Clusters	Merges
Bio.	Base	2.7	1064	1189	5.4	4.9	0.5
	RL	30.4	9883	10091	54.0	45.8	8.2
StackOv.	Base	3.3	1075	1250	4.5	4.0	0.5
	RL	32.0	9246	9511	49.5	40.6	8.8
AoPS	Base	3.9	879	1023	4.5	3.8	0.7
	RL	36.2	7309	7544	56.7	49.7	7.0
Macro avg.	Base	3.3	1006	1154	4.8	4.2	0.6
	RL	32.9	8813	9049	53.4	45.4	8.0

Table 5: **BrowseComp+ statistics.** We report the same statistics on the amortized BrowseComp+ evaluation. Syn. Files, Syn. Tok., and Tot. Tok. again measure answering-agent retrieval behavior, while Act. Files, Clusters, and Merges summarize the active synthetic filesystem produced by the builder.

Dataset	Model	Syn. Files	Syn. Tok.	Tot. Tok.	Act. Files	Clusters	Merges
BrowseComp+	Base	2.7	1069	1235	3.2	2.7	0.6
	RL	26.9	9040	9309	42.2	34.4	7.8

We also showcase a qualitative example of an agent recovering using Provenance Information, when the synthesized documents didn't have the complete information to answer a question in Figure 5.

Finally, we include additional bright transfer results comparing `Librarian Agents` against the base model in Table 6. We see consistent improvement across the nine additional domains within the Bright dataset, using $n = 20$ questions per domain.

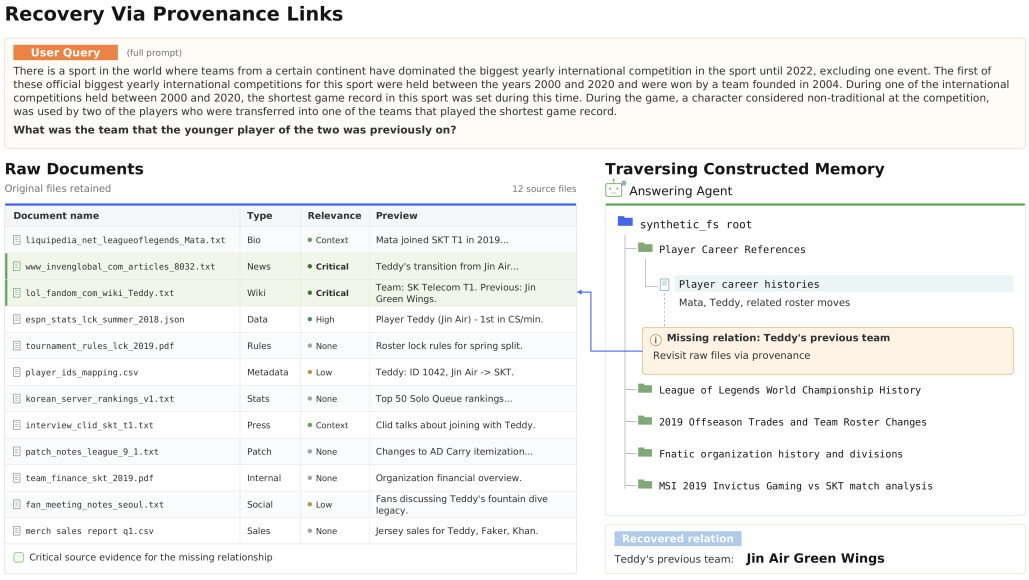


Figure 5: **Recovery on BrowseComp-Plus using provenance information.** When the synthetic filesystem omitted a required detail, the answering agent followed provenance links to the relevant raw source and recovered the correct answer.

Table 6: **Comparison with the Base Model on Additional BRIGHT Domains.** `Librarian Agents` outperforms the base model across all nine additional domains. Results are reported as percentages.

Method	Earth Science	Economics	Psychology	Robotics	Sustainable Living	LeetCode	Pony	TheoremQA Questions	TheoremQA Theorems
Base Model	26.3%	16.3%	32.5%	23.7%	12.5%	28.7%	48.7%	35.0%	38.8%
Librarian	**73.8%**	**72.5%**	**72.5%**	**55.0%**	**43.8%**	**56.2%**	**72.5%**	**63.7%**	**56.2%**

A.3 Algorithm Overview of Librarian Agents

Algorithm 1 RL Training of `Librarian Agents`

Require: Documents $\mathcal{D}$, query distribution $\mathcal{Q}$, builder π_{build}, answerer π_{ans}, horizon H
Ensure: Optimized builder policy π_{build}

for $k = 1, \ldots, K$ **do**
  Sample hidden reward-query batch $\mathcal{B}_k \sim \mathcal{Q}$
  Initialize $\mathcal{F}_0 \leftarrow (\mathcal{D}, \emptyset)$, $\tau \leftarrow \emptyset$
  for $t = 1, \ldots, H$ **do**
    Sample edit $\mathcal{A}_t \sim \pi_{\text{build}}(\cdot \mid \mathcal{F}_{t-1})$
    Initialize $\widehat{r}_t \leftarrow 0$
    if $\mathcal{A}_t = \textsc{Stop}$ **then**
      $\mathcal{F}_t \leftarrow \mathcal{F}_{t-1}$
      Append $(\mathcal{F}_{t-1}, \mathcal{A}_t, \widehat{r}_t, \mathcal{F}_t)$ to τ
      break
    Apply edit: $\mathcal{F}_t \leftarrow \mathcal{A}_t(\mathcal{F}_{t-1})$
    if t is an answerability-probe step **then**
      Estimate $\widehat{S}(\mathcal{F}_t)$ on $\mathcal{B}_k$
      Assign answerability-delta reward $\widehat{r}_t$
    Append $(\mathcal{F}_{t-1}, \mathcal{A}_t, \widehat{r}_t, \mathcal{F}_t)$ to τ
  Evaluate terminal answer score on $\mathcal{B}_k$
  Add the terminal score to the rollout reward
  Update π_{build} with GRPO on rollout τ
return π_{build}

A.4 BROWSECOMP-PLUS REUSE AND BREAK-EVEN ANALYSIS

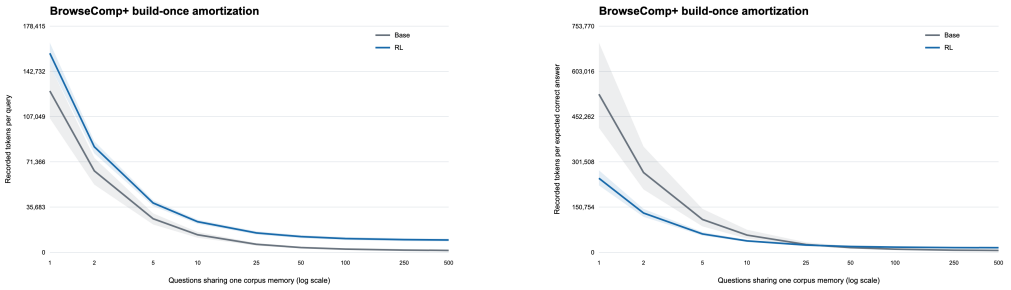


Figure 6: **BrowseComp-Plus reuse curves.** Left: recorded construction and navigation tokens per query. Right: recorded tokens per expected correct answer after normalization by observed full-denominator accuracy. Shaded regions show paired corpus-bootstrap 95% confidence intervals.

We compute reuse curves directly from the frozen BrowseComp-Plus rollout logs. For method m and q questions per corpus, recorded tokens per query are

$$T_m(q) = \frac{B_m}{q} + A_m,$$

where B_m is the observed construction-token cost per corpus and A_m is the observed answer-time workspace-token cost per question. To account for differences in answer quality, we additionally report $T_m(q)/p_m$, where p_m is full-denominator accuracy.

At ten questions per corpus, RL uses 33.8% fewer recorded tokens per expected correct answer than the base builder (RL/Base ratio 0.662, paired-bootstrap 95% CI $[0.504, 0.836]$). The point crossover occurs at 31 questions per corpus, with a paired-bootstrap interval of 18–54 questions. Because the RL answerer explores a richer workspace, its raw recorded tokens per query remain higher; we therefore report raw and accuracy-normalized curves together rather than treating token count independently of answer quality.

A.5 TRAINING CURVE FOR LIBRARIAN AGENTS

Figure 7 reports the aggregate rollout reward during GRPO optimization on the BrowseComp-Plus training split. Reward increases over training as the builder learns construction decisions that produce more answerable downstream memories.

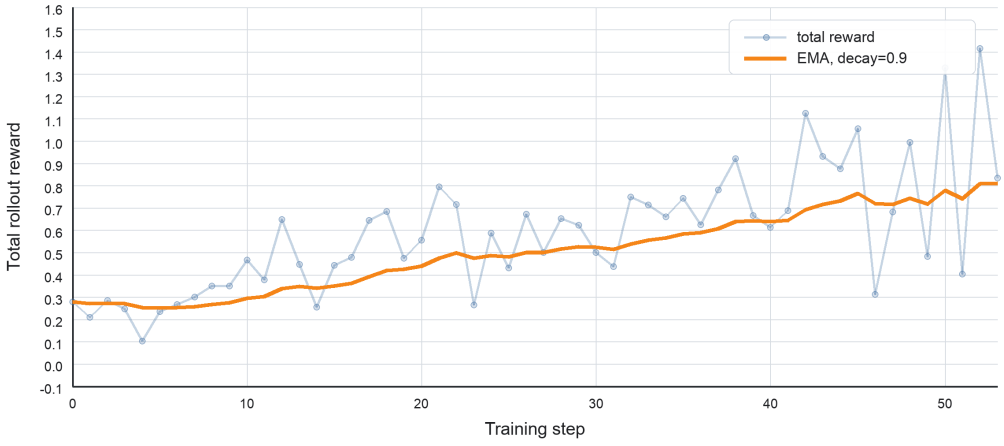


Figure 7: **Training curve for the BrowseComp-Plus builder.** Aggregate rollout reward during GRPO training, smoothed with an exponential moving average.

A.6 LICENSE FOR ARTIFACTS

The code, training scripts, and evaluation framework for `Librarian Agents` will be released under the MIT License. Synthesized filesystem artifacts, logs, and evaluation results generated during our experiments will be released under the Creative Commons Attribution 4.0 International (CC BY 4.0) license. The models used as answering agents, namely Qwen3.5-4B and Gemini 3.1 Flash Lite, are subject to their respective licensing agreements (e.g., the Tongyi Qianwen LICENSE agreement and Google's terms of service). BrowseComp+, BRIGHT, and DeepDive datasets are subject to their creators' respective data licensing terms.

A.7 LIBRARIAN AGENTS TRAINING AND EVALUATION CONFIGURATION

For the primary BrowseComp-Plus experiment, we optimize a Qwen3.5-4B builder with GRPO on the 680-task training split. We use LoRA rank 32, a learning rate of 4×10^{-5}, batch size 16, rollout-group size 4, and a maximum construction horizon of 32 builder turns. Builder generations are limited to 1,024 tokens and complete construction trajectories to 60,000 tokens. Filesystem operations are executed by a frozen Qwen3.5-4B model, while the answering agent and evaluator remain fixed throughout builder optimization.

The training reward measures the downstream usefulness of the evolving filesystem. We probe answerability every eight builder turns, with at most four probes per construction trajectory, and assign reward according to the change in answerability between consecutive probes. A terminal reward evaluates the final filesystem using downstream answer correctness. We use $\gamma = 1$ and no explicit filesystem-structure reward, navigation-cost reward, or construction-cost shaping in the primary run ($\alpha = 0$). Filesystem maturity, organization, and navigation measurements reported in the analysis are therefore diagnostic statistics rather than direct components of the optimization objective.

For pass@4 evaluations, we generate four independent answers per question and count a question as solved when any response is correct. The matched BRIGHT evaluation uses temperature 0.7 and top-$p = 1$ for answer sampling, while single-answer evaluations use deterministic decoding. Within each comparison, all methods use the same source corpus, answering model, response count, answer-generation budget, and scoring procedure. Evaluation questions are never shown to the builder during construction or used to select the reported checkpoint.

A.8 TOOL DEFINITIONS FOR FILESYSTEM ACTION SPACE

The builder agent interacts with the filesystem by delegating a limited set of discrete operations to specialized tool-calling agents. These tools are made available to the builder LLM through structured JSON schemas and can be invoked as needed. We follow the Qwen Tool Calling Format: `https://qwen.readthedocs.io/en/latest/framework/function_call.html`.

The CREATEFILE tool creates a new synthetic file by consolidating information from selected raw documents and/or existing synthetic files. The resulting file is added to the hierarchy and includes provenance links to the sources used to generate it. The MERGEFILES tool combines related synthetic files into a higher-level artifact, such as a summary, index, or directory-like file. This operation introduces hierarchical structure while preserving links to the underlying child files, which become nested under the new summary file.

The DELETE tool removes redundant or unhelpful synthetic files from the active, answer-facing filesystem. It never deletes raw documents in $\mathcal{D}$, ensuring that the original corpus remains available as a fallback. Finally, the STOP tool ends the memory-construction process and returns the current filesystem $\mathcal{F}_t$ as the final memory artifact.

We use the agent's function-calling capability to invoke these specialized subagents, each of which implements one of the individual tools. The subagent is the frozen version of the base model used for optimization.

A.9 LIBRARIAN AGENTS AND BASELINE PROMPTS

To initialize the builder agent, we utilize a detailed system prompt that establishes its task and action space constraints:

Prompt

```
You are building a synthetic filesystem for a later answering agent.

Use the provided filesystem tools to inspect files under:
- raw_docs/
- synthetic_fs/
- history/

Your job is to leave behind a final active synthetic filesystem that
↪  helps later agents answer as many plausible future questions
↪  about this corpus as possible.
Do not optimize for one guessed target question; build broad,
↪  reusable knowledge structure over the corpus.
The later answering agent will start with root-level synthetic
↪  entries, browse deeper synthetic files if needed, and use
↪  raw_docs/ only as fallback.
```

Here is the prompt for the answering agent.

Prompt

```
You are the answering agent for a read-only workspace.

You have filesystem tools over a final workspace that contains:
- README.md explaining the workspace conventions and navigation only
- root-level synthetic entries that are already loaded into context
↪  as the primary entry points
- nested synthetic folders and summary files for more detail

Use this order:
1. start from the already-loaded README.md and root-level synthetic
↪  entries
2. browse deeper synthetic files if needed

Do not guess. If the synthetic files you have read so far do not
↪  contain enough evidence to uniquely identify the answer, continue
↪  exploring the synthetic filesystem before answering. README.md
↪  only explains the workspace and is not sufficient evidence by
↪  itself. Answer only when the synthetic workspace contains enough
↪  evidence.

At each step, either:
- return exactly one compact JSON action:
  {"action":"list_files","path":"."}
  {"action":"read_file","path":"some/path.txt","start_line":1,"num_l
  ↪  ines":80}
- or return exactly one non-empty line:
  FINAL ANSWER: <final answer>

Do not include explanation outside the JSON action or final answer
↪  line.
```

We use the following prompt for Judging with the Gemini 3.1 Flash Lite model as a judge for our method and baselines.

Prompt

```
You are given a question, sampled response, and gold answer.
Decide whether the sampled response is semantically equivalent to the
↪  gold answer.
Minor formatting, capitalization, punctuation, and unit-format
↪  differences are acceptable
when the answer means the same thing.

Return only valid JSON with this shape:
```

```
{{"correctness": 0 or 1}}

QUESTION:
{question}

SAMPLED RESPONSE:
{response}

GOLD ANSWER:
{gold_answer}
```

For baselines, the common answering agent prompt was:

Prompt

```
You are a grounded question-answering assistant.

Answer the question using the retrieved documents below.

Instructions:
1. Treat the retrieved documents as a source of truth.
2. Some documents may be irrelevant, incomplete, duplicated, or
↪  conflicting.
3. Ignore any document content that does not directly help answer the
↪  question.
4. Do not use outside knowledge or make assumptions.
5. If documents conflict, use the answer best supported by the most
↪  direct evidence.
6. Return only the final short answer. Do not include explanation,
↪  citations, markdown, or extra text.

QUESTION:
{question}

RETRIEVED DOCUMENTS:
{documents}

FINAL ANSWER:
```

We also list the prompts for the GraphRag baseline, following the microsoft graph rag system: https://github.com/microsoft/graphrag.

Here is the graph extraction prompt.

Prompt

```
You are building a knowledge graph for retrieval-augmented question
↪  answering.

Goal:
Given one text unit, extract the salient entities and relationships
↪  needed to build a reusable corpus-level
knowledge graph for retrieval-augmented question answering.
Keep specific entities, bridge entities, dates, quantities, events,
↪  works, and facts that could support
future multi-hop questions over this corpus.

Entity types to consider:
{entity_types}

Output format:
Use "{record_delimiter}" between records and finish with
↪  "{completion_delimiter}".
For each entity:
("entity"<|>ENTITY_NAME<|>ENTITY_TYPE<|>ENTITY_DESCRIPTION)
For each relationship:
```

```
("relationship"<|>SOURCE_ENTITY_NAME<|>TARGET_ENTITY_NAME<|>RELATION
↪  SHIP_DESCRIPTION<|>RELATIONSHIP_STRENGTH)

Rules:
- Use canonical, specific entity names.
- Descriptions must be concise and grounded in the text unit.
- Relationship strength is a number from 1 to 10.
- Do not infer facts beyond the text unit.
- If there are no useful records, output only
↪  "{completion_delimiter}".

Text unit metadata:
path: {path}
text_unit_id: {text_unit_id}

Text unit:
{text}
```

Here is the summarization description prompt:

Prompt

```
You are reconciling extracted knowledge graph descriptions.

Given one entity or relationship name and several source
↪  descriptions, write one concise, faithful summary.
Preserve specific names, dates, quantities, and qualifications that
↪  may matter for question answering.
Do not add facts that are not present in the descriptions.

Name:
{name}

Descriptions:
{descriptions}

Return only the merged summary.
{text}
```

Here is the community report prompt:

Prompt

```
You are writing a GraphRAG community report for retrieval-augmented
↪  question answering.

Use only the provided entity and relationship tables. The report
↪  should summarize what this community contains
so it can be reused for later retrieval-augmented question answering.

Return only valid JSON with this shape:
{{
  "title": "short community title",
  "summary": "one paragraph summary grounded in the tables",
  "rating": 0-10,
  "findings": [
    {{"summary": "specific finding", "explanation": "brief
    ↪  explanation with entity or relationship ids"}}
  ]
}}

Maximum report words: {max_report_words}

Community tables:
{community_context}
```

For autocompaction, we use the following prompt for summarization:

Prompt

```
Compress a document stream into query-independent notes for later QA.

Rules:
- The future question is hidden. Do not tailor the notes to any
↪  query.
- Keep only grounded facts: names, dates, numbers, identifiers,
↪  titles, events, relationships, claims, and caveats.
- Prefer dense bullet notes or compact clauses. Remove filler,
↪  repeated wording, and low-value prose.
- If facts conflict, keep the conflict and source path instead of
↪  resolving it.
- Use at most {summary_max_tokens} tokens.
- Return only the updated notes. Do not include reasoning, analysis,
↪  preambles, or markdown fences.

CURRENT SUMMARY:
{current_summary}

NEXT DOCUMENT:
path: {path}
stream_position: {stream_position}
fragment: {fragment_index}/{num_fragments}
content:
{document_text}
```

For agentic rag, we use the following prompt, based on the smolagents implementation: https://huggingface.co/docs/smolagents/en/examples/rag.

Prompt

```
Use the retrieval history below to decide the next tool call.

Available tools:
1. search: retrieve documents for one new query.
2. final_results: stop and choose the final document paths.

Return only valid JSON, with one of these shapes:
{{"thought": "brief reason", "action": "search", "query": "new
↪  dense-retrieval query"}}
{{"thought": "brief reason", "action": "final_results",
↪  "document_paths": ["path/a.txt", "path/b.txt"]}}

Do not answer the question. Search queries should be specific,
↪  diverse, and useful for finding missing evidence.
If agent_searches_remaining is 0, return final_results.

PAYLOAD:
```