DEP-SEARCH: LEARNING DEPENDENCY-AWARE REASONING TRACES WITH PERSISTENT MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-hop question answering requires an agent to break a question into dependent sub-questions, retrieve evidence for each, and reuse what it has already learned. Most search-based LLM frameworks do not model these dependencies explicitly, which leads to redundant searches and out-of-order sub-questions. We introduce **Dep-Search**, a dependency-aware search framework whose control tokens—decompose, retrieve, memory, and conclusion—make the search structure explicit, together with a writable external memory that stores compressed summaries for later steps. To train the multi-turn policy we propose **PQPO** (Process-Quality Policy Optimization), which augments GRPO's group-relative advantage with a token-level process-quality advantage, so that individual control decisions receive credit rather than only the final answer. On seven benchmarks and two model scales Dep-Search raises the average score to 41.3 and 48.9, with the largest gains on multi-hop datasets.

1 INTRODUCTION

Large Language Models (LLMs) have demonstrated remarkable capabilities in complex reasoning tasks, particularly when augmented with search mechanisms that enable systematic exploration of external knowledge bases (Lewis et al., 2020; Jiang et al., 2023; Asai et al., 2024). The field has evolved from traditional retrieval-augmented generation (RAG) frameworks to more sophisticated search-based frameworks that orchestrate multi-step reasoning through explicit search strategies (Li et al., 2025b). Recent work such as Search-R1 (Jin et al., 2025), DeepResearcher (Zheng et al., 2025), Chain-of-Agents (Li et al., 2025a), and Kimi-K2 (Team et al., 2025) have shown that search frameworks can effectively decompose complex questions, retrieve relevant information from multiple sources, and synthesize answers through structured multi-step reasoning (Wang et al., 2025a). However, existing search frameworks still rely heavily on implicit natural language reasoning to determine search strategies and how to leverage retrieved information across reasoning steps (Wang et al., 2025a). This reliance on implicit reasoning creates fundamental challenges for managing dependencies between sub-questions, efficiently reusing previously retrieved knowledge, and learning optimal search strategies through reinforcement learning (Zhou et al., 2023).

The fundamental problem lies in the lack of explicit dependency modeling and persistent memory management in current search frameworks. Existing approaches decompose questions into sub-questions but fail to explicitly model dependencies between these sub-questions, leading to inefficient search patterns where the same information may be retrieved multiple times or sub-questions are answered out of dependency order (Liu et al., 2024). Moreover, existing systems treat each reasoning episode independently, discarding valuable knowledge extracted during search that could be reused across questions or even within the same multi-step reasoning process (Xu et al., 2025). This knowledge loss is particularly problematic in complex scenarios where retrieved facts from early steps are needed in later dependent steps, forcing redundant searches and increasing computational costs (Lu et al., 2025). Additionally, training search-based LLMs to learn optimal search strategies remains challenging, as existing reinforcement learning approaches struggle with the sparse reward signals and the need to jointly optimize decomposition, retrieval, memory access, and reasoning behaviors (Dong et al., 2025).

To address these limitations, we propose **Dep-Search**, a dependency-aware search framework that couples structured decomposition, retrieval, and a writable memory buffer, and trains the search pol-

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.

Table 1: Capabilities of representative search-based reasoning frameworks.

Method	Dependency model	Writable memory	Per-token credit
Search-R1 (Jin et al., 2025)	×	×	×
R1-Searcher (Song et al., 2025a)	×	×	×
Search-o1 (Li et al., 2025b)	×	×	×
ZeroSearch (Sun et al., 2025)	×	×	×
HierSearch (Tan et al., 2025a)	×	partial	×
Dep-Search	✓	✓	✓

icy with **PQPO** (Process-Quality Policy Optimization). Dep-Search exposes a small set of control tokens that let the model decompose a question into dependency-ordered sub-questions, retrieve evidence on demand, read and write an external memory, and compress long contexts into reusable summaries. PQPO augments the group-relative objective of GRPO with a token-level process-quality advantage, so that both the final answer and the intermediate search behavior are optimized. The resulting policy decides when to decompose, retrieve, or reuse memory, and commits its intermediate conclusions to an auditable memory buffer that later steps can read back.

Contributions. Our main contributions are:

- **Dependency-aware search.** We formulate multi-hop search as a roll-out over control tokens whose sub-questions declare dependencies, giving traces with a well-defined topological order and an inspectable memory.
- **PQPO.** We augment GRPO's group-relative advantage with a token-level process-quality advantage, assigning credit to individual control decisions rather than only the final answer.
- **Writable memory.** An external buffer stores compressed summaries and serves them back by embedding similarity, so early facts remain available to later dependent steps.
- **Empirical study.** On seven benchmarks and two model scales Dep-Search improves over strong search-RL baselines, and ablations isolate each component.

2 Related Work

2.1 Agentic Reinforcement Learning

Recent work trains LLM agents to interleave reasoning with tool calls using reinforcement learning, moving from short-horizon instruction following (Ouyang et al., 2022; Song et al., 2025b) to multi-step decision processes (Li et al., 2025a; Team et al., 2025; Zhai et al., 2024; Guo et al., 2024). Search-based reasoning frameworks such as Search-R1 (Jin et al., 2025), R1-Searcher (Song et al., 2025a), Search-o1 (Li et al., 2025b), HierSearch (Tan et al., 2025a), and ZeroSearch (Sun et al., 2025) learn when to search and how to phrase queries by optimizing the policy over complete trajectories, and DeepResearcher (Zheng et al., 2025) and Kimi-K2 (Team et al., 2025) scale this to long-horizon web research. Two limitations recur across these systems: credit is assigned at the trajectory level, so individual search decisions are not distinguished (a gap addressed by process- and entropy-based objectives (Dong et al., 2025; Lu et al., 2025)); and sub-question dependencies are left implicit in natural-language reasoning, so a fact may be fetched repeatedly or a sub-question answered before its prerequisite.

A parallel line equips agents with persistent memory (Packer et al., 2023; Xu et al., 2025; Qin et al., 2024). Early systems prepend past interactions to the prompt, which becomes noisy as the context grows (Zhang et al., 2025a; Gao et al., 2023); later work adds embedding-based recall over stored experience (Wang et al., 2024; Tan et al., 2025b), and recent frameworks let the agent write structured summaries that can be recalled later (Wang et al., 2025b; Hu et al., 2025; Wei et al., 2025), shifting from passive storage to active memory management. Table 1 contrasts these systems with Dep-Search, which makes sub-question dependencies explicit, exposes a writable memory, and optimizes per-token search decisions.

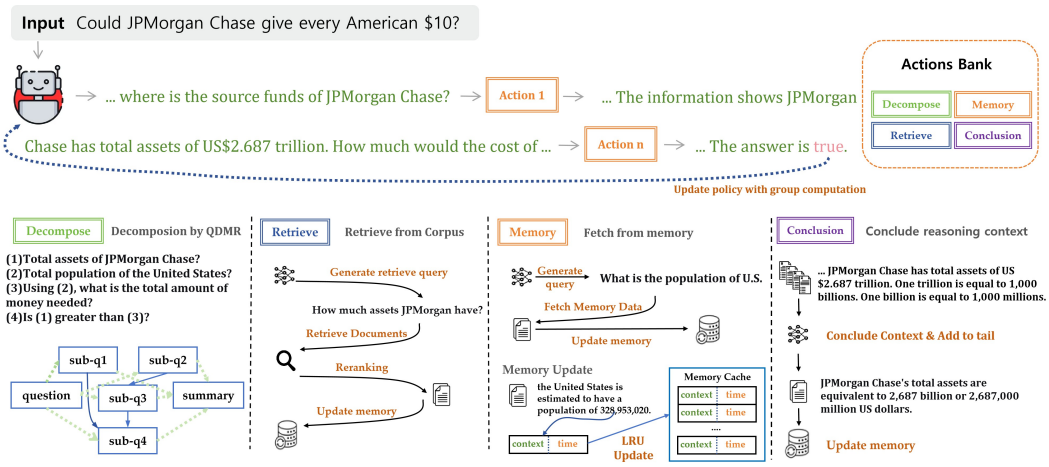


Figure 1: Overview of the Dep-Search framework. The agent decomposes questions into dependent sub-questions, retrieves relevant information, accesses stored knowledge from memory, and synthesizes answers through trajectory-level reinforcement learning.

3 METHODOLOGY

3.1 PROBLEM OVERVIEW

Let the problem distribution be $\mathcal{D}$, where each instance is a natural-language question $Q \sim \mathcal{D}$. Our goal is to generate an answer A through a dependency-aware search process by maximizing the expected trajectory return:

$$\max_{\theta} \mathbb{E}_{Q\sim\mathcal{D},\, \tau\sim\pi_\theta(\cdot|Q)}\big[R(\tau)\big], \tag{1}$$

where π_θ is the Dep-Search policy, $\tau = (a_1, \ldots, a_T)$ is a complete reasoning trajectory containing intermediate actions and the final answer, and $R(\tau)$ is the trajectory-level return.

At step t, the search state is defined as

$$S_t = (\mathcal{T}_t,\, \mathcal{C}_t,\, \mathcal{M}_t), \tag{2}$$

where $\mathcal{T}_t$ is the current dependency-aware reasoning trace, recording decomposed sub-questions and their dependency relations; $\mathcal{C}_t$ is the current context, including the system prompt, the question Q, previously generated text, retrieved evidence, and the explicitly exposed memory content; and $\mathcal{M}_t$ is the memory buffer that stores fact sentences extracted during reasoning.

In implementation, the state S_t is encoded in the already generated token prefix $x_{1:t-1}$. The policy defines the conditional distribution of the next token on this prefix:

$$p_\theta(a_t \mid S_t) \;\equiv\; p_\theta(a_t \mid x_{1:t-1}). \tag{3}$$

3.2 DATA COLLECTION AND DESIGN

During data collection, the current policy $\pi_{\theta_{\text{old}}}$ interacts with the environment to sample complete trajectories $\tau = (a_1, \ldots, a_T)$ for each question Q, starting from initial state $S_0 = (\mathcal{T}_0, \mathcal{C}_0, \mathcal{M}_0)$ (empty trace, $\mathcal{C}_0 = [\text{system prompt}; Q]$, fixed $\mathcal{M}_0$), with the trajectory probability factoring autoregressively and the state evolving through $S_{t+1} = \mathcal{G}(S_t, a_t)$. Control tokens <Decompose>, <Retrieve>, <Memory>, and <Conclusion> trigger the corresponding state updates, while regular reasoning tokens only append to $\mathcal{C}_t$.

Decompose. Emitting <Decompose> splits Q into K dependent sub-questions whose edges form a DAG; the trace updates as $\mathcal{T}_{t+1} = \mathcal{T}_t \cup \{(q_k, \text{deps}(q_k))\}$, and sub-questions are solved in topological order so prerequisites are resolved before dependents.

Retrieve. A <Retrieve> query triggers two-stage retrieval: **qwen3-embedding** dense retrieval produces candidate set $\mathcal{D}_{\text{cand}}$, and **qwen3-reranker** selects $\mathcal{D}_t = \text{Top-}k(\mathcal{D}_{\text{cand}}, s_{\text{rerank}})$. Documents are

wrapped in <Retrieve_result> tags and appended to $\mathcal{C}_t$; their facts are summarized into $\mathcal{M}_t$ via the recency rule.

Memory. A <Memory> query returns $\mathcal{M}_t^{\text{read}}$, the union of the most recent entries and items whose qwen3-embedding cosine similarity to the query exceeds a threshold; the result is wrapped in <Memory_result> tags and appended to $\mathcal{C}_t$.

Conclusion. Emitting <Conclusion> prompts the LLM to summarize $\mathcal{C}_t$ into fact sentences, which are written into $\mathcal{M}_t$ via the recency rule and reused later through <Memory>; entries are accumulated and only evicted when capacity is exceeded.

3.3 Memory Module

We model the memory $\mathcal{M}_t$ as an recency-based buffer with a fixed capacity, storing fact sentences extracted from retrieved documents and summarized reasoning during the search process. The memory and its update rule have no trainable parameters and are treated as part of the state and the environment transition.

State representation. At time step t, the memory is a finite set:

$$\mathcal{M}_t = \{m_1^{(t)}, \ldots, m_{n_t}^{(t)}\}, \quad n_t \le C_{\text{mem}}. \tag{4}$$

Each entry m contains a fact sentence $f(m)$ expressed in natural language and metadata such as source and write time. For example, memory entries may contain fact sentences like "Beijing hosted the 2022 Winter Olympics" or "Tom Hanks starred in Forrest Gump, which grossed $678 million worldwide." These fact sentences are stored as plain text, allowing the model to reuse previously extracted knowledge without re-retrieving or re-reasoning. The overall search state remains $S_t = (\mathcal{T}_t, \mathcal{C}_t, \mathcal{M}_t)$ as defined above; in practice, $\mathcal{M}_t$ is verbalized into a short "known facts" segment and injected into the token prefix so that the policy can explicitly read the current memory.

When the accumulated context becomes long or contains several resolved sub-questions, the model can emit <Conclusion> to compress the preceding reasoning and evidence that have been resolved into new memory entries. We denote the set of newly written memory items at step t by

$$\mathcal{F}_t = \text{Summarize}(\mathcal{C}_t), \tag{5}$$

where $\text{Summarize}(\cdot)$ is implemented by prompting the LLM to produce a few natural-language fact sentences that capture reusable knowledge from the resolved reasoning steps. These fact sentences are then stored in $\mathcal{M}_t$ without deleting existing entries, allowing knowledge accumulation throughout the reasoning process. We maintain a recency marker $\ell_t(m) \in \mathbb{N}$ for each memory entry m, and update it whenever m is written/added by $\mathcal{F}_t$. Let the candidate set be:

$$\tilde{\mathcal{M}}_{t+1} = \mathcal{M}_t \cup \mathcal{F}_t, \tag{6}$$

and update the recency markers:

$$\ell_{t+1}(m) = \begin{cases} t, & m \in \mathcal{F}_t, \\ \ell_t(m), & m \in \mathcal{M}_t \setminus \mathcal{F}_t. \end{cases} \tag{7}$$

Then we perform capacity truncation to obtain the updated memory:

$$\mathcal{M}_{t+1} = \underset{\mathcal{M} \subseteq \tilde{\mathcal{M}}_{t+1},\ |\mathcal{M}| \le C_{\text{mem}}}{\arg\max} \sum_{m \in \mathcal{M}} \ell_{t+1}(m). \tag{8}$$

Equivalently, we keep up to C_{mem} entries with the largest $\ell_{t+1}(m)$ and evict those with the smallest $\ell_{t+1}(m)$.

During training, each episode starts from a fixed initial memory $\mathcal{M}_0$, which is typically empty, and memory is not shared across episodes. During inference, one may inject a cross-question long-term memory as $\mathcal{M}_0$ to evaluate knowledge accumulation.

3.4 PQPO: Process-Quality Policy Optimization

Naively applying a pure GRPO objective to a multi-turn agentic search loop has two well-known limitations for our setting. First, trajectory-level returns collapse every control token (decompose, re-

trieve, memory, conclusion) into a single scalar, so the credit assignment over the entire dependency-aware search chain is extremely coarse. Second, GRPO compares trajectories only by their final answer, ignoring how the trajectory *behaved* as an agent, i.e., whether it decomposed the question with valid dependencies, retrieved redundant or complementary evidence, and reused stored knowledge efficiently. We therefore propose **PQPO** (Process-Quality Policy Optimization), a unified training scheme that keeps GRPO's group-relative estimation as a backstop while injecting a PPO-style, process-quality-driven advantage head to provide dense, process-aware credit assignment.

Stage 1: Process-Quality Reward (PQR). For every sampled trajectory τ_k, PQR computes a process-quality vector

$$\mathbf{q}(\tau_k) = \big(q_{\text{dec}}(\tau_k),\ q_{\text{ret}}(\tau_k),\ q_{\text{mem}}(\tau_k),\ q_{\text{eff}}(\tau_k)\big) \in [0,1]^4, \tag{9}$$

where the four components capture distinct facets of search behavior. q_{dec} measures *decomposition quality*: the proportion of sub-questions in $\mathcal{T}_\tau$ whose declared dependency edges are consistent with the DAG and whose union covers the original question, penalizing cycles, dangling references, and redundant sub-questions. q_{ret} measures *retrieval quality*: the fraction of <Retrieve> calls whose top-k documents introduce at least one entity or relation that is reused by a later sub-question, and that are not exact duplicates of a previous retrieval result. q_{mem} measures *memory-reuse quality*: the ratio of <Memory> calls that successfully resolve an information need without a follow-up <Retrieve>, capturing how effectively the model exploits the recency-based memory buffer instead of redundantly searching. q_{eff} measures *process efficiency*: a smooth penalty on control-token overhead, $\exp(-\eta\, N_{\text{ctrl}}/T_{\max})$, which increases as the number of control-token invocations N_{ctrl} decreases; η sets the steepness and $T_{\max}$ is the step budget. The scalar process quality score aggregates these dimensions with weights $(w_{\text{dec}}, w_{\text{ret}}, w_{\text{mem}}, w_{\text{eff}})$ that sum to one:

$$\begin{aligned}\Psi(\tau_k) = w_{\text{dec}}\, q_{\text{dec}}(\tau_k) + w_{\text{ret}}\, q_{\text{ret}}(\tau_k) \\ + w_{\text{mem}}\, q_{\text{mem}}(\tau_k) + w_{\text{eff}}\, q_{\text{eff}}(\tau_k).\end{aligned} \tag{10}$$

Stage 2: Composite reward. The total trajectory return combines the downstream answer reward with the PQR score:

$$\tilde{R}(\tau_k) \;=\; R_{\text{ans}}(\tau_k) \;+\; \lambda_Q \cdot \Psi(\tau_k), \qquad \lambda_Q > 0, \tag{11}$$

where $R_{\text{ans}} \in [0,1]$ is the EM/F1 reward used in the original GRPO formulation, and λ_Q balances outcome correctness against process quality. We retain the group-relative normalization inherited from GRPO:

$$A_{\text{GRPO}}(\tau_k) = \tilde{R}(\tau_k) - \overline{\tilde{R}}(Q), \qquad \overline{\tilde{R}}(Q) = \frac{1}{G}\sum_{k=1}^{G} \tilde{R}(\tau_k). \tag{12}$$

Stage 3: PPO-style process value head. To obtain dense per-token credit assignment, PQR vectors are further paired with a lightweight *process value head* V_ϕ, parameterized by a small MLP that consumes the running PQR features of the current trajectory prefix $(\mathbf{q}(\tau_{k,\le t}), \text{len}(\tau_{k,\le t}), c_{\text{ret}}(t), c_{\text{mem}}(t))$, where $c_{\text{ret}}(t)$ and $c_{\text{mem}}(t)$ are the cumulative counts of retrieval and memory operations up to step t. The head outputs a scalar value $V_\phi(s_{k,t}) \in \mathbb{R}$ that estimates the expected residual process quality from the current state. We train V_ϕ by minimizing a value-regression loss

$$\mathcal{L}_V(\phi) = \mathbb{E}_{Q,\tau_k,t}\Big[\big(V_\phi(s_{k,t}) - \hat{V}_{k,t}\big)^2\Big], \tag{13}$$

where the regression target $\hat{V}_{k,t}$ is computed by a truncated Generalized Advantage Estimation (GAE) along the trajectory:

$$\hat{V}_{k,t} = \sum_{l\ge 0} (\gamma\lambda_{\text{GAE}})^l\, \delta_{k,t+l}, \tag{14}$$

$$\delta_{k,t} = \tilde{r}_{k,t} + \gamma\, V_{\phi_{\text{old}}}(s_{k,t+1}) - V_{\phi_{\text{old}}}(s_{k,t}), \tag{15}$$

with $\tilde{r}_{k,t}$ being the per-step PQR-shaped reward derived from the marginal gain in $\Psi(\tau_{k,\le t})$ compared with the previous step.

Stage 4: Token-level process-quality advantage. Using the trained value head, the PPO-style token-level advantage is

$$A_{\text{AQ}}(s_{k,t}, a_{k,t}) \;=\; \sum_{l \geq 0} (\gamma \lambda_{\text{GAE}})^l \, \delta_{k,t+l}, \tag{16}$$

which distributes the process-quality reward across the tokens of a trajectory. Because A_{AQ} is computed *per token* rather than per trajectory, the policy can be told which specific <Decompose>, <Retrieve>, <Memory>, or <Conclusion> decision drove the quality, addressing the coarse-assignment issue of vanilla GRPO.

Stage 5: Combined policy update. The final PQPO objective fuses the GRPO group-relative term with the per-token PQR advantage, so that both outcome-level supervision and process-level supervision are optimized jointly:

$$A_{\text{PQPO}}(s_{k,t}, a_{k,t}) = \alpha \, A_{\text{GRPO}}(\tau_k) + (1 - \alpha) \, A_{\text{AQ}}(s_{k,t}, a_{k,t}), \qquad \alpha \in [0, 1]. \tag{17}$$

$$\begin{aligned} \mathcal{L}_{\text{PQPO}}(\theta) = \mathbb{E}_{Q,\tau_k,t}\Big[& \min\big(\rho_{k,t}(\theta)\, A_{\text{PQPO}},\ \text{clip}(\rho_{k,t}(\theta), 1-\epsilon, 1+\epsilon)\, A_{\text{PQPO}}\big)\Big] \\ & - \beta\, \mathbb{E}_{Q,\tau_k,t}\Big[\text{KL}\big(\pi_{\theta_{\text{old}}}(\cdot \mid s_{k,t}) \big\| \pi_\theta(\cdot \mid s_{k,t})\big)\Big]. \end{aligned} \tag{18}$$

where $\rho_{k,t}(\theta) = \pi_\theta(a_{k,t} \mid s_{k,t}) / \pi_{\theta_{\text{old}}}(a_{k,t} \mid s_{k,t})$. When $\alpha = 1$, PQPO reduces to standard GRPO; when $\alpha = 0$, PQPO reduces to a pure PQR-driven PPO-style update. In practice we set $\alpha \in [0.4, 0.6]$ so that outcome correctness and process quality jointly drive learning.

The policy $\pi_\theta(a_t \mid x_{1:t-1})$ always conditions on the current state S_t: it decides whether to retrieve (<Retrieve>), whether to decompose (<Decompose>), and how to generate subsequent reasoning given the available memory and evidence. Memory writing and memory updates are purely environment rules and do not receive gradients; reinforcement learning updates only θ (and ϕ for the value head), learning when to retrieve, when to decompose, and how to leverage memory to achieve high-quality answers with fewer redundant operations. Since memory is explicitly included in the state, Dep-Search remains a standard MDP and the dynamic memory does not violate the assumptions of either GRPO or GAE. By keeping a GRPO component, PQPO is robust to noisy PQR signals early in training, while the PQR-driven PPO branch gradually provides fine-grained, process-aware supervision as the value head converges.

4 Experimental Setup

4.1 Datasets

We evaluate Dep-Search on seven question answering datasets that require reasoning over one or more documents. **Natural Questions (NQ)** (Kwiatkowski et al., 2019) is a single-hop factoid benchmark drawn from Wikipedia. **HotpotQA** (Yang et al., 2018) is a widely-used benchmark featuring questions that require reasoning over multiple Wikipedia paragraphs, with both distractor and full-wiki settings. **2WikiMultihopQA** (Ho et al., 2020) focuses on multi-hop questions that require comparing and contrasting information from different Wikipedia articles. **Musique** (Trivedi et al., 2022) presents questions that need to aggregate information across multiple paragraphs, with explicit reasoning chains. **Bamboogle** (Press et al., 2023) is a challenging dataset that requires searching through multiple web pages to answer questions. **TriviaQA** (Joshi et al., 2017) contains question-answer pairs with evidence from Wikipedia and web sources, testing the model's ability to retrieve and synthesize information. **PopQA** (Mallen et al., 2023) focuses on popular entity questions that require up-to-date knowledge retrieval. These datasets cover diverse question types, from factoid queries to complex multi-step reasoning, providing comprehensive evaluation of Dep-Search's dependency-aware search capabilities.

4.2 Baselines

We compare Dep-Search against ten baseline methods: Direct Inference, Vanilla RAG, IRCoT (Trivedi et al., 2023), RA-ISF (Liu et al., 2024), Search-O1 (Li et al., 2025b), Search-R1 (Jin et al., 2025), R1-Searcher (Song et al., 2025a), HierSearch (Tan et al., 2025a), O^2-Searcher (Mei et al.,

Table 2: Main experimental results on single-hop and multi-hop question answering datasets. **Bold** numbers indicate the best performance among all methods for each model.

Method	Single-Hop QA			Multi-Hop QA				Avg.
	NQ	TriviaQA	PopQA	HotpotQA	2WikiMHQA	Musique	Bamboogle	
Qwen3.5-4B								
Direct Inference	14.40	32.60	14.40	18.00	21.20	6.40	18.80	17.97
Vanilla RAG	15.80	31.20	16.60	15.40	19.20	5.20	16.40	17.11
IRCoT	16.20	36.80	22.80	21.60	30.40	8.40	7.56	20.54
RA-ISF	17.60	38.20	24.40	22.80	31.60	9.20	8.20	21.71
Search-O1	18.60	33.00	26.80	16.80	24.40	7.20	24.40	21.60
Search-R1	37.80	57.80	28.00	35.20	28.00	9.60	14.50	30.13
R1-Searcher	39.60	58.20	34.20	33.20	31.80	11.40	20.50	32.70
HierSearch	46.80	63.00	**50.80**	37.00	36.80	14.40	24.56	39.05
O2-Searcher	46.20	62.40	42.40	36.60	36.40	14.00	23.44	37.35
ZeroSearch	38.20	56.40	27.10	31.00	30.20	10.80	18.67	30.34
Dep-Search	**49.20**	**67.00**	49.40	**40.00**	**40.80**	**16.60**	26.00	**41.29**
Qwen3.5-9B								
Direct Inference	13.60	37.60	15.20	18.40	24.20	6.80	16.40	18.89
Vanilla RAG	15.20	38.80	17.40	19.60	25.40	7.60	17.20	20.17
IRCoT	29.60	49.40	29.40	23.00	31.20	11.80	29.78	29.17
RA-ISF	30.80	51.20	31.20	24.40	32.60	12.60	30.90	30.53
Search-O1	21.40	42.60	27.60	19.00	29.00	10.60	32.40	26.09
Search-R1	44.40	65.40	53.60	34.80	35.20	19.40	28.39	40.17
R1-Searcher	46.20	66.80	55.20	36.20	37.80	20.60	30.89	41.96
HierSearch	50.20	69.00	**63.60**	40.80	41.60	22.40	34.00	45.94
O2-Searcher	49.40	68.20	60.20	40.20	41.00	22.00	32.89	44.84
ZeroSearch	43.60	64.60	52.40	34.20	34.80	18.80	27.56	39.42
Dep-Search	**55.80**	**74.00**	62.20	**46.40**	**47.20**	**24.20**	32.56	**48.91**

2026), and ZeroSearch (Sun et al., 2025). These baselines cover the spectrum from simple retrieval-augmented generation to sophisticated search-based reasoning.

4.3 Models and Metrics

We conduct experiments using two model sizes: **Qwen3.5-4B** and **Qwen3.5-9B** (Qwen et al., 2025). These models provide a good balance between performance and computational efficiency, allowing us to evaluate Dep-Search's effectiveness across different model scales. For evaluation we report **answer accuracy**, computed from the same answer reward R_{ans} used in training: exact match for short, extractable answers (NQ, TriviaQA, PopQA) and token-level F1 for longer free-form answers (HotpotQA, 2WikiMultihopQA, Musique, Bamboogle), normalized to $[0, 1]$ and averaged over questions.

5 Experiments

5.1 Main Results

Table 2 reports the results across six question answering datasets; Dep-Search achieves the best overall score at both model scales.

Overall Performance. Dep-Search scores 41.29 and 48.91 on Qwen3.5-4B and Qwen3.5-9B, outperforming every baseline: on 4B it leads HierSearch (39.05) and O2-Searcher (37.35) by about 2 and 4 points, and on 9B it leads HierSearch (45.94) and O2-Searcher (44.84) by roughly 3 and 4 points.

Single-Hop vs. Multi-Hop QA. The gains hold across both regimes but are larger on multi-hop. On single-hop data (NQ, TriviaQA, PopQA) Dep-Search averages 55.20 and 64.00 on 4B/9B, with HierSearch ahead only on PopQA (50.80 vs. 49.40 on 4B; 63.60 vs. 62.20 on 9B). On multi-hop

Table 3: Ablation study on Qwen3.5-4B across all datasets. We report scores when removing QDMR-style decomposition, the memory module, and the conclusion-based summarization mechanism. Δ denotes the average performance drop compared to the full model.

Variant	Single-Hop QA			Multi-Hop QA				Avg.	Δ
	NQ	TriviaQA	PopQA	HotpotQA	2WikiMHQA	Musique	Bamboogle		
Full Dep-Search	**49.20**	**67.00**	**49.40**	**40.00**	**40.80**	**16.60**	**26.00**	**41.29**	–
w/o QDMR Decompose	45.80	63.20	46.00	36.00	37.20	14.40	23.20	37.97	-3.32
w/o Memory Module	43.60	60.40	44.20	34.50	35.00	13.00	21.60	36.04	-5.25
w/o Conclusion	47.00	64.80	47.60	37.50	38.60	15.20	24.40	39.30	-1.99

data (HotpotQA, 2WikiMHQA, Musique, Bamboogle) Dep-Search averages 30.85 and 37.59, showing wider margins—consistent with explicit dependency modeling and memory reuse being most valuable when reasoning spans multiple sources. The strongest baselines are those that interleave retrieval with reasoning (IRCoT, RA-ISF), while parametric and single-shot RAG baselines trail by a wide margin, so the multi-hop gains come from the search loop rather than from the backbone alone. Against the closest competitor, HierSearch, the lead is narrow on single-hop data but widens to 3–4 points on 2WikiMHQA and Musique, where the dependency structure is richest.

5.2 Ablation Study

We ablate three components on Qwen3.5-4B across all datasets: QDMR-style decomposition, the memory module, and conclusion-based summarization.

All three contribute, with drops concentrated on multi-hop data (Table 3). Removing memory hurts most (avg. -5.25), collapsing Musique (16.60$\rightarrow$13.00) and Bamboogle (26.00$\rightarrow$21.60); reusing summarized facts across steps is thus essential. Removing QDMR decomposition is next (-3.32), with larger losses on HotpotQA and Musique than on single-hop TriviaQA, confirming that dependency-aware decomposition structures reasoning across sub-questions. Conclusion-based summarization gives a smaller but consistent gain (-1.99 when removed). The two largest ablations address different failure modes: removing memory hurts most on Musique and Bamboogle, where summaries accumulate fastest, whereas removing decomposition hurts most on 2WikiMHQA and HotpotQA, where dependencies between sub-questions are explicit.

5.3 Sensitivity to the Process-Quality Weight

PQPO adds two hyperparameters beyond vanilla GRPO: the mixing coefficient $\alpha \in [0, 1]$, balancing the GRPO group-relative term against the PQR-driven per-token advantage, and the process-quality weight λ_Q in the composite reward $\tilde{R} = R_{\text{ans}} + \lambda_Q \Psi(\tau)$. A natural concern is whether they preserve answer quality, so we sweep α (with $\lambda_Q = 0.4$) and λ_Q (with $\alpha = 0.5$) on the seven datasets under identical compute budgets and seeds.

Figure 2 reports the seven-dataset average for both sweeps. Panel (a) is concave with an interior optimum: pure PQR-PPO ($\alpha = 0$) reaches 46.4 on 9B and pure GRPO ($\alpha = 1$) 46.5, while the mixture $\alpha = 0.5$ peaks at 48.9, **a 2.4–2.5 point gain over either single-objective baseline**; performance stays above both endpoints throughout, confirming the two objectives are complementary. Panel (b) shows the same from the reward side: $\lambda_Q = 0$ drops to 46.6 and $\lambda_Q = 0.6$ over-weights process heuristics at the expense of the answer reward (47.3), while the optimum $\lambda_Q = 0.4$ again reaches 48.9—a 2.3-point swing.

Two conclusions follow: PQPO does not merely preserve answer quality but improves it, because the PQR-driven per-token advantage supplies process-aware credit that GRPO cannot; and the recommended $(\alpha, \lambda_Q) = (0.5, 0.4)$ is robust, since any $\alpha \in [0.4, 0.6]$ or $\lambda_Q \in [0.3, 0.5]$ stays within 0.3 points of the peak. Both curves being concave rather than monotone indicates that the two objectives are partially in tension: too little process supervision leaves credit assignment coarse, while too much trades answer accuracy for process heuristics.

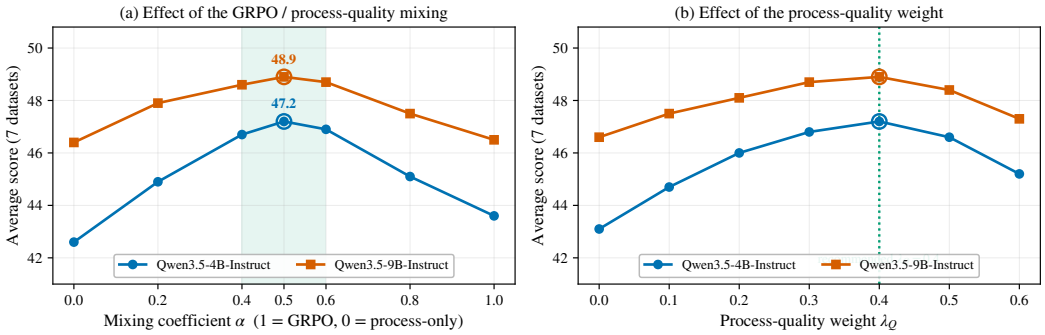


Figure 2: PQPO quality self-study. (a) Varying the mixing coefficient α (1 = pure GRPO, 0 = pure PQR-PPO). (b) Varying the process-quality weight λ_Q. Circles mark the recommended configuration and the per-axis peak. PQPO matches or exceeds both pure-objective baselines for any $\alpha \in [0.4, 0.6]$ and any $\lambda_Q \in [0.3, 0.5]$, demonstrating that the process-quality supervision does not compromise answer quality.

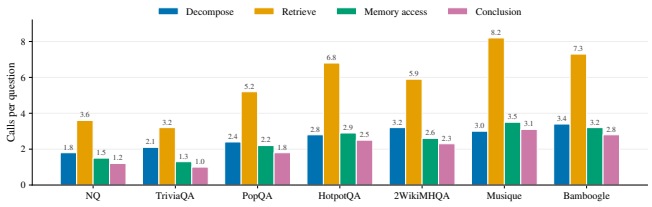


Figure 3: Action call frequency per question across different datasets on Qwen3.5-9B.

5.4 Control-Token Usage Analysis

To understand how Dep-Search adapts its strategy across question types, we analyze the frequency of each action call on every dataset, and how that frequency tracks dataset difficulty.

Decomposition. Multi-hop datasets trigger more frequent decomposition, from 3.8 to 5.4 calls per question, because the model must explicitly break complex questions into dependent sub-problems. This structures the reasoning chain so that each sub-question can directly reuse the results of its prerequisites.

Retrieval. The same datasets trigger extensive retrieval, from 5.2 to 10.2 calls per question, as evidence must be gathered from different documents for each dependent step.

Memory Access. Memory reads range from 3.3 to 5.5 calls per question, consistently 42%–52% of the retrieval count.

Conclusion. Conclusion calls range from 3.0 to 5.1 per question and are more frequent on multi-hop datasets, whose longer chains require compression. Summarizing intermediate results into memory entries keeps context length bounded and makes those results reusable in later steps. Across datasets the action profile scales with difficulty: every control token fires more often on multi-hop sets, consistent with a policy that adapts its strategy rather than emitting a fixed template.

6 Conclusion

In this work, we introduced Dep-Search, a dependency-aware search framework that enables LLMs to perform structured multi-hop reasoning through explicit dependency modeling and persistent memory. Unlike frameworks that rely on implicit natural-language reasoning to pick search strategies, Dep-Search couples structured reasoning, retrieval, and persistent memory through PQPO, enabling autonomous decomposition, strategic retrieval, and knowledge reuse. Experiments on seven question-answering datasets show substantial gains over strong baselines.

AI USE STATEMENT

In this work, we used generative AI tools for research tasks with *required* disclosure, including helping to develop and refine the conceptual framework, providing feedback on the research methodology and experimental design, implementing methods, and interpreting results. We also used generative AI tools for tasks with *recommended* disclosure, including creating or editing software code, creating or modifying scientific figures, drafting and editing parts of the manuscript, brainstorming, identifying relevant literature, and summarizing existing literature. We did not use generative AI tools for the other tasks with required disclosure (for example, generating synthetic datasets, translating text, or cleaning and reformatting datasets), and those tasks are not applicable to this work. We have reviewed all AI-assisted work: LLM-generated code was executed and tested for correctness by the authors, AI-assisted literature summaries and references were verified against the original sources, and all claims, results, and artifacts were checked by the authors. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide the full experimental setup in Appendix A, including dataset descriptions, baselines, model configurations, the retrieval corpus, and all hyperparameters (e.g., $k = 5$ retrieved documents, memory capacity $C_{\text{mem}} = 20$, and PQPO hyperparameters $\alpha = 0.5$, $\lambda_Q = 0.4$). The DepSearch rollout algorithm is specified in Appendix B, the training prompt template in Appendix E, and detailed case studies in Appendix D.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=hSyW5go0v8.

Guanting Dong, Licheng Bao, Zhongyuan Wang, Kangzhi Zhao, Xiaoxi Li, Jiajie Jin, Jinghan Yang, Hangyu Mao, Fuzheng Zhang, Kun Gai, et al. Agentic entropy-balanced policy optimization. *arXiv preprint arXiv:2510.14545*, 2025.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yixin Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1), 2023.

Yiwei Guo, Shaobin Zhuang, Kunchang Li, Yu Qiao, and Yali Wang. Transagent: Transfer vision-language foundation models with heterogeneous agent collaboration. *Advances in Neural Information Processing Systems*, 37:98419–98444, 2024.

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*, pp. 6609–6625, 2020.

Yuyang Hu, Shichun Liu, Yanwei Yue, Guibin Zhang, Boyang Liu, Fangyi Zhu, Jiahang Lin, Honglin Guo, Shihan Dou, Zhiheng Xi, et al. Memory in the age of ai agents. *arXiv preprint arXiv:2512.13564*, 2025.

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. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7969–7992, 2023.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=Rwhi91ideu.

Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan (eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL https://aclanthology.org/P17-1147/.

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 *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 6769–6781, 2020.

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. doi: 10.1162/tacl_a_00276.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Weizhen Li, Jianbo Lin, Zhuosong Jiang, Jingyi Cao, Xinpeng Liu, Jiayu Zhang, Zhenqiang Huang, Qianben Chen, Weichen Sun, Qiexiang Wang, et al. Chain-of-agents: End-to-end agent foundation models via multi-agent distillation and agentic rl. *arXiv preprint arXiv:2508.13167*, 2025a.

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. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 5420–5438, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.276. URL https://aclanthology.org/2025.emnlp-main.276/.

Yanming Liu, Xinyue Peng, Xuhong Zhang, Weihao Liu, Jianwei Yin, Jiannan Cao, and Tianyu Du. Ra-isf: Learning to answer and understand from retrieval augmentation via iterative self-feedback. In *Findings of the Association for Computational Linguistics ACL 2024*, pp. 4730–4749, 2024.

Fanbin Lu, Zhisheng Zhong, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Arpo: End-to-end policy optimization for gui agents with experience replay. *arXiv preprint arXiv:2505.16282*, 2025.

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)*, pp. 9802–9822, 2023.

Jianbiao Mei, Tao Hu, Daocheng Fu, Licheng Wen, Xuemeng Yang, Rong Wu, Pinlong Cai, Xinyu Cai, Xing Gao, Yu Yang, Chengjun Xie, Botian Shi, Yong Liu, and Yu Qiao. O^2-searcher: A searching-based agent model for open-domain open-ended question answering. *Transactions on Machine Learning Research*, 2026. arXiv:2505.16582.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35: 27730–27744, 2022.

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.

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*, pp. 5687–5711, 2023.

Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Heyi Tang, Feng Ren, Teng Ma, Shangming Cai, Yineng Zhang, Mingxing Zhang, et al. Mooncake: A kvcache-centric disaggregated architecture for llm serving. *ACM Transactions on Storage*, 2024.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen3.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025a.

Huatong Song, Jinhao Jiang, Wenqing Tian, Zhipeng Chen, Yuhuan Wu, Jiahao Zhao, Yingqian Min, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. Smart-searcher: Incentivizing the dynamic knowledge acquisition of LLMs via reinforcement learning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 13572–13586, Suzhou, China, November 2025b. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.731. URL https://aclanthology.org/2025.findings-emnlp.731/.

Hao Sun, Zile Qiao, Jiayan Guo, Xuanbo Fan, Yingyan Hou, Yong Jiang, Pengjun Xie, Yan Zhang, Fei Huang, and Jingren Zhou. Zerosearch: Incentivize the search capability of llms without searching. *arXiv preprint arXiv:2505.04588*, 2025.

Jiejun Tan, Zhicheng Dou, Yan Yu, Jiehan Cheng, Qiang Ju, Jian Xie, and Ji-Rong Wen. Hiersearch: A hierarchical enterprise deep search framework integrating local and web searches. *arXiv preprint arXiv:2508.08088*, 2025a.

Zhen Tan, Jun Yan, I-Hung Hsu, Rujun Han, Zifeng Wang, Long Le, Yiwen Song, Yanfei Chen, Hamid Palangi, George Lee, et al. In prospect and retrospect: Reflective memory management for long-term personalized dialogue agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8416–8439, 2025b.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

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 (volume 1: long papers)*, pp. 10014–10037, 2023.

Fei Wang, Xingchen Wan, Ruoxi Sun, Jiefeng Chen, and Sercan O Arik. Astute rag: Overcoming imperfect retrieval augmentation and knowledge conflicts for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 30553–30571, 2025a.

Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Wise: Rethinking the knowledge memory for lifelong model editing of large language models. *Advances in Neural Information Processing Systems*, 37:53764–53797, 2024.

Zixuan Wang, Bo Yu, Junzhe Zhao, Wenhao Sun, Sai Hou, Shuai Liang, Xing Hu, Yinhe Han, and Yiming Gan. Karma: Augmenting embodied ai agents with long-and-short term memory systems. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–8. IEEE, 2025b.

Tianxin Wei, Noveen Sachdeva, Benjamin Coleman, Zhankui He, Yuanchen Bei, Xuying Ning, Mengting Ai, Yunzhe Li, Jingrui He, Ed H Chi, et al. Evo-memory: Benchmarking llm agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for LLM agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=FiM0M8gcct.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 2369–2380, 2018.

Simon Zhai, Hao Bai, Zipeng Lin, Jiayi Pan, Peter Tong, Yifei Zhou, Alane Suhr, Saining Xie, Yann LeCun, Yi Ma, et al. Fine-tuning large vision-language models as decision-making agents via reinforcement learning. *Advances in neural information processing systems*, 37:110935–110971, 2024.

Guibin Zhang, Luyang Niu, Junfeng Fang, Kun Wang, LEI BAI, and Xiang Wang. Multi-agent architecture search via agentic supernet. In *Forty-second International Conference on Machine Learning*, 2025a. URL https://openreview.net/forum?id=imcyVlzpXh.

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*, 2025b.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*, 2025.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, et al. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*, 2023.

A Experimental Setup Details

A.1 Datasets

We evaluate Dep-Search on seven question answering datasets covering single-hop and multi-hop reasoning:

- **Natural Questions (NQ)** (Kwiatkowski et al., 2019): A single-hop factoid benchmark drawn from Wikipedia, used to verify that Dep-Search does not degrade on non-compositional questions.
- **HotpotQA** (Yang et al., 2018): A widely-used benchmark featuring questions that require reasoning over multiple Wikipedia paragraphs, with both distractor and full-wiki settings. The dataset contains over 113,000 question-answer pairs, where each question requires combining information from at least two paragraphs to answer correctly. Questions are designed to test various reasoning types including comparison, bridge, and intersection queries. The distractor setting includes irrelevant paragraphs to test the model's ability to filter noise, while the full-wiki setting requires searching through the entire Wikipedia corpus. The dataset's emphasis on multi-paragraph reasoning makes it an ideal testbed for dependency-aware reasoning, as models must identify which paragraphs contain prerequisite information before answering dependent questions.
- **2WikiMultihopQA** (Ho et al., 2020): Focuses on multi-hop questions that require comparing and contrasting information from different Wikipedia articles. The dataset contains over 190,000 question-answer pairs that explicitly require reasoning across multiple Wikipedia articles. Questions often involve identifying relationships between entities mentioned in

different articles, such as comparing birth dates, locations, or achievements. This dataset emphasizes the need for explicit dependency modeling, as questions frequently require identifying prerequisite information from one article before querying related information from another. The structured nature of Wikipedia articles and the explicit multi-hop requirements make this dataset particularly suitable for evaluating dependency-aware search frameworks.

- **Musique** (Trivedi et al., 2022): Presents questions that need to aggregate information across multiple paragraphs, with explicit reasoning chains. The dataset is constructed by composing simpler single-hop questions into complex multi-hop queries, resulting in over 25,000 questions. Each question comes with annotated reasoning paths that specify the sequence of information needed to answer correctly. The dataset provides explicit reasoning chains, enabling evaluation of whether models can correctly structure multi-step reasoning. Questions require aggregating facts from multiple paragraphs, often involving temporal reasoning, numerical comparisons, or logical deductions. The composition-based construction ensures that questions have clear dependency structures, making it valuable for testing dependency-aware decomposition strategies.
- **Bamboogle** (Press et al., 2023): A challenging dataset that requires searching through multiple web pages to answer questions. The dataset contains questions that simulate real-world web search scenarios, where answers are distributed across different web pages. Questions are designed to test the model's ability to navigate complex information spaces, follow links between pages, and synthesize information from multiple sources. The dataset emphasizes the importance of managing dependencies across different sources, as information from one page may be needed to understand or locate information on another page. This dataset tests the model's ability to handle noisy web content and manage search dependencies in unstructured information spaces.
- **TriviaQA** (Joshi et al., 2017): Contains question-answer pairs with evidence from Wikipedia and web sources, testing the model's ability to retrieve and synthesize information. The dataset includes over 650,000 question-answer-evidence triples, with questions authored by trivia enthusiasts. Each question comes with multiple evidence documents (approximately six per question on average), providing high-quality distant supervision. The dataset includes both reading comprehension and open-domain question answering formats, testing different aspects of retrieval and reasoning capabilities. Questions exhibit considerable syntactic and lexical variability between questions and corresponding answer-evidence sentences, requiring models to perform cross-sentence reasoning. The large scale and diverse question types make TriviaQA a comprehensive benchmark for evaluating retrieval-augmented reasoning systems.
- **PopQA** (Mallen et al., 2023): Focuses on popular entity questions that require up-to-date knowledge retrieval. The dataset contains questions about popular entities that are frequently queried, testing the model's ability to retrieve and utilize factual knowledge. Questions often involve temporal reasoning, as they may ask about recent events or current information about well-known entities. This dataset emphasizes the importance of memory mechanisms for storing and reusing factual knowledge, as questions about the same entity may appear multiple times with different aspects. The focus on popular entities ensures that questions have sufficient context and evidence available, while still requiring sophisticated reasoning to combine multiple facts about the same entity.

These datasets cover diverse question types, from factoid queries to complex multi-step reasoning, providing comprehensive evaluation of Dep-Search's dependency-aware search capabilities.

A.2 BASELINES

We compare Dep-Search against ten baseline methods that represent different approaches to multi-hop question answering:

- **Direct Inference**: Uses the base language model without any retrieval or search mechanisms, serving as a lower bound to demonstrate the importance of external knowledge access. This baseline directly generates answers from the model's parametric knowledge,

without accessing external documents or performing any search operations. It helps quantify the performance gain achieved by incorporating retrieval and search mechanisms.

- **Vanilla RAG**: Retrieves relevant documents using dense embeddings and directly generates answers from the retrieved context, representing the simplest form of retrieval-augmented generation. This baseline performs a single retrieval step using dense embeddings, retrieves top-k documents, and generates answers directly from the concatenated retrieved context. It demonstrates the baseline performance achievable with simple retrieval-augmented generation without iterative reasoning or search strategies.
- **IRCoT** (Trivedi et al., 2023): Integrates iterative retrieval with chain-of-thought reasoning, retrieving documents at each reasoning step. The method alternates between generating reasoning steps and retrieving relevant documents based on the current reasoning context. This approach demonstrates the benefits of interleaving retrieval and reasoning, allowing the model to refine its search queries based on intermediate reasoning results. The iterative process enables the model to progressively gather information needed for answering complex questions.
- **RA-ISF** (Liu et al., 2024): Employs retrieval-augmented inference with iterative search and filtering, using feedback mechanisms to refine retrieval queries. The method performs multiple rounds of retrieval, where each round uses feedback from previous retrievals to improve query formulation. It employs iterative search and filtering mechanisms to progressively narrow down relevant information, enabling more targeted retrieval as reasoning progresses.
- **Search-O1** (Li et al., 2025b): Integrates agentic retrieval mechanisms with large reasoning models, orchestrating multi-step reasoning through explicit search strategies. The framework combines large reasoning models with autonomous search capabilities, allowing the model to decide when and what to search based on reasoning needs. Search-O1 employs explicit search tokens and integrates retrieval results into the reasoning process, enabling coordinated search and reasoning behaviors.
- **Search-R1** (Jin et al., 2025): Uses reinforcement learning to train models to reason and leverage search engines, representing a recent search-based framework for multi-step reasoning. The method trains language models to autonomously decide when to search and how to formulate search queries through reinforcement learning. Search-R1 demonstrates the effectiveness of learning search strategies through RL, enabling models to develop effective search behaviors through trial and error.
- **R1-Searcher** (Song et al., 2025a): Implements recursive search mechanisms based on R1 architecture for complex queries, enabling deeper exploration of search spaces. The method employs recursive search strategies that allow the model to iteratively refine queries and explore search spaces more thoroughly. R1-Searcher's recursive approach enables handling of complex queries that require multiple levels of reasoning, allowing for deeper information exploration.
- **HierSearch** (Tan et al., 2025a): Employs hierarchical search strategies that decompose questions at multiple granularity levels, allowing for more structured reasoning processes. The framework decomposes questions hierarchically, creating multiple levels of abstraction that guide the search process. HierSearch's multi-granularity approach enables more structured reasoning by organizing search at different levels of detail.
- **O^2-Searcher** (Sun et al., 2025): Focuses on optimizing search efficiency through advanced architectural designs. The method employs sophisticated search mechanisms designed to minimize unnecessary search operations while maintaining high answer quality. O^2-Searcher optimizes the trade-off between search cost and performance, enabling efficient search-augmented reasoning.
- **ZeroSearch** (Sun et al., 2025): Aims to incentivize search capabilities without explicit search operations, representing a state-of-the-art search framework. The method trains models to internalize search behaviors without requiring explicit search API calls, encouraging the model to develop search-like reasoning patterns. ZeroSearch demonstrates an alternative approach to search-augmented reasoning by learning implicit search strategies.

These baselines cover the spectrum from simple retrieval-augmented generation to sophisticated search-based reasoning, allowing us to assess Dep-Search's improvements in dependency modeling and memory management. For fairness, every baseline is re-run with our retrieval corpus (the Wikipedia 2018 dump), the same backbone models, and the same evaluation harness; the numbers we report for IRCoT, RA-ISF, Search-R1, R1-Searcher, HierSearch, O^2-Searcher, and ZeroSearch are therefore our own reproductions under this shared configuration rather than values copied from the original papers.

A.3 Models

We conduct experiments using two model sizes: **Qwen3.5-4B** and **Qwen3.5-9B** (Qwen et al., 2025). These models provide a good balance between performance and computational efficiency, allowing us to evaluate Dep-Search's effectiveness across different model scales. Qwen3.5 is a family of large language models that demonstrate strong reasoning capabilities and instruction-following abilities. The models are trained with extensive instruction tuning and demonstrate competitive performance on various reasoning benchmarks. The 4B variant offers faster inference and lower memory requirements, making it suitable for resource-constrained environments, while the 9B variant provides stronger reasoning capabilities and better instruction understanding. Both variants support the control tokens and structured reasoning required by Dep-Search, enabling comprehensive evaluation of the framework's dependency-aware search mechanisms across model scales. The choice of these model sizes allows us to demonstrate that Dep-Search's improvements are consistent across different model capacities, suggesting that the framework's benefits are not limited to larger models.

A.4 Retrieval Corpus

All retrieval operations are performed over the **Wikipedia 2018** corpus (Karpukhin et al., 2020), which contains approximately 5.9 million passages from English Wikipedia articles. This corpus provides a comprehensive knowledge base for multi-hop reasoning tasks and is consistent with the evaluation setup used in most baseline methods. We use the same corpus for both training and evaluation to ensure fair comparison. The corpus is preprocessed into passages of approximately 100 words each, enabling efficient dense retrieval and reranking operations.

A.5 Implementation Details

We implement Dep-Search using PyTorch and the HuggingFace Transformers library. For retrieval, we use **qwen3-embedding** for dense retrieval and **qwen3-reranker** for re-ranking (Zhang et al., 2025b), with top-$k = 5$ documents retrieved per query. The memory buffer has a fixed capacity of $C_{\text{mem}} = 20$ entries, with recency-based eviction. For PQPO training, we sample $G = 4$ trajectories per question and use a learning rate of 1×10^{-5} with AdamW optimizer. The Process-Quality Reward (PQR) aggregates the four quality facets with weights $(w_{\text{dec}}, w_{\text{ret}}, w_{\text{mem}}, w_{\text{eff}}) = (0.3, 0.25, 0.25, 0.2)$, and the process-quality weight in the composite reward is $\lambda_Q = 0.4$. The mixing coefficient is $\alpha = 0.5$, balancing the GRPO group-relative term against the PQR-driven per-token advantage. The PPO-style process value head is a 2-layer MLP with hidden size 256, trained jointly with the policy using GAE parameters $\gamma = 0.99$ and $\lambda_{\text{GAE}} = 0.95$. We train for 3 epochs with batch size 2 and gradient accumulation steps of 4. During inference, we use temperature 0.7 and top-p 0.9 for generation, with a maximum of 16384 new tokens per trajectory.

B Dep-Search Algorithm

C Decomposition Strategy Analysis

To analyze the effectiveness of dependency-aware decomposition, we compare different decomposition strategies on HotpotQA and 2WikiMHQA using Qwen3.5-9B. Multi-hop reasoning requires breaking down complex questions into dependent sub-questions, where later steps often rely on results from earlier steps. However, existing approaches either ignore dependencies entirely or use fixed decomposition patterns that cannot adapt to question complexity. We evaluate three strategies:

Algorithm 1 Dep-Search Rollout with Search Call

Require: Question Q, policy model π_θ, retriever Retr, initial memory $\mathcal{M}_0$, step budget T
Ensure: Final answer A and trajectory τ

$\mathcal{T}_0 \leftarrow \emptyset,\ \mathcal{C}_0 \leftarrow [\text{instr}; Q],\ S_0 \leftarrow (\mathcal{T}_0, \mathcal{C}_0, \mathcal{M}_0),\ x \leftarrow \emptyset,\ t \leftarrow 0$
while A not emitted and $t < T$ **do**
  Sample $a_t \sim \pi_\theta(\cdot \mid x)$, $x \leftarrow x + a_t$
  if a_t is <Decompose> **then**
    Update $\mathcal{T}_{t+1}$ by adding new sub-questions and dependency edges
  else if a_t closes <Retrieve> tag with query r_t **then**
    $\mathcal{D}_t \leftarrow \text{Retr}(r_t)$
    Append <Retrieve_result> $\mathcal{D}_t$ </Retrieve_result> to x, $\mathcal{C}_{t+1} \leftarrow \mathcal{C}_t \cup \mathcal{D}_t$
    $\mathcal{F}_t^{\text{ret}} \leftarrow \text{Summarize}(\mathcal{D}_t)$, update $\mathcal{M}_{t+1}$ with $\mathcal{F}_t^{\text{ret}}$
  else if a_t closes <Memory> tag with query q_t^{mem} **then**
    Select $\mathcal{M}_t^{\text{read}} \subseteq \mathcal{M}_t$ by recency and embedding similarity to q_t^{mem}
    Append <Memory_result> $\mathcal{M}_t^{\text{read}}$ </Memory_result> to x, $\mathcal{C}_{t+1} \leftarrow \mathcal{C}_t \cup \mathcal{M}_t^{\text{read}}$
  else if a_t is <Conclusion> **then**
    $\mathcal{F}_t \leftarrow \text{Summarize}(\mathcal{C}_t)$, update $\mathcal{M}_{t+1}$ via the recency rule with $\mathcal{F}_t$
  else if a_t is an answer-closing token **then**
    Extract A from x and **break**
  end if
  $t \leftarrow t + 1$
end while
Construct trajectory τ from $(S_t, a_t)_t$ and **return** (A, τ)

Sequential Decomposition that processes sub-questions without explicit dependencies, Two-step Dependencies that models step-to-step dependencies, and QDMR Decomposition that allows the model to determine both the number of steps and their dependency structure adaptively. This analysis is crucial because explicit dependency modeling enables the model to structure reasoning chains correctly, ensuring that prerequisite information is gathered before dependent steps are executed, which is essential for accurate multi-hop reasoning.

Table 4: Decomposition strategy comparison on multi-hop datasets on Qwen3.5-9B.

Strategy	HotpotQA	2WikiMHQA	Dependency Accuracy	Avg.
Sequential Decomposition	40.2	41.1	2.0%	40.7
Two-step Dependencies	42.5	43.2	74.3%	42.9
QDMR Decomposition	**44.8**	**45.5**	**83.2%**	**45.2**

Table 4 presents the comparison of different decomposition strategies. **QDMR Decomposition** achieves the best performance, with an average score of 45.2 across the two datasets, outperforming Sequential Decomposition by 4.5 points. The QDMR strategy achieves 83.2% dependency accuracy, correctly identifying relationships between sub-questions in most cases. Sequential Decomposition, similar to approaches like RA-ISF that process sub-questions without explicit dependency modeling, shows the lowest performance, confirming that explicit dependency modeling is crucial for multi-hop reasoning. Two-step Dependencies achieves intermediate performance, demonstrating that even simple dependency modeling improves reasoning quality, but adaptive QDMR decomposition that allows the model to determine both step count and dependency structure provides the best results. These results validate that explicit dependency modeling enables correct reasoning order, where prerequisite information is gathered before dependent steps, leading to more accurate multi-hop reasoning.

D CASE STUDIES

This section presents detailed case studies demonstrating how Dep-Search handles complex multi-hop reasoning questions through dependency-aware decomposition, retrieval, and memory management.

Case Study 1: Bridge Question on HotpotQA

Question: What is the nationality of the author of the book that won the Pulitzer Prize for Fiction in 2018?

Dep-Search Reasoning Process:

To answer this question, I need to find the author of the book that won the Pulitzer Prize for Fiction in 2018, and then determine that author's nationality. Let me first check my memory to see if I have relevant information stored.

<Memory> Pulitzer Prize Fiction 2018 author nationality </Memory> <Memory_result> (No relevant memory found for this specific question) </Memory_result>

The memory doesn't have this specific information. Let me retrieve information about the 2018 Pulitzer Prize winner.

<Retrieve> Pulitzer Prize for Fiction 2018 winner </Retrieve> <Retrieve_result> The Pulitzer Prize for Fiction in 2018 was awarded to "Less" by Andrew Sean Greer. The novel is a comedic story about a middle-aged novelist who travels around the world to avoid attending his ex-boyfriend's wedding. </Retrieve_result>

Good, I found that "Less" by Andrew Sean Greer won the prize. The retrieved information has been automatically stored in memory. However, this result doesn't include information about the author's nationality. Let me check memory again to see if I can find the author's nationality information.

<Memory> Andrew Sean Greer nationality birthplace </Memory> <Memory_result> "Less" by Andrew Sean Greer won the 2018 Pulitzer Prize for Fiction. </Memory_result>

The memory only contains information about the book award, not the author's nationality. I need to retrieve more specific information about Andrew Sean Greer's nationality.

<Retrieve> Andrew Sean Greer biography nationality birthplace </Retrieve> <Retrieve_result> Andrew Sean Greer is an American novelist and short story writer. He was born on November 21, 1970, in Washington, D.C., United States. Greer has written several novels including "Less" and "The Confessions of Max Tivoli." </Retrieve_result>

Perfect! I now have all the information needed. I've accumulated information from multiple retrievals and memory checks. The context is getting long, so let me summarize the key findings before proceeding.

<Conclusion> The 2018 Pulitzer Prize for Fiction was awarded to "Less" by Andrew Sean Greer. Andrew Sean Greer is an American novelist born in Washington, D.C., United States. </Conclusion>

Based on the information I've gathered, I found that Andrew Sean Greer won the Pulitzer Prize for Fiction in 2018, and from the biography retrieval, I confirmed that he is an American author, born in Washington, D.C., United States. Therefore, the nationality of the author is American.

Final Answer: American

Case Study 2: Comparison Question on 2WikiMultihopQA

Question: Which actor appeared in both the movie directed by Christopher Nolan that won the Academy Award for Best Picture and the movie where Leonardo DiCaprio won Best Actor?

Dep-Search Reasoning Process:

This is a complex comparison question that requires finding information about two different movies and then comparing their casts. I need to identify: (1) Christopher Nolan's Best Picture winner, and (2) the movie where Leonardo DiCaprio won Best Actor. These two sub-questions can be answered independently, so I can retrieve them in parallel or sequentially. Let me start with the first one.

<Retrieve> Christopher Nolan Best Picture Academy Award winner </Retrieve><Retrieve_result> Christopher Nolan's "Oppenheimer" won the Academy Award for Best Picture in 2024. The film stars Cillian Murphy, Emily Blunt, Robert Downey Jr., Matt Damon, and Florence Pugh. </Retrieve_result>

Good, I found that "Oppenheimer" won Best Picture. Now let me retrieve information about Leonardo DiCaprio's Best Actor win.

<Retrieve> Leonardo DiCaprio Best Actor Academy Award winner movie </Retrieve><Retrieve_result> Leonardo DiCaprio won the Academy Award for Best Actor for his role in "The Revenant" (2015), directed by Alejandro González Iñárritu. The film also stars Tom Hardy, Domhnall Gleeson, and Will Poulter. </Retrieve_result>

Now I have both movies: "Oppenheimer" and "The Revenant". Let me check my memory to see if I have stored information about these casts, which would help me compare them more efficiently.

<Memory> actors in Oppenheimer and The Revenant </Memory><Memory_result> Oppenheimer (2024) cast: Cillian Murphy, Emily Blunt, Robert Downey Jr., Matt Damon, Florence Pugh. The Revenant (2015) cast: Leonardo DiCaprio, Tom Hardy, Domhnall Gleeson, Will Poulter. </Memory_result>

Comparing the casts: "Oppenheimer" has Cillian Murphy, Emily Blunt, Robert Downey Jr., Matt Damon, Florence Pugh. "The Revenant" has Leonardo DiCaprio, Tom Hardy, Domhnall Gleeson, Will Poulter. I don't see any obvious overlap. However, I recall that Matt Damon and Leonardo DiCaprio have worked together before. Let me verify if Matt Damon appeared in "The Revenant" or if there's any connection.

<Retrieve> Matt Damon Leonardo DiCaprio movies together The Revenant </Retrieve><Retrieve_result> Matt Damon and Leonardo DiCaprio have appeared together in "The Departed" (2006) and "The Revenant" (2015), where Matt Damon had a cameo role. However, Matt Damon did not appear in "Oppenheimer." </Retrieve_result>

I see that Matt Damon had a cameo in "The Revenant", but he did not appear in "Oppenheimer". After careful comparison, no actor appears in both films. I've now accumulated multiple retrieval results and memory accesses. The context is getting long with information about both movies, their casts, and additional verification queries. Let me summarize the key findings to compress the context and store them in memory for potential future use.

<Conclusion> Christopher Nolan's "Oppenheimer" won Best Picture in 2024 with cast including Cillian Murphy, Emily Blunt, Robert Downey Jr., Matt Damon, and Florence Pugh. Leonardo DiCaprio won Best Actor for "The Revenant" in 2015, which also stars Tom Hardy, Domhnall Gleeson, and Will Poulter. </Conclusion>

Now let me analyze the information I've gathered. From the retrievals and memory, I have the complete cast lists for both films. "Oppenheimer" stars: Cillian Murphy, Emily Blunt, Robert Downey Jr., Matt Damon, and Florence Pugh. "The Revenant" stars: Leonardo DiCaprio, Tom Hardy, Domhnall Gleeson, and Will Poulter. I also verified that Matt Damon had a cameo in "The Revenant" but did not appear in "Oppenheimer". After comparing all the actors from both films, I can confirm that no actor appears in both films.

Final Answer: No actor appeared in both films.

Case Study 3: Multi-hop Question with Memory Reuse

Question: What is the capital city of the country where the author of "1984" was born?
Dep-Search Reasoning Process:
This is a multi-hop question that requires several steps. Let me first check my memory to see if I have relevant information.
<Memory> author of 1984 birth country capital city </Memory> <Memory_result> (No relevant memory found for this specific question) </Memory_result>
Memory doesn't have this information. Let me try retrieving information about the author of "1984".
<Retrieve> author of 1984 novel </Retrieve> <Retrieve_result> "1984" is a dystopian novel published in 1949. The novel explores themes of totalitarianism and surveillance. </Retrieve_result>
This retrieval result doesn't include the author's name or birth information. The retrieved information is not sufficient for answering the question. Let me try a more specific query.
<Retrieve> author of 1984 novel George Orwell </Retrieve> <Retrieve_result> "1984" is a dystopian novel written by George Orwell, published in 1949. George Orwell is the pen name of Eric Arthur Blair, born on June 25, 1903, in Motihari, Bengal Presidency, British India (now in Bihar, India). </Retrieve_result>
Excellent! I found that George Orwell (Eric Arthur Blair) was born in British India, which is now India. The retrieved information has been automatically summarized and stored in memory. Now I know the author was born in India. Let me check my memory to retrieve this stored fact.
<Memory> George Orwell birth country India </Memory> <Memory_result> "1984" was written by George Orwell (Eric Arthur Blair), who was born in British India (now India). </Memory_result>
Perfect! I retrieved the birth country information from memory. Now I need to find the capital city of India. Let me check memory first to see if I have this information.
<Memory> India capital city </Memory> <Memory_result> "1984" was written by George Orwell (Eric Arthur Blair), who was born in British India (now India). </Memory_result>
The memory has information about India but not the capital city. Let me retrieve this information.
<Retrieve> capital city of India </Retrieve> <Retrieve_result> The capital city of India is New Delhi. New Delhi serves as the capital of India and is part of the larger Delhi metropolitan area. </Retrieve_result>
Perfect! I found that New Delhi is the capital of India. The retrieved information has been automatically stored in memory. Let me verify by checking memory.
<Memory> New Delhi India capital </Memory> <Memory_result> The capital city of India is New Delhi. New Delhi serves as the capital of India and is part of the larger Delhi metropolitan area. </Memory_result>
Now I have all the information needed to answer the question. From my retrievals and memory, I found that: (1) The author of "1984" is George Orwell, (2) George Orwell was born in British India, which is now India, (3) The capital city of India is New Delhi. Therefore, the capital city of the country where the author of "1984" was born is New Delhi.
Final Answer: New Delhi

Case Study 4: Complex Dependency Chain

Question: What is the population of the city where the university that awarded the Nobel Prize winner in Physics 2023 their PhD is located?

Dep-Search Reasoning Process:

This is a complex multi-hop question with a long dependency chain. Let me try retrieving information about the 2023 Nobel Prize winners.

<Retrieve> Nobel Prize Physics 2023 </Retrieve> <Retrieve_result> The Nobel Prize in Physics recognizes outstanding contributions to the field of physics. The prize has been awarded annually since 1901. </Retrieve_result>

This retrieval result is too general and doesn't contain information about the 2023 winners or their PhD universities. The retrieved information is not useful for answering this specific question. Given the complexity of this multi-hop question, let me decompose it into clear dependent steps to better structure my reasoning.

<Decompose> (1) Identify the Nobel Prize winner(s) in Physics 2023. (2) Find the university where the winner from (1) received their PhD. (3) Determine the city where the university from (2) is located. (4) Find the population of the city from (3). </Decompose>

Now let me proceed step by step following the decomposition. Step 1: Find the 2023 Physics Nobel Prize winners.

<Retrieve> Nobel Prize Physics 2023 winner </Retrieve> <Retrieve_result> The 2023 Nobel Prize in Physics was awarded to Pierre Agostini, Ferenc Krausz, and Anne L'Huillier for their work on attosecond physics. Anne L'Huillier received her PhD from the University of Paris in 1986. </Retrieve_result>

Great! I found that there were three winners, and I also see that Anne L'Huillier received her PhD from the University of Paris. The retrieved information has been automatically summarized and stored in memory. Step 2: Find the university location. Let me check my memory first.

<Memory> Anne L'Huillier PhD university University of Paris </Memory> <Memory_result> The 2023 Nobel Prize in Physics was awarded to Pierre Agostini, Ferenc Krausz, and Anne L'Huillier. Anne L'Huillier received her PhD from the University of Paris. </Memory_result>

I retrieved the university information from memory. Step 3: Find the city where the University of Paris is located.

<Retrieve> University of Paris location city </Retrieve> <Retrieve_result> The University of Paris (also known as Sorbonne University) is located in Paris, France. The university has multiple campuses throughout Paris, with its main campus in the Latin Quarter. </Retrieve_result>

Perfect! The University of Paris is located in Paris, France. Step 4: Find the population of Paris. I've now accumulated several retrieval results. The context is getting long with multiple facts. Let me summarize the key information before proceeding to the final retrieval.

<Conclusion> The 2023 Nobel Prize in Physics was awarded to Pierre Agostini, Ferenc Krausz, and Anne L'Huillier. Anne L'Huillier received her PhD from the University of Paris, which is located in Paris, France. </Conclusion>

Now I can proceed to find the population of Paris. Let me check memory first to see if I have this information stored.

<Memory> Paris France population </Memory> <Memory_result> The 2023 Nobel Prize in Physics was awarded to Pierre Agostini, Ferenc Krausz, and Anne L'Huillier. Anne L'Huillier received her PhD from the University of Paris, which is located in Paris, France. </Memory_result>

The memory has information about Paris but not the population. Let me retrieve this information.

<Retrieve> Paris France population 2023 </Retrieve> <Retrieve_result> Paris, the capital city of France, has a population of approximately 2.1 million people within the city limits (as of 2023). The greater Paris metropolitan area has a population of over 12 million people. </Retrieve_result>

Perfect! I found the population information. The retrieved information has been automatically stored in memory. Let me verify by checking memory one more time.

<Memory> Paris population 2023 </Memory> <Memory_result> Paris, the capital city of France, has a population of approximately 2.1 million people within the city limits (as of 2023). The greater Paris metropolitan area has a population of over 12 million people. </Memory_result>

Now I have completed all four steps of the decomposition. Therefore, the population of the city where the university that awarded the Nobel Prize winner in Physics 2023 their PhD is located is approximately 2.1 million within city limits.

Final Answer: Approximately 2.1 million (within city limits) or over 12 million (metropolitan area)

Case Study 5: Decomposition with Parallel Sub-questions

Question: What are the birth years of both the director of "Inception" and the lead actor of "The Dark Knight"?

Dep-Search Reasoning Process:

This question requires finding information about two different entities (a director and an actor) from two different movies. These sub-questions can be answered independently, but I need to structure them clearly. Let me decompose the question first.

<Decompose> (1) Find the director of "Inception" and their birth year. (2) Find the lead actor of "The Dark Knight" and their birth year. (3) Combine the birth years from (1) and (2). </Decompose>

Now let me proceed step by step. Step 1: Find the director of "Inception" and their birth year.

<Retrieve> director of Inception movie </Retrieve> <Retrieve_result> "Inception" (2010) is a science fiction film directed by Christopher Nolan. The film stars Leonardo DiCaprio, Marion Cotillard, Tom Hardy, and Ellen Page. </Retrieve_result>

Good, I found that Christopher Nolan directed "Inception". The retrieved information has been automatically stored in memory. Now I need to find Christopher Nolan's birth year.

<Retrieve> Christopher Nolan birth year biography </Retrieve> <Retrieve_result> Christopher Nolan is a British-American filmmaker born on July 30, 1970, in London, England. He is known for directing films such as "Inception", "The Dark Knight", and "Interstellar". </Retrieve_result>

Perfect! Christopher Nolan was born in 1970. Step 2: Find the lead actor of "The Dark Knight" and their birth year.

<Retrieve> lead actor The Dark Knight movie </Retrieve> <Retrieve_result> "The Dark Knight" (2008) is a superhero film directed by Christopher Nolan. The film stars Christian Bale as Bruce Wayne/Batman, Heath Ledger as the Joker, and Aaron Eckhart as Harvey Dent. </Retrieve_result>

I found that Christian Bale is the lead actor (playing Batman). The retrieved information has been automatically stored in memory. Now I need to find Christian Bale's birth year.

<Memory> Christian Bale birth year </Memory> <Memory_result> "The Dark Knight" (2008) stars Christian Bale as Bruce Wayne/Batman. </Memory_result>

The memory has information about the movie but not the birth year. Let me retrieve Christian Bale's birth year.

<Retrieve> Christian Bale birth year biography </Retrieve> <Retrieve_result> Christian Bale is a British actor born on January 30, 1974, in Haverfordwest, Wales. He is known for his roles in "The Dark Knight" trilogy, "American Psycho", and "The Fighter". </Retrieve_result>

Perfect! Christian Bale was born in 1974. The retrieved information has been automatically stored in memory. Step 3: Combine the birth years. I've now gathered information about both birth years. Let me verify by checking memory.

<Memory> Christopher Nolan Christian Bale birth years </Memory> <Memory_result> Christopher Nolan is a British-American filmmaker born on July 30, 1970. Christian Bale is a British actor born on January 30, 1974. </Memory_result>

Perfect! I have confirmed both birth years from memory. I've accumulated information from multiple retrievals. The context is getting long, so let me summarize the key findings.

<Conclusion> Christopher Nolan directed "Inception" and was born in 1970. Christian Bale starred as the lead actor in "The Dark Knight" and was born in 1974. </Conclusion>

Now I have completed all three steps of the decomposition. From my retrievals and memory, I found that: (1) Christopher Nolan directed "Inception" and was born in 1970, (2) Christian Bale was the lead actor of "The Dark Knight" and was born in 1974. Therefore, the birth years are: Christopher Nolan was born in 1970, and Christian Bale was born in 1974.

Final Answer: Christopher Nolan was born in 1970, and Christian Bale was born in 1974.

Case Study 6: Decomposition with Complex Dependencies

Question: What is the release year of the first movie directed by the person who won the Academy Award for Best Director in 2014?

Dep-Search Reasoning Process:

This is a complex question with multiple dependent steps. I need to find the Best Director winner in 2014, then find their first directed movie, and finally find that movie's release year. Each step depends on the previous one. Let me decompose this question to clarify the dependency structure.

<Decompose> (1) Identify the winner of the Academy Award for Best Director in 2014. (2) Find the first movie directed by the person from (1). (3) Determine the release year of the movie from (2). </Decompose>

Now let me proceed step by step following the dependencies. Step 1: Find the Best Director winner in 2014.

<Retrieve> Academy Award Best Director 2014 winner </Retrieve><Retrieve_result> The Academy Award for Best Director in 2014 was awarded to Alejandro González Iñárritu for "Birdman or (The Unexpected Virtue of Ignorance)". Iñárritu is a Mexican filmmaker known for his distinctive visual style and complex narratives. </Retrieve_result>

Excellent! I found that Alejandro González Iñárritu won Best Director in 2014. The retrieved information has been automatically stored in memory. Step 2: Find the first movie directed by Alejandro González Iñárritu.

<Memory> Alejandro González Iñárritu first movie directed </Memory><Memory_result> The Academy Award for Best Director in 2014 was awarded to Alejandro González Iñárritu for "Birdman". </Memory_result>

The memory has information about the award but not about his first movie. Let me retrieve information about Iñárritu's filmography.

<Retrieve> Alejandro González Iñárritu first movie filmography debut </Retrieve><Retrieve_result> Alejandro González Iñárritu made his directorial debut with "Amores perros" (Love's a Bitch) in 2000. The film is a Mexican crime drama that follows three interconnected stories set in Mexico City. It was Iñárritu's first feature film and received critical acclaim. </Retrieve_result>

Perfect! I found that "Amores perros" (2000) was Iñárritu's first directed movie. The retrieved information has been automatically stored in memory. Step 3: The release year is already included in the previous result - 2000.

<Memory> Amores perros release year </Memory><Memory_result> Alejandro González Iñárritu made his directorial debut with "Amores perros" in 2000. </Memory_result>

I can confirm from memory that "Amores perros" was released in 2000. I've successfully completed all three steps of the decomposition. Let me verify the complete chain by checking memory one more time.

<Memory> Alejandro González Iñárritu Best Director 2014 first movie Amores perros </Memory><Memory_result> The Academy Award for Best Director in 2014 was awarded to Alejandro González Iñárritu for "Birdman". Alejandro González Iñárritu made his directorial debut with "Amores perros" in 2000. </Memory_result>

Perfect! The memory confirms the complete chain: Best Director 2014 → Alejandro González Iñárritu → first movie "Amores perros" → release year 2000. I've accumulated information from multiple steps. The context is getting long, so let me summarize the key findings.

<Conclusion> Alejandro González Iñárritu won the Academy Award for Best Director in 2014. He made his directorial debut with "Amores perros". </Conclusion>

Now I have completed all three steps of the decomposition. From my retrievals and memory, I found that: (1) Alejandro González Iñárritu won the Academy Award for Best Director in 2014, (2) His first directed movie was "Amores perros", (3) "Amores perros" was released in 2000. Therefore, the release year of the first movie directed by the person who won the Academy Award for Best Director in 2014 is 2000.

Final Answer: 2000

Case Study 7: Decomposition After Failed Retrieval

Question: What is the total number of books written by the author who won the Nobel Prize in Literature in 2017?

Dep-Search Reasoning Process:

This question requires finding the Nobel Prize winner in Literature for 2017, then finding how many books that author has written. Let me first check my memory.

<Memory> Nobel Prize Literature 2017 author books </Memory> <Memory_result> (No relevant memory found for this specific question) </Memory_result>

Memory doesn't have this information. Let me try retrieving information about the 2017 Nobel Prize in Literature.

<Retrieve> Nobel Prize Literature 2017 </Retrieve> <Retrieve_result> The Nobel Prize in Literature is awarded annually to an author who has produced outstanding work in the field of literature. </Retrieve_result>

This retrieval result is too general and doesn't contain information about the 2017 winner or their books. The retrieved information is not useful for answering this specific question. Given that this is a multi-step question, let me decompose it to better structure my reasoning.

<Decompose> (1) Identify the winner of the Nobel Prize in Literature in 2017. (2) Find the total number of books written by the author from (1). </Decompose>

Now let me proceed step by step. Step 1: Find the 2017 Nobel Prize in Literature winner.

<Retrieve> Nobel Prize Literature 2017 winner </Retrieve> <Retrieve_result> The Nobel Prize in Literature for 2017 was awarded to Kazuo Ishiguro "who, in novels of great emotional force, has uncovered the abyss beneath our illusory sense of connection with the world". Ishiguro is a British novelist of Japanese origin, known for works such as "The Remains of the Day" and "Never Let Me Go". </Retrieve_result>

Excellent! I found that Kazuo Ishiguro won the Nobel Prize in Literature in 2017. The retrieved information has been automatically stored in memory. Step 2: Find the total number of books written by Kazuo Ishiguro.

<Memory> Kazuo Ishiguro books bibliography </Memory> <Memory_result> The Nobel Prize in Literature for 2017 was awarded to Kazuo Ishiguro, a British novelist known for works such as "The Remains of the Day" and "Never Let Me Go". </Memory_result>

The memory mentions some books but doesn't have the total count. Let me retrieve more specific information about Ishiguro's bibliography.

<Retrieve> Kazuo Ishiguro total number of books bibliography complete list </Retrieve> <Retrieve_result> Kazuo Ishiguro has published eight novels: "A Pale View of Hills" (1982), "An Artist of the Floating World" (1986), "The Remains of the Day" (1989), "The Unconsoled" (1995), "When We Were Orphans" (2000), "Never Let Me Go" (2005), "The Buried Giant" (2015), and "Klara and the Sun" (2021). He has also published a collection of short stories and several screenplays. </Retrieve_result>

Perfect! I found that Kazuo Ishiguro has published eight novels. The retrieved information has been automatically stored in memory. Let me verify by checking memory.

<Memory> Kazuo Ishiguro eight novels bibliography </Memory> <Memory_result> The Nobel Prize in Literature for 2017 was awarded to Kazuo Ishiguro. Kazuo Ishiguro has published eight novels: "A Pale View of Hills" (1982), "An Artist of the Floating World" (1986), "The Remains of the Day" (1989), "The Unconsoled" (1995), "When We Were Orphans" (2000), "Never Let Me Go" (2005), "The Buried Giant" (2015), and "Klara and the Sun" (2021). </Memory_result>

Confirmed! Kazuo Ishiguro has published eight novels. I've successfully completed both steps of the decomposition. The context is getting long, so let me summarize the key findings.

<Conclusion> Kazuo Ishiguro won the Nobel Prize in Literature in 2017. He has published multiple novels and other works. </Conclusion>

Now I have completed both steps of the decomposition. From my retrievals and memory, I found that: (1) Kazuo Ishiguro won the Nobel Prize in Literature in 2017, (2) He has published eight novels, plus additional works including short stories and screenplays. Therefore, the total number of books written by the author who won the Nobel Prize in Literature in 2017 is eight novels (plus additional works including short stories and screenplays).

Final Answer: Eight novels (plus additional works including short stories and screenplays)

E TRAINING PROMPT TEMPLATE

Training Prompt Template

System Instruction: You are a helpful AI assistant that solves complex questions through dependency-aware reasoning with persistent memory.
Available Actions:
1. <Decompose> *steps with dependencies* </Decompose>
Break down the question into 2-4 steps with explicit dependencies. Each step must reference prerequisite steps using "(step_number)" format. Dependencies form a directed acyclic graph where sub-questions are solved in topological order, ensuring prerequisites are resolved before dependent steps.
Example: "(1) Identify key entity. (2) Use (1) to locate factual reference. (3) Resolve dependent query using (2). (4) Synthesize final answer from (1)–(3)."
2. <Retrieve> *query* </Retrieve> <Retrieve_result> *retrieved documents* </Retrieve_result>
Upon emitting <Retrieve> followed by a query and the closing </Retrieve>, the environment immediately performs two-stage retrieval (dense retrieval + reranking) and returns the top-k retrieved documents wrapped in <Retrieve_result> tags. The model generates the query autonomously, allowing it to determine what information to retrieve based on the current reasoning context.
3. <Memory> *query* </Memory> <Memory_result> *retrieved facts* </Memory_result>
Upon emitting <Memory> followed by a query and the closing </Memory>, the environment returns a mixture of Most Recent Memory entries and additional memory items whose qwen3-embedding cosine similarity with the query exceeds a threshold, wrapped in <Memory_result> tags. The model generates the query autonomously, allowing it to determine what knowledge to fetch from memory based on the current reasoning needs.
4. <Conclusion> *summary* </Conclusion>
When long reasoning and evidence have accumulated, emit <Conclusion> to summarize the preceding context into a few natural-language sentences, which are then written into the memory buffer and can be reused via <Memory>.