RAG-DOLL: Zero-Shot LLM Multi-Hop Reasoning by Simultaneously Planning Over Strategy and Sub-Problems

Anonymous ACL submission

Abstract

Most solution methods for zero-shot multi-hop reasoning suffer in performance due to an inability to backtrack on wrong retrieval decisions or to modify the plan for decomposing the multi-hop question given new information. We introduce RAG-DOLL, a zero-shot LLM multi-hop reasoning formulation that plans over *both* selecting a good strategy to decompose queries *and* selecting the right sub-actions to take given a strategy. We structure this formulation as a Markov Decision Process (MDP) that we solve using Monte-Carlo Tree Search (MCTS) with Action Progressive Widening (APW) over zero-shot LLM-sampled actions. Through this, we demonstrate strong exact-match performance against other training-free prompting approaches on the QA tasks HotPotQA, MuSiQue, and 2WikiMultihopQA, as well as on closed-book reasoning with GPQA-diamond and BIG-Bench Hard. Our method yields competitive results for zero-shot multi-hop reasoning, enabling better use of test-time compute on complex tasks.

1 Introduction

Although Large Language Models (LLMs) are widely deployed for reasoning and retrieval tasks (Lewis et al., 2021), multi-step tasks pose unique challenges (Chen et al., 2024; Wu et al., 2024). LLMs exhibit strong retrieval performance on straightforward one-step queries that do not require decomposition. However, this performance notably degrades for multi-hop queries, which require sequential information-gathering steps (Yang et al., 2018; Kočiský et al., 2017).

Consider the query: *Who is the director of the 2003 film shot in Quality Cafe?* One approach to addressing it is to prompt the LLM to break it down. For example, the LLM could produce the sub-queries "What 2003 films are shot in Quality Cafe?" and "Who is the director of the film?" To produce these queries, the LLM is effectively *strategizing* about the problem.

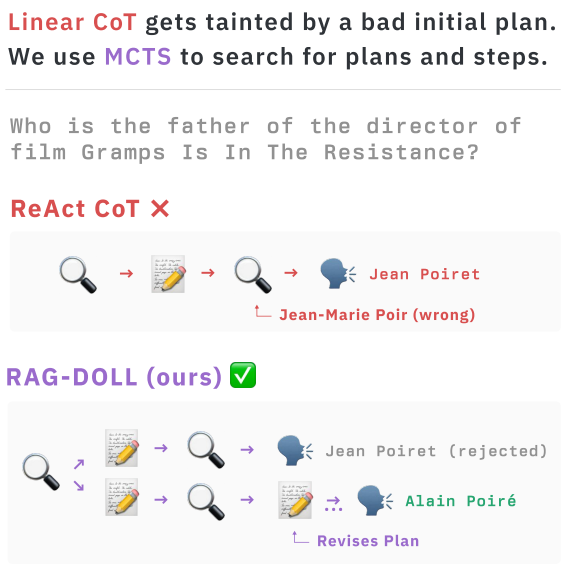


Figure 1: RAG-DOLL is a Monte-Carlo tree search based solution for multi-hop tasks that simultaneously plans over decomposition strategies as well as steps, preventing bad initial plans from affecting outcomes.

A simple, commonly-used approach to query decomposition is Chain-of-Thought (CoT) prompting (Wei et al., 2022). Since these chains can, by definition, be divided into steps, each step can be individually augmented with the corresponding retrieved or parametric knowledge (Trivedi et al., 2023). However, CoT-based approaches are inherently single-shot, i.e., they consist of a single plan. As a result, an error at any step of the process may degrade the decisions that follow it.

Tree-based search solutions attempt to mitigate this weakness by formulating a plan over multiple possible sub-queries at each step and allowing backtracking between them (Qi et al., 2021; Khattab et al., 2022; Sarthi et al., 2024; Fan et al., 2024;

Bi et al., 2024; Jiapeng et al., 2024; Lee et al., 2024; Ren et al., 2025). These approaches significantly improve multi-hop performance, but they are limited to branching within a single higher-level plan. The drawbacks of committing to a fixed plan are particularly apparent in zero-shot approaches. Without the opportunity to revise a flawed strategy, both zero-shot prompting and zero-shot RAG are susceptible to error propagation from the original query approach (Li et al., 2025).

In response, we introduce **RAG-DOLL**, a zero-shot planning-based approach for solving multi-hop queries that simultaneously plans over high-level strategy and low-level action selection. RAG-DOLL leverages the efficient exploration of Monte-Carlo Tree Search (MCTS) (Silver and Veness, 2010): unlike fixed tree approaches which grow exponentially as a function of depth (which may be irreducible for multi-hop problems), MCTS can explore the tree to variable depths, concentrating compute on areas that are most promising. We use zero-shot LLMs to guide the simultaneous choices of both the *plan* for solving a particular query and the *steps* with which to answer it (Figure 1) by incorporating the retrieval plan into the MCTS state space—enabling the tree to backtrack on bad plans given new information.

We demonstrate strong performance on **both multi-hop question-answering datasets and multi-hop reasoning tasks** compared to other zero-shot prompting methods. Our work introduces three main contributions:

1. We formulate planning over sub-query strategy and document selection as a single planning problem with a joint state space.

2. We solve the problem using Monte-Carlo Tree Search with Action Progressive Widening (Couëtoux et al., 2011), using zero-shot LLMs to guide the search and assign rewards.

3. We show how (1) and (2) can be integrated to form an LLM-based multi-hop QA and reasoning solver, and demonstrate strong zero-shot performance on retrieval datasets HotPotQA Fullwiki (Yang et al., 2018), MuSiQue (Trivedi et al., 2022), and 2WikiMultihopQA (Ho et al., 2020), as well as on the reasoning datasets GPQA-diamond (Rein et al., 2024) and BIG-Bench Hard (Srivastava et al., 2023).

Our approach demonstrates strong performance-token scaling properties relative to baselines. We release the code and tool for performing retrieval.[1]

2 Related Work

Multi-Hop Information Retrieval Our work primarily contributes to the study of multi-hop retrieval. CoQA (Reddy et al., 2019), HotPotQA (Yang et al., 2018), and BeerQA (Qi et al., 2021) offer strong baselines and benchmarks for this area, extended by decomposition datasets such as MuSiQue (Trivedi et al., 2022). Approaches used to tackle such problems include methods that use explicit summarization at the document level (synthesizing multiple documents into one relevant one), either recurrently (Khattab et al., 2022), by decomposing the retrieval action via chain-of-thought (Trivedi et al., 2023), via a tree (Khattab et al., 2022; Bi et al., 2024), or even with explicit beam search (Zhang et al., 2024). To efficiently leverage tree-like approaches, MCTS has also been previously proposed (Lee et al., 2024; Ren et al., 2025) as a way to sequence the exploration of different sub-query plans. Finally, to ensure generation stability, methods have employed verifiers of the retrieved content (Fan et al., 2024; Bi et al., 2024).

Zero-Shot Prompting Strategies Our solution to the sequential retrieval and reasoning problem involves leveraging an effective zero-shot prompting method. Work in this area began with the discovery that Chain-of-Thought prompting (Wei et al., 2022) significantly improves LLM performance. CoT was subsequently extended by methods that leverage input feedback such as ReAct (Yao et al., 2023c) and MRKL (Karpas et al., 2022). More recently, search-inspired prompting methods such as Tree of Thoughts (Yao et al., 2023a) and Graph of Thoughts (Besta et al., 2024) have emerged that further improve the efficacy of zero-shot prompts. Packages such as DSPy (Khattab et al., 2024), along with a family of "prompt optimizers" (Opsahl-Ong et al., 2024), have also emerged to bootstrap zero-shot prompts into few-shot systems.

MDP Solution Methods Modeling complex sequential decision problems as a Partially Observable Markov Decision Process has been successful in many domains, such as aviation safety (Kochenderfer et al., 2012) and robotics (Lauri et al., 2022); for language models in particular, successful Partially-Observable MDPs (POMDPs) have

[1] Available upon publication.

been formulated for web language agents to sequence complex multi-step interactions (Yao et al., 2022; Putta et al., 2024). Solving a POMDP exactly is intractable (Papadimitriou and Tsitsiklis, 1987; Madani et al., 2003), but online solvers that use tree search (Silver and Veness, 2010; Somani et al., 2013; Sunberg and Kochenderfer, 2018) were able to generate effective actions. Most recently, deep neural network planning methods (Zhang and Yu, 2020; Moss et al., 2024) remove the dependence on manually designed selection heuristics and instead learn a solution through neural networks.

3 Methods

RAG-DOLL is a zero-shot prompting pipeline that solves multi-hop reasoning tasks that have clean single-hop decompositions. We will now describe the construction of this pipeline. First, we formulate the prompting task of planning over *both* decomposition strategies and sub-problem solutions as a single MDP. Second, we introduce an MCTS method with Action Progressive Widening (APW) over zero-shot LLM-sampled actions to solve the MDP, yielding a performant multi-hop QA pipeline. Finally, we describe the details of using zero-shot language model programs to construct each of our method's constituent components.

3.1 Problem Formulation

A Markov Decision Process (MDP) (Kochenderfer et al., 2022; Bellman, 1957) is a model formulation in reinforcement learning that enables sequential decision making over observable states. MDPs are characterized by the tuple $\langle S, A, T, R, \gamma \rangle$. Given the current state $s \in S$ and action $a \in A$, the agent transitions to a new state $s' \sim T(\cdot \mid s, a)$ and receives reward $R(s, a, s')$. The goal of an MDP is to maximize the expected sum of future rewards.

The task of multi-hop reasoning and retrieval is especially well-suited to this formulation, because the quality of subsequent decisions about what information to seek or what to solve next depends heavily on the quality of the information previously obtained. Attempts have been made to formulate retrieval as an MDP, but always with a state space defined in terms of the substeps themselves rather than the documents used (Lee et al., 2024; Ren et al., 2025; Bi et al., 2024; Jiapeng et al., 2024).

Here, we provide an MDP formulation of reasoning whose state **simultaneously encodes the plan for the solution and the actual substeps**, enabling the agent to use the information it has gathered to change the subgoals needed to solve a problem.

3.1.1 State Representation

We construct our state by formulating it in terms of two writable context "buffers": one containing the current subproblem, and the other containing the current plan. Which subproblem our agent is considering and which step of the plan its on naturally forms the state of our MDP.

Our state is a tuple $(C, G, g, d) \in S$ where S is our state space. Let D be the set of all subproblems in the corpus: G is a plan for solving the current question, $g \in G$ is the current step of the plan, $C \subset D$ is the current context, and $d \in D$ is the current subproblem to explore.

3.1.2 Actions and Transitions

For the open-book case, we have access to a retriever tool that yields a ranked list of recommended documents to explore $d_1, \ldots, d_n \sim \text{Retrieve}(x)$ for some query x, with documents $d_j \in D$ returned in ranked order (i.e., such that d_1 is the most relevant document with respect to x). Similarly, for the closed-book case, we build a prompt that explicitly asks the language model to break down a query to form ranked substeps to explore next $d_1, \ldots, d_n \sim \text{Propose}(x)$ with respect to some higher-level goal x. The current state $s = (C, G, g_j, d_k)$ uses the conventions described in Section 3.1.1, and a planning-prompted LLM $\text{Replanner}(C, G)$ (described in Section 3.4) is used to modify the current plan given the context.

Our MDP has four discrete actions, each of which yields a distinct possible output distribution in transition.

a_1: step We take the next step in the plan while committing to the current context; namely, we transition to $s' = (C \cup \{d_k\}, G, g_{j+1}, d_1 \sim F(g_{j+1}))$, where $F = \text{Retrieve}$ for the open-book case and $F = \text{Propose}$ for the closed-book case.

a_2: choose document We ask the LLM to sample a document or substep that it deems most relevant; this can be thought of as an LLM-guided reranking. Let $s' = (C, G, g_j, d_{k+1})$ WLOG by treating the sub-steps as already sorted in order of relevance.

a_3: modify plan Given the current context C, we generate a new plan that extends the steps already taken. This yields $s' = (C, \{g_1, \ldots, g_{j-1}\} \cup G' \sim$

Replanner$(C, G), g_1', d_1 \sim \mathrm{F}(g_1'))$. Note that we also run F on this new step.

a_4: answer The final action terminates the simulation and answers the question using a language model given the retrieved context C.

Notice that transitions between states are only non-deterministic with respect to the outputs of the retriever and the planner. This dramatically cuts down on uncertainty, and allows the user to trade off speed of convergence against retrieval performance by simply fixing some or all transitions to be deterministic via caching. This choice of actions naturally decomposes the tasks of *planning the retrieval* and *selecting the documents* into one unified MDP.

The Retrieve, Propose and Replanner tools are described in further detail in Section 3.4.

3.1.3 Reward

Unlike previous works that incorporate hand-crafted process-level supervision, we use a very simple heuristic as our reward. We simply prompt a language model Correct$(s, q, a) \in \{1, \ldots, 5\}$ to output an integer on a 1-5 scale for question q, answer a, and state s, judging whether the context supports the answer.

For process-level reward assignments, we prompt another language model Relevant$(C, q) \in \{1, \ldots, 5\}$ on the same 1-5 scale to judge whether the context is relevant to the question.

Thus, for scaling factors α_1 and α_2, we can write

$$R(s, a, s') = \begin{cases} \alpha_1 \left[\frac{1}{5}\text{Correct}(s, q, a)\right], & \text{if } a = a_4 \\ \alpha_2 \left[\frac{1}{5}\text{Relevant}(s, q)\right], & \text{otherwise} \end{cases} \tag{1}$$

We describe the design of the LLM-driven Correct and Relevant functions in Section 3.4.

3.2 Monte-Carlo Tree Search (MCTS)

In order to determine the sequence of actions in our MDP that leads to a good final answer, we perform Monte-Carlo Tree Search (MCTS) (Coulom, 2006; Silver and Veness, 2010). All variants of Monte-Carlo tree search execute four basic steps—selection, expansion, simulation, and backup—for which our implementation is described below:

Selection In the selection stage, we select an action $a \in \{a_1, a_2, a_3, a_4\}$ from a given context (state s) within the tree, balancing exploration of possible actions with exploitation of useful ones. A particularly good heuristic for this is the Upper-Confidence Bound (UCB) (Kocsis and Szepesvári, 2006), whereby at a given state we select the action a_j as follows:

$$\arg\max_j Q(s, a_j) + c\sqrt{\frac{\log \sum_{k=1}^{4} N(s, a_k)}{N(s, a_j)}} \tag{2}$$

where $Q(s, a)$ is the Q-value estimate of the state-action pair s, a and $N(s, a)$ is the number of visits that our tree has made to this pair.

Expansion Second, in the expansion stage, we take our selected action a_j from the previous step and execute it to obtain $s' \sim T(\cdot \mid s, a)$. In particular, we apply the appropriate state modifications and transitions outlined in Section 3.1.2 to obtain the next state s' of our tree.

Simulation Next, for the simulation step, we follow AlphaZero (Zhang and Yu, 2020) and BetaZero (Moss et al., 2024) in using a heuristic—in this case, a language model—to determine the value of a leaf state s. After performing selection and expansion up to a finite depth, we estimate the leaf value by forcing action a_4 (i.e., answering the question immediately given the current state) and using the resulting reward as the value at the leaf. This is appropriate because all multi-hop datasets have a finite (and usually small, between 1 and 5) number of hops, so a well-trained tree at a reasonable depth should have enough information in the state to appropriately answer the question.

Backup Finally, for the backup step, the discounted cumulative values obtained at the root node, along with intermediate process-level rewards collected at each step, are propagated along the entire path of the tree. In particular, we update our Q-values at each node by computing

$$q = R(s, a, s') + \gamma Q(s', a') \tag{3}$$

for the s, a and s' obtained in the selection and expansion process. We then perform a running average to update the stored Q-values, namely we increment our visit count $N(s, a) := N(s, a) + 1$ and then set

$$Q(s, a) \leftarrow \frac{q - Q(s, a)}{N(s, a)} \tag{4}$$

Tree Policy and Data Collection After construction, for a given query the MCTS tree root node

contains a collection of expanded actions (subqueries), each weighted based on visit counts and Q-values.

To actually sample the actions we take at each stage, we then instantiate a "tree policy" π_{tree}, a weighting over the actions at our given state s, following

$$\pi_{\text{tree}}(s,a) \propto \left(\left(\frac{e^{Q(s,a)}}{\kappa_Q} \right)^{z_q} \left(\frac{e^{N(s,a)}}{\kappa_N} \right)^{z_n} \right)^{\frac{1}{\tau}} \tag{5}$$

where $\kappa_Q = \sum_{a' \in \{a_1,a_2,a_3,a_4\}} \exp\left(Q(s,a')\right)$ and $\kappa_N = \sum_{a' \in \{a_1,a_2,a_3,a_4\}} \exp\left(N(s,a')\right)$ with hyperparameters z_q that weights Q-maximizing actions and z_n that weights frequently visited ("robust") actions, as well as a temperature τ. This follows insights from recent work that trades off exploration of less-visited actions against exploitation of highly valued ones (Moss et al., 2024).

3.3 Action-Progressive Widening

Although the presentation in Section 3.1.2 treats actions as discrete $(a_1, \ldots, a_4)$, note that all actions are parameterized by an LLM output (such as the actual answer or the retrieved results). Due to the huge action space that language model sampling offers (indeed, with high sampling temperature it is possible that, for a fixed context, no generated follow-up queries within a fixed horizon overlap), we in practice sample actions through Action Progressive Widening (Couëtoux et al., 2011). For a given node s whose set of already-visited child actions is A_s, we only add a new action to select from if $|A_s| \leq k_a N(s)^{\alpha_a}$, for parameters k_a, α_a.

3.4 Design of Zero-Shot LLM Functions

Throughout the methods above, we call a number of LLM-driven functions to perform various tasks. We will now describe each of them in detail.

All of the tools described here, as well as the entire retrieval pipeline, are constructed as tool calls given to the model under test using their standardized tool-call tokenization schemes.

Retrieve The most important tool for multi-hop QA is naturally the one-hop retriever on which it is based. Retrieve : $G \to 2^D$ is a function that maps a subgoal g to a set of documents $d_1, \ldots, d_n \in D$ relevant to achieving this goal. We implement this following a classic two-phase, document-order RAG pipeline (Laitenberger et al., 2025). We first perform a rough Okapi BM25 retrieval (Jones et al., 2000) via the goal text g to obtain the top k most relevant documents (in practice, we set $k = 200$). Following Laitenberger et al. (2025) and other similar baselines, we then perform a *local dense reranking* using a small embedding language model, ordering the resulting documents' embedding vectors by their cosine similarity with that of g. Finally, we return the top n most relevant documents as $d_1, \ldots, d_n$.

Propose Similar to the retriever design, Propose : $G \to 2^D$ is a function that maps a goal g to a set of "documents" describing an intermediate computation. We implement this as a single chain-of-thought module that generates in sequence: first a descriptive approach to the sub-problem (e.g., for "3×12", "find the solution by multiplying 3×12"), then a proposed answer (e.g., a value), and finally a description of the solution (e.g., "the solution to 3×12 is 36"). Multiple such sequences are generated in parallel and stored as "context" documents. The prompt is given in Section C.9.

Solve Subproblem Unlike the Retrieve tool above, in the closed-book reasoning case there is no opportunity for the model to conduct retrieval to solve a particular subtask generated by the Propose tool. Instead, we ask the language model to pick a subproblem to solve from the proposed steps and then use chain-of-thought reasoning to solve the step it deems most relevant. The prompt is given in Section C.10.

Replanner To draft a new plan, we use a single chain-of-thought (CoT) prompt that asks the language model to reflect on the flaws of the current plan given the context, and then formulate a new, revised plan. This prompt is given in Section C.5.

Answer Answer is, again, a simple CoT prompt that takes the context and the full multi-hop query and produces the answer. Its exact construction is given in Section C.3.

Relevant and Correct We use two prompts that each score a context—along with an optional answer—for its context relevance, thus forming our reward function. With scoring-based prompts, care must be taken to constrain the language model to only output numbers in the range we expect; thus, we instantiate the values as a categorical classification task with 5 choices from worst to best (i.e.,

instead of asking the LLM to choose a number directly). The details of these prompts are given in Section C.11.

4 Experiments

4.1 Dataset and Embeddings

Retrieval Datasets We evaluate our approach using a suite of standard benchmarks across retrieval and reasoning. For the retrieval corpus, we consider the standard Multi-Hop QA dataset HotpotQA (Yang et al., 2018) in the full-wiki setting, as well as the harder MuSiQue (Trivedi et al., 2022) and 2WikiMultihopQA (Ho et al., 2020) corpora. For the retrieval base, we use the Wikipedia corpus, consisting of 5,233,329 documents. We use `sqlite-fst5`[2] to perform sparse retrieval and `snowflake-arctic-embed-m-v1.5`[3] to obtain document-query embeddings for dense retrieval.

Reasoning Datasets We additionally evaluate our approach on difficult multi-hop reasoning datasets. First, we consider the multi-hop reasoning dataset GPQA-diamond (Rein et al., 2024) in a closed-book setting, a multiple-choice QA benchmark whose questions require multiple reasoning steps to answer correctly. Second, we consider the difficult multiple-choice multi-domain benchmark BIG-Bench Hard (Srivastava et al., 2023), which evaluates the model on both its multi-hop reasoning capability and its ability to recall parametric knowledge.

4.2 Harness and Model Selection

Primary Evaluations. Our primary evaluation involves measuring the performance of our method compared to baselines in both reasoning and retrieval settings. In particular, we compare our method against our reimplementations of baselines, enabling granular reporting of token usage, runtime, and matched model selection. For this evaluation, we conduct our experiments using `gpt-5.4-mini` (without "thinking", since we cannot control the time or token usage precisely; we discuss this further in Figure 10), a strong API model that is cheap and fast enough for many parallel rollouts.

[2]https://sqlite.org/fts5.html

[3]https://huggingface.co/Snowflake/snowflake-arctic-embed-m-v1.5

External Benchmarks. For the retrieval datasets, we perform a benchmark against published results without reimplementation due to the standardized benchmarking they can offer. This provides a strong, harness-independent benchmark of our performance. However, this also limits us to using an older model, `gpt-4o-mini`, in order to match results published in the literature. The choice of retrieval setup and vector embeddings used in our primary evaluations is consistent with the external benchmarks as well.

Per-Component Ablations. Lastly, on the reasoning dataset, we conduct an ablation of every single component of our design. We measure the performance of our approach using linear inference allowed to call the tools we provide, MCTS without any tools beyond retrieval, and MCTS with planning but no replanning.

4.3 Baselines

External Baselines We report results of our method against RAG (Karpukhin et al., 2020), standard Chain-of-Thought (Wei et al., 2022), adaptive methods including ReAct prompts (Yao et al., 2023c), Query2Doc (Wang et al., 2023), and Self-RAG (Asai et al., 2024), and MCTS-based methods like HG-MCTS (Ren et al., 2025).

Implemented Baselines In addition to leaderboard results, we reimplement a number of baselines to precisely measure their time and token use. Specifically, we report time, token use, and performance of standard Chain-of-Thought (Wei et al., 2022), ReAct (Yao et al., 2023c), Tree of Thoughts (Yao et al., 2023b), and a standard MCTS without planning (Ren et al., 2025).

4.4 Metrics

For retrieval, as is standard with other Multi-Hop QA experiments, we report two primary metrics: Exact Token-For-Token Match (EM) and Token F1 (F1), the latter computed as a mean F1 score between the list of tokens produced by our policy and the gold tokens. When comparing matches across both metrics, we disregard case differences, punctuation, and stopwords as detected by NLTK (Bird and Loper, 2004), which is standard for QA datasets. For reasoning, we report exact match on the 4-choice multiple-choice task.

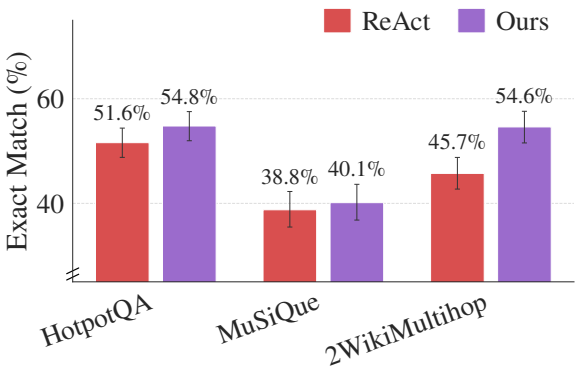


Figure 2: Token-wise exact-match of our approach compared to the ReAct baseline on three multi-hop retrieval datasets using `gpt-5.4-mini`; errors shown as 10–90 Beta credible intervals over 6 seeds.

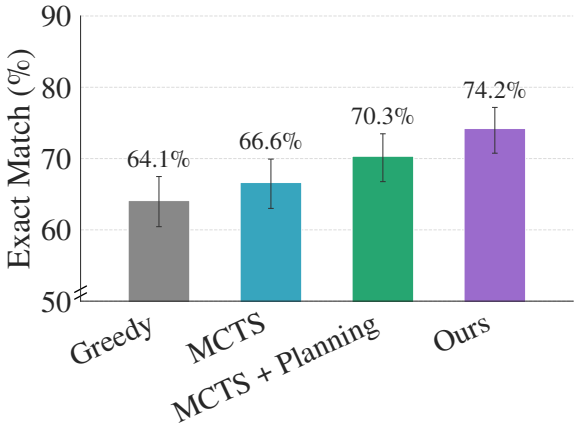
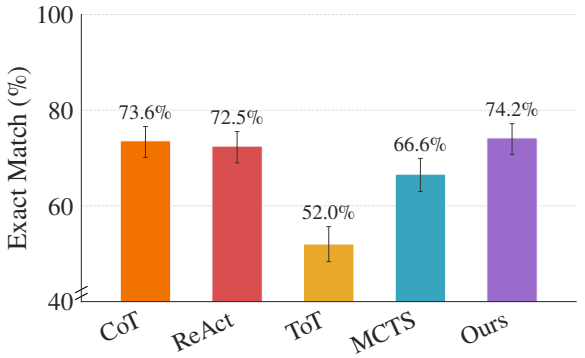
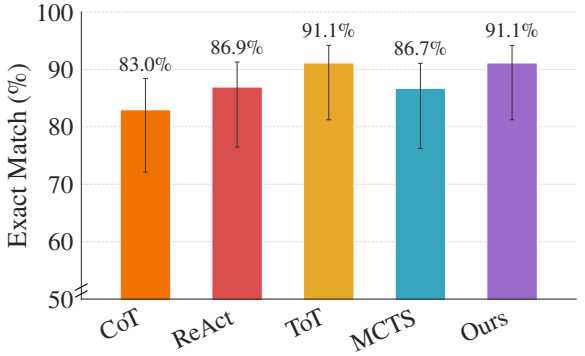


Figure 3: GPQA-diamond exact-match for simple greedy sampling over our action space, MCTS over subproblems only, MCTS with planning, and our full approach. 10–90 Beta credible intervals over 6 seeds shown as error bars.



(a) GPQA-diamond



(b) BIG-Bench Hard

Figure 4: Selection exact-match of our approach compared to various baselines on GPQA-diamond and BIG-Bench Hard using `gpt-5.4-mini`; errors shown as 10–90 Beta credible intervals over 6 seeds.

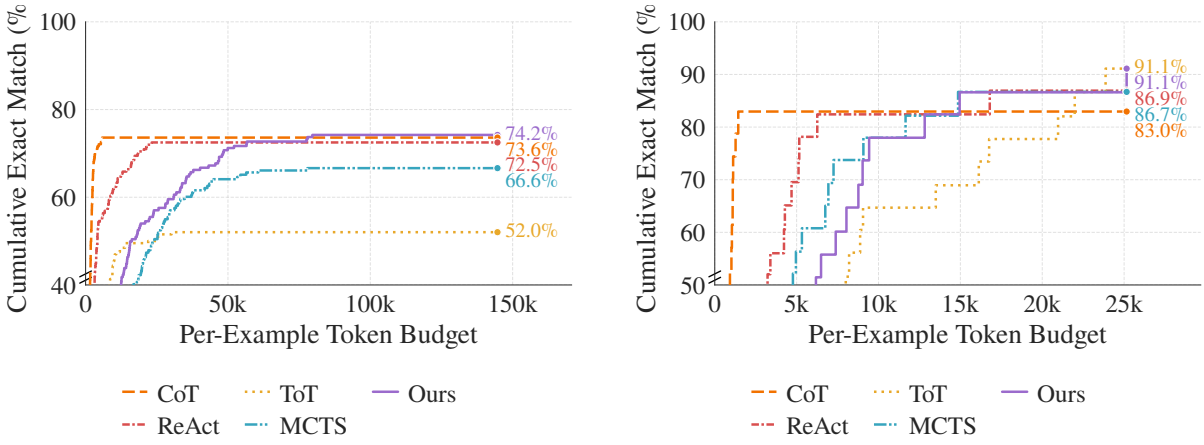
5 Results

Our method is performant against all baselines across retrieval and reasoning. As highlighted in Figure 2 and Figure 4, our method is performant against all baselines across both retrieval and reasoning tasks. On many tasks, such as 2WikiMultihop—a difficult multi-hop retrieval dataset—our method achieves significant improvements of almost 10% EM over the ReAct baseline.

Our method continues to improve at larger token budgets, and is more efficient at smaller ones. On BIG-Bench Hard, our method meets (Figure 4b) but does not exceed the performance of the Tree-of-Thoughts baseline. Yet, as seen in Figure 5, after just 5k tokens our method outperforms ToT at every given token budget, and continues to scale more efficiently up to 25k tokens, at which point the two approaches' performance meets. Our continuous improvements can also be seen strongly in the retrieval datasets as well Figure 6, where our method continues to improve way past the plateau of compariable test-time compute used in ReAct.

Our approach outperforms every baseline in the external benchmark setting. As highlighted in Table 1, our method outperforms all baselines' published results using the same backbone, including ones that also use MCTS.

Every component of our design contributes to performance gain. As demonstrated in Figure 3, every component of our design contributes to a measurable degree of performance gain. Notably, even adding the "replanning" action yields a mean 4% increase in GPQA-Diamond EM, indicating that initial plans are often flawed and can be improved as subproblems are solved.



(a) GPQA-Diamond

(b) BIG-Bench Hard

Figure 5: The performance of our method compared to various baselines at every given token budget; we evaluate the performance of each approach at a given budget by treating every question not answered by the allotted budget as incorrect.

	HotPot QA		2WikiMultihopQA		MuSiQue	
	EM	F1	EM	F1	EM	F1
CoT (Wei et al., 2022)	26.47	32.35	24.12	26.47	13.53	18.24
RAG (Lewis et al., 2021)	41.18	52.94	25.88	27.65	11.76	16.47
ReAct (Yao et al., 2023c)	35.88	42.35	29.41	30.00	10.00	12.35
Self-RAG (Asai et al., 2024)	38.82	47.65	26.47	27.65	13.53	16.47
IRCoT (Trivedi et al., 2023)	42.30	54.10	48.10	59.20	19.10	29.40
Query2doc (Wang et al., 2023)	44.71	54.71	29.41	29.41	19.41	24.71
HG-MCTS (Ren et al., 2025)	41.76	58.69	52.94	53.53	23.67	26.04
Ours	**48.80**	**59.60**	**55.39**	**63.42**	**24.95**	**34.31**

Table 1: Performance of our method using `gpt-4o-mini` compared to other zero-shot baselines. Baselines were measured by Ren et al. (2025) using the same model and data subsets. Sampling temperature is 0.8, over 6 seeds.

6 Conclusion

We introduced RAG-DOLL, a zero-shot prompting framework that uses local actions on a writable context to formulate multi-hop reasoning as a planning problem over both the *strategy* of complex query decomposition and the actual *selection* of documents to place into context. We then introduced a zero-shot, language-model driven solution of our formulation using Monte-Carlo Tree Search (MCTS) with Action Progressive Widening (APW) over LLM-sampled actions.

We demonstrate strong performance on retrieval datasets HotPotQA, MuSiQue, and 2WikiMultihopQA as well as on reasoning datasets GPQA-diamond and BIG-Bench Hard against zero-shot baselines, including other MCTS-based methods. We find that this zero-shot prompting scheme can be used to decompose many multi-hop tasks into simple, approachable steps—enabling better leverage of test-time inference for more capable language models.

Limitations

Necessity of Decomposition Our method relies on problems that are indeed decomposable, even if the optimal decomposition is not obvious. However, for complex reasoning questions for which decomposition itself is not necessary (such as ARC-AGI tasks), our method cannot currently effectively create tree-search branches for non-separable problems.

Initial Cost For a fixed very small token budget, our method is not the most cost-efficient method to achieve multi-hop reasoning compared to sim-

ple CoT. Our method begins to scale with test-time compute at medium to large token regimes (e.g., those that would be comparable to Tree-of-Thoughts), where the model is able to perform sufficient tree-search rollouts to discover potentially good subplans; yet, at small token regimes, this is not possible and thus we do not outperform more efficient direct methods.

Tool Dependence Our method's current implementation relies on a number of tools that the LLM has to call in sequence. Although this is an efficient way to bootstrap our capabilities onto an existing LLM, it prevents weaker models that cannot effectively call tools or maintain state from implementing our approach effectively. Using our method as synthetic data for training a smaller model is an exciting potential future direction to address this.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *iclr*.

Richard Bellman. 1957. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michał Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of thoughts: solving elaborate problems with large language models. In *AAAI Conference on Artificial Intelligence (AAAI)*, AAAI'24/IAAI'24/EAAI'24. AAAI Press.

Zhenyu Bi, Daniel Hajialigol, Zhongkai Sun, Jie Hao, and Xuan Wang. 2024. STOC-TOT: Stochastic Tree-of-Thought with Constrained Decoding for Complex Reasoning in Multi-Hop Question Answering. *arXiv preprint*. ArXiv:2407.03687 [cs].

Steven Bird and Edward Loper. 2004. NLTK: The natural language toolkit. In *Proceedings of the ACL Interactive Poster and Demonstration Sessions*, pages 214–217, Barcelona, Spain. Association for Computational Linguistics.

Jiawei Chen, Hongyu Lin, Xianpei Han, and Le Sun. 2024. Benchmarking Large Language Models in Retrieval-Augmented Generation. In *AAAI Conference on Artificial Intelligence (AAAI)*, pages 17754–17762.

Adrien Couëtoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, and Nicolas Bonnard. 2011. Continuous upper confidence trees. In *Learning and Intelligent Optimization (LION)*, pages 433–445. Springer.

Rémi Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International Conference on Computers and Games*, pages 72–83. Springer.

Yue Fan, Hu Zhang, Ru Li, YuJie Wang, Hongye Tan, and Jiye Liang. 2024. FRVA: Fact-Retrieval and Verification Augmented Entailment Tree Generation for Explainable Question Answering. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 9111–9128, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Google. Gemma 4 model overview.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Li Jiapeng, Liu Runze, Li Yabo, Zhou Tong, Li Mingling, and Chen Xiang. 2024. Tree of Reviews: A Tree-based Dynamic Iterative Retrieval Framework for Multi-hop Question Answering. *arXiv preprint*. ArXiv:2404.14464 [cs].

K. Sparck Jones, S. Walker, and S. E. Robertson. 2000. A probabilistic model of information retrieval: development and comparative experiments. *Inf. Process. Manage.*, 36(6):779–808.

Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, and 1 others. 2022. Mrkl systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Omar Khattab, Christopher Potts, and Matei Zaharia. 2022. Baleen: Robust Multi-Hop Reasoning at Scale via Condensed Retrieval. *arXiv preprint*. ArXiv:2101.00436 [cs].

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *iclr*.

Mykel J Kochenderfer, Jessica E Holland, and James P Chryssanthacopoulos. 2012. Next-generation airborne collision avoidance system. *Lincoln Laboratory Journal*, 19(1):17–33.

Mykel J Kochenderfer, Tim A Wheeler, and Kyle H Wray. 2022. *Algorithms for decision making*. MIT press.

Tomáš Kočiský, Jonathan Schwarz, Phil Blunsom, Chris Dyer, Karl Moritz Hermann, Gábor Melis, and Edward Grefenstette. 2017. The NarrativeQA Reading Comprehension Challenge. *Preprint*, arXiv:1712.07040.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European Conference on Machine Learning (ECML)*, pages 282–293. Springer.

Alex Laitenberger, Christopher D Manning, and Nelson F. Liu. 2025. Stronger baselines for retrieval-augmented generation with long-context language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 32559–32569, Suzhou, China. Association for Computational Linguistics.

Mikko Lauri, David Hsu, and Joni Pajarinen. 2022. Partially observable markov decision processes in robotics: A survey. *IEEE Transactions on Robotics*, 39(1):21–40.

Seongmin Lee, Jaewook Shin, Youngjin Ahn, Seokin Seo, Ohjoon Kwon, and Kee-Eung Kim. 2024. Zero-shot multi-hop question answering via monte-carlo tree search with large language models. *arXiv preprint arXiv:2409.19382*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-augmented generation for knowledge-intensive NLP tasks. *Preprint*, arxiv:2005.11401 [cs].

Yuan Li, Qi Luo, Xiaonan Li, Bufan Li, Qinyuan Cheng, Bo Wang, Yining Zheng, Yuxin Wang, Zhangyue Yin, and Xipeng Qiu. 2025. R3-RAG: Learning step-by-step reasoning and retrieval for LLMs via reinforcement learning. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 10491–10507, Suzhou, China. Association for Computational Linguistics.

Omid Madani, Steve Hanks, and Anne Condon. 2003. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artificial Intelligence*, 147(1-2):5–34.

Robert J. Moss, Anthony Corso, Jef Caers, and Mykel Kochenderfer. 2024. BetaZero: Belief-state planning for long-horizon POMDPs using learned approximations. *Reinforcement Learning Journal*, 1:158–181.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 9340–9366, Miami, Florida, USA. Association for Computational Linguistics.

Christos H Papadimitriou and John N Tsitsiklis. 1987. The complexity of markov decision processes. *Mathematics of operations research*, 12(3):441–450.

Pranav Putta, Edmund Mills, Naman Garg, Sumeet Motwani, Chelsea Finn, Divyansh Garg, and Rafael Rafailov. 2024. Agent q: Advanced reasoning and learning for autonomous ai agents. *arXiv preprint arXiv:2408.07199*.

Peng Qi, Haejun Lee, Oghenetegiri "TG" Sido, and Christopher D. Manning. 2021. Answering Open-Domain Questions of Varying Reasoning Steps from Text. *arXiv preprint*. ArXiv:2010.12527 [cs].

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2019. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level google-proof q&a benchmark. In *Conference on Language Modeling (COLM)*.

Ruiyang Ren, Yuhao Wang, Junyi Li, Jinhao Jiang, Wayne Xin Zhao, Wenjie Wang, and Tat-Seng Chua. 2025. LLM-based search assistant with holistically guided MCTS for intricate information seeking. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, page 1098–1108.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. 2024. RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval.

David Silver and Joel Veness. 2010. Monte-Carlo Planning in Large POMDPs. In *Advances in Neural Information Processing Systems (NIPS)*, volume 23.

Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. 2013. DESPOT: Online POMDP planning with regularization. In *Advances in Neural Information Processing Systems (NIPS)*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, Agnieszka Kluska, Aitor Lewkowycz, Akshat Agarwal, Alethea Power, Alex Ray, Alex Warstadt, Alexander W. Kocurek, Ali Safaya, Ali Tazarv, and 431 others. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*. Featured Certification.

Zachary Sunberg and Mykel Kochenderfer. 2018. Online algorithms for POMDPs with continuous state, action, and observation spaces. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 28, pages 259–263.

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, and 1 others. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

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

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10014–10037, Toronto, Canada. Association for Computational Linguistics.

Liang Wang, Nan Yang, and Furu Wei. 2023. Query2doc: Query expansion with large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9414–9423, Singapore. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Shirley Wu, Shiyu Zhao, Michihiro Yasunaga, Kexin Huang, Kaidi Cao, Qian Huang, Vassilis N. Ioannidis, Karthik Subbian, James Zou, and Jure Leskovec. 2024. STaRK: Benchmarking LLM retrieval on textual and relational knowledge bases. In *Advances in Neural Information Processing Systems (NeurIPS)*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems (NeurIPS)*, pages 20744–20757.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pages 11809–11822.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023b. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 11809–11822.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023c. React: Synergizing reasoning and acting in language models. In *iclr*.

Hongming Zhang and Tianyang Yu. 2020. Alphazero. *Deep Reinforcement Learning: Fundamentals, Research and Applications*, pages 391–415.

Jiahao Zhang, Haiyang Zhang, Dongmei Zhang, Liu Yong, and Shen Huang. 2024. End-to-end beam retrieval for multi-hop question answering. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1718–1731, Mexico City, Mexico. Association for Computational Linguistics.

A Additional Experiments

Retrieval benchmarks. We present in Figure 6 further analysis of our approach in terms of the time and token budgets it takes to complete retrieval. Especially for 2WikiMultihop—a dataset which requires more hops on average than the others—our method confers significant performance gains at large token-usage budgets, where other methods stop improving.

Tool-call distribution by approach. As seen in Figure 8, we find that although replanning accounts for a very small percentage of actions taken (a mean of $N{=}52$ on the retrieval datasets), it is still a significant contributor to performance as shown in Figure 3.

Backbone-model robustness. We reproduce our results on GPQA-diamond using a variety of open-source models, including Gemma 4 (Google), **Kimi K2.6** (Team et al., 2025), and Qwen 3.5 (Yang et al., 2025). We find in Figure 9 that our method remains performant against ReAct baselines even with a locally hostable Gemma 4 31B, and that **Kimi K2.6 in particular delivers strong performance under our pipeline**—demonstrating that the gains from RAG-DOLL transfer to capable open-source backbones. We find that the approach underperforms slightly with Qwen 3.5, where most of the errors appear to be due to hallucinated tool calls to functions that did not exist. In this case we report tokenwise F1 (case sensitive) due to the models often not adheringd to the user instruction to produce the letter of the output only.

Native-thinking ablation. As seen in Figure 10, enabling thinking only confers a slight performance gain; this suggests that the majority of the performance gain in our approach comes from our approach's decomposition, rather than the LLM's own ability to reason.

B Experiment Hyperparameters

Table 2 lists the exact hyperparameters used for every experiment reported in the main body. The retrieval and reasoning settings share the same APW parameters and discount, but differ in search depth, simulation budget, UCB exploration constant, and rollout depth.

	Reasoning (GPQA, BBH)	**Retrieval** (HotPot, 2Wiki, MuSiQue)
MCTS		
`num_simulations`	16	32
`depth`	8	12
`c` (UCB)	1.0	1.4
`max_rollout_depth`	1	3
`discount` γ	0.995	0.995
Action Progressive Widening		
`apw_k` (k_a)	0.7	0.7
`apw_a` (α_a)	0.4	0.4
Backbone		
`model`	`gpt-5.4-mini`	
sampling temperature	0.8	
seeds	6	
Baselines		
ReAct `max_steps`	8	12
ToT `depth`	8	—
ToT `width`	2	—
ToT `beam_width`	4	—

Table 2: Hyperparameters used for every experiment reported in the body. The reasoning column applies to the GPQA-diamond and BIG-Bench Hard results (Figure 4, Figure 3, Figure 5); the retrieval column applies to HotPotQA, 2WikiMultihopQA, and MuSiQue (Figure 2, Table 1). Baselines are the implemented reimplementations described in Section 4.

C Exact Prompts Used for Each LLM Function

RAG-DOLL is implemented as a tool-using agent: the top-level LM is given a system prompt and a set of named tools, and every action it takes is either a plain assistant message or a function call into one of those tools. The tool *names* and *descriptions* below are exactly what the LM sees in its tool schema—they are the prompts. Several tools (e.g., `answer`, `plan`, `replan`, `subproblem`, `solve_subproblem`) take no parameters from the top-level LM and instead delegate to a one-off *surrogate* call that forks the conversation and asks for the action's content (the answer text, the plan, a chosen subproblem, etc.). We give the surrogate instructions verbatim alongside each handler.

C.1 System Prompt

System prompt

```
You are RAG-DOLL, an agent for solving complicated
↪ questions in multiple steps. Users will ask you a
↪ question, and you must find the answer. For instance,
↪ the user may ask "What's the name of the color of the
↪ Golden Gate Bridge?", and your answer should be
↪ "international orange" only.

To help you, you will have a number of tools available to
↪ you. You may and should call a number of them
↪ back-to-back until you arrive at a good answer.
```

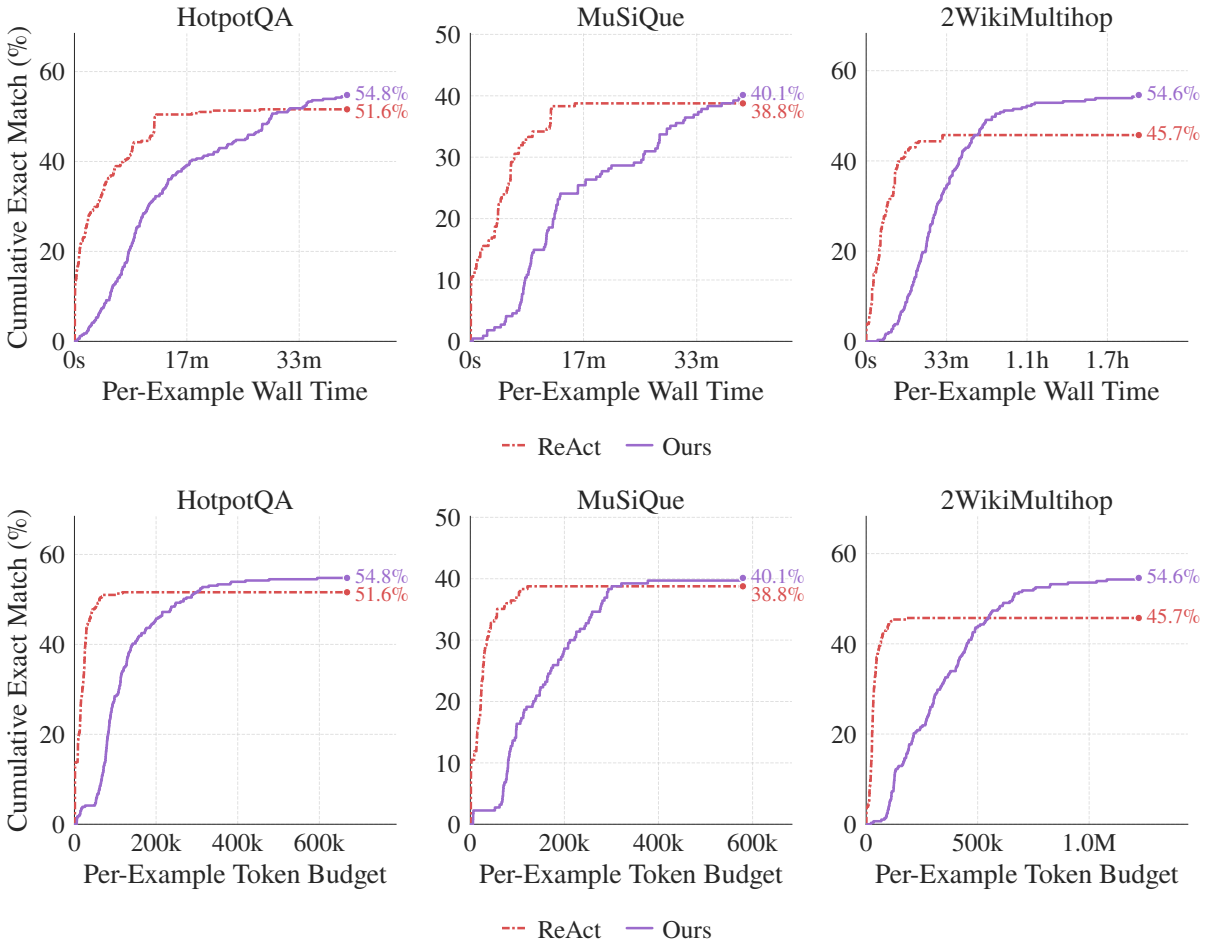


Figure 6: The performance of our method compared to ReAct on the three retrieval datasets (HotPotQA, MuSiQue, and 2WikiMultihopQA) as a function of wall-clock time and tokens. We evaluate the performance of each approach at a given time budget by counting a question not answered within the budget as incorrect, and otherwise evaluating the exact match of the retrieved context at the end of the time budget.

```
To submit your answer, use the `answer` tool. You can
↪  provide explanations to the user output as needed.

You will have a few steps before `answer`. DO NOT just
↪  call the `answer` tool right after the user's question
↪  unless its trivial to answer; you should try to break
↪  down prompts into multiple steps.
```

Previous-actions reminder

```
You have already explored the following actions from this
↪  exact decision point on previous attempts. Q is the
↪  current value estimate (higher is better; Q=N/A means
↪  it has not been evaluated yet). Choose a *different*
↪  action this time so we can explore the search space —
↪  do not repeat any of the actions below verbatim.

  • [Q=<value>] <tool>(<args>)
  • ...
```

C.2 Previous-Actions Reminder (Injected at Each Step)

Before each provider call, the agent injects a transient user message listing all actions previously sampled from the current state along with their current Q-value estimates. This biases the LM towards exploring the search space rather than re-issuing identical tool calls.

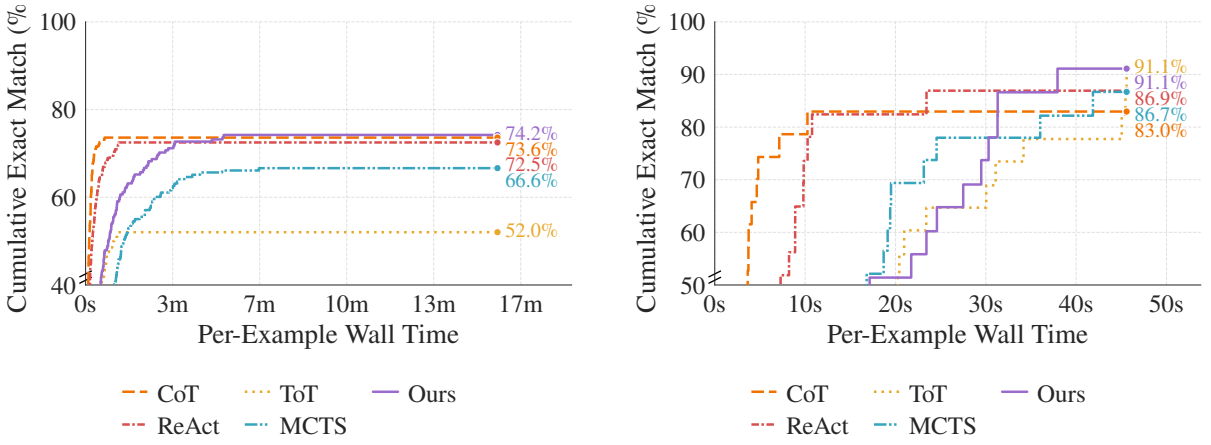
C.3 Tool: `answer`

Zero-argument tool. Invoking it triggers a surrogate call that drafts the final answer from the running conversation; the LM does not write the answer text itself.

Tool: answer

```
Tool name:        answer
Tool description: Submit your final answer. The answer text
↪  is drafted automatically from the conversation so far.

Surrogate instruction:
```



(a) GPQA-diamond

(b) BIG-Bench Hard

Figure 7: The performance of our method compared to ReAct on the two reasoning datasets (GPQA-diamond and BIG-Bench Hard) as a function of wall-clock time. We evaluate the performance of each approach at a given time budget by counting a question not answered within the time budget as incorrect, and otherwise evaluating the exact match of the retrieved context at the end of the time budget.

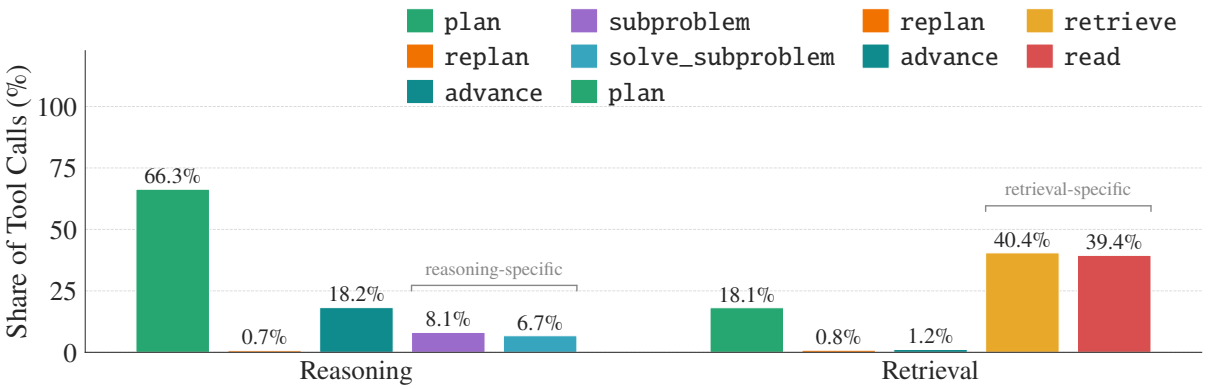


Figure 8: Distribution of tool calls taken by our approach in retrieval and reasoning datasets.

```
Based on the conversation so far, give your FINAL answer
↪ to the user's original question. Reply with ONLY the
↪ answer text — no preamble, no explanation, no
↪ tool-call syntax.
```

C.4 Tool: `plan`

Zero-argument tool, available only before any plan has been generated. The surrogate drafts a numbered plan from the conversation, which is parsed back into a list of steps; the agent then enters "step 0" of that plan.

Tool: `plan`

```
Tool name:        plan
Tool description: Come up with a step-by-step plan to
↪ guide your solution. The plan is generated for you
↪ (you do not supply it). The returned function output
↪ contains the plan and your current step. Call this
↪ before taking any other actions. For complex queries
↪ try to call this first.

Surrogate instruction:
```

```
Based on the conversation so far, devise a focused
↪ step-by-step plan to solve the user's question.
↪ Reply with ONLY the plan as a numbered list (one
↪ step per line, `1. ...`, `2. ...`). No extra
↪ commentary, no preamble.
```

C.5 Tool: `replan`

Zero-argument tool, available only after a plan exists. The handler runs two surrogate calls in sequence: a brief reflection on why the existing plan is inadequate, then a fresh numbered plan that incorporates the reflection.

Tool: `replan`

```
Tool name:        replan
Tool description: Request a new plan, replacing the
↪ current one. Both a brief reflection on the current
↪ plan and the new plan itself are generated for you.

Reflection surrogate instruction:
  Reflect briefly on why the current plan is inadequate
  ↪ given what you've learned so far. Reply with one or
  ↪ two sentences — no preamble.
```

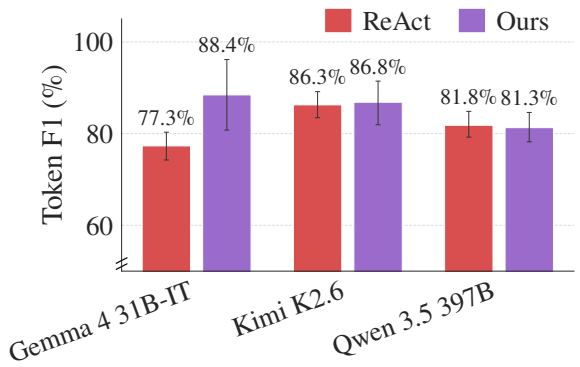


Figure 9: GPQA-diamond F1 for our method using a variety of open-source models, including Gemma 4, Kimi K2.6, and Qwen 3.5. 10–90 Beta credible intervals over 6 seeds shown as error bars.

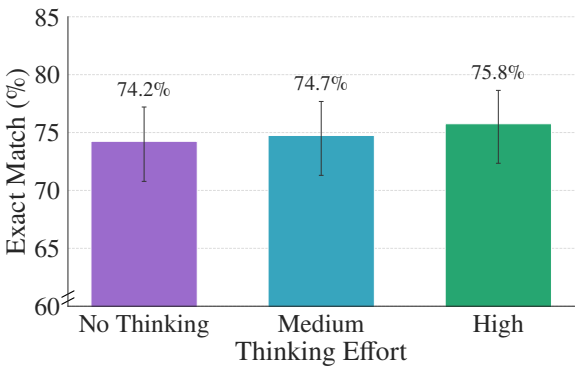


Figure 10: GPQA-diamond exact-match for our method compared to a native-thinking ablation, where the agent under test can use the built-in CoT to reason before taking actions. 10–90 Beta credible intervals over 6 seeds shown as error bars.

```
Plan-regeneration surrogate instruction:
  The previous plan was inadequate. Reflection:
  ↪  <reflection>

  Based on the conversation so far, devise a focused
  ↪  step-by-step plan to solve the user's question.
  ↪  Reply with ONLY the plan as a numbered list (one
  ↪  step per line, `1. ...`, `2. ...`). No extra
  ↪  commentary, no preamble.
```

C.6 Tool: advance

Zero-argument tool, available only after a plan exists. Moves the agent to the next step of the current plan; no surrogate call.

Tool: advance

```
Tool name:        advance
Tool description: move on to the next step of the plan
```

C.7 Tool: retrieve (retrieval agent)

The first of two parameterized tools: the top-level LM picks the query string itself, so semantically distinct queries are distinct actions in the MCTS tree. The handler calls a sparse FTS search and returns the result as a numbered list of titles.

Tool: retrieve (retrieval agent)

```
Tool name:        retrieve
Tool description: The query for the retrieval; it cannot
↪  handle complex queries, so provide a single noun or
↪  keyword that could match.
Parameter `query` (string, required):
                  A single keyword or short noun phrase to
                  ↪  search for. Avoid full sentences.
```

C.8 Tool: read (retrieval agent)

The second parameterized tool: the LM provides the integer ID of a previously retrieved title. The handler fetches that article's summary and returns it.

Tool: read (retrieval agent)

```
Tool name:        read
Tool description: Call this tool after calling `retrieve`.
↪  Provide which document you want to read from retrieval
↪  output, identified by the integer ID shown in the
↪  retrieve result (e.g. ``ID 5: . . .``). You must call
↪  ``retrieve`` at least once before ``read``.
Parameter `id` (integer, required):
                  The integer ID of the retrieved element
                  ↪  to read, as shown in a prior
                  ↪  ``retrieve`` result.
```

C.9 Tool: subproblem (reasoning / tool agents)

Zero-argument tool. Decomposes the current plan step into 2–4 self-contained subproblems via a surrogate call; each subproblem gets a fresh integer ID and is added to the agent's working state.

Tool: subproblem (reasoning / tool agents)

```
Tool name:        subproblem
Tool description: Decompose the current plan step into
↪  smaller self-contained subproblems. Returns a numbered
↪  list of subproblem IDs that you can then reason over
↪  via `solve_subproblem`.

Surrogate instruction:
  Current plan step: <step>

  Decompose this step into 2-4 smaller, self-contained
  ↪  subproblems whose solutions, combined, would solve
  ↪  the step. Reply with one subproblem per line,
  ↪  numbered (`1. ...`, `2. ...`). No extra commentary.
```

C.10 Tool: solve_subproblem (reasoning agent)

Zero-argument tool. A single surrogate call both picks an open subproblem and writes its solution, which the handler then files back into the subproblem state.

Tool: `solve_subproblem` (reasoning agent)

```
Tool name:        solve_subproblem
Tool description: Pick one open subproblem and record
↪  reasoning for it. Both the choice of subproblem and
↪  the solution text are drafted automatically; you only
↪  choose to invoke this tool.

Surrogate instruction:
  From the following OPEN subproblems, pick ONE to solve
  ↪  now and write your reasoning. Reply in EXACTLY this
  ↪  format and nothing else:
  ID: <integer>
  Solution: <your solution on one line or more>

  Open subproblems:
    ID <k>: <subproblem text>
    ...
```

C.11 MCTS trajectory-judge scoring prompt

Outside the agent's tool surface, our MCTS solver issues one auxiliary scoring call per visited state to form the LM-judge reward. The same prompt is used for intermediate and terminal states; the magnitude of the resulting reward is scaled differently in the two cases (`intermediate_reward_magnitude` vs. `correctness_reward_magnitude` in `MCTSConfig`). The judge runs as a separate two-message conversation (system + user), not as a tool call, so it does not appear in the agent's trajectory.

MCTS trajectory-judge prompt

```
System: You are a strict evaluator of agent trajectories.

User: Given the question and the steps, determine if the
↪  steps are helpful for answering the question on a 1-5
↪  scale. (5 = clearly correct and well supported by
↪  context or intermediate work; 1 = wrong, unsupported,
↪  or no context). Reply with ONLY the integer.

=== Trajectory ===
USER: <user question>
CALL <tool>(<args>)
RESULT: <tool output>
ASSISTANT: <free-text reasoning>
...
```