RAG-DOLL: Zero-Shot LLM Multi-Hop Question Answering by Simultaneously Planning Over Documents and Strategies

Houjun Liu*, **Amelia F. Hardy, Samuel Akinwande, Robert J. Moss, Duncan Eddy**
Christopher D. Manning and **Mykel J. Kochenderfer**
Stanford University
*houjun@stanford.edu

Abstract

Language models demonstrate surprisingly strong zero-shot multi-hop question-answering (MHQA) capabilities; however, most prompting schemes for zero-shot MHQA suffer in performance either due to a lack of ability to backtrack on wrong retrieval decisions or modify the plan for decomposing the multi-hop question given new information. In response, we introduce RAG-DOLL, a zero-shot LLM MHQA formulation that uses a Markov decision process (MDP) to simultaneously plans over *both* selecting a good strategy to decompose the MHQA query *and* selecting the right documents to be retrieved given that strategy. We solve this formulation with Monte-Carlo Tree Search (MCTS) and demonstrate state-of-the-art exact match performance on HotPot QA Fullwiki. We find also that even a simple, weighted stochastic policy leveraging our action space outperforms chain-of-thought prompting. Our method thus sets a new, competitive baseline for zero-shot prompts on hard, multi-step reasoning challenges, paving the way to better leverage test-time inference for retrieval-heavy tasks.

1 Introduction

Large Language models (LLM) can store and compress an immense wealth of information (Petroni et al., 2019) as well as perform reasoning with natural language at scale (Kaplan et al., 2020). Yet, despite these advancements, recent work highlights that there are still significant gaps in multi-step information retrieval (Chen et al., 2024; Wu et al., 2024).

In order to enable accurate information retrieval with language models with minimal hallucination, Lewis et al. (2021) proposes Retrieval Augmented Generation (RAG), an information retrieval pipeline external to a language model that fulfills user queries by using a sequential process that first encodes the user query to match against true documents in a corpus, and then conditions an answer generator (an LLM) against these matched documents to produce the final answer.

Though a breakthrough technique for truthful LLMs, the success of these RAG systems is tempered by the initial retrieval step. When information needed to fulfill the user query is not fully present in the user query, the query-once approach given by RAG degrades in performance. This effect can be observed particularly in datasets for which the query explicitly requires sequential information gathering steps to answer (Yang et al., 2018) or in long-form retrieval tasks where the answer is buried in context (Kočiský et al., 2017).

One set of techniques that have found success in mitigating problems like this involves prompting the LLM to perform sequential reasoning that breaks the large query down, essentially *strategizing* the method of solution before solving the problem. A very basic way to achieve this effect is Chain-of-Thought (CoT) prompting (Wei et al., 2022), which asks the language model to explain its stepwise retrieval decisions. Each retrieval step can even be augmented with retrieved knowledge individually (Trivedi et al., 2023).

A limitation of CoT approaches involves the fact that errors early on during reasoning will propagate to later stages without backtracking, leading to erroneous answers. In response, tree-based search solutions have emerged that plans over multiple possible sub-queries at each stage (Qi et al., 2021; Khattab et al., 2022; Sarthi et al., 2024; Fan et al., 2024; Bi et al., 2024; Jiapeng et al., 2024), sometimes even leveraging Monte-Carlo tree search to trade off efficiency of exploration and retrieval outcomes (Lee et al., 2024; Ren et al., 2025). While significantly improving multi-hop performance, planning over sub-queries still means that the retriever still commits to one fixed plan: it is simply searching over the best queries to help fulfill it. Thus, espe-

cially for zero-shot approaches that do not incorporate carefully designed domain-specific heuristics (Zhao et al., 2020; Qi et al., 2021), erroneous initial query strategies can still propagate through to the retrieval outcome.

In response, we introduce **RAG-DOLL, a zero-shot planning-based approach for solving multi-hop retrieval that simultaneously plans over sub-query choices as well as query decomposition plans**. In particular, RAG-DOLL leverages the efficient exploration of Monte-Carlo Tree Search (MCTS) (Silver and Veness, 2010) and the effective heuristic given by zero-shot LLMs to simultaneously choose both the *plan* for solving a particular complex query as well as the actual *documents* to be explored. We do this by incorporating the sequential query plan into the state space of MCTS exploration, and choosing actions that either execute or revise each step of the plan. To prevent the action space from being intractable, we limit our actions to be local to the step of the plan we are executing (i.e., to revise the previous step of the plan, we must take two actions to first move up the plan being executed and then to revise it).

We implement this planning strategy using DSPy (Khattab et al., 2024), and demonstrate state-of-the-art performance on multi-hop question-answering datasets compared to other zero-shot prompting methods. Our work thus introduces three main contributions:

1. We propose a way to simultaneously formulate multi-hop retrieval plans and the actual documents to explore as a single planning problem by composing local actions on two writable context tapes.

2. We introduce a language-model weighted Q-function initialization scheme for Monte-Carlo tree search that prioritizes exploring valuable actions during planning, drastically speeding convergence.

3. We show how these two methods can be integrated together to form a LLM based multi-hop QA solver, and demonstrate state-of-the-art zero-shot performance on HotPotQA Fullwiki (Yang et al., 2018).

We hope this system provides both insights into the design of adaptive language model retrieving pipelines, and can serve as a useful baseline for multi-hop retrieval.

2 Related Work

Multi-Hop Information Retrieval Our project 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 on the document level (synthesizing multiple documents in 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 have also been previously proposed (Lee et al., 2024; Ren et al., 2025) as a well to sequence exploring 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 problem involves leveraging an effective zero-shot prompting method. Work in this area began through the discovery that Chain-of-Thought prompting (Wei et al., 2022) significantly improves LLM performance, and was subsequently extended by methods that leverage input feedback such as ReAct (Yao et al., 2023) and MRKL (Karpas et al., 2022). More recently, search-inspired prompting methods such as tree of thoughts (Yao et al.) and graph of thoughts (Besta et al., 2024) have also emerged that further improves 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 emerged to bootstrap zero-shot prompts into few-shot systems.

POMDP Solution Methods Modeling complex sequential decision problems as a Partially Observable Markov Decision Process has shown to be successful in many domains such as aircraft safety (Kochenderfer et al., 2012) and robotics (Lauri et al., 2022); for language models in particular, successful POMDPs have been formulated for web language agents (Yao et al., 2022; Putta et al., 2024) to sequence complex multi-step interactions. Solving a POMDP exactly is intractable (Spaan and Vlassis, 2005) but online solvers that use tree search

(Silver and Veness, 2010; Somani et al., 2013; Sunberg and Kochenderfer, 2018) have excelled in performance. Most recently, deep hybrid planning methods (Zhang and Yu, 2020; Moss et al., 2024) remove the dependence of prior methods on selection heuristics and instead learns a solution through neural networks.

3 Methods

RAG-DOLL is a zero-shot prompting pipeline that solves the multi-hop retrieval task when provided access to a single tool that queries the underlying knowledge base. We will now describe the construction of this pipeline. First, we formulate the retrieval task of planning over *both* retrieval strategies and documents to read as a single MDP. Second, we introduce a language-model Q-weighted MCTS method in to solve the MDP we propose to yield 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) is a model formulation in reinforcement learning to enable sequential decision making over observable states. MDPs are characterized by the tuple $\langle S, A, T, R, s_0, \gamma \rangle$. Given the current state $s \in S$ and action $a \in A$ to transition to a new state $s' \sim T(\cdot|s, a)$ with rewards $R(s, a, s')$. The goal of an MDP is to maximize the expected sum of future rewards.

The task of multi-hop retrieval is especially well-suited for this formulation because the quality of next decisions in what information to seek is heavily dependent on the quality of previous information already sought. Attempts have been made to formulate retrieval as an MDP, but always with a state space in terms of documents to be retrieved (Lee et al., 2024; Ren et al., 2025; Bi et al., 2024; Jiapeng et al., 2024).

Here, we provide a MDP formulation of retrieval whose state **simultaneously encodes the plan for retrieval and the retrieved documents**, enabling the agent to take the information it gathered to completely 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 "tapes": one containing the current retrieval results, and the other containing the current plan. The current "state" of the machine (i.e., which retrieval result it is considering and which step of the plan it is considering) naturally forms the state of our MDP.

Formally, our state forms a four tuple:

$$(C, G, g, d) \in S \tag{1}$$

for the set of all documents in the corpus D, a plan G for solving the current question the current step of the plan $g \in G$, the current context $C \subset D$, and the current document to explore $d \in D$.

3.1.2 Actions and Transitions

Our MDP has four discrete actions, each of which yields a distinct possible output distribution in transition. During transitions, we have access to a retriever tool that yields $d_1, \ldots d_n \sim \text{Retrieve}(x)$ for some query x and documents $d_j \in D$ returned in ordered rank (i.e., such that d_1 is the most relevant document with respect to x), current state $s = (C, G, g_j, d_k)$ written in conventions described in section 3.1.1, and a planning-prompted LLM used to modify the current plan given context $\text{Replanner}(C, G)$ described in section 3.4.

a_1**: next step** We take the next step in the plan and add the current document being considered into context, namely, we transition to $s' = (C \cup \{d_k\}, G, g_{j+1}, d_1 \sim \text{Retrieve}(g_{j+1}))$.

a_2**: next document** We swap the current document being considered for the one with the next highest relevance. $s' = (C, G, g_j, d_{k+1})$. This transition is thus entirely deterministic since the retrieval output of $\text{Retrieve}(g_j)$ is always cached.

a_3**: modify plan** Given the current context C, we come up with a new plan that extends the steps already taken. This makes $s' = (C, \{g_1 \ldots g_{j-1}\} \cup G' \sim \text{Replanner}(C, G), g'_1, d_1 \sim \text{Retrieve}(g'_1))$. Note that we also retrieve a new context for the current document given this new plan.

a_4**: answer** The final action terminates the simulation and answers the question using a language model given the retrieved context C. This action can be generated by sampling a variety of different answers at a high temperature given the same context.

In particular, notice that transitions between states are only non-deterministic with respect to the output of the retriever and planner. This dramatically cuts down on uncertainty, and allows the

user to trade-off the speed of convergence with retrieval performance by simply fixing some or all transitions to be deterministic via caching.

Observe also that these choices of action naturally decomposes the task of *planning the retrieval* and *selecting the documents* into one unified MDP.

Both the Retrieve and Replanner tools are described in further detail in section 3.4.

3.1.3 Reward

Unlike previous works that incorporate handcrafted process-level supervision, we use a very simple heuristic as our reward. We simply prompt a language model $\text{Correct}(s, q, a) \in [x, y] \in \mathbb{R}^+$ for some positive range $[x, y]$ to output an integer within a fixed range for question q, answer a, and state s to judge whether the context supports the answer.

For process-level reward assignments, we prompt another language model $\text{Relevant}(C, q) \in [x, y] \in \mathbb{R}^+$ for some positive range to judge whether the context is relevant to the question.

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

$$R(s, a, s') = \begin{cases} \alpha_1 \left[\frac{1}{5}\text{Correct}(s, q, a)\right], & \text{if } a = a_4 \\ \alpha_2 \left[\frac{1}{5}\text{Relevant}(s, q)\right], & \text{otherwise} \end{cases} \tag{2}$$

We describe the design of the LLM driven Correct and Relavent functions in section 3.4.

3.2 Monte-Carlo Tree Search (MCTS)

In order to actually determine the sequence of actions in our MDP that leads us to obtaining a good final answer, we perform Monte-Carlo Tree Search (MCTS)(Coulom, 2006; Silver and Veness, 2010). All variants of Monte-Carlo tree search executes 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; we do so by trying to balance exploration of possible actions and 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:

$$\arg\max_j Q(s, a_j) + c\sqrt{\frac{\log \sum_{k=1}^{4} N(s, a_k)}{N(s, a_j)}} \tag{3}$$

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 take the appropriate state modifications and transitions outlined in section 3.1.2, giving us the next state of our tree s'.

Simulation Next, for the simulation step, we follow AlphaZero (Zhang and Yu, 2020) and BetaZero (Moss et al., 2024) in using a heuristic—and 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., to answer the question immediately given current state), and using the resulting reward as the value at leaf. This is appropriate because all multi-hop datasets have a finite (and usually small, between 1 and 5) number of hops, 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, is propagated along the entire path of the tree. In particular, we update our Q values at each node by computing:

$$q = R\big(s, a, s'\big) + \gamma Q\big(s', a'\big) \tag{4}$$

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

$$Q(s, a) := \frac{q - Q(s, a)}{N(s, a)} \tag{5}$$

Tree Policy and Data Collection After construction, for a given query, the MCTS tree root node contains a collection of expanded actions (subqueries), each of which has a weighting based on visit counts and Q-values.

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

$$\pi_{\text{tree}}(s,a) \propto \left(\left(\frac{e^{Q(s,a)}}{\sum_{a' \in \{a_1,a_2,a_3,a_4\}} e^{Q(s,a')}}\right)^{z_q} \left(\frac{e^{N(s,a)}}{\sum_{a' \in \{a_1,a_2,a_3,a_4\}} e^{N(s,a')}}\right)^{z_n}\right)^{\frac{1}{\tau}} \quad (6)$$

with hyperparameters that weights Q-maximizing counts z_q and multiple visit ("robust") counts z_n, as well as the temperature τ. This follows insights from recent work that trades exploration of less visited actions with highly-valuable actions (Moss et al., 2024).

3.3 Language-Model Q-Weighted MCTS

Although the method described in section 3.2 serves as a sufficient baseline for multi-hop QA, it can require many tree expansion cycles before converging to a reasonable policy.

One reason why convergence may be slow is that some actions are almost certainly more valuable than others (e.g., answering directly without any context will definitely be bad), but a naive tree policy will need to spend exploration cycles trying those actions.

To improve the rate of convergence, we initialize our Q values with a language model's judgment of the best action to take at every given state. We create a language model prompt $\text{Recommend}(s) : S \to [1,5]^4$ which provides a numerical score for each of the four actions given each state in terms of how useful the language model thinks the action may be given context. We describe the design of this prompt further in section 3.4. We thus initialize our Q values before any tree search occurs with:

$$Q_{\text{init}}(s_k, a_j) = \frac{\text{Recommend}(s_k)_j}{\sum_{i=1}^{4} \text{Recommend}(s_k)_i} \quad (7)$$

by normalizing the value scores of each action provided by the recommendation prompt.

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, is constructed with DSPy (Khattab et al., 2024). In particular, each tool is instantiated as a single DSPy Program, which leverages the JSON-filling interface available for most major hosted LLMs to provide a structured input and output schema.

Retrieve The most important tool for multi-hop QA is naturally the one-hot retriever on which it is based. $\text{Retrieve} : G \to 2^D$ is a function that maps a subgoal g to a set of documents relevant to achieving this goal $d_1 \ldots d_n \in D$. We implement this following a classic two-phase, document-order RAG pipeline (Laitenberger et al., 2025). We first use an initial prompt (appendix A.1) to generate a proposal for some keywords $k_1, \ldots, k_m$ as well as a sub-question q from our goal g. Then, we perform a rough Okapi BM25 retrieval (Jones et al., 2000) via $k_1 \ldots k_m$ to obtain the top $O\left(m \times 10^2\right)$ most relevant documents. Then, following Laitenberger et al. (2025) and other similar baselines, we then perform a *local dense reranking* using a small embedding language model by ordering the resulting documents' embedding vectors via their cosine similarity with that of q. Finally, we return the top n most relevant documents as $d_1 \ldots d_n$.

Replanner In order 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 context, and formulate a new, revised plan. This prompt is given in appendix A.2.

Answer Answer is again a very simple CoT prompt that takes the context and the full multi-hop query, and asks the language model to answer the question. Its exact construction is given in appendix A.3.

Relavent and Correct We use two prompts that each score a context along with an optional answer for their 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 that we expect; thus, we instantiate the values as a categorical classification task 5 choices from worst to best (i.e., instead of asking the LLM to choose a number). The details of these prompts are given in appendix A.4.

Recommend Lastly, we use a prompt to generate the initial Q values evaluating each state. To do this, we ask the language model to use CoT to reflect upon the value of taking each action given the goals being considered, the context, and the current subgoal. Each action is then assigned a positive integer, which is then normalized to initialize the final Q values. This prompt is given in appendix A.5.

4 Experiments

4.1 Model Selection

Our primary experiments are conducted using `gpt-4o-mini`, a state-of-the-art, API level language model. We choose this model for two primary reasons: that evaluation is fast and cheap, and that many other comparable methods on the same dataset uses the same model. To perform sparse retrieval, we use `sqlite-fst5`[1]; to obtain document-query embeddings for dense retrieval, we use a small language model `snowflake-arctic-embed-m-v1.5`[2].

During evaluation, we allow MCTS to take at most 6 actions; should no answer result within 6 steps, we terminate and consider the answer to be a single empty string for evaluation (which will not match ground truth in any token and yield both EM and F1 scores of 0).

4.2 Evaluations

Corpora We evaluate our approach using the standard Multi-Hop QA dataset HotpotQA (Yang et al., 2018) in the full-wiki setting. We use the Wikipedia corpus, consisting of 5,233,329 documents, as our retrieval corpus.

Sampling Strategy To align with prior work, we sample 130 dev-set samples for evaluation (Jiang et al., 2025; Ren et al., 2025). To maintain etiological validity due to the large variance in performance, however, our results are reported in terms of a bootstrapped average of 300 independently drawn subsets of 130 samples. This allows us to report a low standard error bound while maintaining the same small sample size to match prior work.

Evaluation Metrics As is standard with other Multi-Hop QA experiments, we report two primary metrics: Exact Token-For-Token Match (EM), as well as Token F1 (F1), whereby an mean F1 measure is computed between the list of tokens produced by our policy versus gold. When comparing matches across both metrics, we disregard case differences, punctuation, as well as stopwords as detected by NLTK (Bird and Loper, 2004). Since the evaluation is in English only, we use spaces as a rough homogenization metric.

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

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

4.3 Baselines

We experiment and report a variety of baselines reported in both literature as well as ablations of our own method. This allows us to uncover which parts of our method contributes to the most performance gains, as well as how our method compares against the state of the art.

Baseline Zero-Shot Methods We report results of our method against one-shot methods like RAG (Karpukhin et al., 2020), standard Chain-of-Thought (Wei et al., 2022), and adaptive methods including ReAct prompts (Yao et al., 2023), Query2Doc (Wang et al., 2023), and Self-RAG (Asai et al., 2024). We additionally baseline against zero-shot MCTS-based methods, including HG-MCTS (Ren et al., 2025), MZQA-BC (Lee et al., 2024).

Baseline Ablations To further gain insights into the performance of our method, we additionally benchmark against a few ablative baselines. We examine the performance of greedily taking the highest-scoring action from the Recommend prompt (section 3.4) linearly at every step (i.e., instead of MCTS), sampling directly from the LM recommendations and took the sampled action linearly, and taking entirely random actions.

We run and report all baselines via `gpt-4o-mini`.

5 Results

Our method outperforms all baselines in both exact match and tokenwise F1 metrics. As highlighted in table 1, our method outperforms all baselines, including ones that also use MCTS as the solution method. In particular, we achieve state-of-the-art zero shot results while using models at the same scale or smaller—as in the case of MZQA-BC (Lee et al., 2024).

Prompting alone does not give good performance. The results in table 2 highlights that the prompting scheme alone, without searching over optimal actions with MCTS, does not give performant retrieval results. We believe this is due to the fact that a linear search scheme with these sampling methods will not provide the opportunity to take information seeking steps and backtrack from them, even despite the fact that our action space searches over strategies and documents.

	HotPot QA	
	Ans. Exact Match	Ans. Tokenwise F1
CoT (Wei et al., 2022)	26.47	32.35
RAG (Lewis et al., 2021)	41.18	52.94
ReAct (Yao et al., 2023)	35.88	42.35
Self-RAG (Asai et al., 2024)	38.82	47.65
IRCoT (Trivedi et al., 2023)	42.3	54.1
Query2doc (Wang et al., 2023)	44.71	54.71
HG-MCTS (Ren et al., 2025)	41.76	58.69
MZQA-BC (Lee et al., 2024)	47.0	**59.6**
Ours	**48.8**	**59.6**

Table 1: Performance of our method using `gpt-4o-mini` compared to other zero-shot baselines. MZQA-BC uses `gpt-4o` as the backbone, while all other baselines use `gpt-4o-mini`. Measurements were conducted the HotPot QA Fullwiki Dev set, using procedure as described in section 4. MZQA results were collected in Lee et al. (2024), while all other baselines were measured by Ren et al. (2025) using the same model and same data subsets as in our work. Sampling temperature is 0.8.

	HotPot QA	
	Ans. Exact Match	Ans. Tokenwise F1
Greedy Sampling	4.2	5.1
Weighted Sampling	28.0	35.0
Random Sampling	24.2	30.0
Ours	**48.8**	**59.6**

Table 2: Performance of our method compared to baseline ablations leveraging the same prompts as in section 4.3 on `gpt-4o-mini` using HotPot QA Fullwiki Dev set. In particular, we measure the performance of policies taking our same action space by sampling actions greedily (i.e., using the LM's most recommended action at each step); sampling actions either uniformly or weighted by LM recommendations; as well as the actual MCTS policy that we propose. In all weighted sampling tasks, including for LM token generation, we set sampling temperature is 0.8.

Even weighted stochastic sampling with our action space outperforms Chain of Thought. Curiously, table 2 indicates that *sampling*, whether using an LLM-guided heuristic or entirely randomly, achieves a significant performance boost over taking the greedy action. In fact, the weighted sampling version in our approach outperforms CoT prompting on HotPot QA (table 1). We believe this is the case because the structure of our action space allows for taking *smart, local steps* such that any walk along them will yield a reasonable retrieval context.

6 Conclusion

We introduced RAG-DOLL, a zero-shot prompting framework that uses local actions on a writable context to formulate multi-hop question answering as a planning problem over both the *strategy* of complex query decomposition as well as 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), along with a novel language-model weighted Q-function initialization scheme.

We demonstrate state-of-the-art performance on HotPot QA Fullwiki against zero-shot baselines, even among approaches that used a larger backbone model and with MCTS. We additionally highlight that, by leveraging our action space design, even a weighted stochastic policy can outperform chain-of-thought. We hope that this zero-shot prompting scheme can be useful to decompose many multi-hop tasks into simple, approachable steps—enabling better leverage of test-time inference for more capable language models.

References

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

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 *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, 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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17754–17762.

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.

Jinhao Jiang, Jiayi Chen, Junyi Li, Ruiyang Ren, Shijie Wang, Xin Zhao, Yang Song, and Tao Zhang. 2025. RAG-star: Enhancing deliberative reasoning with retrieval augmented verification and refinement. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7064–7074, Albuquerque, New Mexico. Association for 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.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. *Preprint*, arXiv:2001.08361.

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 *The Twelfth International Conference on Learning Representations*.

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

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*, 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 32547–32557, 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].

Robert J Moss, Anthony Corso, Jef Caers, and Mykel Kochenderfer. 2024. Betazero: Belief-state planning for long-horizon pomdps using learned approximations. In *Reinforcement Learning Journal*.

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.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. 2019. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China. Association for Computational Linguistics.

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.

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 *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '25, page 1098–1108, New York, NY, USA. Association for Computing Machinery.

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*, volume 23. Curran Associates, Inc.

Adhiraj Somani, Nan Ye, David Hsu, and Wee Sun Lee. 2013. Despot: Online pomdp planning with regularization. *Advances in neural information processing systems*, 26.

Matthijs TJ Spaan and Nikos Vlassis. 2005. Perseus: Randomized point-based value iteration for pomdps. *Journal of artificial intelligence research*, 24:195–220.

Zachary Sunberg and Mykel Kochenderfer. 2018. Online algorithms for pomdps with continuous state, action, and observation spaces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pages 259–263.

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 *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

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 *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

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*, 35:20744–20757.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. 36:11809–11822.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

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.

Chen Zhao, Chenyan Xiong, Xin Qian, and Jordan Boyd-Graber. 2020. Complex Factoid Question Answering with a Free-Text Knowledge Graph. In *Proceedings of The Web Conference 2020*, pages 1205–1216. ArXiv:2103.12876 [cs].

A Exact Prompts Used for Each LLM Function

Here we give the exact DSPy signatures and modules used to obtain each language model based function used throughout our work. Each of these sections are referred above in the main text whenever each of these programs are used, and are not separately described here since multiple programs may be chained together to form one function descried in the text.

A.1 Question Generation

```
class Subquestion(dspy.Signature):
    """You are trying to find the answer to a multi-hop retrieval task
     using intermediate documents; given a plan to solve the question
     and the current step in the plan we want to tackle, generate a list
     of good titles to explore as well as query that could be useful to
     answer the current step of the plan. You are not allowed to use
     existing knowledge and should only use the context given, and
     instead should pose questions to exactly answer the current step
     in the current goal."""

    question: str = dspy.InputField()
    context: list[Context] = dspy.InputField()
    goals: list[str] = dspy.InputField()

    current_goal: str = dspy.InputField()

    titles_to_explore: list[str] = dspy.OutputField(
        desc="Return a list of best wikipedia entities (a single "
             "noun phrase or article title) to explore which would "
             "help answer the current goal. A good place to start "
             "are entities mentioned in the goal. Do not use "
             "parentheses or additional information or qualifiers. "
             "Be as extensive as possible and cover all of your bases.")
    query_to_explore: str = dspy.OutputField(desc="A single sub-question that
                                                   could re-rank the documents above
                                                   achieve the current goal")
subquestion = dspy.ChainOfThought(Subquestion)
```

A.2 Replanning

```
class Replanner(dspy.Signature):
    """You are trying to find the answer to a multi-hop retrieval task using
       intermediate documents; you are presented with the question, and you
       must come up with a plan to solve it. You will only have the summary
       of around one Wikipedia article to answer each goal, so be sure to
       break down your goals accordingly such that the information you seek
       would be contained within some Wikipedia article. Break down your
       questions accordingly. You are not allowed to use existing knowledge
       and should only use the context given."""

    question: str = dspy.InputField()
    context: list[Context] = dspy.InputField()

    goals: list[str] = dspy.InputField()
    current_goal: str = dspy.InputField()

    critique: str = dspy.OutputField(desc="Describe why the old goals may be bad, "
                                          "and how you plan to fix it in the new "
                                          "goals. Reiterate that your goals will "
                                          "not start from scratch, and instead will
                                          "start from the step of the current goal
    new_goals: list[str]  = dspy.OutputField(desc="Break the query down into a sequ
                                                   "of sequential sub-questions tha
                                                   "lead into another. You will onl
                                                   "the summary of around one Wikip
                                                   "article to answer each goal, so
                                                   "sure to break down your goals a
                                                   " such that the information you
                                                   " be contained within some "
                                                   "Wikipedia article.")
replanner = dspy.ChainOfThought(Replanner)
```

A.3 Question Answering

```
class Answer(dspy.Signature):
    """Using solely informatino from the context, answer the question."""

    question: str = dspy.InputField()
    context: list[Context] = dspy.InputField()

    answer: str = dspy.OutputField(
        desc="the answer to the question, according to context. It
              "should be short like \"Lincoln Tunnel\" or \"yes\"."
    )
answerer = dspy.ChainOfThought(Answer)
```

A.4 Context Relevance

```
class Status(Enum):
    zero = 0
    one = 1
    two = 2
    three = 3
    four = 4
class ScoreAction(dspy.Signature):
    """Given the question and the context, determine if the
       context is helpful for answering the question"""
    question: str = dspy.InputField()
    context: list[Context] = dspy.InputField()
    rating: Status = dspy.OutputField(
        desc="rate how on track the context is to answering
              the question; 0 means very far away, 2 means
```

```
            somewhat relavent or in a similar genre, 4 means
            the context exactly answer the question fully with
            no additional information needed."
    )

edge_utility = dspy.Predict(ScoreAction)
```

A.5 Action Relevance

```
class Action(dspy.Signature):
    """
    You are trying to find the answer to a multi-hop
    retrieval task using intermediate documents; you
    are given a plan to answer the question, the
    retrieved context so far, and with four tools.
    You have to give a score between 1 (least
    useful) to 5 (most useful) in terms of how
    useful each tool will be guven the context.

    Tools:
    - answer_subquestion: answer the current subquestion
                          in the plan using the context given,
                          and move on to the next subquestion
    - answer_question: answer the full multi-hop query using the
                       context given
    - next: read the next document that was retrieved to
            answer the current subquestion
    - replan: given the context, change the current step of the
              plan and generate a new subquestion
    """

    question: str = dspy.InputField()
    context: list[Context] = dspy.InputField()
    goals: list[str] = dspy.InputField()

    current_goal: str = dspy.InputField()
    current_subquestion: str = dspy.InputField()

    answer_subquestion_score: float = dspy.OutputField(
        desc="Usefulness from 1 (least) to 5 (most) for"
             "answer_subquestion tool"
    )
    answer_question_score: float = dspy.OutputField(
        desc="Usefulness from 1 (least) to 5 (most) for answer_question tool"
    )
    next_score: float = dspy.OutputField(
        desc="Usefulness from 1 (least) to 5 (most) for next tool"
    )
    replan_score: float = dspy.OutputField(
        desc="Usefulness from 1 (least) to 5 (most) for replan tool"
    )

action_weights = dspy.ChainOfThought(Action)
```