PLAN*RAG: EFFICIENT TEST-TIME PLANNING FOR RETRIEVAL AUGMENTED GENERATION

Prakhar Verma[1] **Sukruta Prakash Midigeshi**[2] **Gaurav Sinha**[2] **Arno Solin**[1]
Nagarajan Natarajan[2] **Amit Sharma**[2]

[1] Aalto University [2] Microsoft Research

ABSTRACT

We introduce Plan*RAG, a novel framework that enables structured multi-hop reasoning in retrieval-augmented generation (RAG) through test-time reasoning plan generation. While existing approaches such as ReAct maintain reasoning chains within the language model's context window, we observe that this often leads to plan fragmentation and execution failures. Our key insight is that by isolating the reasoning plan as a directed acyclic graph (DAG) outside the LM's working memory, we can enable *(1)* systematic *exploration* of reasoning paths, *(2) atomic* subqueries enabling precise retrievals and grounding, and *(3) efficiency* through parallel execution and bounded context window utilization. Moreover, Plan*RAG's modular design allows it to be integrated with existing RAG methods, thus providing a practical solution to improve current RAG systems. On standard multi-hop reasoning benchmarks, Plan*RAG consistently achieves improvements over recently proposed methods such as RQ-RAG and Self-RAG, while maintaining comparable computational costs.

1 INTRODUCTION

Retrieval-Augmented Generation (RAG, Lewis et al., 2020; Petroni et al., 2020) has emerged as a promising approach for grounding language model (LM) responses in external knowledge. However, RAG systems struggle with multi-hop queries that require reasoning across multiple retrieved documents (Tang & Yang, 2024; Wei et al., 2022). A key challenge lies in the initial retrieval step, which often fails to retrieve sufficient relevant documents due to the query's lack of full contextual information (Ma et al., 2023). This limitation has been highlighted in recent surveys (Torfi et al., 2020; Zhao et al., 2023) as a fundamental barrier to reliable AI systems, particularly given the widespread deployment of large language models (Brown et al., 2020) across critical domains.

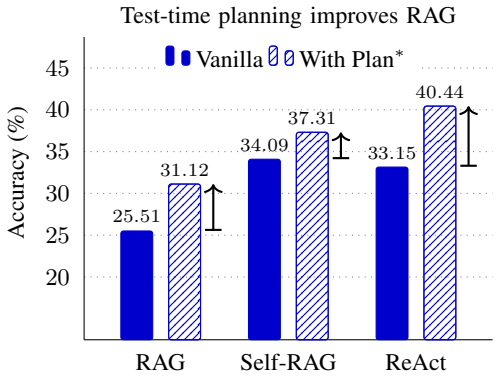

Figure 1: **Plan*RAG** improves performance on the HotpotQA benchmark substantially compared to various existing RAG methods, demonstrating the value of externalizing planning as a DAG

Consider the query: *"Rumble Fish was a novel by the author of the coming-of-age novel published in what year by Viking Press?"* Answering this requires an iterative retrieval process: identifying the Rumble Fish's author, connecting to their coming-of-age novel, and determining its publication year. Single-step retrieval in RAG systems often fails in such cases, as it may retrieve documents about Rumble Fish's author and Viking Press without recognizing the intermediate fact—the author's coming-of-age novel—must first be established. Furthermore, Leng et al. (2024); Shuster et al. (2021) demonstrate that even when relevant documents are retrieved, LMs struggle to reason across them due to fixed context windows, leading to information loss and broken reasoning chain. These limitations pose risks in critical domains such as healthcare and finance (Pal et al., 2023; Zhao et al., 2024), where accurate multi-step reasoning is essential.

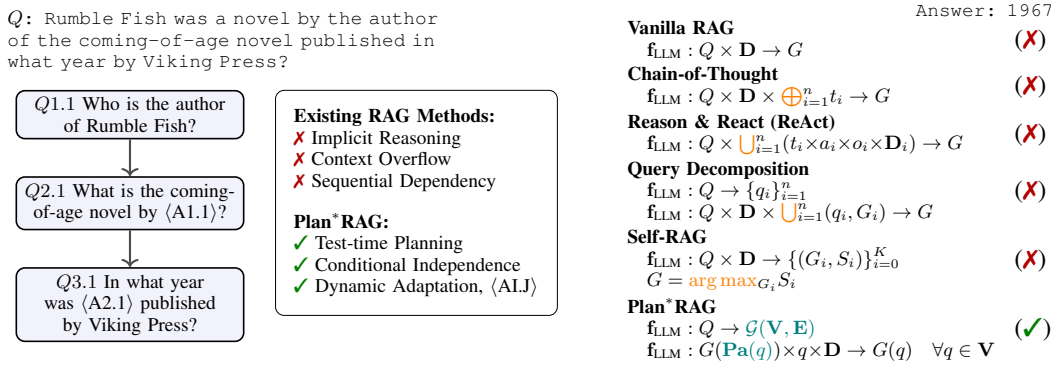

Figure 2: **Comparison of RAG approaches** on a HotpotQA multi-hop query. Traditional RAG methods struggle with context management and implicit reasoning, where sequential operators create implicit dependencies. In contrast, Plan*RAG generates an explicit reasoning plan ($\mathcal{G}$) at *test-time*, with special tags ⟨AI,J⟩ enabling dynamic information flow through parent subqueries ($\mathbf{Pa}(q)$).

Recent research has attempted to address these limitations through structured reasoning frameworks. Chain-of-Thought (CoT) prompting (Wei et al., 2022) and systematic query decomposition (Patel et al., 2022) have introduced explicit reasoning steps, enabling more granular thought processes and targeted retrievals. Building upon these foundations, Yao et al. (2023) proposed ReAct—a framework that creates a reasoning chain inside the LM's context and dynamically coordinates between thoughts, retrieval operations (actions), and information processing (observations). However, a fundamental limitation of ReAct is that the reasoning and retrieval plan exists entirely within the LLM's context window, leading to context overflow as reasoning chains grow, plan fragmentation, and high latency from sequential execution. While recently proposed frameworks like Self-RAG (Asai et al., 2023), RA-ISF (Liu et al., 2024), and RQ-RAG (Chan et al., 2024) have advanced the field through innovations in dynamic reasoning and adaptive retrieval mechanisms, the core problem still remains. These methods often struggle to balance reasoning depth and computational efficiency (see Sec. 3 for an example). This trade-off results in either shallow analysis that misses critical reasoning steps, or impractical latency that limits their deployment in real-time applications.

To address these challenges, we propose Plan*RAG, a novel approach that externalizes the reasoning process as a directed acyclic graph (DAG). Unlike existing methods that maintain reasoning chains within the LM's context or generate isolated sequential sub-queries, Plan*RAG decomposes the main query into interrelated *atomic* and *dynamic* sub-queries, where nodes represent atomic reasoning steps and edges capture the conditional dependencies between them. Atomic sub-queries can be answered by a single retrieval, and further decomposition does not provide additional useful granularity. Additionally, sub-queries are tagged with special tags ⟨AI.J⟩, allowing them to adapt based on updated knowledge during generation. This external representation addresses core limitations of existing approaches in several ways. First, by following the DAG structure for generation, only the parent nodes need to be included in the context window for any given step (and that the system follows the intended plan). This prevents context overflow and ensures that only relevant information is processed at each stage. Second, nodes at the same depth in the DAG can be processed in parallel, significantly reducing latency compared to sequential approaches. Third, nodes are dynamic, containing tags that allow them to adapt based on their parent's generation, resulting in more targeted sub-queries. Furthermore, the DAG structure enables explicit verification at each node, facilitating error correction and backtracking. As generation involves traversing through the reasoning DAG, Plan*RAG can be incorporated with both traditional RAG frameworks and recent approaches like Self-RAG (Asai et al., 2023), enhancing their multi-hop reasoning capabilities (Fig. 1) while maintaining comparable computational efficiency in terms of input-output tokens (*cf.* Fig. 4).

Contributions We summarize our contributions as follows. *(i)* We introduce a *test-time* reasoning approach that externalizes the reasoning process as a DAG, fundamentally altering how multi-hop queries are processed in RAG systems. *(ii)* We propose a novel DAG-based reasoning structure that decomposes complex queries into *atomic* and *dynamic* sub-queries, enabling parallel processing while maintaining conditional dependencies. *(iii)* We demonstrate that our approach can enhance both traditional RAG frameworks and modern approaches like Self-RAG, improving multi-hop reasoning capabilities with comparable computation.

2 RELATED WORK

We review two key areas relevant to our work: *(1)* reasoning in LLMs, which focuses on methods that enhance multi-step reasoning, and *(2)* retrieval-augmented generation (RAG), which integrates external information to improve LLM performance in knowledge-intensive tasks.

Reasoning in LLMs Recent advancements in LLMs have significantly enhanced their reasoning capabilities through various approaches. Chain-of-Thought (CoT) prompting (Wei et al., 2022) improves model performance by guiding LLMs through intermediate reasoning steps. Building on this, the Tree of Thoughts (ToT) framework (Yao et al., 2024) enables LLMs to explore and evaluate multiple reasoning paths simultaneously. Wang et al. (2023) proposed self-consistency, a technique that samples multiple reasoning chains and selects the most likely answer through majority voting. Similarly, least-to-most prompting (Zhou et al., 2023) decomposes complex questions into simpler subquestions, addressing them sequentially. Recently, Hao et al. (2023) propose Reasoning via Planning (RAP), where LLMs act as both world models and reasoning agents. Additionally, Sun et al. (2024) introduced Think-on-Graph (ToG), incorporating knowledge graphs into multi-hop reasoning for deeper and more interpretable reasoning processes. Recent works from Welleck et al. (2024); Chen et al. (2024) also analyzed tree search for reasoning. While these methods have demonstrated impressive results, they still rely on implicit LM reasoning capability or on sequential reasoning steps, which can lead to increased context usage and computational costs. In contrast, Plan*RAG externalizes the reasoning process as a DAG, mitigating these inefficiencies.

Retrieval Augmented Generation RAG enhances LLMs by integrating relevant external documents, leading to notable performance improvements, particularly in knowledge-intensive tasks (Lewis et al., 2020; Guu et al., 2020). Retrieval strategies in RAG models can be categorized into three paradigms based on the frequency of retrievals: *(1)* one-time retrieval, *(2)* retrieval every k tokens, and *(3)* adaptive retrieval. Models employing one-time retrieval include DrQA (Chen et al., 2017), REALM (Guu et al., 2020), and ATLAS (Izacard et al., 2023). Retrieval at fixed intervals (every k tokens) is used by RALM (Ram et al., 2023), RETRO (Borgeaud et al., 2022), and InstructRetro (Wang et al., 2024a). In contrast, adaptive retrieval approaches—such as Self-RAG (Asai et al., 2023), SPALM (Yogatama et al., 2021), Adaptive kNN (Drozdov et al., 2022), and Active-Retriever (Jiang et al., 2023)—dynamically adjust the frequency and nature of document retrieval based on task requirements and input context. FLARE (Jiang et al., 2023) uses token probability distributions to trigger retrievals, while RETRO (Borgeaud et al., 2022) employs a specialized architecture for fixed-interval document retrieval. Wang et al. (2024b) proposed RAFT, combining chain-of-thought reasoning with RAG through iterative thought refinement. Other approaches include RA-ISF (Liu et al., 2024), which combines dataset-specific specialized models, and RQ-RAG (Chan et al., 2024), which enables query rewriting and decomposition for handling complex queries. Hsu et al. (2024) also focus on multi-hop query performance of RAG but take an orthogonal approach.

Recent work by Ranaldi et al. (2024), explores reasoning in RAGs through C-RAG which generates explanations explicitly contrasting the relevance of retrieved passages to support the final answer. Plan*RAG takes a different approach by proposing a novel framework that performs reasoning at test-time outside the LM's working memory, and performs generation in a computationally efficient manner, enabling LLMs to tackle complex multi-hop queries with greater accuracy while producing a verifiable reasoning trace.

3 MOTIVATION

Multi-hop reasoning queries, requiring information from multiple documents, pose significant challenges for current Retrieval-Augmented Generation (RAG) systems. Consider an example query from HotPotQA (Yang et al., 2018): *"Rumble Fish was a novel by the author of the coming-of-age novel published in what year by Viking Press?"*. The query demands multiple reasoning steps: identifying the Rumble Fish's author, connecting to their coming-of-age novel, and determining its publication year. In this section, we will use this query as a running example *a)* to articulate the failure modes of standard as well as recently-proposed RAG approaches on such multi-hop reasoning queries (Fig. 2); and *b)* motivate the core idea of our proposed solution. For this analysis, we fix the model to be Llama-3.1-instruct$_{8B}$ and the retriever to be Contriever (Izacard et al., 2022). In the remainder of the paper, Q denotes queries, D denotes documents, and G denotes generations.

Table 1: **Reasoning DAG depth** (percentage/count) for multi-hop (HotpotQA, StrategyQA, MuSiQue) and single-hop (PopQA) data sets. For single-hop queries, the DAG primarily has depth 0, which is desirable, while multi-hop queries require deeper reasoning paths.

Dataset	Depth 0	Depth 1	Depth 2	Depth 3	Depth≥4
HotpotQA (Multi-hop)	0.5% (35)	12.8% (945)	79.5% (5884)	6.8% (507)	0.4% (34)
StrategyQA (Multi-hop)	0.9% (22)	42.9% (978)	51.2% (1175)	4.5% (103)	0.3% (6)
MuSiQue (Multi-hop)	0.0% (0)	2.11% (51)	66.4% (1604)	25.5% (617)	5.9% (145)
PopQA (Single-hop)	77.9% (1090)	0.8% (11)	18.9% (264)	2.4% (34)	0.0% (0)

Standard Baselines A straight-forward RAG approach is to issue the given query Q to a retriever, retrieve documents D, and prompt a language model (LM) with Q and D to generate an answer. For the running example, this approach produces an incorrect answer (*1975*), because *a)* the retrievals are incomplete; it fails to get the ground-truth correct document on the coming-of-age novel *The Outsiders*, which is not surprising as the query makes only an indirect reference to the novel, and *b)* in turn, LM generates an incorrect answer without any reasoning.

This motivates using better reasoning at inference time. A natural idea is to prompt the LM with Q and D, but also elicit a reasoning for the answer in a CoT manner (Wei et al., 2022).

However, for our example, RAG with CoT generates incorrect reasoning: *[S.E. Hinton wrote Rumble Fish → Rumble Fish was published in 1975 → The author of The Outsiders also published in 1975]*, leading to the incorrect answer *1975*. Here, we see the reason for failure is that the model hallucinated (third step of the chain) as it was trying to break down the complex query. Thus, the multi-hop query requires careful reasoning. Next, we turn to approaches that use more *test-time* compute.

Reason & Act (ReAct) ReAct (Yao et al., 2023) extends traditional RAG frameworks by implementing a structured interaction loop between reasoning and retrieval. The framework decomposes each iteration into three components: a *Thought* phase for planning, an *Action* phase for doing retrievals, and an *Observation* phase for incorporating retrieved information. For our example, ReAct produces the following plan (and retrievals): *(T1) Need to find the author of the coming-of-age novel, then find its Viking Press publication year* → ⟨ Retrieve ⟩ → *(T2) Found S.D. Smith, need to verify Viking Press publication* → ⟨ Retrieve ⟩ → *(T3) Located publication info for Rumble Fish*, ultimately answering *1975*. This trace reveals how ReAct, despite using more test-time compute, fails to keep track of the original query intent. ReAct approach relies entirely on the LM's in-context memory, that keeps growing with iterations and retrievals. It becomes challenging for the LM to track and update the plan periodically in light of the new retrievals. As shown in Fig. 2, while ReAct enables dynamic reasoning, it is vulnerable to error propagation, and struggles to maintain entity relationships across steps.

Query Decomposition (QD) Query decomposition methods (Patel et al., 2022) address the limitations of previous approaches by breaking queries into independent, sequential subqueries. LM first decomposes the query into subqueries, then solves them sequentially while maintaining previous subquery-answer pairs in context. For our example, it generates two subqueries: *"What is the author of the coming-of-age novel published by Viking Press?"* yielding *"S.E. Hinton"*, followed by *"In what year was the coming-of-age novel published by Viking Press?"* producing *"1975"*. Firstly, we note that the decomposition is not very helpful. Secondly, the LM answers the subqueries based on the first document retrieved for the original query, which is on *Rumble Fish* (and gets the author right for the first sub-query). Despite explicitly decomposing the query, the method still fails to maintain coherent reasoning across subqueries, and loses critical entity relationships between subsequent sub-queries. Recent approaches such as RQ-RAG (Chan et al., 2024) and RA-ISF (Liu et al., 2024) extend query decomposition, and incorporate query rewriting tricks, but inherit similar limitations.

Self-RAG Asai et al. (2023) recently proposed Self-RAG that introduces a novel self-evaluation mechanism. The LM generates multiple candidate responses and scores them based on their faithfulness to retrieved evidence as well as relevance to the original query. This approach aims to improve reasoning quality through verification. However, Self-RAG still fundamentally relies on the language model's implicit reasoning capabilities within a single context window, similar to the previously discussed approaches. When tested on our example query, it exhibits the same limitations, producing the incorrect answer *1975* despite its sophisticated verification mechanism.

The aforementioned methods incorporate different strategies and varying degrees of test-time compute to handle reasoning in RAG systems. But they all share fundamental limitations, listed below, that lead to failures even on 2–3 hop queries such as our running example. We identify two observations that motivate our proposed framework Plan*RAG:

- **First**, by accumulating intermediate reasoning steps in the context window, they create information overload as the context grows. It becomes challenging for the LM to identify and resolve the dependencies between subqueries and faithfully execute a reasoning plan.
- **Second**, state-of-the-art RAG approaches fail to leverage the inherent independence between different reasoning paths in multi-hop queries. For instance, finding an author's identity and locating a publication date can be answered independently, without overburdening the LM's context with all reasoning paths.

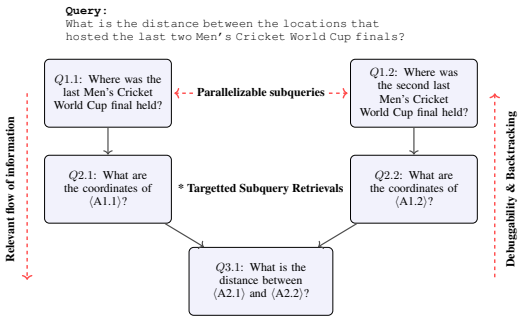

Figure 3: **Reasoning plan example:** A Reasoning DAG highlighting key advantages: only relevant information flows to each subquery, subqueries on the same depth can be executed in parallel, and the DAG structure allows for debugging and backtracking.

4 PLAN*RAG: TEST-TIME PLANNING

We present Plan*RAG, a novel framework for multi-hop reasoning through *dynamic, test-time* planning via Directed Acyclic Graphs (DAGs). The key ideas of our formulation, motivated by the issues faced by the current RAG systems discussed in Sec. 3, are:

1. We want to isolate the reasoning plan, outside of the LM's in-context memory, as a global data structure that can help with tracking, execution, and verification. We can generate the plan once statically, refine, and execute the plan using data structure operations. In this work, we use Directed Acyclic Graphs (Fig. 3) that are appropriate to represent complex plans.
2. DAG structure allows executing any node conditioned only on the necessary context along its path, unlike state-of-the-art approaches that accumulate the entire trace (plan and retrievals) in the LM's in-context memory. This also helps make the LM calls more economical in terms of the number of tokens.
3. DAG structure allows processing nodes independently, conditioned on the execution of parent nodes, helping with efficiency and end-to-end latency of RAG system.

At a high level, Plan*RAG operates in two phases: *(1)* decomposing complex queries into a DAG structure, and *(2)* executing the DAG while preserving dependencies. Besides the merits of serializing the query plan as a data structure—such as control and efficiency listed above, Plan*RAG is complementary to state-of-the-art approaches like Self-RAG in the following sense. We can seamlessly integrate Plan*RAG with methods like Self-RAG, to solve the sub-queries in its internal nodes. We formalize the reasoning plan DAG in Sec. 4.1, followed by detailed analysis on properties and benefits of Plan*RAG in Sec. 4.3. The full algorithm is showcased in Alg. 1.

4.1 REASONING PLAN: DAG

At test-time, Plan*RAG generates a reasoning plan as a Directed Acyclic Graph (DAG) $\mathcal{G}(\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ represents the set of generated subqueries and $\mathbf{E}$ denotes the directed edges between them. The root nodes of $\mathcal{G}$ correspond to independently answerable atomic subqueries, while subsequent nodes represent dependent queries that build upon their parent nodes' answers. This hierarchical structure follows the Markov assumption, ensuring that each node's answer depends only on its direct predecessors, thus enabling efficient parallel computation.

Formally, for any subquery $q \in \mathbf{V}$, its answer is computed as:

$$G(q) = \mathbf{f}_{\text{LLM}}(G(\mathbf{Pa}(q)), q, \mathbf{D}_q)\,; \quad \mathcal{G} = \mathbf{f}_{\text{LLM}}(Q)\,, \tag{1}$$

Table 2: **Evaluation with standard prompting methods:** Performance comparison of Plan*RAG against traditional RAG approaches across models.

	Method	HotpotQA	StrategyQA	MuSiQue	PopQA
GPT-3.5$_{\text{turbo}}$	Vanilla-LLM	30.45	55.37	6.53	28.45
	Vanilla-RAG	29.83	54.24	4.27	32.13
	CoT-RAG	29.96	60.66	6.50	30.02
	QD-RAG	29.31	59.91	4.85	36.03
	Plan*RAG	**31.72**	**62.07**	**6.71**	**38.62**
Llama2-chat$_{13B}$	Vanilla-LLM	19.33	44.47	2.81	22.30
	Vanilla-RAG	22.67	39.11	2.65	31.25
	CoT-RAG	25.93	29.26	5.02	34.17
	QD-RAG	26.97	42.79	4.73	36.88
	Plan*RAG	**27.51**	**54.57**	**5.32**	**41.45**
Llama3.1-instruct$_{8B}$	Vanilla-LLM	21.34	48.43	4.18	23.52
	Vanilla-RAG	25.51	47.36	4.39	39.1
	CoT-RAG	30.10	59.51	6.00	37.49
	QD-RAG	28.68	62.38	2.65	41.13
	Plan*RAG	**31.12**	**68.16**	**6.28**	**41.51**

Algorithm 1 Plan*RAG framework

Input: Q: Query
Output: Generation G
Get a reasoning plan: $\mathcal{G} \leftarrow \mathbf{f}_{\text{LLM}}(Q)$
Identify root nodes of $\mathcal{G}$: $\mathbf{q}_{root}$
Calculate depth of each node from root:
$l_q \leftarrow maxdist(q, \mathbf{q}_{root})$
for i: 0 to $\max_q(l_q)$ **do**
  for parallel: q in $\{q : l_q = i\}$ **do**
    Get parent questions and answers:
    $M_q \leftarrow \mathbf{Pa}(q)$ & $M_a \leftarrow G_{M_q}$
    $\tilde{q} \leftarrow \mathbf{f}_{\text{LLM}}(q, M_q, M_a)$ {Generate the subquery}
    $G_q \leftarrow \mathbf{f}_{\text{LLM}}(\tilde{q})$ {Obtain generated answer for query q}
  end for
end for
$G \leftarrow \Omega(\mathbf{q}, G_1, G_2, \ldots, G_q)$ {Dataset Dependent Operator}
return G

where $\mathbf{Pa}(q)$ denotes the set of parent nodes of q in the DAG, $\mathbf{D}_q$ represents the retrieved documents relevant to q, and Q is the main query. The language model $\mathbf{f}_{\text{LLM}}$ generates responses by jointly considering the subquery, its parent nodes' answers, and retrieved documents. When applied recursively, this formulation enables systematic reasoning from root nodes to leaves, with document retrieval potentially interleaved at each step.

To facilitate dynamic reasoning, we introduce a simple node indexing scheme. Each node is uniquely identified as $\langle i.j \rangle$, where i denotes the node's depth from the root and j indicates its position among nodes at depth i. Furthermore, we introduce a special tag $\langle$**AI.J**$\rangle$ that enables dynamic dependency tracking between subqueries. In this notation, **I** and **J** are integer values representing the Question IDs required to complete a subquery. For instance, as illustrated in Fig. 3, when subquery *Q2.1* depends on the answer to *Q1.1*, the tag $\langle$A1.1$\rangle$ enables dynamic answer propagation at inference time, allowing the system to adapt to updated knowledge.

4.2 Generating a Reasoning Plan as a DAG

The first step during inference involves generating a reasoning plan by prompting the LLM with a specialized prompt, as detailed in App. D.1. While this initial plan is static, it can be dynamically instantiated, and refined, given the indexing scheme $\langle$AI.J$\rangle$ as discussed in Sec. 4.1. The plan therefore is a template initially which during generation dynamically materializes as the LM traverses through the DAG.

To ensure reliable plan generation, we explore two approaches: *(1)* fine-tuning a language model specifically for this structured output, and *(2)* leveraging more capable language models that can follow complex prompting instructions. Our experiments show both approaches are effective, with a fine-tuned Llama3.1-instruct$_{8B}$ achieving comparable performance to GPT-4o (Sec. 5.4).

4.3 Reasoning Plan: Properties & Benefits

The effectiveness of Plan*RAG stems from its structured reasoning process, which offers several key advantages over traditional RAG-based methods. By leveraging a Directed Acyclic Graph (DAG) to represent reasoning plans, Plan*RAG ensures efficient information flow, reduces computational overhead, and improves query formulation. The DAG structure naturally enables systematic verification and granular control at the subquery level—errors can be isolated to specific nodes and automated interventions (like additional retrievals or alternative decompositions) can be targeted at specific reasoning steps. While we leave the full exploration of these verification and intervention capabilities to future work, this structural advantage distinguishes our approach from sequential reasoning methods. Below, we analyze the fundamental properties that enhance multi-hop reasoning while maintaining computational efficiency.

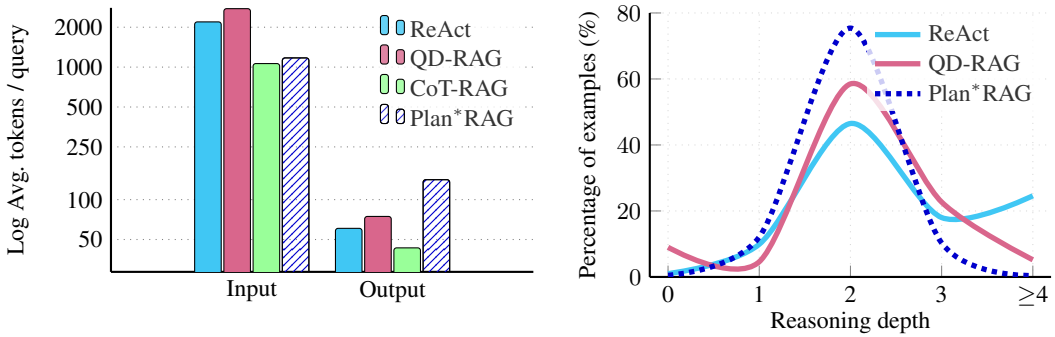

Figure 4: **Efficiency analysis:** (a) Token utilization comparison showing that Plan*RAG maintains comparable computational efficiency with baseline methods. (b) Analysis of reasoning depth on HotpotQA demonstrates that Plan*RAG naturally adapts to the dataset's 2-hop nature, achieving optimal depth for 80% of queries while other methods show inconsistent reasoning depths.

Atomic Subqueries Plan*RAG decomposes complex queries into *atomic* subqueries—unlike existing query decomposition methods that perform sequential, local decompositions. An atomic refers to subqueries that request a single piece of information and thus can be answered by a single retrieval—further decomposition does not yield additional useful granularity. This atomic nature enables precise retrievals (Table 4) and reduces hallucination risk by constraining the scope of each reasoning step. For instance, consider the query in Fig. 2, instead of broad queries like *"In what year was the coming-of-age novel published by Viking Press?"* (like QD method), Plan*RAG generates specific atomic queries like *"In what year was The Outsiders published by Viking Press?"* for *Q3.1*, where the intermediate generation for $\langle$A2.1$\rangle$ is dynamically replaced with the corresponding generation.

Parallelization & Efficiency The DAG structure explicitly captures conditional independence among subqueries through its edge relationships. This independence enables parallel execution of reasoning paths, significantly reducing latency compared to sequential approaches like QD or ReAct. By controlling information flow through explicit parent-child relationships, Plan*RAG minimizes token overhead and improves computational efficiency (Fig. 4). Formally, for sequential methods,

$$T^{\text{seq}} = \sum_{i=1}^{n} (t_{\text{gen}}(q_i) + t_{\text{ret}}(\mathbf{D}_i)) \,, \tag{2}$$

where $t_{\text{gen}}(q_i)$ is the generation time for subquery q_i and $t_{\text{ret}}(\mathbf{D}_i)$ is the retrieval time for documents $\mathbf{D}_i$. In contrast, Plan*RAG enables parallel execution of independent nodes at the same depth, reducing the total computation time to:

$$T^{\text{Plan*RAG}} = \sum_{d=0}^{D} \max_{q \in \mathbf{V}_d} (t_{\text{gen}}(q) + t_{\text{ret}}(\mathbf{D}_q)) \,, \tag{3}$$

where $\mathbf{V}_d$ represents the set of nodes at depth d and D is the maximum depth. Since, $T^{\text{Plan*RAG}} \leq T^{\text{seq}}$, the efficiency gain increases with the number of parallel branches. This parallelization significantly reduces latency compared to sequential approaches like query decomposition or ReAct.

Context Window Utilization Plan*RAG achieves optimal context window utilization through selective information propagation. For sequential reasoning methods like QD or ReAct, the context size grows linearly,

$$C_t^{\text{seq}} = |Q| + \sum_{i=1}^{t} (|q_i| + |G(q_i)| + |\mathbf{D}|) \,, \tag{4}$$

where $|Q|$ is the main query length, q_i are intermediate questions, $G(q_i)$ their generations, and $\mathbf{D}_i$ the retrieved documents. In contrast, Plan*RAG maintains a constant context size at each node q,

$$C_q^{\text{ours}} = |q| + |\mathbf{D}_q| + \sum_{p \in \mathbf{Pa}(q)} (|p| + |G(p)|) \,. \tag{5}$$

This selective propagation of only parents information, rather than entire reasoning history, enables efficient scaling to multi-hop queries while maintaining reasoning quality.

5 Experiments

We conduct extensive experiments to evaluate Plan*RAG across diverse reasoning scenarios, demonstrating significant improvements across multiple datasets.

Datasets We evaluate Plan*RAG on four datasets that test different aspects of reasoning capabilities: HotpotQA (Yang et al., 2018) and StrategyQA (Geva et al., 2021) for 2-hop reasoning, MuSiQue (Trivedi et al., 2022) for complex multi-hop reasoning (>2 hops), and PopQA (Mallen et al., 2022) for single-hop queries. This diverse selection enables comprehensive evaluation of Plan*RAG's adaptability to varying reasoning depths. We consider an answer correct if the predicted answer contains the ground truth, providing a relaxed version of exact-match to account for variations in answer phrasing. Dataset details are provided in App. A.

Retriever For all experiments, we use the Contriever-MS MARCO (Izacard et al., 2022) retriever with embeddings based on the 2018 English Wikipedia. The Wikipedia articles are segmented into non-overlapping 100-word chunks, and we retrieve the top-5 documents for each query.

Baselines and Methods We evaluate Plan*RAG against several baselines as discussed in Sec. 3: LM with no retrieval (Vanilla-LLM), retrieval with direct prompting (Vanilla-RAG), CoT prompting with retrievals (CoT-RAG), QD with retrievals (QD-RAG), and state-of-the-art RAG methods, like Self-RAG, ReAct, and RQ-RAG. These methods have outperformed SAIL (Luo et al., 2023), Toolformer (Schick et al., 2024), and commercial systems like Perplexity.ai and ChatGPT. We conduct experiments using three base models: GPT-3.5$_{\text{turbo}}$, Llama2-chat$_{\text{13B}}$, and Llama3.1-instruct$_{\text{8B}}$.

We implement two variants of Plan*RAG: *(1)* a base version that uses documents retrieved from the original query; *(2)* Plan*RAG $_{\text{SubQ}}$ that performs iterative retrievals for each subquery in the DAG, enabling precise retrievals. Given Plan*RAG's agnostic design discussed in Sec. 4, we also evaluate its integration with Self-RAG (Plan*-Self-RAG).

5.1 Accuracy Performance

We evaluate Plan*RAG against both standard prompting techniques and state-of-the-art RAG frameworks in Table 2 and Table 3. For fair comparison with standard prompting methods, all approaches use the same set of top-5 retrievals via Contriever from the original query. As shown in Table 2, we compare against vanilla LLMs and three RAG variants (standard retrieval, chain-of-thought prompting, and query decomposition) using GPT-3.5$_{\text{turbo}}$, Llama2-chat$_{\text{13B}}$, and Llama3.1-instruct$_{\text{8B}}$. Plan*RAG demonstrates consistent performance gains across all datasets and baseline models. For state-of-the-art methods that employ iterative retrieval, we evaluate Plan*RAG $_{\text{SubQ}}$. As shown in Table 3, Plan*RAG $_{\text{SubQ}}$ achieves significant improvements over recent frameworks like Self-RAG, ReAct, and RQ-RAG across respective base models. Additionally, integrating Self-RAG to Plan*RAG's reasoning DAG (Plan*-Self-RAG) yields substantial performance gains, validating the effectiveness of the *test-time, external* planning approach.

These results establish that Plan*RAG's *test-time* planning provides clear advantages over existing reasoning frameworks, including those designed for multi-hop reasoning.

5.2 Reasoning DAG Depth

Table 1 shows the reasoning DAG depths for all the datasets. As expected, the multi-hop query datasets exhibit a DAG depth greater than 1, indicating multiple atomic queries must be answered at different depths to answer the main query. In contrast, the single-hop dataset typically shows a reasoning depth of 0, which is both expected and desired, as simpler queries do not require further decomposition into subqueries. This showcases that the reasoning DAG effectively adapts its complexity based on the query complexity. On HotpotQA (a 2-hop dataset), we compare depth distributions for sequential methods like QD-RAG and ReAct in Fig. 4. Both the methods exceed the ideal reasoning depth, with a significant proportion of samples having depths greater than 3, indicating suboptimal reasoning paths.

Information Gain (IG) We introduce *cumulative information gain (IG)* to measure information aggregation across reasoning depths. Formally, at depth d, $IG(Q, \mathcal{G}, d) = IG(Q, \{q_i\}_{i=0}^{d})$, where Q represents the main query and $\{q_i\}_{i=0}^{d}$ denotes the set of subqueries up to depth d. A higher IG indicates substantial new information contribution toward the final answer, while a similar value as the previous depth suggest redundant or unnecessary reasoning steps. We experiment with the HotpotQA dataset and employ GPT-3.5$_{\text{turbo}}$ model to score the information gain between 1-10 across the reasoning plan. Table 6 showcases the average IG score over reasoning depth, demonstrating consistent incremental information gain as the DAG is traversed.

Table 3: **Comparison with state-of-the-art methods:** Performance of Plan*RAG against advanced RAG methods (RQ-RAG, Self-RAG) and their Plan*-augmented variants. Plan*RAG $_{\text{SubQ}}$ achieves consistent improvements across both single-hop and multi-hop tasks.

Base	Method	HotpotQA	StrategyQA	MuSiQue	PopQA
Llama2$_{7B}$	RQ-RAG	33.78	47.46	13.43	32.66
	Self-RAG	31.48	43.71	7.07	43.67
	Plan*-Self-RAG	**36.65**	**60.67**	**14.76**	**44.06**
Llama2$_{13B}$	Self-RAG	34.09	42.75	8.31	45.18
	Plan*-Self-RAG	**37.31**	**62.87**	**16.72**	**45.23**
Llama3.1 instruct$_{8B}$	ReAct	33.15	54.67	13.36	35.81
	Plan*RAG $_{\text{SubQ}}$	**40.44**	**65.45**	**14.88**	**41.98**

Table 4: **Retrieval effectiveness:** Evaluation of retrieval quality on HotpotQA distractor setting. Plan*RAG achieves higher precision and recall.

Method	Precision	Recall	Accuracy
ReAct	0.04	34.61	25.68
QD-RAG$_{\text{SubQ}}$	0.02	32.16	23.54
Plan*RAG$_{\text{SubQ}}$	**27.43**	**36.57**	**31.27**

Table 5: **Reasoning planners:** Accuracy comparison showing a *fine-tuned* Llama3.1-instruct$_{8B}$ planner achieves comparable performance to GPT-4o.

Vanilla-RAG	CoT-RAG	QD-RAG	Plan$_{4o}$	Plan$_{8B}$
24.83	30.11	28.25	31.97	31.28

5.3 Targeted Subqueries and Retrievals

A key feature of Plan*RAG's reasoning plan is the *atomic, dynamic* subqueries. As discussed in Sec. 4.3, the subqueries in the reasoning plan are more targetted and help improve retrievals. We evaluate retrieval effectiveness with the HopotQA Distractor setting, which provides ground truth labels for supporting documents. As shown in Table 4, Plan*RAG significantly outperforms both ReAct and QD-RAG$_{\text{SubQ}}$ in both precision and recall, contributing to improved accuracy. High precision is particularly crucial for multi-hop reasoning, as it minimizes the introduction of irrelevant information that could propagate errors through the reasoning chain.

5.4 Reasoning Plan Generation

So far in the experiments we have utilized the more capable GPT-4o model for generating reasoning plans. However, to demonstrate that Plan*RAG framework is not dependent on powerful LLMs, we finetune a Llama3.1-instruct$_{8B}$ reasoning planner with LoRA adapters on a small subset of HotpotQA-train dataset. We use input-output pairs of questions and their corresponding reasoning DAGs in the required JSON format as training data. Experiments on a HotpotQA-dev subset (1300 samples) demonstrate comparable performance to GPT-4o, achieving [31.28] accuracy versus [31.97] with GPT-4o (Table 5). These results indicate that the success of our *test-time* planning approach does not hinge on specific LLMs; and that modest training data is sufficient to fine-tune LMs to generate structured output.

6 Discussion and Conclusion

In this paper, we presented Plan*RAG, a framework that transforms multi-hop reasoning in RAG through *test-time* planning. Our key insight—externalizing the reasoning structure as a DAG outside the LM's context—addresses fundamental limitations of existing approaches. Empirical results validate three core benefits: systematic exploration of reasoning paths, efficient parallel execution, and improved performance on multi-hop datasets. Significant improvements in retrieval precision demonstrates that atomic subqueries enable more focused document retrieval, while bounded context windows overcome the overflow challenges faced by sequential approaches. A key strength of Plan*RAG is its modular design, demonstrated by successful integration with existing RAG methods like Self-RAG. This flexibility, combined with comparable computational costs, establishes Plan*RAG as a practical solution for real-world multi-hop reasoning tasks

A key limitation in Plan*RAG is reliance on a static plan generated a priori. While the plan may need updates as more information is collected, the DAG structure facilitates subquery updates and backtracking since it is a formal object outside the LLM's context window. Future work can explore: *(1)* verification and backtracking using the reasoning DAG, *(2)* feedback loops between reasoning steps for dynamic plan refinement, and *(3)* extending the framework to tasks like fact verification and mathematical deduction.

ACKNOWLEDGEMENTS

This work was primarily conducted while PV was an intern at Microsoft Research. PV and AS acknowledge funding from the Research Council of Finland (grants 362408, 339730). We acknowledge CSC – IT Center for Science, Finland, and the Aalto Science-IT project for providing computational resources. Finally, we appreciate the valuable suggestions and feedback provided by the anonymous reviewers.

REFERENCES

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *The Twelfth International Conference on Learning Representations*, 2023.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego De Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, Tom Hennigan, Saffron Huang, Loren Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack Rae, Erich Elsen, and Laurent Sifre. Improving language models by retrieving from trillions of tokens. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 2206–2240. PMLR, 2022.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. RQ-RAG: Learning to refine queries for retrieval augmented generation. In *First Conference on Language Modeling*, 2024.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1870–1879. Association for Computational Linguistics, 2017.

Ziru Chen, Michael White, Raymond Mooney, Ali Payani, Yu Su, and Huan Sun. When is tree search useful for llm planning? it depends on the discriminator. *arXiv preprint arXiv:2402.10890*, 2024.

Andrew Drozdov, Shufan Wang, Razieh Rahimi, Andrew Mccallum, Hamed Zamani, and Mohit Iyyer. You can't pick your neighbors, or can you? When and how to rely on retrieval in the kNN-LM. In *Findings of the Association for Computational Linguistics (EMNLP)*, pp. 2997–3007, 2022.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. Did Aristotle use a laptop? A question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, 2021.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *Proceedings of the International Conference on Machine Learning*, pp. 3929–3938. PMLR, 2020.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Sheryl Hsu, Omar Khattab, Chelsea Finn, and Archit Sharma. Grounding by trying: LLMs with reinforcement learning-enhanced retrieval. *arXiv preprint arxiv:2410.23214*, 2024.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. Unsupervised dense information retrieval with contrastive learning. *Transactions on Machine Learning Research*, 2022.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Atlas: Few-shot learning with retrieval augmented language models. *Journal of Machine Learning Research*, 24(251): 1–43, 2023.

Zhengbao Jiang, Frank F Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7969–7992, 2023.

Quinn Leng, Jacob Portes, Sam Havens, Matei Zaharia, and Michael Carbin. Long context rag performance of large language models. *arXiv preprint arXiv:2411.03538*, 2024.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 9459–9474. Curran Associates, Inc., 2020.

Yanming Liu, Xinyue Peng, Xuhong Zhang, Weihao Liu, Jianwei Yin, Jiannan Cao, and Tianyu Du. Ra-isf: Learning to answer and understand from retrieval augmentation via iterative self-feedback. *arXiv preprint arXiv:2403.06840*, 2024.

Hongyin Luo, Tianhua Zhang, Yung-Sung Chuang, Yuan Gong, Yoon Kim, Xixin Wu, Helen M. Meng, and James R. Glass. Search augmented instruction learning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5303–5315, 2023.

Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Hannaneh Hajishirzi, and Daniel Khashabi. When not to trust language models: Investigating effectiveness and limitations of parametric and non-parametric memories. *arXiv preprint arXiv:2212.10511*, 2022.

T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal. Can a suit of armor conduct electricity? A new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018.

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. Med-halt: Medical domain hallucination test for large language models. In *Proceedings of the 27th Conference on Computational Natural Language Learning (CoNLL)*, pp. 314–334, 2023.

Pruthvi Patel, Swaroop Mishra, Mihir Parmar, and Chitta Baral. Is a question decomposition unit all we need? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 4553–4569, 2022.

Fabio Petroni, Patrick Lewis, Aleksandra Piktus, Tim Rocktäschel, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. How context affects language models' factual predictions. In *Automated Knowledge Base Construction*, 2020.

Ori Ram, Yoav Levine, Itay Dalmedigos, Dor Muhlgay, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. In-context retrieval-augmented language models. *Transactions of the Association for Computational Linguistics*, 11:1316–1331, 2023.

Leonardo Ranaldi, Marco Valentino, and Andrè Freitas. Eliciting critical reasoning in retrieval-augmented language models via contrastive explanations. *arXiv preprint arXiv:2410.22874*, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36. Curran Associates, Inc., 2024.

Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. Retrieval augmentation reduces hallucination in conversation. *arXiv preprint arXiv:2104.07567*, 2021.

Jiashuo Sun, Chengjin Xu, Lumingyuan Tang, Saizhuo Wang, Chen Lin, Yeyun Gong, Lionel Ni, Heung-Yeung Shum, and Jian Guo. Think-on-graph: Deep and responsible reasoning of large language model on knowledge graph. In *The Twelfth International Conference on Learning Representations*, 2024.

Yixuan Tang and Yi Yang. Multihop-RAG: Benchmarking retrieval-augmented generation for multi-hop queries. In *First Conference on Language Modeling*, 2024.

Amirsina Torfi, Rouzbeh A Shirvani, Yaser Keneshloo, Nader Tavaf, and Edward A Fox. Natural language processing advancements by deep learning: A survey. *arXiv preprint arXiv:2003.01200*, 2020.

H. Trivedi, N. Balasubramanian, T. Khot, and A. Sabharwal. Musique: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022.

Boxin Wang, Wei Ping, Lawrence Mcafee, Peng Xu, Bo Li, Mohammad Shoeybi, and Bryan Catanzaro. InstructRetro: Instruction tuning post retrieval-augmented pretraining. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 51255–51272. PMLR, 2024a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Zihao Wang, Anji Liu, Haowei Lin, Jiaqi Li, Xiaojian Ma, and Yitao Liang. RAT: Retrieval augmented thoughts elicit context-aware reasoning in long-horizon generation. *arXiv preprint arXiv:2403.05313*, 2024b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022.

Sean Welleck, Amanda Bertsch, Matthew Finlayson, Hailey Schoelkopf, Alex Xie, Graham Neubig, Ilia Kulikov, and Zaid Harchaoui. From decoding to meta-generation: Inference-time algorithms for large language models. *arXiv preprint arXiv:2406.16838*, 2024.

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

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

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. In *Advances in Neural Information Processing Systems*, volume 36. Curran Associates, Inc., 2024.

Dani Yogatama, Cyprien de Masson d'Autume, and Lingpeng Kong. Adaptive semiparametric language models. *Transactions of the Association for Computational Linguistics*, 9:362–373, 2021.

Huaqin Zhao, Zhengliang Liu, Zihao Wu, Yiwei Li, Tianze Yang, Peng Shu, Shaochen Xu, Haixing Dai, Lin Zhao, Gengchen Mai, et al. Revolutionizing finance with llms: An overview of applications and insights. *arXiv preprint arXiv:2401.11641*, 2024.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

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

APPENDICES

The supplementary document is organized as follows: App. A presents the characteristics and specifications of datasets used in the evaluations. App. B details baseline models employing traditional prompting approaches, their implementation and specific prompts. App. C describes the state-of-the-art RAG methods evaluated in the experiments. App. D elaborates on the proposed Plan*RAG framework, encompassing reasoning plan generation, dynamic subquery generation, and answer generation through DAG traversal. Finally, App. E provides details on the experiment setup.

A DATASET DETAILS

In this section, we discuss the datasets used in the experiments. The datasets are particularly characterized into multi-hop and single-hop depending on the nature of the queries they contain.

A.1 MULTI-HOP QA

For multi-hop queries, we focus on three datasets: HotpotQA (Yang et al., 2018), StrategyQA (Geva et al., 2021), and MuSiQue-Ans (Trivedi et al., 2022)

HotpotQA HotpotQA is a multi-hop dataset collected from English Wikipedia. The questions are diverse and not constrained to any pre-existing knowledge bases or knowledge schemas. The dataset contains 7,405 queries in the *dev-fullwiki* split. Although, each question in the dataset comes with two gold paragraphs, as well as a list of sentences in these paragraphs that crowd workers identify as supporting facts necessary to answer the question, we do not use these in our experiments and use contriever to fetch the relevant documents from the Wikipedia embeddings.

StrategyQA StrategyQA is a question-answering benchmark requiring multiple reasoning steps to answer each question. Questions are short, topic-diverse, and require specific reasoning strategies for answers. We use the dataset that is available on the Self-RAG repository Asai et al. (2023). The dataset consists of 2,234 question-answer pairs. The dataset covers a broad range of topics including science, history, and common sense reasoning.

MuSiQue-Ans MuSiQue-Ans is a multi-hop question answering dataset comprising 2,417 questions spanning 2 to 4 reasoning hops. Compared to existing QA datasets, MuSiQue-Ans presents a higher level of difficulty, evidenced by a threefold increase in the gap between human and machine performance. It is specifically designed to be resilient against models that rely on disconnected or shallow reasoning patterns.

A.2 SINGLE-HOP QA

To evaluate model performance on single-hop queries, we use PopQA (Mihaylov et al., 2018).

PopQA PopQA is an open-domain question-answering dataset designed to assess a model's ability to retrieve and generate answers based on factual knowledge. The dataset consists of factual questions, many of which require specific knowledge of popular culture, history, and general world facts. In total the size of the dataset is 1,399 question-answer pairs.

B STANDARD PROMPTING METHODS

In this section we discuss various standard prompting methods experimented in Sec. 5.1 (*cf.* Table 2) along with the specific prompts. For all the models, we set temperature=0 and as all the RAG methods discussed below use the query to retrieve top-k documents, all the methods use the same set of retrieved documents.

Note that for the Llama2 model, we wrap the prompts with the required system tags: `<s>,[INST],<<SYS>>,<</SYS>>`.

Vanilla-LLM We experiment with three LLM models without any retrievals to answer the queries using its parametric knowledge: GPT-3.5$_{\text{turbo}}$, Llama2-chat$_{\text{13b}}$, and Llama3.1-instruct$_{\text{8b}}$. Below is the prompt that we use:

```
Be precise and give answer to the query. Response should be a
valid JSON, that can be passed to json.loads directly, with a key
 as Response which only has 2-3 words. Do not use complete
sentences or punctuation. In JSON, put every value as a string
always, not float.

Example:

Query: What is the capital of France?
{"Response": "Paris"}
Query: How do you make coffee?
{"Response":  "Brew ground beans"}
```

Vanilla-RAG We experiment with the standard RAG method, *i.e.* retrieving top-k relevant documents for each query and providing them as context to the LLM. For our experiments, we use Contriever (Izacard et al., 2022) as the retriever with k=5. Similar to Vanilla-LLM, we experiment with 3 models: GPT-3.5$_{\text{turbo}}$, Llama2-chat$_{\text{13b}}$, and Llama3.1-instruct$_{\text{8b}}$. Below is the prompt that we use:

```
You are a concise answering assistant. Use the Retrievals while
generating the answer and keep the answer grounded in the
retrievals. Generate a JSON with a single key "Response" and a
value that is a short phrase or a few words. In JSON, put every
value as a string always, not float.
Note: Generate only JSON and no explanation or repeating the
query or retrievals.

Example:

Query: Who was the PM of India when India performed its first
nuclear test?
Retrievals: [["Indira Gandhi was the PM of India in 1974"], ["
Indira Gandhi was the first woman PM of India."]]
Generation: {
        "Response": "Indira Gandhi"
}

Query: What is the capital of France?
Retrievals: [["Paris is the capital of France."]]
Generation: {
        "Response": "Paris"
}
```

CoT-RAG We combine chain-of-thought (CoT) prompting with RAG, using Contriever as the retriever with k=5. We experiment with the three models: GPT-3.5$_{\text{turbo}}$, Llama2-chat$_{\text{13b}}$, and Llama3.1-instruct$_{\text{8b}}$. Below is the prompt that we use:

```
You are a highly intelligent assistant skilled at solving
problems by reasoning step-by-step.  For each query, explain your
 reasoning clearly and logically in multiple steps, and then
provide the final answer (Response). Use retrievals and keep your
 generation grounded in the retrievals. Produce a list of
reasoning steps along with the Answer (Response) (precise and max
 3-4 words).
Note: Generate only reasoning steps and response. no explanation
or repeating the query or retrievals. Strictly follow the below
template and only return the JSON with no explanation.

Example:

Query: What is the sum of 15 and 27?
Retrievals: []
```

```
Generation: {
        "Reasoning_steps" : ["The first number is 15",  "The
        second number is 27", "Adding these gives 15 + 27 = 42"],
        "Response": "42"
}

Query: Who was the PM of India when India performed its first
nuclear test?
Retrievals:[["Indira Gandhi was the PM of India in 1974"], ["
Indira Gandhi was the first woman PM of India."]]
Generation:{
        "Reasoning_steps" : ["India performed its first nuclear
        test in 1974", "Indira Gandhi was the PM of India in
        1974."],
        "Response": "Indira Gandhi"
}
```

QD-RAG We experiment with query decomposition (QD) based RAG, where the original query is first broken down into simpler sub-queries. We use Contriever as the retriever with k=5 and evaluate three models: GPT-3.5$_{\text{turbo}}$, Llama2-chat$_{\text{13b}}$, and Llama3.1-instruct$_{\text{8b}}$. Below is the prompt used for query decomposition:

```
You are a helpful assistant that breaks complex queries into
simpler ones that are easy to answer. Therefore, your job is to
simplify complex queries into multiple queries that can be
answered in isolation to eachother. If the query is simple, then
keep it as it is.
NOTE: Always return a python list of subqueries that be passed to
 eval() directly.

Examples:

Query: Did Microsoft or Google make more money last year?
Subqueries: ['How much profit did Microsoft make last year?', '
How much profit did Google make last year?']

Query: What is the capital of France?
Subqueries: ['What is the capital of France?']

Query: Who has the highest score in the last two ODI cricket
world cups?
Subqueries: ['Who has the highest score in the last ODI cricket
world cup?', 'Who has the highest score in the second last ODI
cricket world cup?', 'Who scored the highest score among these
two?']
```

After decomposing the query into sub-queries, we solve each sub-query sequentially. The answers to previous sub-queries are provided as "Known answers" to help maintain consistency and build upon intermediate findings. Below is the prompt used for answering each sub-query:

```
You are a concise answering assistant. Use the Retrievals while
generating the answer and keep the answer grounded in the
retrievals. If Known answers are given, use them while generating
 the response. Generate a JSON with a single key "Response" and a
 value that is a short phrase or a few words. In JSON, put every
value as a string always, not float.
Strictly follow the format below, and provide only the "
Generation" part.

Example:

Query: Who was the PM of India when India performed its first
nuclear test?
```

```
Retrievals: [["Indira Gandhi was the PM of India in 1974"], ["
Indira Gandhi was the first woman PM of India."]]
Known answers:  Q=When did India perform the first nuclear test?
A=India conducted its first nuclear test on May 18, 1974, at the
Pokhran Test Range in Rajasthan, India.
Generation: {
        "Response": "Indira Gandhi was the PM of India when India
         performed its first nuclear test in 1974."
}

Query: What type of literature did John Keble write?
Retrievals: [["John Keble (1792-1866) was an English clergyman,
poet, and theologian, best known for his contributions to
religious poetry"], ["Keble wrote essays and sermons emphasizing
the importance of tradition, the authority of the Church, and the
 significance of the sacraments."]]
Generation: {
        "Response": "Religious and devotional poetry"
}
```

C State-of-the-Art RAG Methods

In this section we discuss various state-of-the-art RAG methods experimented in Sec. 5.1 (*cf.* Table 3).

Self-RAG Asai et al. (2023) proposed Self-RAG, wherein the base Llama2 models are trained to learn a set of special reflection tokens. The reflection tokens are then used at the time of inference to judge the requirement for retrievals, relevance of the retrieved documents and the accuracy of the output. We tested Self-RAG$_{7b}$ and Self-RAG$_{13b}$ on both single and multi-hop datasets. As a retriever, we use contriever and set k=5. The temperature for both the models was set to 0 to maintain non-stochasticity. We use the official codebase released by the authors for all the experiments.

RQ-RAG Chan et al. (2024) proposed RQ-RAG, a framework where the base Llama2 model is trained to enable it to dynamically refine search queries through rewriting, decomposing, and clarifying ambiguities. Further, control tokens are used to direct the generation process. In addition, the authors use three different sampling methods which includes selection based on perplexity (PPL), confidence, and an ensemble approach, in order to select the final answer. In our experiments, total of k=5 documents are retrieved at each depth for any given query and the maximum depth is set to 2. The ensemble answer is used in all the experiments as that gave the best results. Similar to Self-RAG, temperature was set to 0 to remove any stochasticity. We use the official codebase released by the authors for all the experiments.

ReAct ReAct (Yao et al., 2023) combines reasoning and acting with language models to solve diverse language reasoning and decision-making tasks. The model uses an interleaved sequence of Thought, Action, and Observation steps to answer questions. For the experiment, we use Contreiver as a retriever, and the actions are limited to two options: *'search'* and *'finish'*. Therefore, the prompt for the models were changed accordingly. We experimented using both Llama3.1-instruct$_{8b}$ and GPT-3.5$_{turbo}$ models, with the temperature set to 0 in both cases. We used only one retrieval (k=1) per thought as setting k=5 documents per thought gave poor results due to exploding contexts. The prompt used is as follows:

```
You are a question answering agent, you need to solve the given
question with interleaving Thought, Action, Observation steps.
You need to do this one step at a time, given some previous steps
. Thought can reason about the current situation, and Action can
be of two types:
Search[entity], which searches and returns the top 1 relevant
article.
Finish[answer], which returns the answer and finishes the task.
```

```
Action can be only of the above two forms. They need to either
include Search or Finish with the entity or answer enclosed in
the bracket respectively.

Here are some examples:

Question: Musician and satirist Allie Goertz wrote a song about
the "The Simpsons" character Milhouse, who Matt Groening named
after who?

Thought 1: I only need to search Milhouse and find who it is
named after.
Action 1: Search[Who was Milhouse named after?]
Observation 1: [Lisa kisses Milhouse. Lisa told Milhouse he
should not give up searching for other girls and that life has
unexpected things to offer. She told him he is cute in the
moonlight, which caused him to fall off a cliff nearly to his
death, but a bald eagle caught him, which left him saying "
everything is coming up Milhouse!" Milhouse was designed by Matt
Groening for a "Butterfinger" commercial, and it was decided to
use the character in the series. Milhouse was named after U.S.
President Richard Nixon, whose middle name was Milhous. The name
was the most "unfortunate name"]

Thought 2: Milhouse was named after U.S. president Richard Nixon,
 so the answer is Richard Nixon.
Action 2: Finish[Richard Nixon]
```

D PLAN*RAG

We present the prompts and details of Plan*RAG, including its reasoning plan, tag replacement, and answer generation LM. For all the models, we set temperature=0. Note that for the Llama2 model, we wrap the prompts with the required system tags: `<s> [INST] <<SYS>>` and `<</SYS>>`.

D.1 REASONING PLAN

The reasoning plans, as discussed in Sec. 4.2, are created by prompting the language model with the query and a set of contextual examples. The prompt guides the LLM to generate a minimal DAG where each sub-query's answer depends only on its parent nodes. We employ the special tag ⟨AI.J⟩ in sub-queries to explicitly denote dependencies on parent node answers. The prompt used for the DAG generation is:

```
You are a reasoning DAG generator expert. The goal is to make a
reasoning DAG with minimum nodes. Given a query, if it is complex
 and requires a reasoning plan, split it into smaller,
independent, and individual subqueries. The query and subqueries
are used to construct a rooted DAG so make sure there are NO
cycles and all nodes are connected, there is only one leaf node
with a single root and one sink. DAG incorporates Markov property
 i.e. you only need the answer of the parent to answer the
subquery. The main query should be the parent node of the initial
 set of subatomic queries such that the DAG starts with it.
Return a Python list of tuples of parent query and the subatomic
query which can be directly given to eval().

Strictly follow the below template for output.

For the subquery generation, input a tag <AI.J> where the answer
of the parent query should come to make the query complete.

NOTE: Make the DAG connected and for simple queries return the
original query only without any reasoning DAG.
```

```
Example:

Query: Who is the current PM of India?
DAG: "Q: Who is the current PM of India?"

Query:What is the tallest mountain in the world and how tall is
it?
DAG: [
("Q: What is the tallest mountain in the world and how tall is it
?", "Q1.1: What is the tallest mountain in the world?"),
("Q1.1: What is the tallest mountain in the world?", "Q2.1: How
tall is <A1.1>?")
]

Query: What percentage of the worlds population lives in urban
areas?
DAG: [
("Q: What percentage of the worlds population lives in urban
areas?", "Q1.1: What is the total world population?"),
("Q: What percentage of the worlds population lives in urban
areas?",  "Q1.2: What is the total population living in urban
areas worldwide?"),
("Q1.1: What is the total world population?", "Q2.1: Calculate
the percentage living in urban areas worldwide when total
population is <A1.1> and population living in urban areas is <A1
.2>?"),
("Q1.2: What is the total population living in urban areas
worldwide?", "Q2.1: Calculate the percentage living in urban
areas worldwide when total population is <A1.1> and population
living in urban areas is <A1.2>?")
]
```

D.2 Dynamic Subquery Generation

The interdependence of child nodes on their parents is captured through special ⟨AI.J⟩ tags. During sub-query answer generation, we prompt the LLM to replace these tags with their corresponding parent answers to create coherent sub-query question. Below is the prompt used for tag replacement:

```
You are provided a question with tags where the corresponding tag
 answers need to be replaced. Replace tags with answers of the
previous question in such a way that the final question is
coherent and logical.
Just replace all parts of the answers in the main question. Do
not reason or answer the question. Your role is just to replace
tags with all parts of the answer.

NOTE: Only output the question with no explanation or any other
details.

Example:

Query: Q2.1: Who was the president of India when the captain of
the Indian cricket team was <A1.1> and vice-captain was <A1.2> in
 2018?
Q1.1: Who was the captain of India cricket team in 2018?
A1.1: The captain of Indian cricket team in 2018 was M.S.Dhoni.
Q1.2: Who was the vice-captain of India cricket team in 2018?
A1.2: The vice-captain of Indian cricket team in 2018 was Virat
Kohli.
Output: Q2.1: Who was the president of India when the captain of
the Indian cricket team was M.S.Dhoni and vice-captain was Virat
Kohli?
```

D.3 GENERATOR LLM

Plan*RAG involves traversing the reasoning plan to generate answers for each sub-query and ultimately resolve the main query. The following prompt is used for each answer generation:

```
You are a concise answering assistant. If relevant and provided,
use the Retrievals while generating the answer or use your own
knowledge. If Known answers are given, use them while generating
the response. Generate a JSON with a single key "Response" and a
value that is a short phrase or a few words. In JSON, put every
value as a string always, not float. Strictly follow the format
below, and provide only the "Generation" part.

Example:

Query: Who was the PM of India when India performed its first
nuclear test?
Retrievals: [["Indira Gandhi was the PM of India in 1974"], ["
Indira Gandhi was the first woman PM of India."]]
Known answers: Q=When did India perform the first nuclear test? A
=India conducted its first nuclear test on May 18, 1974, at the
Pokhran Test Range in Rajasthan, India.
Generation: {
        "Response": "Indira Gandhi was the PM of India when India
         performed its first nuclear test in 1974."
}

Query: What type of literature did John Keble write?
Retrievals: [["John Keble (1792-1866) was an English clergyman,
poet, and theologian, best known for his contributions to
religious poetry"], ["Keble wrote essays and sermons emphasizing
the importance of tradition, the authority of the Church, and the
 significance of the sacraments."]]
Generation: {
        "Response": "Religious and devotional poetry"
}
```

E EXPERIMENTS

This section presents comprehensive details of our experimental evaluations. We begin with an analysis of a HotpotQA example query (Fig. 2), demonstrating comparative method outputs in App. E.1. We then evaluate performance on multi-hop and single-hop datasets in App. E.2, presenting results in Table 2 and Table 3. Following this, we analyze Plan*RAG's retrieval effectiveness in App. E.3 and information gain results in App. E.4. Finally, we describe the reasoning plan generation experiments using a finetuned Llama3.1-instruct$_{8b}$ model in App. E.5.

E.1 HOTPOTQA EXAMPLE QUERY

We analyze a query from HotpotQA to demonstrate how different methods approach multi-hop reasoning:

Query: "Rumble Fish was a novel by the author of the coming-of-age novel published in what year by Viking Press?"

Ground Truth: 1967

As discussed in Sec. 3, we compare various methods using Llama3.1-instruct$_{8b}$ with temperature=0 and Contriever as the retriever (k=5). Below are the outputs for each method:

Vanilla-LLM generates *S.E. Hinton 1975*.

Vanilla-RAG generates *1975*.

Chain-of-Thought (CoT) - RAG produces the following reasoning chain:
S.E. Hinton wrote Rumble Fish → Rumble Fish was published in 1975 → The author of The Outsiders also published in 1975
Final generation: *1975*

ReAct generates the following sequence:
Thought 1: I need to find the author of the coming-of-age novel, then find the year it was published by Viking Press.
Action 1: Search[Author of the coming-of-age novel published by Viking Press]
Thought 2: The author of the coming-of-age novel is S.D. Smith, but I am not sure if he published it with Viking Press.
Action 2: Search[Coming-of-age novel by S.D. Smith published by Viking Press]
Thought 3: I found that S.E. Hinton published Rumble Fish, and it was published by Viking Press in 1975.
Action 3: Finish[1975]
Final generation: *1975*

Query Decomposition (QD) - RAG decomposes the query as follows, we show both the query and the generation: *Q1:* What is the author of the coming-of-age novel published by Viking Press?
A1: S.E. Hinton
Q2: In what year was the coming-of-age novel published by Viking Press?
A2: 1975
Q3: What is the title of the novel written by the author of the coming-of-age novel?
A3: The Outsiders
Final generation: *1975*

Self-RAG using the fine-tuned $\text{Llama2}_{\text{13b}}$ model generates *1975*.

Plan*RAG generates the following reasoning plan:
Q1.1: Who is the author of Rumble Fish? → *Q2.1:* What is the coming-of-age novel by ⟨A1.1⟩? → *Q3.1:* In what year was ⟨A2.1⟩ published by Viking Press?

During the generation the subquery and generation are as follows:
Q1.1: Who is the author of Rumble Fish?
A1.1: S. E. Hinton
Q2.1: What is the coming-of-age novel by S. E. Hinton?
A2.1: The Outsiders
Q3.1: In what year was The Outsiders published by Viking Press?
A3.1: 1967
Final generation: *1967*

Therefore, Plan*RAG is the only method that arrives at the correct answer of *1967*.

E.2 Performance Experiment

Table 2 and Table 3 demonstrate the performance limitations of traditional prompting RAG methods and recently proposed advanced RAG approaches, while highlighting the superior performance of the proposed method (Plan*RAG).

For Table 2, all methods use same set of retrievals obtained via Contriever with k=5 on the main query. The prompts employed by each model are detailed in App. B. To eliminate stochasticity, we set temperature=0 across all models. For Table 3, we evaluate against recently proposed advanced RAG methods: Self-RAG (Asai et al., 2023), RQ-RAG (Chan et al., 2024), and ReAct (Yao et al., 2023). We utilize the official implementations for Self-RAG and RQ-RAG, while adapting ReAct's prompt for the RAG framework (detailed in App. C).

For comparison with state-of-the-art methods, we employ Plan*RAG $_{\text{SubQ}}$, which retrieves a fresh set of documents for each sub-query, maintaining consistency with the experimental setup of other state-of-the-art approaches.

For both experiments, we employ *accuracy contains* as the evaluation metric. This metric represents a relaxed version of the *exact-match* score, considering a prediction correct when the true answer ap-

pears as a substring within the prediction. This approach accounts for variations in answer phrasing across different language models.

E.3 RETRIEVAL EFFECTIVENESS OF PLAN*RAG

In this experiment, we evaluate the effectiveness of Plan*RAG $_{\text{SubQ}}$'s reasoning plan nodes, which generate targeted subatomic queries leading to improved retrievals. Using a random sample of 1000 queries from the HotpotQA *distractor* dataset, which provides golden sentences for evaluation, we assess retrieval performance against ReAct and QD-RAG$_{\text{SubQ}}$. As shown in Table 4, Plan*RAG $_{\text{SubQ}}$ demonstrates superior performance across all threee metrics: Precision, Recall, and Accuracy.

Table 6: **Information Gain (IG)**: IG at different depths of the reasoning DAG, demonstrating systematic information accumulation across reasoning steps on HotpotQA, validating the structural coherence of the reasoning DAG.

Depth	0	1	2	3	4	5	6
IG	7.23	8.85	8.91	9.02	9.47	9.75	10.0

To account for potential fragmentation of golden sentences during embedding generation, we segment the golden sentences into chunks of size 50. A retrieval is considered successful if any chunk of the golden sentence is present in the retrieval.

E.4 INFORMATION GAIN (IG)

We evaluate the effectiveness of Plan*RAG's reasoning plan by analyzing the cumulative information gain (IG) across DAG-depth. A valid reasoning plan should demonstrate increasing IG with depth, as each subquery contributes additional relevant information toward answering the main query. Conversely, stagnant or decreasing IG would indicate subqueries at that depth fail to contribute meaningful information.

Using the HotpotQA dataset (7,405 queries), we employ GPT-3.5$_{\text{turbo}}$ to compute and return IG scores at each depth of the reasoning DAG. We showcase the average IG score at each depth in Table 6. The evaluation prompt is

```
Suppose you are an expert in quantifying information gain. Given
a main query and a set of subqueries quantify how much knowledge
is gained by this whole set of subqueries. Range of information
gain: 1-10. 1 being the least and 10 being the most.
The special tag <AI.J> is used to denote the place where the
answer to QI.J will be placed.
NOTE: Only output the Information Gain number that can be passed
to eval() directly. No explanations or anyother tag. Strictly
follow the below template.

Example:

Main Query: What is the DoB of the current President of Finland?
Subqueries: [Q1.1: Who is the current President of Finland?]
Information Gain: 5

Main Query: What is the DoB of the current President of Finland?
Subqueries: [Q1.1: Who is the current President of Finland?, Q2
.1: When was <A1.1> born?]
Information Gain: 10

Main Query: What is the DoB of the current President of Finland?
Subqueries: [Q1.1: What is the capital of Finland?]
Information Gain: 0
```

E.5 REASONING PLAN GENERATION

While our primary experiments use GPT-4o to generate the reasoning plan, we demonstrate that Plan*RAG's performance is not dependent on the large language models. For this, we finetune the Llama3.1-instruct$_{8b}$ model with LoRA adapters on a small subset of HotpotQA-*train* dataset (3,700 query-plan pairs), using plans generated by GPT-4 as training data. The results show that a relatively small set of training data is sufficient to finetune a language model to generate structure output required to construct the DAG.

The finetuning process employs the *Parameter-Efficient Fine-Tuning (PEFT)* library with the following hyperparameters: 5,000 training steps, LoRA rank of 8, LoRA alpha of 16, batch size of 4, and gradient accumulation steps of 2. Table 5 demonstrates the competitive accuracy of the finetuned model compared to other methods on a subset of HotpotQA-*fullwiki* dataset, confirming that Plan*RAG's enhanced performance is not dependent on a large language model.