SR-RAG: Verifiable Multi-Hop Reasoning via On-the-fly Symbolic Graph Construction

Zehua Wang[1,] , **Zhaojing Zhang**[1,] , **Boyu Qiu**[1,] , **Xiaolong Weng**[1,],
Ying Xiong[1*] , **Buzhou Tang**[1,2*], **Min Zhang**[1]
[1]Department of Computer Science, Harbin Institue of Technology (Shenzhen), Shenzhen, China
[2]Peng Cheng Laboratory, Shenzhen, China
wangzehua@stu.hit.edu.cn,
tangbuzhou@gmail.com

Abstract

Retrieval-Augmented Generation (RAG) has been widely adopted to enhance large language models (LLMs) by incorporating external knowledge. However, the two main existing paradigms struggle with multi-hop reasoning: *aggregate-first* approaches suffer from high construction costs and limited adaptability to dynamic knowledge, while *dynamic-first* approaches rely heavily on LLM reasoning and are prone to error propagation across reasoning steps. To address these limitations, we propose SR-RAG, a symbolic reasoning framework for multi-hop question answering. SR-RAG integrates the advantages of both paradigms by dynamically generating sub-questions, performing information retrieval and symbolic encoding based on an on-the-fly graph, and using a symbolic verifier to formally validate intermediate reasoning steps to ensure the correctness of intermediate answers and the completeness of the reasoning chain . We evaluate SR-RAG on multiple multi-hop benchmarks and a medical dataset. Experimental results demonstrate that it significantly improves both accuracy and robustness.

1 Introduction

Retrieval-Augmented Generation (RAG) has emerged as a dominant paradigm for enhancing large language models (LLMs) with external knowledge, enabling them to perform knowledge-intensive tasks beyond their parametric memory (Lewis et al., 2020). However, despite these advances, multi-hop reasoning remains a persistent challenge for RAG-based methods. Building on the basic framework, various paradigms have been proposed to tackle multi-hop reasoning, as illustrated in Figure 1, we categorize them into two families: *aggregate-first* and *dynamic-first* RAG .

The former, such as graph-based RAG systems, construct offline knowledge graphs based on whole knowledge bases and perform reasoning over aggregated structures. While effective in enforcing relational constraints (Edge et al., 2024; Han et al., 2025), these methods incur high construction and maintenance costs and struggle to adapt to dynamic or query-specific knowledge (Chen et al., 2025; Zhang et al., 2025). In contrast, latter methods, including agentic and chain-of-thought-driven RAG frameworks, perform reasoning incrementally by interleaving retrieval and generation steps (Li et al., 2025b). Although more flexible, these approaches

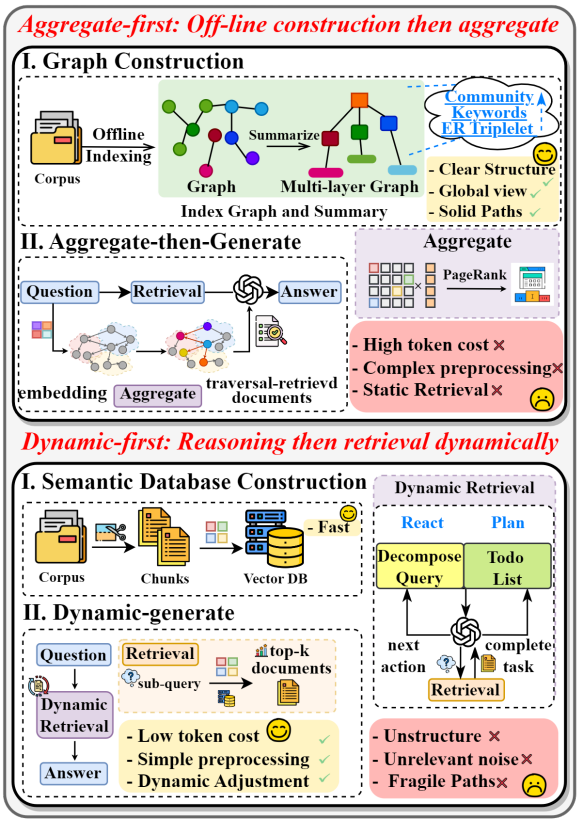

Figure 1: **Two RAG Paradigms**: Upper part shows the *aggregate-first* paradigm, exemplified by GraphRAG. Lower part illustrates the *dynamic-first* paradigm, represented by React and Plan. Each has its own strengths and weaknesses in handling multi-hop reasoning tasks.

*Corresponding authors.

Findings of the Association for Computational Linguistics: ACL 2026, pages 38588–38606
July 2-7, 2026

rely heavily on LLM-internal reasoning, rendering them vulnerable to error propagation: an incorrect intermediate inference can irreversibly bias subsequent retrieval and reasoning steps, leading to cascading failures(Banerjee et al., 2025; Wang et al., 2025).

Crucially, both paradigms lack mechanisms to formally validate and update intermediate states: *aggregate-first* approaches validate reasoning implicitly through static graph structures, while *dynamic-first* methods rely on LLM reasoning heuristics. Therefore, they only provide limited guarantees on the logical validity of intermediate conclusions and lack interpretability, which is particularly problematic in multi-hop scenarios.

To address this gap, we propose SR-RAG, a symbolic reasoning framework that introduces explicit symbolic verification into the reasoning process and synergistically combines the dynamic adaptability of *dynamic-first* methods with the relational rigor of *aggregate-first* approaches. Specifically, it dynamically constructs query-specific graphs from retrieved evidence and encodes them into first-order logical premises, then employs a symbolic verifier to obtain correct answers based on these premises, mitigating the spread of reasoning errors. In addition, we have also introduced a symbol-guided feedback loop to ensure the symbolic grounding and the integrity of dynamically updated graphs.

As shown in Table 1, this design yields key advantages. **First,** symbolic verification acts as a formal guardrail over LLM-driven reasoning, enforcing logical consistency. **Second,** by constructing symbolic structures on demand, it preserves the adaptability without incurring the rigidity of static graph construction. **Third,** the resulting reasoning process is interpretable and verifiable, producing explicit logical traces that expose how conclusions are derived.

We evaluate SR-RAG on multi-hop benchmarks and a medical QA dataset. Experimental results demonstrate that our method outperforms existing RAG methods, as shown in Figure 2. We summarize the contributions as follows:

1. We identify the absence of intermediate reasoning validation as a core limitation of existing RAG paradigms for multi-hop reasoning.
2. We introduce SR-RAG, a new framework that integrates dynamic retrieval with step-level symbolic verification.
3. We demonstrate through extensive experiments that explicit symbolic constraints substantially improve robustness and interpretability in multi-hop RAG systems.

2 Preliminaries

In this section, we present the theoretical foundations of symbolic verification in SR-RAG. First, a formal definition of the basic concepts of symbolization is provided. Subsequently, the transformation mechanism for mapping knowledge graph structures to logical expressions is elaborated in detail. Finally, we formulate the verification task as a variable matching problem aimed at identifying a specific result variable.

2.1 Symbolic Logic Formulation

Symbolization entails the abstraction of unstructured natural language knowledge into a structured logic formal representation. The logic language of SR-RAG is First-Order Logic (FOL). Let $\mathcal{L}$ denote a first-order logic language, and is composed of a tuple $\langle \mathcal{C}, \mathcal{X}, \mathcal{P} \rangle$, defined as follows:

- $\mathcal{C}$ is a set of **constant symbols** representing distinct entities within the domain of discourse.
- $\mathcal{X}$ is a set of **variable symbols** (e.g., x, y, z) acting as placeholders that range over domain elements.
- $\mathcal{P}$ is a set of **predicate symbols** denoting properties or relations among terms. Each predicate $P \in \mathcal{P}$ is assigned a arity $n \geq 1$.

Definition 1 (Atomic Formula)**.** *An atomic formula (atom) is an expression of the form* $P(t_1, \ldots, t_n)$ *where* $P \in \mathcal{P}$ *and each* $t_i \in \mathcal{C} \cup \mathcal{X}$.

For example, $\text{Man}(\text{Socrates})$ is a one arity atom where Man is a predicate, 'Socrates' is a constant. In addition, combined with logical connectives $(\neg, \wedge, \vee, \rightarrow)$ and quantifiers $(\forall, \exists)$ we can build complex rules such as $\forall x.\, \text{Man}(x) \rightarrow \text{Mortal}(x)$.

Definition 2 (Premises)**.** *Premises* Π *is a finite set of atoms in* $\mathcal{L}$ *that serves as the axiomatic basis for deduction.*

Practically, Π is composed of two parts: facts (ground atoms that record observations) and rules (domain knowledge). Formally, let $\mathcal{F}_{facts}$ denote the set of facts and $\mathcal{R}_{rules}$ denote the set of rules, then:

$$\Pi = \mathcal{F}_{facts} \cup \mathcal{R}_{rules}.$$

Property		Traditional	Graph-based	Agentic	**SR-RAG**
1.*Adaptive*	*dynamic queries*	?		✔	✔
2.*Structural*	*structured constraints*		✔		✔
3. *Interpretable*	*reasoning paths*		?	✔	✔
4. *Robust*	*error correction*				✔

Table 1: **SR-RAG matches all properties**, while baselines miss more than one property.

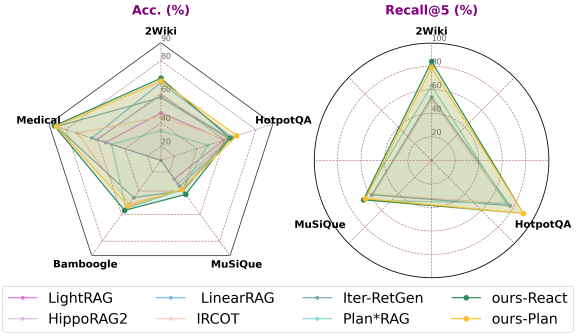

Figure 2: Acc(%) and recall@5 (%) of different methods on the datasets. Acc is the average of Acc@R and Acc@L.

2.2 Logic-based Knowledge Representation

To symbolize unstructured knowledge, we introduce graph structures as an intermediate representation because it can capture the semantic information of entities and their relationships. Consider a Knowledge Graph denoted as $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{A}, \mathcal{T})$, where $\mathcal{E}$ is the set of entities, $\mathcal{R}$ is the set of relations, $\mathcal{A}$ is the set of attributes, and $\mathcal{T}$ represents the set of triples.

We define a mapping function $f : \mathcal{G} \rightarrow \Pi_{\mathcal{G}}$ to encode graph elements into first-order logic premises. The mapping is specified as follows:

1. **Entity Mapping:** Each entity $e \in \mathcal{E}$ is mapped to a unique constant symbol $c_e \in \mathcal{C}$.
2. **Relation Mapping:** A relational triple can be represented two arity predicate. Specifically, a triple $(h, r, t) \in \mathcal{T}$ is formalized as the ground atom $p_r(c_h, c_t)$.
3. **Attribute Mapping:** An attribute $a \in \mathcal{A}$ associated with an entity is mapped to a predicate $p_a(x, v)$ or $p_a(x)$, where $x \in \mathcal{C}$ represents the entity and v represents the attribute value (literal or constant).

2.3 Symbolic Constraint Solving

Based on the $\Pi_{\mathcal{G}}$, the final stage of our framework involves symbolic constraint solving. This process transforms the natural language problem into a logical deduction task, executed by an external symbolic prover.

Question Formulation. Let $\mathcal{Q}$ denote a question, and $g(\mathcal{Q}|\Pi_{\mathcal{G}}) \rightarrow \mathcal{L}$ be a semantic parsing function that translates the input question into a logical expression, denoted as the goal formula ψ. The goal formula ψ is checked against premises to obtain the result returned by the prover.

Deduction via External Interpreter. Symbolic deduction is performed by verifying whether the goal formula ψ is entailed by the $\Pi_{\mathcal{G}}$. We employ an external logic prover (SWI-Prolog interpreter[1]) as the inference engine. Formally, the prover seeks a substitution θ for the variables in ψ, such that the grounded formula $\psi\theta$ satisfies:

$$\Pi_{\mathcal{G}} \models \psi\theta \tag{1}$$

where $\models$ denotes the logical entailment and θ constitutes the answer to the original question $\mathcal{Q}$.

bottom part illustrates the "Answer Generator"

3 Proposed Method: SR-RAG

For a complex question Q, and knowledge base K, the task aims to find an answer a from K. The overall framework of SR-RAG consists of three main components: Dynamic Generator, Symbolic Verifier, and Answer Generator. The framework is shown in Figure 3 and detail pseudocode is provided in Appx. C.

3.1 Dynamic Generator

Inspired by recent advances in agentic reasoning, we design two dynamic generator variants to produce reasoning paths for complex questions.

The first is a ReAct-based Dynamic Generator, which interleaves reasoning and actions based on observations. At step t, the model generates sub-query q_t based on the original question Q and the accumulated history $H_{t-1} = \{(q_1, a_1), \ldots, (q_{t-1}, a_{t-1})\}$, where each (q_i, a_i) pair represents a previous sub-query and its corresponding answer. The generation process is defined as:

$$q_t = \text{GenerateStep}(Q, H_{t-1}) \tag{2}$$

[1]https://github.com/yuce/pyswip

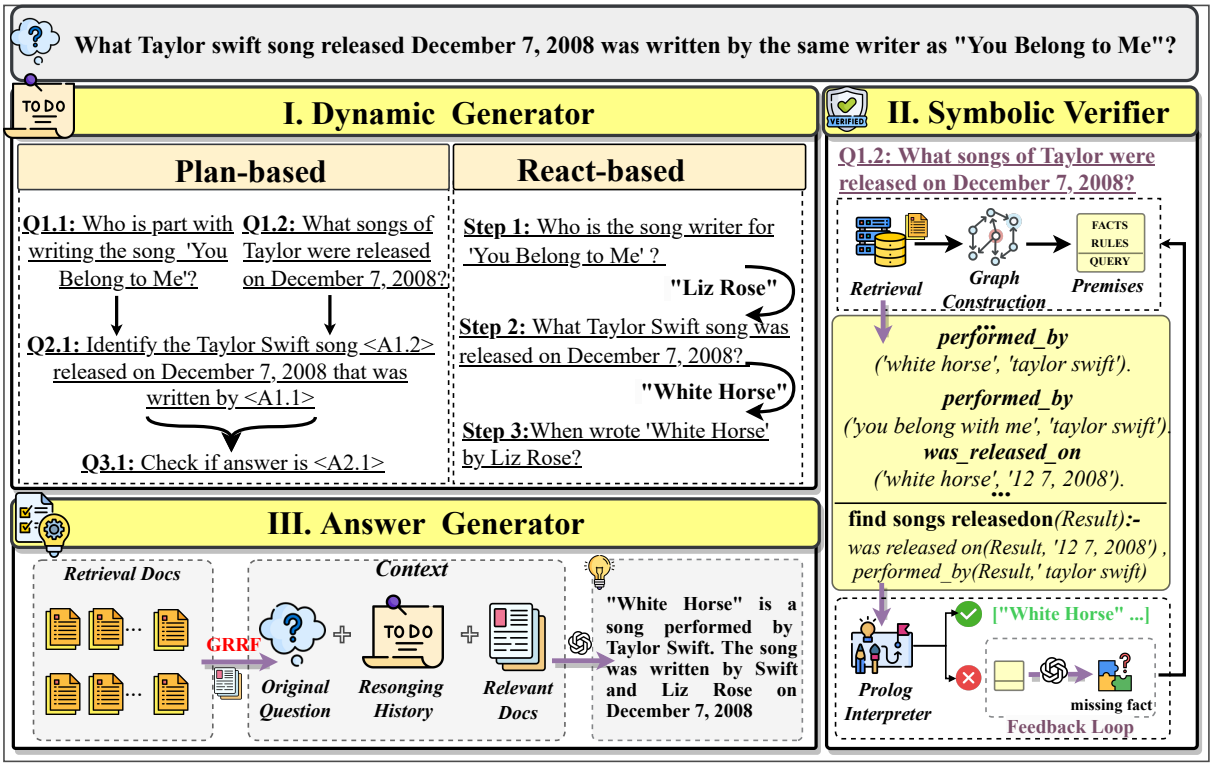

Figure 3: **Framework of SR-RAG.** The top part shows the Dynamic Generator. The bottom part illustrates Answer Generator while the right side illustrates the Symbolic Verifier. The entire process iteratively generates sub-queries, verifies their answers symbolically, and finally synthesizes the final answer.

The second is a Plan-based Dynamic Generator, which first produces a high-level plan and then follows it to generate detailed steps. To enable parallel processing, the plan is represented as a Directed Acyclic Graph $\mathcal{P}_{DAG}$: nodes correspond to reasoning steps and edges to their dependencies. We use a masking mechanism so child-node queries are dynamically adjusted based on parent answers, preserving flexibility while following the plan. This is formalized as:

$$\begin{aligned} \mathcal{P}_{DAG} &= \text{GeneratePlanDAG}(Q), \\ v_t &= \text{PopNode}(\mathcal{P}_{DAG}), \\ q_t &= \text{MaskingAdjust}\big(v_t, \\ &\qquad \{a_p \mid p \in \text{Parents}(v_t)\}\big). \end{aligned} \tag{3}$$

The function $\text{Parents}(v_t)$ returns the parent nodes of v_t in the DAG and $\{a_p\}$ are their validated answers. The adjusted sub-query q_t is then issued to the retrieval and verification pipeline.

For clarity, we collectively refer to the two formats as the Dynamic Generator. The sub-query produced at step t is defined as

$$q_t = \text{DynamicGenerator}(Q, H_{t-1} \mid \text{Mode}), \tag{4}$$

where $\text{Mode} \in \{\text{Plan, ReAct}\}$ specifies which generator format is employed.

3.2 Symbolic Verifier

The Symbolic Verifier is designed to obtain the answer of q_t and verify the validity of it. This process follows a rigorous pipeline: retrieve relevant documents, construct a query-specific graph, transform it into symbolic premises, perform logic verification and feedback correction if necessary. The overall process can be summarized as: $D_t \rightarrow \mathcal{G}_t \rightarrow \Pi_t \rightarrow a_t$.

On-the-fly Graph Construction. To answer the q_t, we first retrieve a set of supporting documents D_t from the knowledge base K. Then, we construct a dynamic, query-specific subgraph $\mathcal{G}_t = (\mathcal{E}_t, \mathcal{R}_t, \mathcal{A}_t, \mathcal{T}_t)$ on the fly. Information Extraction (IE) module is used to extract entities and relations from D_t. This step structures the unstructured textual evidence into a graph format tailored to the current reasoning context.

Logic-based Verification. Once the subgraph $\mathcal{G}_t$ is constructed, following the mapping rules (see in Section 2.2), $\mathcal{G}_t$ is transformed into premises

$\Pi_{\mathcal{G}_t}$ using the mapping function f. Simultaneously, the natural language q_t is parsed into a goal formula $\psi_t = g(q_t|\Pi_{\mathcal{G}_t})$. The verification is then executed by an external symbolic prover. We check for logical entailment, if the prover finds a valid substitution θ, the verification returns TRUE, and the substituted value is accepted as the answer a_t. If no such substitution exists, the verification enters a feedback loop.

Symbolic-Guided Feedback Loop. If the prover fails to entail ψ_t from Π_t, SR-RAG utilizes the failure trace as feedback to iteratively refine the graph $\mathcal{G}_t$, rather than terminating. Specifically, an LLM-based reflection analyzes the goal ψ_t, and current premises Π_t to abductively propose a missing fact f_{miss} required for entailment. This fact guides a focused graph completion:

$$\mathcal{E}'_t, \mathcal{R}'_t, \mathcal{A}'_t = \text{UpdateGraph}(D_t, f_{\text{miss}}) \quad (5)$$

The new entities $\mathcal{E}'_t$, attributes $\mathcal{A}'_t$, and relations $\mathcal{R}'_t$ are integrated into $\mathcal{G}_t$ to yield an updated premise set Π'_t. The verifier then invokes verification on Π'_t, repeating this cycle up to a maximum iteration limit.

3.3 Answer Generator

The Answer Generator synthesizes the final answer by aggregating the entire reasoning history. Let $H = \{(q_1, a_1, D_1), \ldots, (q_n, a_n, D_n)\}$ represent the sequence of sub-queries, their answers, and supporting documents. Specifically, we propose Global Reciprocal Rank Fusion (GRRF), which fuses the document sets $\bigcup D_i$ collected across all reasoning steps. This prioritizes documents that are consistently relevant throughout the reasoning chain. The GRRF is implemented in Algorithm 1.

Finally, the original question Q, the structured reasoning history H, and the re-ranked documents D_{fused} are fed into the LLM to generate the final answer:

$$A_{final} = \text{LLM}(Q, H, D_{fused}) \quad (6)$$

Based on the above components, SR-RAG iteratively generates sub-queries, verifies their answers symbolically, and finally synthesizes the final answer.

Algorithm 1: Global Reciprocal Rank Fusion

Input: Sub-queries $\mathcal{Q}_{sub}$, Fact Set $\mathcal{F} = (\mathcal{E} + \mathcal{R} + \mathcal{A})$, Top-$K$, Weights $\mathbf{w} = [w_{sp}, w_{de}]$, η

Output: Ranked Documents $\mathcal{D}_{final}$

Initialize Score Map $S \leftarrow \emptyset$;
foreach *query* $q \in \mathcal{Q}_{sub}$ **do**
  $R_{sp} = \text{SparseSearch}(q, \mathcal{F})$;
  $R_{de} = \text{DenseSearch}(q, \mathcal{F})$;
  /* Iterate search */
  foreach $(\mathcal{L}, w) \in \{(R_{sp}, w_{sp}), (R_{de}, w_{de})\}$ **do**
    foreach *rank* r, *fact* $f \in \mathcal{L}$ **do**
      $S(f) \leftarrow S(f) + \frac{w}{\eta + r + 1}$;
/* Top-K Diversified Selection */
$S_{sorted} = \text{Sort}(S, \text{key} = score, \text{DESC})$;
$\mathcal{D}_{final} \leftarrow []$, $\mathcal{V}_{seen} \leftarrow \emptyset$;
foreach *fact* f, *score* $\in S_{sorted}$ **do**
  $d_{doc} \leftarrow \text{GetDocument}(f)$;
  if $d_{doc} \notin \mathcal{V}_{seen}$ **then**
    $\mathcal{V}_{seen} \leftarrow \mathcal{V}_{seen} \cup \{d_{doc}\}$;
    $\mathcal{D}_{final}$.append(d_{doc});
    if $|\mathcal{D}_{final}| \geq K$ **then**
      break;
Return $\mathcal{D}_{final}$;

4 Experiments

We conducted experiments to answer the following research questions (RQ):

RQ1. **Effectiveness:** To what degree does SR-RAG succeed on other RAG methods?
RQ2. **Ablation Study:** Which components or design choices of SR-RAG are indispensable for its performance?
RQ3. **Interpretability:** How does SR-RAG improve interpretability?
RQ4. **Trade-off:** How to make a trade-off between different paradigms?

4.1 Datasets and Evaluation

We evaluated our method on multiple multi-hop benchmarks, including HotpotQA (Yang et al., 2018), 2WikiMultiHopQA (Ho et al., 2020), MuSiQue (Trivedi et al., 2022b), Bamboogle (Press et al., 2022) and the Medical dataset from GraphRAG-Bench (Xiang et al., 2025). Because answers in open-ended QA are textual, we fellow the evaluation metrics of (Zhuang et al., 2025), primarily using Accuracy (denoted Acc@R) to verify whether model-generated responses contain the key information from the reference answers and

Method	HotpotQA		2Wiki		MuSiQue		Bamboogle		Medical	AVG	
	Acc@R	Acc@L	Acc@R	Acc@L	Acc@R	Acc@L	Acc@R	Acc@L	Acc@L	Acc@R	Acc@L
aggregate-first											
Vanilla RAG	50.70	52.40	35.10	29.80	21.20	24.30	36.00	36.00	55.43	35.75	39.59
RAPTOR	55.90	58.30	50.10	42.10	23.30	27.40	-	-	55.75	43.10	45.89
LightRAG	60.30	59.50	55.20	39.00	27.40	28.60	-	-	54.36	47.63	45.36
HippoRAG	57.00	59.30	66.10	59.90	29.30	24.10	-	-	55.04	50.80	49.58
HippoRAG2	62.90	64.30	62.70	55.00	31.00	35.00	-	-	60.77	52.20	53.77
GFM-RAG	62.70	65.60	66.80	59.60	29.90	34.60	-	-	56.07	53.13	53.97
LinearRAG	64.30	66.50	70.20	63.70	33.90	37.00	-	-	63.72	56.13	57.73
dynamic-first											
CoT	54.40	53.40	38.70	29.80	26.70	30.50	46.40	48.80	69.45	41.55	46.39
Self-ASK	56.80	58.60	43.40	48.40	26.90	28.80	45.60	48.80	69.69	43.17	50.86
IRCoT	62.10	59.60	50.70	38.20	36.30	41.50	51.20	52.00	73.08	50.08	52.88
Iter-RetGen	**70.50**	55.30	63.20	51.90	**41.80**	37.70	48.00	44.00	85.55	55.88	54.89
Plan*RAG	51.00	45.30	39.20	28.20	39.40	38.40	53.60	56.80	50.19	45.80	43.78
GenGround	61.20	58.70	59.30	52.60	34.20	33.40	42.80	45.60	77.98	49.38	53.66
IterKey	59.60	44.70	55.00	55.10	30.30	28.80	47.20	44.80	85.60	48.03	51.80
SR-RAG $_{react}$	65.20	62.40	**72.30**	67.00	**41.80**	**44.50**	**55.20**	**58.40**	**86.86**	**58.62**	**63.83**
SR-RAG $_{plan}$	68.10	**68.30**	68.40	**67.60**	38.40	38.60	52.80	53.60	85.64	56.92	62.75

Table 2: **Evaluation on benchmark datasets.** denotes our proposed method. The **best** results are shown in bold, and the second-best results are underlined. AVG is the average of Acc@R and Acc@L.

Acc@L to measure semantic and factual consistency between the predicted and reference answers. Detailed descriptions of the benchmarks and evaluation metrics are provided in Appx. A.

4.2 Baselines

We compared SR-RAG with baselines across two paradigms of RAG methods: **1)** *aggregate-first* RAG methods such as Naive RAG, RAPTOR (Sarthi et al., 2024), LightRAG (Guo et al., 2024), HippoRAG1&2 (Jimenez Gutierrez et al., 2024; Gutiérrez et al., 2025), GFM-RAG (Luo et al., 2025), and LinearRAG (Zhuang et al., 2025); **2)** *dynamic-first* RAG methods, including IRCoT (Trivedi et al., 2022a), Self-ASK (Press et al., 2022), Iter-RetGen (Shao et al., 2023), Plan*RAG (Verma et al., 2024), GenGround (Shi et al., 2024), IterKey (Hayashi et al., 2025). We also evaluated SR-RAG variants that use ReAct-based and Plan-based strategies. Detailed descriptions of baselines are provided in Appx. B.

4.3 Results and Analysis

4.3.1 Effectiveness (RQ1)

Table 2 shows that SR-RAG consistently outperforms all baselines. Not only multi-hop benchmarks, SR-RAG also achieves the best result on the Medical dataset, demonstrating effectiveness in complex open-domain settings. Moreover, our method achieves a steady improvement in results across all datasets, unlike the baselines which show significant fluctuations in performance across different datasets. Between the two SR-RAG variants, the react-based approach excels in most datasets, while the plan-based variant performs better on HotpotQA, suggesting that different reasoning strategies may be more effective depending on the dataset characteristics.

4.3.2 Ablation Study (RQ2)

We studied the effect of different design choices in SR-RAG, including symbolic verifier and answer generator. The partial results are shown in Table 3, and the full results are provided in Appx. D.1.

Symbolic Verifier. We compared two alternative strategies for sub-question answering: (i) relying solely on the LLM's direct output; and (ii) using the retrieved documents directly as the answer. The results show that introducing the symbolic verifier improves performance; relying only on the LLM or directly using retrieved documents both degrade performance, with the latter causing a larger drop.

In addition, we analyzed the effectiveness of the symbolic feedback loop. As shown in Table 4, the Valid Prolog Rate (VPR) is very high, indicating that LLMs can convert problems into effective Prolog code. And with more feedback iterations, the Trigger Rate (TR) improves significantly. This indicates that the feedback loop can improve the knowledge graph and premises to enhance the ability of the symbolic verifier to successfully find valid infer-

ence paths, thereby improving the overall accuracy of SR-RAG.

Settings / Variants	HotpotQA Acc@R	HotpotQA Acc@L	MuSiQue Acc@R	MuSiQue Acc@L
Full Component	**65.20**	**62.40**	**41.80**	**44.50**
Sub-question Strategy				
– LLM Direct Output	62.50	61.90	37.80	37.60
– Retrieved Docs Directly	61.80	61.20	36.50	34.20
Answer Generator Components				
– Replace GRRF w/ Top-5 Docs	63.20	62.40	38.50	41.30
– Remove Relevant Docs	63.80	61.10	36.20	35.80
– Remove Reasoning History	64.50	62.30	39.90	41.60

Table 3: **Ablation study of design choices** in SR-RAG $_{react}$. We compared against alternative strategies and component removals.

Answer Generator. We replaced GRRF with the top-k retrieved documents most relevant to the origin question, results shows degraded performance, indicating that GRRF effectively aggregates information from multiple sources. For the context used in answer generation, we assessed the impact of removing relevant-documents and the reasoning history. It found that both components contribute positively to performance. Specifically, removing relevant-document leads to a significant performance drop (e.g.,Musique Acc@L: 44.50 $\rightarrow$ 35.80). Overall, results confirm that each design choice in SR-RAG is essential for achieving optimal performance in multi-hop question answering.

	HotpotQA VPR	HotpotQA TR	Musique VPR	Musique TR
React-based				
No feedback	97.67	66.67	97.33	40.17
feedback × 1	97.67	72.83 (↑6.17)	97.33	59.67 (↑19.50)
feedback × 2	97.67	80.00 (↑13.33)	97.33	79.83 (↑39.67)
Plan-based				
No feedback	97.91	65.46	97.70	39.12
feedback × 1	97.91	73.66 (↑8.21)	97.70	60.88 (↑21.76)
feedback × 2	97.91	80.15 (↑14.69)	97.70	80.34 (↑41.22)

Table 4: **Symbolic Verifier Effectiveness Analysis.** VPR: percentage of Prolog code that is syntactically valid and executable. TR: proportion of "True" (successful reasoning path found). Sampling 200 examples from HotpotQA and Musique.

4.3.3 Interpretability (RQ3)

Figure 6 illustrates how SR-RAG enhances interpretability. The left part shows retrieved documents that lack a clear relational chain and contain multiple personal names that confuse the model and cause incorrect outputs. The right panel shows constructing Facts from the documents and encodes rules ("if A is B's son and B is male, then B is A's father") to identify matching relational entities. For the question "Who is the grandfather of Raghmall Mac Ruaidhrí?", the method first infers that his father is Ruaidhrí Mac Ruaidhrí, then identifies that Ruaidhrí Mac Ruaidhrí's father is Ailean Mac Ruaidhrí, producing the correct conclusion. This symbolic reasoning process not only improves answer accuracy but also provides verifiable logical steps, thereby increasing interpretability.

4.3.4 Trade-off (RQ4)

As shown in Table 2, both paradigms of RAG methods demonstrate advantages. Specifically, *aggregate-first* approaches excel on tasks with explicit relational chains, such as HotpotQA and 2Wiki, while *dynamic-first* methods show clear advantages on open-ended medical datasets and MuSiQue, where relationships are more implicit. This finding reveals that different paradigms possess distinct strengths when tackling specific problem types.

These complementary strengths motivate a hybrid solution. To that end, our SR-RAG integrates the advantages of both paradigms. Moreover, Figure 4 compares robustness to noise and shows that SR-RAG suffers a smaller performance drop than *dynamic-first* methods under high-noise conditions, which indicates improved reliability in challenging scenarios. In summary, by combining the mechanism of the two paradigms, SR-RAG provides a more effective and robust approach for complex multi-hop tasks.

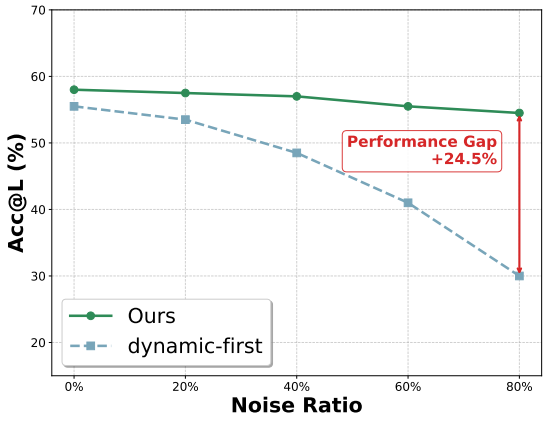

Figure 4: **Robustness to Noise Analysis.** Sampling 200 examples from HotpotQA and Musique.

4.4 Failure Analysis

To investigate the source of performance gains, we conducted a manual error analysis on 50 sampled failure cases from the baseline (Traditional RAG) and applied SR-RAG to these cases. As shown in Figure 5, the failures of the baseline predominantly categorized into Retrieval Failures (72%) and LLM Hallucinations (28%). Detailed analysis shows that the SR-RAG is particularly effective in mitigating retrieval issues: 21 retrieval errors were successfully corrected. In addition, the improvement in context quality and clear reasoning history help reduce hallucinations, with 8 such cases corrected. These results confirm that SR-RAG enhances robustness primarily by ensuring accurate evidence retrieval, thereby providing a basis for the generation of LLMs and mitigating the occurrence of hallucinations.

4.5 Inference time and latency analysis

Table 5 details the time cost of our framework, showing an average processing time of approximately 4.5s per iteration. Notably, the latency is dominated by the retrieval and information extraction phase (3.8s), whereas the core symbolic verification remains highly efficient (0.7s).

While this "on-the-fly" construction incurs higher latency compared to static retrieval methods, we frame this as a deliberate adoption of the "System 2" reasoning paradigm: trading test-time computation for logical rigor. In safety-critical fields such as medical, this latency is a justifiable cost to ensure interpretability and minimize hallucinations, and our method has also achieved significant results in medical QA. Furthermore, the current bottleneck stems primarily from the generation speed of general-purpose LLMs; in practical deployments, this engineering overhead can be significantly mitigated by parallelizing the extraction process or using cache, lightweight models for graph construction.

4.6 API Resources

The number of API calls and tokens for each component in SR-RAG are shown in Table 6. Caused by the multi-round retrieval, some documents may be retrieved multiple times, leading to a reduction in the overall token count.

Stage	Average Time (seconds)
Retrieval & Graph	3.8
Prolog Verify (1st)	0.7
Feedback Loop	2.4
Total (w/o feedback)	4.5
Total (feedback $\times$ 1)	6.9

Table 5: **Latency breakdown of different stages** : (1) Retrieval & Graph — document retrieval and entity/relation extraction to build the KG. (2) Prolog Verify (1st) — Prolog generation and solving. (3) Feedback Loop — LLM diagnosis and refinement when verification fails.

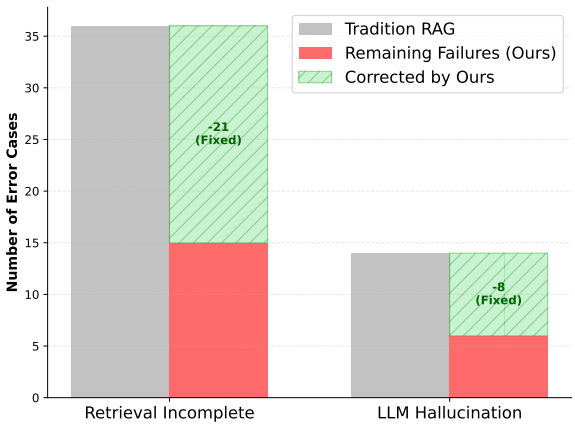

Figure 5: **Failure Analysis.** Error analysis on 50 sampled failure cases from Traditional RAG. The green hatched area shows that improvement brought by SR-RAG.

5 Related Works

Aggregate- and Dynamic-first RAG To address limitations of traditional RAG, recent work has followed two complementary directions. The first augments retrieval by organizing documents into structured representations, such as graphs (*aggregate-first* RAG). This line of work focuses on graph construction (e.g., LinearRAG (Zhuang et al., 2025)) and graph-aware retrieval algorithms (e.g., LightRAG (Guo et al., 2024), HippoRAG 1&2 (Jimenez Gutierrez et al., 2024; Gutiérrez et al., 2025)). These methods remain constrained by entity disambiguation, rigid schema definitions, and limited cross-document knowledge integration (Zhang et al., 2025).

The second direction leverages LLM reasoning to alternate retrieval and generation dynamically (*aggregate-first* RAG). It decomposes complex queries into multi-turn retrieve-and-generate

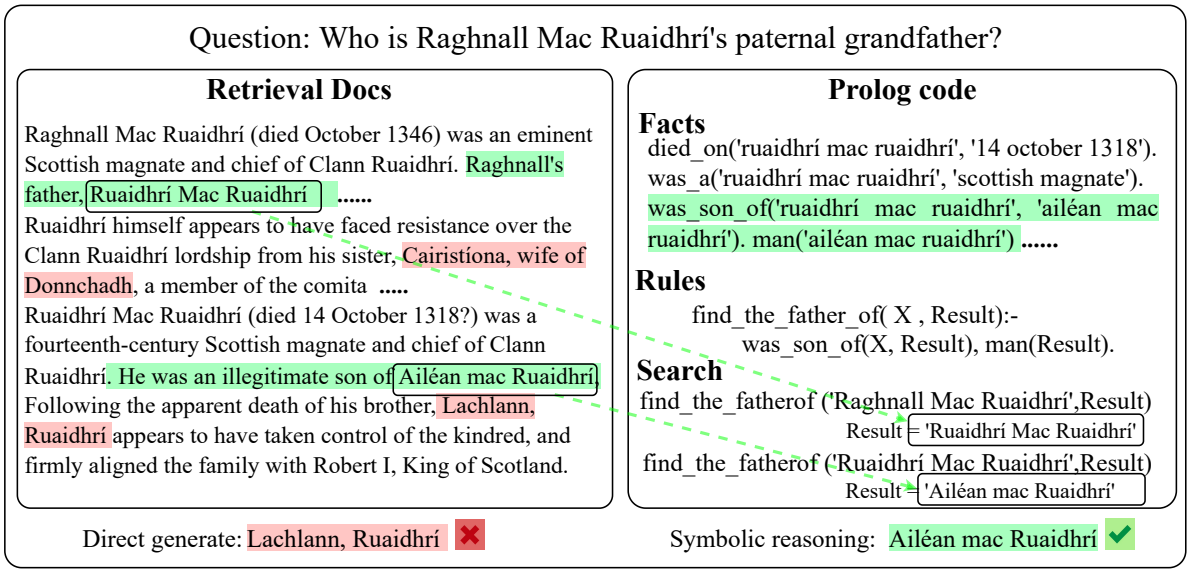

Figure 6: **SR-RAG is interpretable.** In this example, SR-RAG generates answers iteratively, strictly adhering to Facts and Rules, providing process-level interpretability and accurate answers compared to directly generating from text.

Component	API Call # / Iteration	Token # for Input	Token # for Output
Generator (React$_{base}$)	3	$\sim$ 450	$\sim$ 110
Generator (Plan$_{base}$)	$\sim$ 3	$\sim$ 560	$\sim$ 200
Graph	3	$\sim$ 800	$\sim$ 200
Symbolic	3	$\sim$ 1400	$\sim$ 80
Answer	1	$\sim$ 550	$\sim$ 110
Overall	-	$\sim$ 10k	$\sim$ 1.9k

Table 6: **Number of API calls and tokens** for compoents of SR-RAG.

procedures, as in ReAct (Yao et al., 2023) and Plan-then-Execute (Wang et al., 2023). While modular and extensible, this approach can suffer from error propagation, over-reliance on intermediate results, and a lack of mechanisms to evaluate and correct intermediates. Some research has integrated both paradigms (Ni et al., 2025; Li et al., 2025a), but they still remain separate in practice. SR-RAG uses symbolization as a bridge, closely combining the advantages of these two paradigms, and enhancing interpretability and reasoning ability.

Symbolic Reasoning with Logic Programming Logic programming derives conclusions from explicit facts and rules and provides a principled framework for symbolic reasoning. Among formalisms, First-Order Logic (FOL) is particularly suited for representing complex, sequential deductions (Qi et al., 2025). Recent work has integrated symbolic logic to improve the robustness of LLM reasoning (Pan et al., 2023; Han et al., 2024). Rather than prompting LLMs to produce FOL formulas directly, we construct structured knowledge graphs to organize facts extracted from documents and map them to FOL representations, enabling symbolic verification of multi-hop reasoning.

6 Conclusions

In this paper, we proposed SR-RAG, a symbolic retrieval-augmented generation framework for multi-hop reasoning. SR-RAG introduces an explicit symbolic verification layer that formalizes and validates intermediate reasoning states, thereby preventing cascading errors across multi-step inference. Crucially, SR-RAG innovates via dynamic, query-centric grounding: it constructs lightweight knowledge graphs from retrieved documents on-the-fly and maps them into premises, ensuring that every inference step is logically grounded and verifiable. Extensive experiments on multi-hop benchmarks demonstrate that symbolic verification plays a decisive role in improving both accuracy and robustness. By explicitly checking the logical entailment of intermediate answers, it mitigates error propagation, while producing interpretable reasoning chains grounded in formal logic.

Limitations

Despite these advantages, SR-RAG has several limitations. The effectiveness of symbolic verification depends on the quality of information extraction and relation grounding from retrieved documents; extraction errors may lead to incomplete or overly restrictive symbolic premises. We improved this through a feedback mechanism, but extreme cases with highly irrelevant or contradictory documents can still mislead the reasoning process. Furthermore, symbolic reasoning introduces additional computational overhead and latency. Although our design limits verification to small, query-specific graphs, thereby keeping costs within a manageable range in practical applications, but further improvements are expected by integrating more powerful retrieval and ranking models, as well as enhancing the domain adaptability of symbolic abstraction.

More broadly, this work suggests that explicit symbolic constraints can serve as a principled interface between retrieval and reasoning in RAG systems. We hope SR-RAG encourages future research on tighter neural–symbolic integration, advancing retrieval-augmented language models toward more reliable, interpretable, and verifiable multi-hop reasoning.

Acknowledgments

This study is partially supported by National Natural Science Foundation of China (62276082), National Key R&D Program of China (2023YFC3502900), Shenzhen Science and Technology Research and Development Fund (KJZD20240903102802003), Shenzhen Science and Technology Research and Development Fund for Sustainable Development Project (GXWD20231128103819001, 20230706140548006) and Guangdong Provincial Key Laboratory Grant (2023B1212060076).

References

Sourav Banerjee, Ayushi Agarwal, and Saloni Singla. 2025. Llms will always hallucinate, and we need to live with this. In *Intelligent Systems Conference*, pages 624–648. Springer.

Shengyuan Chen, Chuang Zhou, Zheng Yuan, Qinggang Zhang, Zeyang Cui, Hao Chen, Yilin Xiao, Jiannong Cao, and Xiao Huang. 2025. You don't need pre-built graphs for rag: Retrieval augmented generation with adaptive reasoning structures. *arXiv preprint arXiv:2508.06105*.

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

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. 2024. Lightrag: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779*.

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. 2025. From rag to memory: Non-parametric continual learning for large language models. *arXiv preprint arXiv:2502.14802*.

Haoyu Han, Yaochen Xie, Hui Liu, Xianfeng Tang, Sreyashi Nag, William Headden, Yang Li, Chen Luo, Shuiwang Ji, Qi He, et al. 2025. Reasoning with graphs: Structuring implicit knowledge to enhance llms reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 25698–25714.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Wenfei Zhou, James Coady, David Peng, Yujie Qiao, Luke Benson, et al. 2024. Folio: Natural language reasoning with first-order logic. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 22017–22031.

Kazuki Hayashi, Hidetaka Kamigaito, Shinya Kouda, and Taro Watanabe. 2025. Iterkey: Iterative keyword generation with llms for enhanced retrieval augmented generation. *arXiv preprint arXiv:2505.08450*.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. *arXiv preprint arXiv:2011.01060*.

Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. 2024. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in Neural Information Processing Systems*, 37:59532–59569.

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

Jiaoyang Li, Junhao Ruan, Shengwei Tang, Saihan Chen, Kaiyan Chang, Yuan Ge, Tong Xiao, and Jingbo Zhu. 2025a. Subqrag: Sub-question driven dynamic graph rag. *arXiv preprint arXiv:2510.07718*.

Yangning Li, Weizhi Zhang, Yuyao Yang, Wei-Chieh Huang, Yaozu Wu, Junyu Luo, Yuanchen Bei, Henry Peng Zou, Xiao Luo, Yusheng Zhao, et al. 2025b. Towards agentic rag with deep reasoning:

A survey of rag-reasoning systems in llms. *arXiv preprint arXiv:2507.09477*.

Linhao Luo, Zicheng Zhao, Gholamreza Haffari, Dinh Phung, Chen Gong, and Shirui Pan. 2025. Gfm-rag: graph foundation model for retrieval augmented generation. *arXiv preprint arXiv:2502.01113*.

Tengjun Ni, Xin Yuan, Shenghong Li, Kai Wu, Ren Ping Liu, Wei Ni, and Wenjie Zhang. 2025. Stepchain graphrag: Reasoning over knowledge graphs for multi-hop question answering. *arXiv preprint arXiv:2510.02827*.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Wang. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3806–3824.

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. 2022. Measuring and narrowing the compositionality gap in language models. *arXiv preprint arXiv:2210.03350*.

Chengwen Qi, Ren Ma, Bowen Li, He Du, Binyuan Hui, Jinwang Wu, Yuanjun Laili, and Conghui He. 2025. Large language models meet symbolic provers for logical reasoning evaluation. In *The Thirteenth International Conference on Learning Representations*.

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. 2024. Raptor: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*.

Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. *arXiv preprint arXiv:2305.15294*.

Zhengliang Shi, Shuo Zhang, Weiwei Sun, Shen Gao, Pengjie Ren, Zhumin Chen, and Zhaochun Ren. 2024. Generate-then-ground in retrieval-augmented generation for multi-hop question answering. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7339–7353.

Saba Sturua, Isabelle Mohr, Mohammad Kalim Akram, Michael Günther, Bo Wang, Markus Krimmel, Feng Wang, Georgios Mastrapas, Andreas Koukounas, Andreas Koukounas, Nan Wang, and Han Xiao. 2024. jina-embeddings-v3: Multilingual embeddings with task lora. *Preprint*, arXiv:2409.10173.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022a. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. *arXiv preprint arXiv:2212.10509*.

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

Prakhar Verma, Sukruta Prakash Midigeshi, Gaurav Sinha, Arno Solin, Nagarajan Natarajan, and Amit Sharma. 2024. Plan* rag: Efficient test-time planning for retrieval augmented generation. *arXiv preprint arXiv:2410.20753*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634.

Yuxin Wang, Shicheng Fang, Bo Wang, Qi Luo, Xuanjing Huang, Yining Zheng, and Xipeng Qiu. 2025. Multi-hop reasoning via early knowledge alignment. *arXiv preprint arXiv:2512.20144*.

Zhishang Xiang, Chuanjie Wu, Qinggang Zhang, Shengyuan Chen, Zijin Hong, Xiao Huang, and Jinsong Su. 2025. When to use graphs in rag: A comprehensive analysis for graph retrieval-augmented generation. *arXiv preprint arXiv:2506.05690*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W Cohen, Ruslan Salakhutdinov, and Christopher D Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. *arXiv preprint arXiv:1809.09600*.

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*.

Hairong Zhang, Jiaheng Si, Guohang Yan, Boyuan Qi, Pinlong Cai, Song Mao, Ding Wang, and Botian Shi. 2025. Rakg: Document-level retrieval augmented knowledge graph construction. *arXiv preprint arXiv:2504.09823*.

Luyao Zhuang, Shengyuan Chen, Yilin Xiao, Huachi Zhou, Yujing Zhang, Hao Chen, Qinggang Zhang, and Xiao Huang. 2025. Linearrag: Linear graph retrieval augmented generation on large-scale corpora. *arXiv preprint arXiv:2510.10114*.

A Appendix: Datasets and Evaluation

A.1 Datasets

For evaluating SR-RAG, we used the following open-source multi-hop datasets. We used the entire Bamboogle dataset for evaluation, while for the remaining datasets we applied the sampling strategy from (Zhuang et al., 2025).

- **2WikiMultiHopQA** (Ho et al., 2020). Designed to assess multi-step reasoning, 2WikiMultiHopQA integrates the structured knowledge of Wikidata with unstructured Wikipedia text to form natural QA pairs. The dataset comprises over 192,606 examples, split into 167,454 training and 25,152 test samples. It categorizes inquiries into four distinct reasoning types: comparison, inference, compositional, and bridge-comparison.

- **HotpotQA** (Yang et al., 2018). HotpotQA is a large-scale dataset featuring 112,779 Wikipedia-based pairs, specifically curated to foster multi-hop reasoning and explainable QA. Unlike systems limited by traditional knowledge schemas, it employs diverse natural language questions that require synthesizing information from multiple documents. Notably, the dataset provides sentence-level supporting facts to explicitly benchmark the reasoning process.

- **MuSiQue** (Trivedi et al., 2022b). To mitigate reasoning shortcuts, MuSiQue employs a bottom-up construction method that composes existing single-hop questions into multi-hop challenges. It contains approximately 248,814 examples with 2-4 hops. The dataset enforces connected reasoning—where intermediate outputs are requisite for subsequent steps—thereby minimizing disconnected shortcuts and significantly widening the human-machine performance gap.

- **Bamboogle** (Press et al., 2022). Bamboogle is a manually curated dataset of 125 questions designed to probe diverse combinatorial reasoning. Unlike automatically generated sets, these handcrafted 2-hop questions are structured to prevent direct retrieval via search engine snippets. This design necessitates genuine multi-fact synthesis rather than simple memorization, effectively measuring deep reasoning capabilities. must perform real combinatorial reasoning across multiple facts, effectively measuring its deep reasoning ability.

- **Meddical** (Xiang et al., 2025). The dataset is from structured clinical data from the National Comprehensive Cancer Network (NCCN) guidelines. These guidelines provide standardized treatment regimens, drug interaction hierarchies, and diagnostic criteria. The dataset includes four tasks with increasing complexity: factual retrieval, complex reasoning, contextual summarization, and creative generation, totaling 4076 questions across all difficulty levels.

A.2 Evaluation

In the main text, ACC_R and ACC_L are used as evaluation metrics, and their calculation formulas are as follows:

- **Acc@R**. The Acc@R determines whether a given ground truth answer is fully included within a predicted answer. For a single evaluation instance, this can be represented as:

$$\text{Acc@R} = \mathbb{I}(\ \mathcal{G} \subseteq \text{O}) \tag{7}$$

where $\mathcal{G}$ is golden answer and O is the predicted answer by LLMs.

- **Acc@L** . Acc@L evaluates the overall correctness of the predicted answer by leveraging an LLM as a judge. This metric assesses whether the predicted answer fully aligns with the meaning and key information of the Golden Answer. It's useful for evaluating open-ended generation tasks where exact string matching might be too strict. The evaluation is typically performed using a prompt template like the one below, where an LLM is asked to verify the correctness of the predicted answer against the golden answer. The prompt used is as follows:

Prompt Template of Answer Verification by LLM

Please evaluate if the generated answer is correct by comparing it with the gold answer. Generated answer: **{prediction}**. Gold answer: **{golden}**. The generated answer should be considered correct if it: 1. Contains the key information from the gold answer 2. Is factually accurate and consistent with the gold answer 3. Does not contain any contradicting information Respond with ONLY correct or incorrect. Response:

B Appendix: Baselines

We compared SR-RAG with several classical types of methods, categorized as follows:

B.1 *aggregate-first* RAG

- **RAPTOR** (Sarthi et al., 2024) constructs a hierarchical tree structure via recursive clustering and abstractive summarization, enabling the retrieval of information across varying levels of semantic granularity.

- **LightRAG** (Guo et al., 2024) introduces a dual-level indexing framework that integrates graph-based and textual representations, harmonizing fine-grained entity relations with coarse-grained high-level themes.

- **HippoRAG1&2** (Jimenez Gutierrez et al., 2024; Gutiérrez et al., 2025). HippoRAG is a training-free framework leveraging Personalized PageRank to guide single-step and multi-hop retrieval based on query concepts. Its successor, HippoRAG2, enhances performance by refining paragraph contextualization and optimizing seed node selection, thereby strengthening both factual retention and associative memory capabilities.

- **GFM-RAG** (Luo et al., 2025) implements a GraphRAG paradigm by generating document-derived graphs and utilizing a graph-enhanced mechanism to improve the precision of relevant document retrieval.

- **LinearRAG** (Zhuang et al., 2025) is a framework designed to simplify complex explicit relationship graphs, convert them into a linear, easily indexable view, and focus solely on modeling the semantic relationships between target entities and underlying text paragraphs, thereby avoiding unstable relationship modeling.

B.2 *dynamic-first* RAG

- **Self-ASK** (Press et al., 2022) utilizes an elicitive prompting strategy to decompose complex queries. Instead of answering directly, the model explicitly constructs and resolves a sequence of intermediate sub-questions, systematically building a logical path to the final answer.

- **IRCoT** (Trivedi et al., 2022a) targets knowledge-intensive multi-hop scenarios by interleaving Chain-of-Thought (CoT) reasoning with dynamic retrieval. It alternates between reasoning and retrieval steps, using the rationale generated in the current step to guide the acquisition of new information for the next.

- **Iter-RetGen** (Shao et al., 2023) optimizes performance through a synergistic loop of retrieval and generation. It leverages the output from the previous generation cycle as a refined, context-aware query to steer subsequent retrieval, thereby progressively enhancing the relevance and accuracy of the retrieved knowledge.

- **Plan*RAG** (Verma et al., 2024) addresses context overflow and reasoning inefficiencies in multi-hop tasks by externalizing the reasoning workflow. It structures the planning process as a Directed Acyclic Graph (DAG), enabling more organized navigation through complex queries.

- **GenGround** (Shi et al., 2024) introduces a generate-then-ground strategy for multi-hop question answering. It uses the "generation" capability of LLM to decompose problems and provide preliminary answers, and then utilizes the "grounding" capability combined with external knowledge for fact-checking and correction.

- **IterKey** (Hayashi et al., 2025) operates via a three-stage recursive framework: keyword generation, answer synthesis, and verification. In instances where verification fails, the model iteratively refines its search keywords to optimize retrieval precision until a valid response is derived.

C Appendix: Algorithm

The pseudocode of the overall SR-RAG process is shown in Algorithm 2. It consists of three main components: Dynamic Generator, Symbolic Verifier, and Answer Generator.

And the workflow of Global Reciprocal Rank Fusion (GGRF) proposed in the main text is as follows: According to the sub-queries, it performs hybrid retrieval for each sub-query, combining sparse and dense retrieval results. The retrieved facts are then scored and ranked based on their relevance to the sub-queries. Finally, a top-k documents linked to the highest-scoring facts are selected as the final retrieval results, ensuring diversity by avoiding multiple selections from the same document.

Algorithm 2: SR-RAG Process

Input: Question $\mathcal{Q}$, KB $\mathcal{K}$, Mapping Functions f, Parsing Function g, Symbolic Interpreter S, Mode (Plan/ReAct)

Output: Final Answer A_{final}

Initialize History $H \leftarrow \emptyset$;
while *not* FINISHED **do**
  /* Dynamic Generator */
  $q_t = \text{DynamicGenerator}(\mathcal{Q}, H_{t-1} \mid \text{Mode})$;
  if q_t *is* STOPSIGNAL **then**
    break;
  /* Symbolic Verifier */
  $D_t = \text{Retrieve}(\mathcal{K}, q_t)$;
  $\mathcal{G}_t = \text{GraphConstruct}(D_t)$;
  $\Pi_t = f(\mathcal{G}_t), \quad \psi_t = g(q_t \mid \Pi_t)$;
  $\theta = S(\Pi_t, \psi_t)$;
  if $\Pi_t \models \psi_t\theta$ **then**
    $a_t = \text{ApplySub}(\theta)$;
  else
    /* Feedback Loop */
    $iter \leftarrow 0$;
    while *not* $\Pi_t \models \psi_t\theta$ *and* $iter <$ FEEDBACK ITER **do**
      $f_{miss} = \text{LLM}(\psi_t, \Pi_t)$;
      $\mathcal{G}_t \leftarrow \text{UpdateGraph}(\mathcal{G}_t, f_{miss})$;
      $iter \leftarrow iter + 1$;
    if $\Pi_t \models \psi_t\theta$ **then**
      $a_t = \text{ApplySub}(\theta)$;
    else
      /* Fallback Generation */
      $a_t = \text{LLM}(q_t \mid \mathcal{G}_t)$;
  $H \leftarrow H \cup \{(q_t, a_t, D_t)\}$;
/* Answer Generator */
$D_{fused} = \text{GRRF}(\{D \mid (q, \cdot, D) \in H\})$;
$A_{final} = \text{LLM}(\mathcal{Q}, H, D_{fused})$;
Return A_{final};

D Appendix: Additional Experiments

D.1 Ablation Study

The ablation study results of SR-RAG on different benchmarks are shown in Table 7. The compoents of SR-RAG, are all effective across various datasets, demonstrating the robustness and generalizability of our designs. We observed that the multi-hop datasets benefit significantly from the symbolic verifier, as it helps ensure the logical consistency of the reasoning steps. The medical dataset shows less improvements from the design choices, likely due to it depends more on intensive knowledge rather than complex reasoning.

D.2 Effectiveness on Retriever

We evaluated the effectiveness of SR-RAG on different retrievers, including BM25, All-mpnet-base-v2[2], and Jina-embedding-v3 (Sturua et al., 2024). The results are shown in Table 8, we can observe that retriever matters significantly for RAG. This indicates that a more effective retriever can provide higher-quality evidence, which in turn enhances the overall performance in multi-hop reasoning tasks.

E Appendix: Reproducibility

E.1 Implementation Details

We implemented SR-RAG and all baselines based on the GPT-4o-mini. The retriever and knowledge base were the same as (Zhuang et al., 2025). The number of retrieved documents per sub-query was set to 3. Other key hyperparameter settings are shown in Table 9.

E.2 Prompt Templates

We provide the detailed prompt templates used in SR-RAG for reproducibility.

For react-based dynamic generation, the following prompt is used to initiate the reasoning loop. It guides the model to decompose complex questions into manageable sub-questions, facilitating step-by-step reasoning.

Prompt Template of React-based Generator

Now there is a complex question and related context information.
Question: **{question}**
Context: **{previous_steps}**
Let's think step by step to obtain a definitely accurate answer. You must consider what information is still required to answer this question and not omit any details.
Your response should start after "Thought:" and conclude with "Answer:".
Answer: the information you wish to obtain, in the form of an interrogative sentence.
Thought:

[2]https://huggingface.co/sentence-transformers/all-mpnet-base-v2

Settings / Variants	HotpotQA		2WikiMultiHopQA		MuSiQue		Bamboogle		Medical	AVG	
	Acc@R	Acc@L	Acc@R	Acc@L	Acc@R	Acc@L	Acc@R	Acc@L	Acc@L	Acc@R	Acc@L
SR-RAG $_{react}$											
Full Component	**65.20**	**62.40**	**72.30**	**67.00**	**41.80**	**44.50**	**55.20**	**58.40**	**86.86**	**58.62**	**63.83**
Sub-question Strategy											
– LLM Direct Output	62.50	61.90	65.20	64.80	37.80	37.60	52.80	56.80	86.71	54.58	61.56
– Retrieved Docs Directly	61.80	61.20	62.80	63.40	36.50	34.20	50.40	51.20	86.67	52.88	59.33
Answer Generator Components											
– Replace GRRF w/ Top-5 Docs	63.20	62.40	67.50	65.90	38.50	41.30	52.00	56.00	86.13	55.30	62.35
– Remove Relevant Docs	63.80	61.10	65.90	67.20	36.20	35.80	50.40	49.60	83.70	54.08	59.48
– Remove Reasoning History	64.50	62.30	67.30	69.50	39.90	41.60	56.80	54.40	86.71	57.13	62.90
SR-RAG $_{plan}$											
Full Component	**68.10**	**68.30**	**68.40**	**67.60**	**38.40**	**38.60**	**52.80**	**53.60**	**85.64**	**56.92**	**62.75**
Sub-question Strategy											
– LLM Direct Output	65.10	64.40	67.60	67.20	34.90	36.70	52.00	51.20	85.30	54.90	60.96
– Retrieved Docs Directly	63.20	65.60	65.30	64.10	30.80	33.40	48.80	49.60	85.45	52.03	59.63
Answer Generator Components											
– Replace GRRF w/ Top-5 Docs	67.90	67.60	65.80	68.20	35.40	33.90	49.60	50.40	85.20	54.68	61.06
– Remove Relevant Docs	66.40	66.80	63.70	66.90	33.60	35.20	48.80	51.20	85.01	53.13	61.02
– Remove Reasoning History	67.20	68.00	67.50	69.10	35.10	36.90	53.60	52.00	85.25	55.85	62.25

Table 7: **Comprehensive ablation study of SR-RAG across all datasets.** The **Full Model** (highlighted in gray) is compared against specific component replacements and removals. **Bold** indicates the performance of our proposed full method.

Retriever	HotpotQA			2WikiMQA			Musique		
	Recall@5	Acc@R	Acc@L	Recall@5	Acc@R	Acc@L	Recall@5	Acc@R	Acc@L
BM25	88.65	62.60	61.70	92.75	75.30	68.10	60.81	38.40	40.20
All-mpnet-base-v2	90.06	65.20	62.40	84.08	72.30	67.00	66.88	41.80	**44.50**
Jina-embedding-v3	**91.90**	**68.80**	**64.50**	**93.10**	**75.90**	**71.20**	**68.80**	**43.20**	42.40

Table 8: **Recall@5 and accuracy of SR-RAG with different retrievers** on multi-hop reasoning datasets.

	Top-K	w_{sp}	w_{de}	η	FI	RI
Default	5	0.6	0.4	1.0	1	3

Table 9: **Hyperparameter Settings of SR-RAG.** FI denotes feedback iteration, RI denotes React iteration.

For queries requiring structured planning, the following prompt instructs the model to decompose the main query into a logical, non-cyclic graph of sub-queries, ensuring dependencies are correctly ordered before execution.

Prompt Template of Plan-based Generator

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 list of tuples of parent query and the subatomic query.
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: 1. Make the DAG connected and for simple queries return the original query only without any reasoning DAG. 2. There is only a unique leaf node to obtain the final answer.

{example}
Now, please generate the reasoning DAG for the following query.
Query: **{query}**
DAG:

Few-shot Exemplars for DAG

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

Example II:
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>?")
]

To construct the domain-specific knowledge graph, we proceed in two stages. First, the **Entity Extraction** prompt identifies and categorizes key named entities from the retrieved passages, forming the nodes of the graph. Subsequently, the **Relation Extraction** prompt is employed to establish semantic links between the identified entities.

Prompt Template for Entity Extraction

 # Task Description
Your task is to extract named entities from the given paragraph. Respond with a JSON list of entities. Follow the output format as shown in the Example.

{example}
 # Now, give output for the following passages: {passages}
Output:

Prompt Template for Relation Extraction

 # Task Description
Your task is to construct an RDF (Resource Description Framework) graph based on the given paragraphs and list of named entities. Respond in the form of a JSON list of triples, where each triple represents a relationship in the RDF graph.
Please note the following requirements:
1.Each triple should contain at least one, and preferably two, named entities from each paragraph list.
2.Clearly resolve pronouns to their specific names to maintain clarity.
3.Follow the output format as shown in the Example.
{example}
 # Now, give output for the following passages:{passages}
Named Entity List:**{entitys}**
Output:

The prompt below directs the model to generate valid Prolog query statements based on extracted facts, specific logical rules and question.

Prompt Template for Prolog Generation

I need you to provide me with a complete Prolog query statement based on the facts I give you, combined with the provided questions, and in accordance with the rules.
References
The Prolog query must follow the rules below:**{prolog rules}**
The example Prolog outputs are as follows:
{prolog example}
Now, please generate the Prolog based on the facts and the question: The question is:
{question}.
The facts are as follows:
{prolog facts}
Only provide the Prolog code without any additional explanations.

Prolog Construction Rules

 # Basic Structure Pattern Rules as follows:
% Overall explanation of the rule structure
find_predicate_name(fixed_atom, Result) :-
% Step 1 explanation
sub_predicates(atom, IntermediateVar1),
% Step 2 explanation
sub_predicates(IntermediateVar1, IntermediateVar2),
% Step 3 Rmeaning
sub_predicates(IntermediateVar2, Result).
 # Elements Explanation:
find_predicate_name: The name of the predicate to be found, following snake_case naming and starting with find_.
fixed_atom: A fixed atom representing the query subject, enclosed in single quotes (e.g., "jason wei").
Result: The expected result to be found, and it is denoted by Result.
sub_predicates: The intermediate subgoal predicates represent each step of reasoning, and the predicates must use the basic predicates provided above context.
IntermediateVar: Intermediate variables used to pass results between subgoals, starting with an uppercase letter.
comment: Each subgoal must be followed by a comment and comments must start with % symbol

When the prover unsuccessfully finds a substitude answer, the following analysis prompt is used to find missing abstract entities or relationships in the knowledge graph.

Prompt Template for analysis of Prolog Query Failure

Based on the provided Prolog query and the Prolog premises, analyze the reasoning path and identify any gaps or missing information.
Provide a detailed explanation of why the Prolog query fails to find a substitution answer given the current Prolog premises.
 # Now, please analyze the following Prolog query and premises:
Prolog Query: **{prolog query}**
Prolog premises: **{prolog premises}**
Output:

Prompt Template for refine graph

Your task is to identify abstract entities or relationships that may be missing from the knowledge graph based on the analysis of the failed Prolog query.
Provide a list of specific entities or relationships that should be added to the knowledge graph to enable successful reasoning and substitution answers.
 # Now, please identify missing entities or relationships based on the following analysis:
Analysis: **{analysis}**
Retrieval Context: **{retrieval doc}**
Output:

Finally, this instruction ensures the model integrates the reasoning history and retrieved documents to produce a conclusive answer.

Prompt Template for Answer Generation

As an advanced reading comprehension assistant, your task is to analyze text passages and corresponding questions meticulously. Your response start after "Thought: ", where you will methodically break down the reasoning process, illustrating how you arrive at conclusions. Conclude with "Answer: " to present a concise, definitive response, devoid of additional elaborations.
Reasoning History: **{reasonging history}**
Relevant Documents: **{docs}**
Question: **{question}**
Thought:

E.3 Case Study Details

We present a case study of SR-RAG in Figures 7 and 8, with the first step shown in detail in Figure 7. This example illustrates how the method handles multi-hop reasoning tasks. Irrelevant documents and attributes have been omitted to show only the key information. The reasoning steps in the figures demonstrate that SR-RAG effectively employs symbolic verification to ensure logical consistency in multi-hop reasoning.

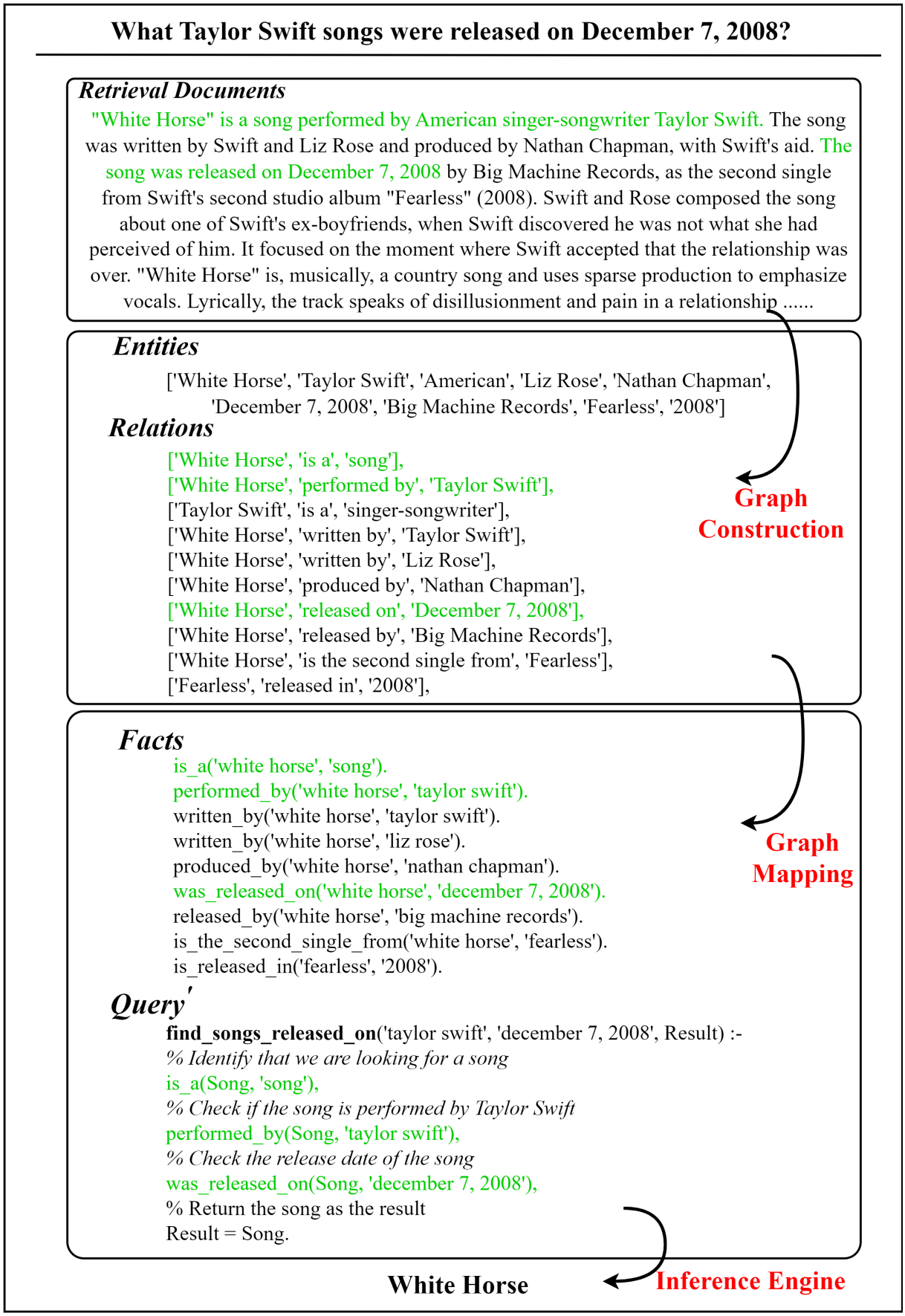

Figure 7: A case study of Symbolic Verification. We show how SR-RAG utilizes symbolic verification to ensure logical consistency in multi-hop reasoning. The green bold text indicates golden facts.

Figure 8: Case study of SR-RAG. The reasoning path is shown with green arrows, and the final context in the Answer Generator is displayed.