π-CoT: Prolog-Initialized Chain-of-Thought Prompting for Multi-Hop Question-Answering

Anonymous authors
Paper under double-blind review

Abstract

Chain-of-Thought (CoT) prompting significantly enhances large language models' (LLMs) problem-solving capabilities, but still struggles with complex multi-hop questions, often falling into circular reasoning patterns or deviating from the logical path entirely. This limitation is particularly acute in retrieval-augmented generation (RAG) settings, where obtaining the right context is critical. We introduce **P**rolog-**I**nitialized **C**hain-**o**f-**T**hought (π-CoT), a novel prompting strategy that combines logic programming's structural rigor with language models' flexibility. π-CoT reformulates multi-hop questions into Prolog queries decomposed as single-hop sub-queries, which are resolved systematically through `SLICE`—a procedure that seamlessly bridges symbolic and neural reasoning by translating each sub-query to natural language for LLM-powered question-answering, then converting answers back to an evolving knowledge base. By grounding each retrieval step in Prolog's systematic query resolution, we maintain a focused reasoning trajectory used to initialize the final CoT reasoning step. Extensive experimental evaluation demonstrates that π-CoT significantly outperforms RAG and in-context baselines on multi-hop question-answering benchmarks.

1 Introduction

Chain-of-thought (CoT) reasoning has emerged as a powerful paradigm for enhancing the problem-solving capabilities of large language models, substantially improving performance on arithmetic, commonsense, and symbolic reasoning tasks (Wei et al., 2022; Kojima et al., 2022). By encouraging models to articulate their reasoning process through intermediate steps, CoT enables more systematic and interpretable problem-solving approaches (Zhang et al., 2022; Wang et al., 2022). However, as the complexity of reasoning tasks increases—particularly in multi-hop scenarios where multiple interconnected inferences must be made—CoT systems have been observed to generalize poorly (Dziri et al., 2023) and become trapped in circular reasoning patterns (Lo et al., 2023; Yao et al., 2023).

This limitation becomes especially pronounced in retrieval-augmented generation (RAG) systems, where CoT excels at single-hop questions that require straightforward document retrieval and reasoning, but struggles significantly with multi-hop queries that demand the integration of information across multiple sources and reasoning steps (Asai et al., 2023). The fundamental challenge lies in CoT's inherent trade-off: while its flexibility allows for creative and adaptive reasoning, this same flexibility can lead to unstructured exploration that fails to maintain logical consistency across complex reasoning chains.

Recent work has explored decomposition-based approaches that break multi-hop questions into manageable single-hop questions (Khot et al., 2023; Zhou et al., 2023; Min et al., 2019). However, even with decomposition, critical gaps remain: models struggle to generate high-quality decompositions without supervision (Patel et al., 2022; Wolfson et al., 2020), fail at reliable fact composition across steps (Press et al., 2023), and lose track of intermediate state in long reasoning chains (Yen et al., 2024; Liu et al., 2024). These failures suggest the need for a more principled reasoning framework that can enforce structure while maintaining flexibility.

In contrast to natural-language reasoning pioneered by CoT, the structured reasoning paradigm has been extensively studied for decades in artificial intelligence and logic programming (Kowalski and

Smoliar, 1982; Russell et al., 1995; McCarthy et al., 1960; Simon and Newell, 1971). Prolog, a declarative programming language explicitly designed for structured reasoning tasks, exemplifies this approach through its systematic query resolution mechanisms and logical rule-based inference (Robinson, 1965; Kowalski and Smoliar, 1982; Clocksin and Mellish, 2003). While Prolog's rigid structure ensures logical consistency, it lacks the flexibility to handle ambiguous natural language, cannot easily incorporate unstructured text from documents, and requires precise logical formulations that may not capture the nuanced reasoning needed for real-world questions.

Recognizing that Prolog and CoT possess complementary strengths and weaknesses, we introduce **Prolog-Initialized Chain-of-Thought (π-CoT)**, a novel prompting strategy that combines the structural rigor of logic programming with the contextual flexibility of natural language reasoning. Our approach begins by algorithmically reformulating complex multi-hop reasoning questions into equivalent Prolog queries, where each query is deliberately decomposed into a sequence of single-hop sub-queries. These sub-queries are then resolved systematically: each is translated into natural language and posed to a RAG (Lewis et al., 2020; Gao et al., 2023) or in-context CoT system, which retrieves relevant documents and generates answers. The resulting answers are translated back into Prolog facts and incorporated into the evolving knowledge base.

The key insight underlying π-CoT is that by structuring the reasoning process through Prolog's query resolution mechanism, we ensure that the retrieved context remains highly relevant. Rather than allowing the model to freely explore the reasoning space, potentially losing track of relevant information or pursuing irrelevant tangents, (Yao et al., 2023; Dziri et al., 2023), our approach maintains a structured trajectory that systematically builds toward the final answer. At the completion of the Prolog resolution process, we concatenate the original question, all retrieved documents, and the structured Prolog derivation to create a comprehensive context that initializes the final CoT reasoning step.

Through extensive experimental evaluation, we demonstrate that π-CoT significantly outperforms traditional RAG and in-context systems on multi-hop question-answering (QA) benchmarks, including HotpotQA, 2WikiMultiHopQA, MuSiQue, and PhantomWiki. Our results suggest that the principled integration of symbolic reasoning structures with neural language models offers a promising direction for developing more reliable and interpretable reasoning systems.

2 Related Works

Decomposition for multi-hop question-answering. Breaking down a complex problem into smaller, manageable parts is a common technique in LLM prompting (Zhou et al., 2023; Khot et al., 2023; Wei et al., 2022). For open-domain QA, Press et al. (2023) prompt the model to generate follow-up questions, and Trivedi et al. (2023) take each new sentence in a CoT as input to the retriever. Importantly, the language model decomposes the question in natural-language steps. Recent works have also explored the use of *explicit plans*, usually in the form of Python programs (Surís et al., 2023; Khattab et al., 2022). While we do not provide a direct comparison due to different model sizes and/or retrieval setups, Monte Carlo Tree Search (Tran et al., 2024), test-time scaling (Wang et al., 2025), and reinforcement learning methods (Li et al., 2025; Jin et al., 2025a; Song et al., 2025) are emerging as promising approaches to open-domain QA.

Use of logic programming languages with LLMs. Weber et al. (2019) propose a weak unification strategy in Prolog based on semantic similarity. However, their approach requires training and uses pre-defined predicates extracted from the training text. Wu and Liu (2025) translate natural language questions to Prolog queries using Prolog definite clause grammar (DCG) parsing, but rely on access to a pre-determined knowledge base. The method closest to our work is that of Chen et al. (2019). They train an LSTM "programmer" to generate programs, which are executed using a BERT-based "reader" to produce answers. In this work, we contribute a training-free strategy and demonstrate its effectiveness with recent LLMs like Llama-3.3-70B-Instruct and Deepseek-R1-Distill-Qwen-32B, using both sparse and dense retrievers. The results of Chen et al. (2019) are also limited by the strictness of a pure Prolog execution.

Fact extraction and summarization. Extracting knowledge graph triples from unstructured text is a classical problem in NLP, also known as open information extraction (OpenIE) (Angeli et al., 2015; Pei et al., 2023; Zhou et al., 2022). To enhance conventional retrieval techniques, LightRAG

(Guo et al., 2024) and HippoRAG (Jimenez Gutierrez et al., 2024; Gutiérrez et al., 2025) demonstrate the effectiveness of fact extraction and GraphRAG (Edge et al., 2024) and RAPTOR (Sarthi et al., 2024) propose methods for clustering and summarization. Among these methods, HippoRAG 2 from Gutiérrez et al. (2025) performs the best on HotpotQA, 2WikiMultiHopQA, and MuSiQue. We provide a direct comparison of our method to HippoRAG 2 in the results section below.

3 PRELIMINARIES

Dating back to the 1970s, Prolog is a powerful way to represent factual knowledge and perform logical inference (Colmerauer and Roussel, 1996; Russell et al., 1995; Sterling and Shapiro, 1994). Solutions in Prolog are verifiable and compositional, making it particularly well-suited for multi-hop question answering where intermediate steps must be chained reliably.

Representing factual knowledge. Consider the the following English sentence about *Harry Potter*:

"Lockhart was a Defense against the Dark Arts teacher at Hogwarts."

This statement can be represented as the following Prolog **fact**:

```
DADA_teacher("Lockhart", "Hogwarts").
```

In Prolog, `DADA_teacher` is a **predicate** and "Lockhart" and "Hogwarts" are **constants**. A large amount of real-world knowledge can be encoded in this structured, relational form. In fact, as of early 2025, Wikidata contained over 1.65 billion such knowledge triples.[1] Throughout the paper, we refer to a collection of Prolog facts as a **knowledge base**.

Multi-hop questions as Prolog queries. Prolog doesn't just store facts—it also allows us to query them. A **Prolog query** comprises one or more predicates with unassigned variables and asks whether any satisfying variable assignment exists. Consider the following example:

"Who is the wife of the Defense against the Dark Arts teacher at Hogwarts who was also a werewolf?" (1)

One way of answering this question involves first identifying all the Defense against the Dark Arts teachers at Hogwarts, then filtering for those who were also werewolves, and finally retrieving the wives of the selected people. Each step depends on resolving the previous step(s), and the intermediate space of possible answers can be large. (Note that there are seven Defense against the Dark Arts teachers at Hogwarts, so a language model would have to consider seven separate reasoning paths to answer the question.) A concise way of expressing question (1) is the following Prolog query:

```
DADA_teacher(X, "Hogwarts"),
werewolf(X),
wife(X, Y),
```
(2)

with `Y` representing the answer. Translating questions in natural language to structured queries is a classical problem in the community (Zelle and Mooney, 1996; Zettlemoyer and Collins, 2012).

Assuming we have a pre-populated knowledge base, the answer is obtained by directly executing the query. For example, a knowledge base containing the facts `DADA_teacher("Lockhart", "Hogwarts")`, `werewolf("Lupin")`, and `wife("Lupin", "Tonks")` yields the following solution to query (2):

```
X = "Lupin",
Y = "Tonks".
```

Complex reasoning. While several implementations of the Prolog programming language are available, we use SWI-Prolog implementation (Wielemaker et al., 2012) due to its inclusion of advanced features beyond standard Prolog. For example, the question, "How many Defense against the Dark Arts teachers have been at Hogwarts?" can be mapped to the Prolog query, `aggregate_all(count, DADA_teacher(X, "Hogwarts"), Y)`, with `Y` representing the desired numerical answer. Here, the `aggregate_all` predicate is built into SWI-Prolog. Other built-in predicates provide functionality for arithmetic and if-then-else logic.

[1] https://en.wikipedia.org/wiki/Wikidata

4 METHOD: π-COT — PROLOG-INITIALIZED CHAIN-OF-THOUGHT

While Prolog provides a precise and verifiable framework for multi-hop reasoning, it assumes access to a structured database, which can be difficult to obtain in practice. On the other hand, LLMs can adeptly retrieve and extract information from unstructured text, but cannot guarantee logical consistency across reasoning steps. Our method, π-CoT, combines the two: it uses Prolog to guide symbolic execution (see Fig. 1, left) and LLMs to interface with the unstructured text. π-CoT augments vanilla CoT by using passages and notes resulting from symbolic execution to initialize the prompt of an LLM for chain-of-thought reasoning (see Fig. 1, right). With this structure, π-CoT improves logical coherence while operating over unstructured inputs. Fig. 1 provides a high-level overview of our method.

To acquire the Prolog-guided execution trace, π-CoT proceeds as follows. First, given a question in natural language, we prompt the LLM to generate a structured **Prolog query**,

$$Q = (q_1, q_2, \ldots, q_T), \tag{3}$$

where each **sub-query** q_i represents an intermediate step in solving the original multi-hop question. We provide the query generation prompt in Sec. A.1. We also instruct the model to generate the **target variable** τ corresponding to the final answer. We use T to denote the total number of steps (or "hops") in the question. For the sake of illustration, the example question (1) yields the Prolog query (2) with $T = 3$ and target variable $\tau = \texttt{Y}$ (see Fig. 1, top left).

As we process each sub-query step-by-step, we simultaneously update our knowledge by inserting facts into a **knowledge base**. We also define the **state** S_t at a certain step t to be the set of variable assignments at step t. At each step, a sub-query q_t is resolved by a `SLICE` module (detailed in §4.1), which takes the current state S_{t-1}, and updates the state with S_t. Details of chaining the `SLICE` modules are discussed in §4.2. After reaching the final state S_{final}, we collect intermediate outputs from each `SLICE` module and use them to initialize the CoT reasoning before the model outputs the final answer (see §4.3).

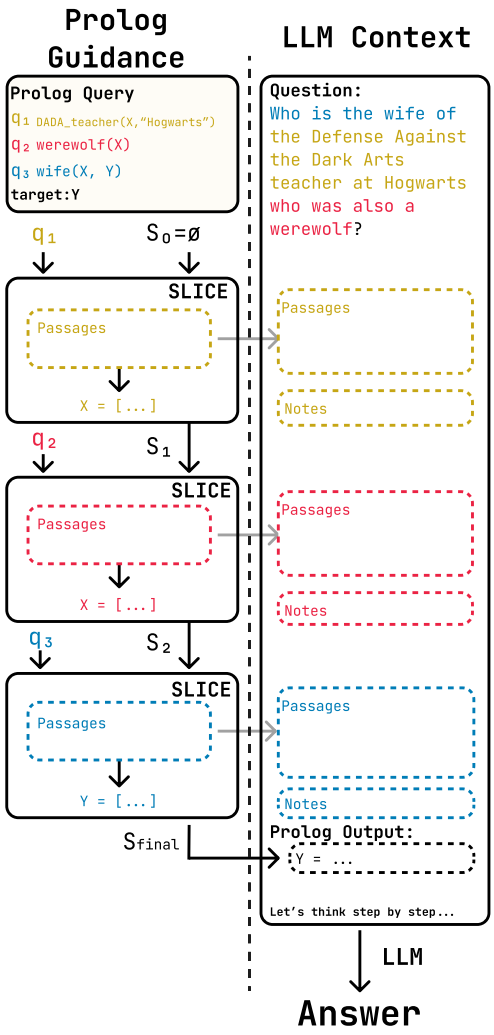

Figure 1: **Overview of π-CoT.** Left: π-CoT follows the guidance from an LLM generated Prolog query, and resolves each sub-query q_t with `SLICE`. `SLICE` updates the state S_{t-1} to S_t at each step, iteratively solving unknown variables in the Prolog query, until all the variables are solved after the final step. Right: Intermediate outputs from `SLICE` modules (passages, notes) and the target variable assignment from the final state are used as initialization of the CoT reasoning for the LLM.

4.1 SINGLE-STEP EXECUTION WITH `SLICE`

We now introduce our procedure for resolving each sub-query with `SLICE` (**S**ingle-step **L**ogical **I**nference with **C**ontextual **E**vidence). `SLICE` bridges symbolic execution and natural language

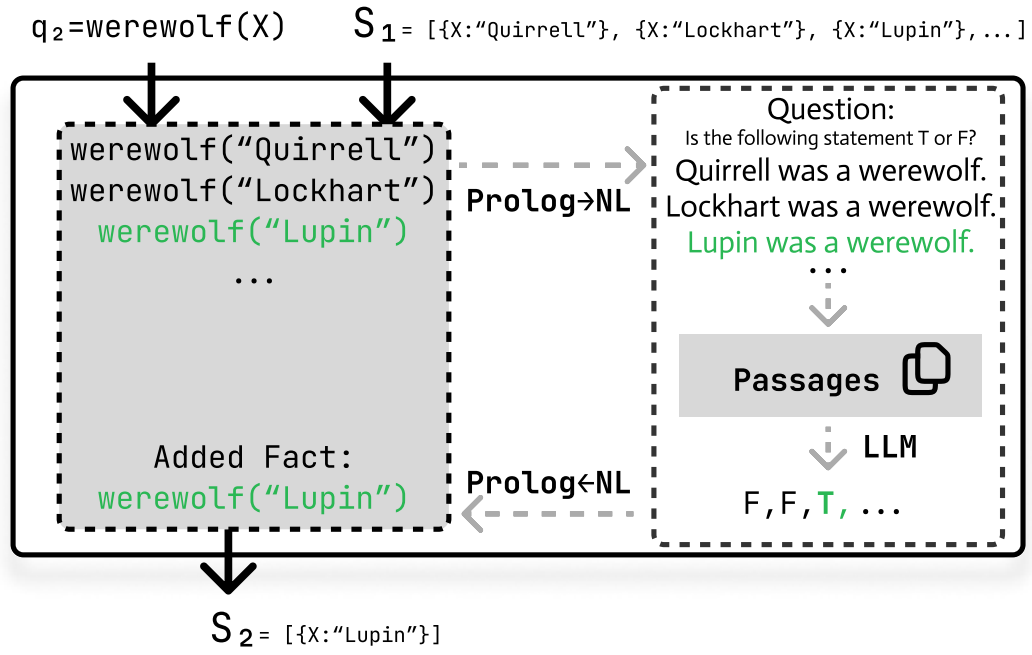

Figure 2: **a SLICE module for fact verification in the RAG setting.** At step $t = 2$, the module takes in the previous state S_1 containing variable assignments, the current sub-query q_2, and the corpus $\mathcal{C}$ (not shown) as inputs and outputs S_2. Only the Prolog fact corresponding to the correct statement is added to the knowledge base (green).

understanding: it grounds one Prolog sub-query at a time by gathering relevant text and updating both the knowledge base and a running notepad of natural language notes.

Inputs. At each step t, SLICE takes as input:

- The t-th Prolog sub-query, q_t;
- The state from the previous step, S_{t-1};
- A corpus of unstructured text, $\mathcal{C}$.

We propose a simple training-free method for translating between **Prolog and natural language (NL)**. We start by assigning values to as many variables in q_t using the set of variable assignments, S_{t-1}. In Fig. 2 (top left), only X has assigned values in S_1, and we proceed by substituting X in q_1 with "Quirrell," "Lockhart," "Lupin," etc. A challenge with using the Prolog sub-query q_t directly is that it losses semantic meaning about the original question: only by looking at the original question do we know that the predicate werewolf(X) means "X was a werewolf" or that the predicate DADA_teacher symbolizes Defense against the Dark Arts teacher. To solve this issue, we prompt the LLM to generate a **definition** for each q_i in addition to generating the Prolog query (3). We provide the definitions generation prompt in Sec. A.1. Specifically, each definition comprises:

- A **question template**, which maps q_i to a natural-language question (e.g., "Who were the Defense against the Dark Arts teachers at Hogwarts?").
- A **statement template**, which maps an instantiated q_i to a factual claim (e.g., "Lockhart was a Defense against the Dark Arts teacher at Hogwarts.").

The question template is used for *fact extraction* and the statement template is used for *fact verification*. We describe these two sub-routines as follows:

Fact extraction. When q_t introduces a new unassigned variable, SLICE uses the question template to map q_t to a natural-language question (see **Prolog→NL** in Fig. 2). Next, we call the LLM using

chain-of-thought prompting to answer this single-hop question (see **LLM** in Fig. 2). We provide the LLM prompt in Sec. A.2. In this work, we consider two strategies to retrieve the relevant evidence for the single-hop question:

(S1) In the **RAG setting**, we use an external retriever to obtain the top-k passages from $\mathcal{C}$.

(S2) In the **in-context setting**, all passages are provided in full.

Using its innate reading comprehension abilities, the LLM locates the answer to the question from the provided passages and responds with (potentially) multiple answers (e.g., Quirrell, Lockhart, Lupin). These answer are parsed back into Prolog facts (see **Prolog**←**NL** in Fig. 2) and added to the knowledge base.

Fact verification. When all variables in q_t can be assigned values from S_{t-1}, we must check a fact. `SLICE` uses the statement template to generate an entailment question (Bowman et al., 2015). For example, `werewolf("Quirrell")` maps to the question, "Is the following statement true or false? Quirell was a werewolf." Again, we call an LLM to answer this question (see **LLM** in Fig. 2), leveraging S1 and S2 in the same way as during fact extraction. A statement that evaluates to true is added to the knowledge base as a fact; a statement that evaluates to false is simply ignored (and nothing is added to knowledge base). In our running example, the only fact added to the knowledge base at step $t = 1$ is `werewolf("Lupin")` (see green text in Fig. 2).

Output. Finally, the output of `SLICE` is the updated set of partial solutions, which we obtain by querying the knowledge base with the query $(q_1, \ldots, q_t)$. We denote the result of this query S_t.

4.2 SLICE CHAINING

To execute the full Prolog query Q, we start with an empty state $S_0 = \emptyset$ and sequentially apply `SLICE`:

$$S_t = \texttt{SLICE}(q_t, S_{t-1}, \mathcal{C}) \quad \text{for} \quad t = 1, 2, \ldots, T.$$

After executing the current step and updating the state with `SLICE`, we instantiate the next query with the resolved variables and solve it again with `SLICE`. This iterative execution produces our final state $S_{\text{final}} = S_T$ with all variables resolved. The answer to the original question corresponds to the value of the target variable τ in S_{final}. In our running example, $S_{\text{final}} = [\{\texttt{Y} : \texttt{"Tonks"}\}]$, yielding `"Tonks"` as the final answer.

Beyond the final answer, this process generates three key components for the subsequent reasoning phase:

- A collection of retrieved passages from each `SLICE` execution (in the RAG setting);
- A grounded knowledge base containing all corpus-derived facts necessary to derive the answer;
- Natural language notes that verbalize each Prolog fact using statement templates.

4.3 COMBINING SYMBOLIC AND NATURAL LANGUAGE REASONING

Having completed the structured Prolog resolution, we now leverage its outputs to initialize a final CoT reasoning step (see Fig. 1, right). We construct a comprehensive prompt containing: (i) the natural language notes derived from Prolog facts via statement templates, providing a structured reasoning trace; (ii) the Prolog-derived answer, serving as a strong reasoning anchor; and (iii) all retrieved passages from the sub-query executions; and (iv) all retrieved passages from the original question, supplying the necessary factual context. This rich initialization guides the LLM to produce a coherent final answer that benefits from both the logical rigor of the Prolog execution and the natural language understanding capabilities of the LLM. By grounding the CoT reasoning in this structured context, we prevent the model from diverging into irrelevant reasoning paths while maintaining the flexibility to generate natural, contextually appropriate responses.

5 MAIN RESULTS

We consider two settings: **(1) open-domain question-answering**, when the corpus is too large to fit within the model's context window, and **(2) in-context question-answering**, when the corpus does

Table 1: **Accuracy of RAG methods on open-domain QA.** We report mean ± 1 standard error for exact match (EM) and F1 score across 500 questions from each dataset. For each dataset and metric pair, we perform a repeated measures ANOVA followed by Tukey BSD with $\alpha = 0.05$ to test significance of paired differences between methods. Bold indicates that no other method performs significantly better.

	HotpotQA		2WikiMultiHopQA		MuSiQue	
Method	EM ↑	F1 ↑	EM ↑	F1 ↑	EM ↑	F1 ↑
Standard RAG	**38.8 ± 2.2**	**52.6 ± 2.0**	37.2 ± 2.2	40.4 ± 2.1	11.0 ± 1.4	18.1 ± 1.5
Self-Ask	19.2 ± 1.8	28.0 ± 1.8	15.8 ± 1.6	21.8 ± 1.7	5.6 ± 1.0	9.1 ± 1.2
IRCoT	**40.4 ± 2.2**	**52.9 ± 2.0**	32.4 ± 2.1	42.5 ± 2.0	**17.6 ± 1.7**	**24.5 ± 1.8**
Memento (Ours)	**42.0 ± 2.2**	**59.1 ± 1.9**	**49.4 ± 2.2**	**57.5 ± 2.1**	**15.2 ± 1.6**	**25.7 ± 1.7**

Table 2: **Efficiency of RAG methods on open-domain QA.** We report mean ± 1 standard error number of calls to the retriever (BM25) across 500 questions from each dataset. Bold indicates the method with the lowest number of calls.

Method	**HotpotQA**	**2WikiMultiHopQA**	**MuSiQue**
Self-Ask	3.36 ± 0.04	3.44 ± 0.04	**3.29 ± 0.05**
IRCoT	**3.07 ± 0.03**	3.47 ± 0.03	3.51 ± 0.03
Memento (Ours)	3.82 ± 0.22	**3.14 ± 0.04**	4.42 ± 0.17

fit within the model's context window. The open-domain and in-context QA settings are also known as the *fullwiki* and *distractor* settings in the literature (Yang et al., 2018). To overcome the context limitations in the open-domain QA setting, we use retrieval augmented generation (RAG) to first fetch relevant passages before generation. Since π-CoT is a prompting method, we require a strong instruction-tuned model and employ the Llama-3.3-70B-Instruct model from Grattafiori et al. (2024). We also provide results utilizing the Deepseek-R1-Distill-Qwen-32B model from (Guo et al., 2025) in the in-context question-answering setting.

5.1 Open-Domain Question-Answering

We evaluate π-CoT on three multi-hop QA datasets: (1) HotpotQA from Yang et al. (2018), (2) 2WikiMultiHopQA from Ho et al. (2020), and (3) MuSiQue from Trivedi et al. (2022). Since these datasets are curated from Wikipedia, we can assess Prolog's effectiveness in handling real-world knowledge. For our retrieval setup, we use the preprocessed December 18, 2020 corpus from FlashRAG (Jin et al., 2025b), which contains 20M chunks each of size 100 words, and use BM25 (Robertson et al., 2009) as our retriever. We provide supplementary experiment details in Sec. B.1.

Tab. 1 compares π-CoT to standard RAG and two RAG baselines that also rely on decomposition to handle multi-hop reasoning: **(1) Self-Ask** from Asai et al. (2023) and **(2) IRCoT** from (Trivedi et al., 2023). Among all training-free methods in the FlashRAG repository, IRCoT was the top-performing training-free method on HotpotQA and the second top-performing method on 2WikiMultiHopQA at the time of writing. On HotpotQA, standard RAG, IRCoT, and π-CoT are comparable in terms of accuracy, with neither method significantly outperforming the other two as determined by a Tukey BSD test with $\alpha = 0.05$. On 2WikiMultiHopQA, π-CoT significantly outperforms all other methods in terms of exact match and F1 score. On MuSiQue, IRCoT and π-CoT achieve comparable accuracy. Despite requiring only a single retriever call, standard RAG achieves surprisingly competitive accuracy on HotpotQA. Min et al. (2019); Chen and Durrett (2019) reveal that multi-hop reasoning is not required for many examples in HotpotQA, possibly explaining our findings. Notably, we find that π-CoT never performs worse than standard RAG, and does significantly better in the case of 2WikiMultiHopQA and MuSiQue. In terms of efficiency, π-CoT uses a similar number of retriever calls as IRCoT and Self-Ask, as shown by Tab. 2.

Table 3: **Comparison to the state-of-the-art OpenIE method.** We report exact match (EM) and F1 on 1000 random samples from the splits provided by Gutiérrez et al. (2025, Tab. 2). We use Llama-3.3-70B-Instruct for generation and NV-Embed-v2 for embedding passages with $k = 5$ per query.

	HotpotQA		2WikiMultiHopQA		MuSiQue	
Method	EM ↑	F1 ↑	EM ↑	F1 ↑	EM ↑	F1 ↑
Standard RAG	61.2	74.5	58.3	63.2	34.9	44.8
HippoRAG 2 (Gutiérrez et al., 2025)	**62.6**	75.3	65.5	72.0	37.6	49.5
π-CoT (Ours)	60.3	**76.8**	**71.1**	**79.6**	**38.5**	**56.2**

Next, we compare π-CoT to **HippoRAG 2**, an OpenIE-augmented retrieval method that outperforms GraphRAG (Edge et al., 2024), RAPTOR (Sarthi et al., 2024), and LightRAG (Guo et al., 2024) on multi-hop QA. We follow the experimental setup of Gutiérrez et al. (2025, Tab. 2), which uses the NV-Embed-v2 embedding model of Lee et al. (2024) for retrieval. Instead of using full Wikipedia as the corpus, this experiment uses 9811 passages for HotpotQA, 6119 passages for 2WikiMultiHopQA, and 11656 passages for MuSiQue. Tab. 3 reports mean exact match and F1 score on the provided 1000 samples for each dataset. These results show π-CoT outperforming both Standard RAG and HippoRAG 2, demonstrating that offline fact extraction is not necessary for good accuracy on multi-hop question-answering tasks.

5.2 In-Context Question-Answering

	HotpotQA		2WikiMultiHopQA		MuSiQue	
Method	EM ↑	F1 ↑	EM ↑	F1 ↑	EM ↑	F1 ↑
Llama-3.3-70B-Instruct						
CoT	62.4 ± 2.2	77.4 ± 1.6	76.4 ± 1.9	83.9 ± 1.5	51.2 ± 2.2	63.7 ± 1.9
π-CoT (Ours)	58.0 ± 2.2	77.7 ± 1.5	72.8 ± 2.0	82.5 ± 1.5	46.4 ± 2.2	63.1 ± 1.9
DeepSeek-R1-Distill-Qwen-32B						
CoT	57.0 ± 2.2	74.2 ± 1.7	75.2 ± 1.9	82.9 ± 1.6	45.8 ± 2.2	57.3 ± 2.0
π-CoT (Ours)	56.8 ± 2.2	75.2 ± 1.6	74.6 ± 1.9	84.2 ± 1.5	46.0 ± 2.2	59.6 ± 2.0

(a) Real-world (Wikipedia-based) multi-hop QA datasets

	PW-S		PW-M	
Method	EM ↑	F1 ↑	EM ↑	F1 ↑
Llama-3.3-70B-Instruct				
CoT	52.8 ± 1.3	71.9 ± 1.0	27.5 ± 1.2	41.7 ± 1.1
π-CoT (Ours)	**78.8 ± 1.1**	**91.4 ± 0.6**	**31.1 ± 1.2**	**56.9 ± 1.0**
DeepSeek-R1-Distill-Qwen-32B				
CoT	54.4 ± 1.3	75.6 ± 0.9	17.5 ± 1.0	28.0 ± 1.0
π-CoT (Ours)	**82.7 ± 1.0**	**88.2 ± 0.8**	18.4 ± 1.0	**31.1 ± 1.0**

(b) Synthetic multi-hop QA datasets

Table 4: **Accuracy on in-context QA.** We report exact match (EM) and F1 score on HotpotQA (HP), 2WikiMultiHopQA (2Wiki), MuSiQue (MSQ), PhantomWiki with a corpus size of 50 articles (PW-S), and PhantomWiki with a corpus size of 500 articles (PW-M). Bold indicates that π-CoT significantly outperforms CoT ($p < 0.05$). Specifically, we use a paired samples t-test for each combination of dataset and metric.

When all the necessary information fits in the context window of the model, a natural question is *when does formal reasoning (via π-CoT) benefit reasoning in natural-language (via CoT)?*. To investigate this, we employ the distractor variants of HotpotQA, 2WikiMultiHopQA, and MuSiQue, where the gold passages are presented in-context alongside a small number of irrelevant ("distractor") passages. We also include two versions of the PhantomWiki benchmark from Gong et al. (2025): PW-S and PW-M. Unlike the other three datasets that are curated from Wikipedia, PhantomWiki generates challenging multi-hop questions from fictional universes to ensure contamination-free LLM evaluation. We provide supplementary experiment details in Sec. B.2.

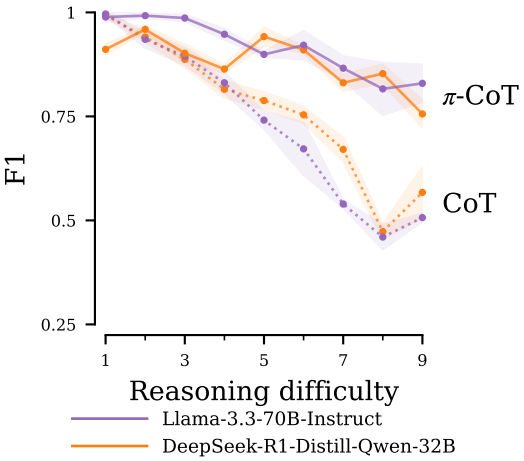

Figure 3: **F1 score vs. difficulty, as measured by number of reasoning steps.** We use the synthetic PW-S benchmark from Gong et al. (2025) and display mean ± 1 standard error. For each model, we evaluate CoT and π-CoT prompting.

Tab. 4(a) shows that on HotpotQA, 2WikiMultiHopQA, and MuSiQue, there is no significant difference in accuracy between CoT and π-CoT. Providing the gold passages to model makes the task considerably easier than considering all of Wikipedia (Min et al., 2019). Thus, Llama-3.3-70B-Instruct with CoT may be hitting a performance ceiling. On PW-S, Llama-3.3-70B-Instruct with CoT achieves an F1 score of 71.9 ± 1.0%. This increases to 91.4 ± 0.6% with π-CoT. On PW-M, which has a corpus 10 times the size of PW-S, the performance of Llama-3.3-70B-Instruct with CoT drops to 41.7 ± 1.1% F1. We posit this drop is mainly due to the inherent challenges of long-context retrieval. On PW-M, π-CoT significantly boosts the F1 score to 56.9 ± 1.0 F1—a relative gain of 36%! In Tab. 4(b), we report similar findings using Deepseek-R1-Distill-Qwen-32B as the language model.

Finally, to understand where the gains on PW-S and PW-M come from, we plot accuracy versus the ground-truth difficulty level that comes associated with each question. Gong et al. (2025) defines this difficulty level as the number of hops required to answer the question. According to Fig. 3, the accuracy of π-CoT and CoT is similar for questions with low difficulty. However, π-CoT diverges from CoT as the difficulty increases. Since higher difficulty questions require traversing more reasoning paths than lower difficulty questions, our results show that π-CoT is better able to keep track of multi-hop, multi-branch reasoning than CoT.

6 Conclusions & Future Work

In this work, we investigate how formal reasoning can guide reasoning in natural language. We introduce π-CoT, a novel prompting strategy that initializes the context of an LLM with the intermediate outputs of Prolog-guided execution. Our results show that even strong LLMs like Llama-3.3-70B-Instruct and Deepseek-R1-Distill-Qwen-32B benefit from being guided through structured reasoning steps, especially on complex, multi-step tasks. More broadly, our work demonstrates the potential of bringing explicit planning and state tracking into language model behavior.

Despite being a purely prompting-based strategy, π-CoT has potential implications for training future language models to serve as agents that can piece together knowledge across large, dynamic corpora. Inspired by DeepSeek R1 (Guo et al., 2025), significant effort has been made to couple reasoning with retrieval using reinforcement learning (Li et al., 2025; Jin et al., 2025a; Song et al., 2025). An interesting future direction is training language models to generate structured queries instead. π-CoT provides a way to execute these queries without a pre-existing database. Some steps in π-CoT may not even require calling an LLM, if the information resides directly in a pre-existing database. Thus, enabling π-CoT to leverage both unstructured and structured data is a natural next step.

Ethics Statement

Our work adheres to the ICLR Code of Ethics, and does not pose any societal, personal, or organizational risks.

Reproducibility Statement

To encourage reproducibility, we use free and open-source software and LLMs. We also include details of our experimental setups in Sec. B. We report sample sizes, standard errors, and random seeds where possible.

References

Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D Manning. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 344–354, 2015. (Cited on page 2.)

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. (Cited on pages 1 and 7.)

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. The snli corpus. 2015. (Cited on page 6.)

Jifan Chen and Greg Durrett. Understanding dataset design choices for multi-hop reasoning. *arXiv preprint arXiv:1904.12106*, 2019. (Cited on page 7.)

Xinyun Chen, Chen Liang, Adams Wei Yu, Denny Zhou, Dawn Song, and Quoc V Le. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *International Conference on Learning Representations*, 2019. (Cited on page 2.)

William F Clocksin and Christopher S Mellish. *Programming in PROLOG*. Springer Science & Business Media, 2003. (Cited on page 2.)

Alain Colmerauer and Philippe Roussel. The birth of prolog. In *History of programming languages—II*, pages 331–367. 1996. (Cited on page 3.)

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, et al. Faith and fate: Limits of transformers on compositionality. *Advances in Neural Information Processing Systems*, 36: 70293–70332, 2023. (Cited on pages 1 and 2.)

Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. From local to global: A graph rag approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024. (Cited on pages 3 and 8.)

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

Albert Gong, Kamilė Stankevičiūtė, Chao Wan, Anmol Kabra, Raphael Thesmar, Johann Lee, Julius Klenke, Carla P Gomes, and Kilian Q Weinberger. Phantomwiki: On-demand datasets for reasoning and retrieval evaluation. In *Forty-second International Conference on Machine Learning*, 2025. (Cited on page 9.)

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. (Cited on page 7.)

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. (Cited on pages 7 and 9.)

Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang. Lightrag: Simple and fast retrieval-augmented generation. *arXiv preprint arXiv:2410.05779*, 2024. (Cited on pages 3 and 8.)

Bernal Jiménez Gutiérrez, Yiheng Shu, Weijian Qi, Sizhe Zhou, and Yu Su. From rag to memory: Non-parametric continual learning for large language models. In *Forty-second International Conference on Machine Learning*, 2025. (Cited on pages 3 and 8.)

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop qa dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, 2020. (Cited on page 7.)

Bernal Jimenez Gutierrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su. Hipporag: Neurobiologically inspired long-term memory for large language models. *Advances in Neural Information Processing Systems*, 37:59532–59569, 2024. (Cited on page 3.)

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025a. (Cited on pages 2 and 9.)

Jiajie Jin, Yutao Zhu, Zhicheng Dou, Guanting Dong, Xinyu Yang, Chenghao Zhang, Tong Zhao, Zhao Yang, and Ji-Rong Wen. Flashrag: A modular toolkit for efficient retrieval-augmented generation research. In *Companion Proceedings of the ACM on Web Conference 2025*, pages 737–740, 2025b. (Cited on page 7.)

Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive nlp. *arXiv preprint arXiv:2212.14024*, 2022. (Cited on page 2.)

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*, 2023. (Cited on pages 1 and 2.)

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35: 22199–22213, 2022. (Cited on page 1.)

Robert Kowalski and Steve Smoliar. Logic for problem solving. *ACM SIGSOFT Software Engineering Notes*, 7(2):61–62, 1982. (Cited on pages 1 and 2.)

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023. (Cited on page 15.)

Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Nv-embed: Improved techniques for training llms as generalist embedding models. In *The Thirteenth International Conference on Learning Representations*, 2024. (Cited on page 8.)

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020. (Cited on page 2.)

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. *arXiv preprint arXiv:2501.05366*, 2025. (Cited on pages 2 and 9.)

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024. (Cited on page 1.)

Robert Lo, Abishek Sridhar, Frank F Xu, Hao Zhu, and Shuyan Zhou. Hierarchical prompting assists large language model on web navigation. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 10217–10244, 2023. (Cited on page 1.)

John McCarthy et al. *Programs with common sense*. RLE and MIT computation center Cambridge, MA, USA, 1960. (Cited on page 2.)

Sewon Min, Eric Wallace, Sameer Singh, Matt Gardner, Hannaneh Hajishirzi, and Luke Zettlemoyer. Compositional questions do not necessitate multi-hop reasoning. *arXiv preprint arXiv:1906.02900*, 2019. (Cited on pages 1, 7, and 9.)

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*, pages 4553–4569, 2022. (Cited on page 1.)

Kevin Pei, Ishan Jindal, Kevin Chen-Chuan Chang, ChengXiang Zhai, and Yunyao Li. When to use what: An in-depth comparative empirical analysis of openie systems for downstream applications. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 929–949, 2023. (Cited on page 2.)

Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, 2023. (Cited on pages 1 and 2.)

Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009. (Cited on page 7.)

John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)*, 12(1):23–41, 1965. (Cited on page 2.)

Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25(27):79–80, 1995. (Cited on pages 2 and 3.)

Parth Sarthi, Salman Abdullah, Aditi Tuli, Shubh Khanna, Anna Goldie, and Christopher D Manning. Raptor: Recursive abstractive processing for tree-organized retrieval. In *The Twelfth International Conference on Learning Representations*, 2024. (Cited on pages 3 and 8.)

Herbert A Simon and Allen Newell. Human problem solving: The state of the theory in 1970. *American psychologist*, 26(2):145, 1971. (Cited on page 2.)

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *arXiv preprint arXiv:2503.05592*, 2025. (Cited on pages 2 and 9.)

Leon Sterling and Ehud Y Shapiro. *The art of Prolog: advanced programming techniques*. MIT press, 1994. (Cited on page 3.)

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11888–11898, 2023. (Cited on page 2.)

Hieu Tran, Zonghai Yao, Junda Wang, Yifan Zhang, Zhichao Yang, and Hong Yu. Rare: Retrieval-augmented reasoning enhancement for large language models. *arXiv preprint arXiv:2412.02830*, 2024. (Cited on page 2.)

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. MuSiQue: Multihop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554, 2022. (Cited on page 7.)

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10014–10037, 2023. (Cited on pages 2, 7, and 15.)

Liang Wang, Haonan Chen, Nan Yang, Xiaolong Huang, Zhicheng Dou, and Furu Wei. Chain-of-retrieval augmented generation. *ArXiv*, abs/2501.14342, 2025. URL https://api.semanticscholar.org/CorpusID:275906944. (Cited on page 2.)

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022. (Cited on page 1.)

Leon Weber, Pasquale Minervini, Jannes Münchmeyer, Ulf Leser, and Tim Rocktäschel. Nlprolog: Reasoning with weak unification for question answering in natural language. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2019. (Cited on page 2.)

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. *Advances in neural information processing systems*, 35:24824–24837, 2022. (Cited on pages 1 and 2.)

Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. Swi-prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012. (Cited on page 3.)

Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. Break it down: A question understanding benchmark. *Transactions of the Association for Computational Linguistics*, 8:183–198, 2020. (Cited on page 1.)

Katherine Wu and Yanhong A Liu. Lp-lm: No hallucinations in question answering with logic programming. *arXiv preprint arXiv:2502.09212*, 2025. (Cited on page 2.)

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, 2018. (Cited on page 7.)

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. (Cited on pages 1 and 2.)

Howard Yen, Tianyu Gao, Minmin Hou, Ke Ding, Daniel Fleischer, Peter Izsak, Moshe Wasserblat, and Danqi Chen. Helmet: How to evaluate long-context language models effectively and thoroughly. *arXiv preprint arXiv:2410.02694*, 2024. (Cited on page 1.)

John M Zelle and Raymond J Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055, 1996. (Cited on page 3.)

Luke S Zettlemoyer and Michael Collins. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*, 2012. (Cited on page 3.)

Zhuosheng Zhang, Aston Zhang, Mu Li, and Alex Smola. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022. (Cited on page 1.)

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, et al. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*, 2023. (Cited on pages 1 and 2.)

Shaowen Zhou, Bowen Yu, Aixin Sun, Cheng Long, Jingyang Li, Haiyang Yu, Jian Sun, and Yongbin Li. A survey on neural open information extraction: Current status and future directions. *arXiv preprint arXiv:2205.11725*, 2022. (Cited on page 2.)

A Prompt Templates

A.1 Prolog Query and Definitions Generation

We use the following prompt template for the Prolog query generation of Sec. 4:

```
You will be provided a question. Your goal is to devise a
Prolog query to answer this question. Your response must end in
"**Query:** <query>\n**Target:** <target>\n**Definition:**
<definition>", where <query> is a Prolog query that when
executed, will yield the answer to the question, <target>
is the target variable in the Prolog query to be returned
as the final answer, and <definition> defines the semantic
meaning of predicates in the Prolog query.

Here are some examples:
(START OF EXAMPLES)
{examples}
(END OF EXAMPLES)

Question: {question}
Answer:
```

To form the prompt, `examples` is replaced with few-shot examples specific to each dataset and `question` is replaced with the natural-language question.

A.2 Chain-of-Thought Fact Extraction and Verification

```
You are given the following evidence:
(BEGIN EVIDENCE)
{{evidence}}
(END EVIDENCE)

You will be provided a question. Your response must end in the
following sentence: The answer is <answer>.
Here, <answer> must be either a single answer or a
list of answers separated by '{constants.answer_sep}'.

Here are some examples:
(START OF EXAMPLES)
{{examples}}
(END OF EXAMPLES)

Question: {{question}}
Answer:
```

To form the prompt, `evidence` is replaced by relevant passages, `examples` is replaced with few-shot examples specific to each dataset, and `question` is replaced with the natural-language question (in the case of fact extraction) or entailment question (in the case of fact verification). To ensure that the answer can be added to the Prolog database, our few-shot examples format `<answer>` as Prolog literals.

B Supplementary Experiment Details

B.1 FullWiki experiment details

Retrieval setup. We use the `wiki18_100w` corpus from https://huggingface.co/datasets/RUC-NLPIR/FlashRAG_datasets and use the code from https://github.

com/RUC-NLPIR/FlashRAG to build our BM25 index. We allow $k = 14$, $k = 16$, and $k = 8$ chunks per retrieval call for HotpotQA, 2WikiMultiHopQA, and MuSiQue, respectively.

Baseline implementations. For standard RAG, we use the Python implementation of `CoTRAGAgent` from https://github.com/kilian-group/phantom-wiki and write few-shot examples for each dataset. For Self-Ask, we use the Python implementation and few-shot examples from https://github.com/RUC-NLPIR/FlashRAG. For IRCoT, we use the Python implementation from https://github.com/RUC-NLPIR/FlashRAG and the GPT3 (`code-davincii-002`) few-shot examples from https://github.com/StonyBrookNLP/ircot (see also (Trivedi et al., 2023, App. G)). We set the maximum iterations for Self-Ask and IRCoT to be 4.

LLM configuration. We run Llama-3.3-70B-Instruct on 8 A6000s using vLLM (Kwon et al., 2023) and use greedy decoding with maximum generation tokens 4096. We use the full 128K context length.

B.2 Distractor Experiment Details

LLM configuration. For the Llama-3.3-70B-Instruct results, we use vLLM running on 8 A6000s and use greedy decoding with maximum generation tokens 4096. For the Deepseek-R1-Distill-Qwen-32B results, we use vLLM running on 6 A6000s and use sampling temperature 0.6, top-p 0.95, and max generation tokens 16384. We use the full 128K context length for both models.

PhantomWiki dataset. For the experiment of Tab. 4(b), we use the code at https://github.com/kilian-group/phantom-wiki to generate two synethic multi-hop QA datasets. Tab. 5 lists the configurations for PW-S and PW-M. Each dataset has 1500 questions.

Table 5: **Configurations for PhantomWiki dataset generation.**

Parameter	PW-S	PW-M
Question Depth	20	20
Number of family trees	10	100
Max family tree size	50	50
Max family tree depth	20	20
Mode	Easy	Easy
Number of questions per template	10	10
Seeds	1, 2, 3	1, 2 ,3

B.3 LLM Usage Statement

Large language models were used for proofreading, revising, and literature search. All claims and arguments were drafted and verified by the authors.