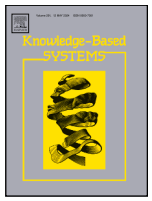
Knowledge-Based Systems 330 (2025) 114526

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys



TreeQA: Enhanced LLM-RAG with logic tree reasoning for reliable and interpretable multi-hop question answering

Xiangrui Zhang [a], Fuyong Zhao [a], Yutian Liu [a], Panfeng Chen [a], Yanhao Wang [b], Xiaohua Wang [a], Dan Ma [a], Huarong Xu [a], Mei Chen [a], Hui Li [a,*]

[a] State Key Laboratory of Public Big Data, College of Computer Science and Technology, Guizhou University, Guiyang, 550025, China
[b] School of Data Science and Engineering, East China Normal University, Shanghai, 200062, China

ARTICLE INFO

Keywords:
Multi-hop question answering
Large language model
Retrieval-augmented generation
Logic tree

ABSTRACT

Multi-Hop Question Answering (MHQA), crucial for complex information retrieval, remains challenging for current Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) systems, which often suffer from hallucination, reliance on incomplete knowledge, and opaque reasoning processes. Existing RAG methods, while beneficial, still struggle with the intricacies of multi-step inference and ensuring verifiable accuracy. This research introduces TreeQA, a novel framework designed to significantly enhance the reliability and interpretability of LLM-RAG systems in MHQA tasks. TreeQA addresses these limitations by decomposing complex multi-hop questions into a hierarchical logic tree of simpler, verifiable sub-questions, integrating evidence from both structured knowledge bases (e.g., Wikidata) and unstructured text (e.g., Wikipedia), and employing an iterative, evidence-based validation and self-correction mechanism at each reasoning step to dynamically rectify errors and prevent their accumulation. Extensive experiments on four benchmark datasets (WebQSP, QALD-en, AdvHotpotQA, and 2WikiMultiHopQA) demonstrate TreeQA's superior performance, achieving Hit@1 scores of 87 %, 57 %, 53 %, and 59 %, respectively, representing improvements of 4 %-12 % over state-of-the-art LLM-RAG methods. These findings highlight the significant impact of structured, verifiable reasoning pathways in developing more robust, accurate, and interpretable knowledge-intensive AI systems, thereby enhancing the practical utility of LLMs in complex reasoning scenarios. Our code is publicly available at https://github.com/ACMISLab/TreeQA.

1. Introduction

Multi-Hop Question Answering (MHQA) [1], which involves extracting and integrating different pieces of information through multi-step reasoning processes, has garnered extensive research attention over the last decade and is regarded as a crucial component in modern Natural Language Processing (NLP) systems. Recently, since Large Language Models (LLMs) have demonstrated strong capabilities in text generation, understanding, and reasoning [2,3], there have been several applications of LLMs to MHQA tasks [4], especially in the open-domain setting [5,6]. However, MHQA still poses a significant challenge for LLMs because their usage can be severely limited by issues such as hallucination, dependence on static knowledge, and opaque reasoning processes [7,8].

Retrieval-Augmented Generation (RAG) aims to improve the reliability of LLMs by incorporating external knowledge [9] and has been introduced to improve LLMs' performance on MHQA tasks [10,11]. However, standard RAG methods often falter during the complex multi-step reasoning process. Vector-based retrieval methods, which rely primarily on text similarity, fail to effectively capture the logical relationships between information. Structured reasoning can be facilitated through graph-based retrieval; however, existing graph-based RAG methods are limited by the completeness of Knowledge Graphs (KGs). Missing information can undermine the credibility of the entire reasoning chain. Such limitations are illustrated in Fig. 1 using a multi-hop question: "*Who was the President of the United States when humans made their third landing on the moon?*" Using standard text retrieval, no relevant information is provided. With graph-based RAG, the entities related to *moon landing* can be identified from the KG. However, since information about the specific *third* landing (i.e., "*Apollo 14*") is not pinpointed, likely due to the incompleteness of the KG, no information is provided.

Hybrid RAG methods, which integrate both text retrieval and KG retrieval, represent a promising direction that leverages the strengths

* Corresponding author.
E-mail addresses: gs.zhangxr23@gzu.edu.cn (X. Zhang), gs.fyzhao24@gzu.edu.cn (F. Zhao), gs.liuyt24@gzu.edu.cn (Y. Liu), pfchen@gzu.edu.cn (P. Chen), yhwang@dase.ecnu.edu.cn (Y. Wang), gzzywxh@hotmail.com (X. Wang), dma@gzu.edu.cn (D. Ma), hrxu@gzu.edu.cn (H. Xu), mchen@gzu.edu.cn (M. Chen), cse.huili@gzu.edu.cn (H. Li).

https://doi.org/10.1016/j.knosys.2025.114526
Received 5 June 2025; Received in revised form 22 August 2025; Accepted 19 September 2025
Available online 27 September 2025

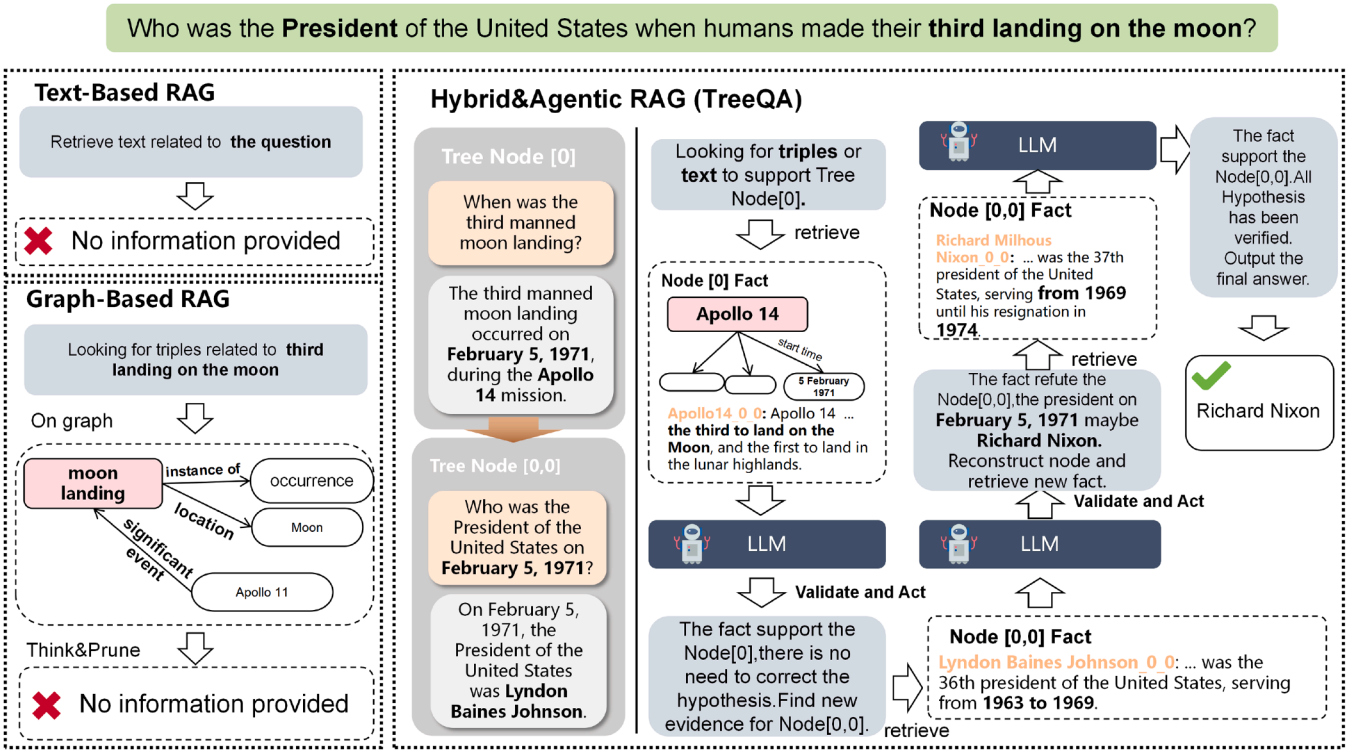


Fig. 1. Comparison of different RAG approaches for answering an exemplar multi-hop question.

of both modalities [12,13]. In hybrid RAG, structured precision can be offered by KG retrieval, while broad context can be provided by text retrieval. However, current hybrid methods often lack effective mechanisms to verify the reasoning with evidence and dynamically correct errors. These approaches still depend on predefined search strategies or require additional model training, which makes it difficult to ensure that the correct reasoning path is reliably and interpretably found and followed across diverse multi-hop questions.

To bridge this gap, we introduce **TreeQA**, a framework designed for reliable and interpretable multi-hop question answering. Specifically, TreeQA systematically decomposes a multi-hop question into a series of simpler, interconnected sub-questions organized in a logic tree structure [14]. Each node in the tree contains a sub-question and a corresponding hypothesis generated by a pre-trained LLM. Crucially, hybrid knowledge sources are consulted – facts are drawn from structured KGs (e.g., Wikidata) and context from unstructured text (e.g., Wikipedia pages) – to validate these hypotheses iteratively. Through this grounding in up-to-date external evidence, hallucination can be significantly reduced. When evidence contradicts a hypothesis, the LLM identifies inconsistency, corrects the hypothesis, and adjusts the logic tree to refine subsequent reasoning. This iterative validation and correction mechanism is key to mitigating cascading errors and enhancing the reliability and interpretability of reasoning. Fig. 1 illustrates the process of TreeQA in answering an example question: "*Who was the President of the United States when humans made their third landing on the moon?* The question is decomposed into several sub-questions ("*When was the third manned moon landing?*", leading to "*Who was the President of the United States on February 5, 1971?*"). Then, an initial hypothesis is generated ("*Lyndon Baines Johnson*") while conflicting evidence is also retrieved (i.e., the facts identifying "*Apollo 14*" as the third landing and "*Richard Nixon*" as president in 1971). qThe hypothesis is then validated against the facts, the reasoning path is corrected, and the correct answer is finally reached ("*Richard Nixon*"). Finally, extensive experiments demonstrate that TreeQA achieves Hit@1 scores of 87 %, 57 %, 53 %, and 59 % on the WebQSP, QALD-en, AdvHotpotQA, and 2WikiMultiHopQA datasets, respectively, achieving improvements of 4–12 % over the state-of-the-art LLM-RAG methods. These results highlight that breaking down reasoning and using iterative evidence-based validation helps build more robust and interpretable knowledge-intensive AI systems.

In summary, the main contributions of this paper are as follows.

- We introduce a logic tree structure to decompose multi-hop questions into simpler sub-questions, addressing the limitations of text-based retrieval methods in accurately handling multi-hop problems.
- We implement an iterative self-correction mechanism to dynamically refine reasoning paths, overcoming the inability of graph-based retrieval methods to optimize inference trajectories, thereby reducing error propagation and model hallucinations for enhanced reliability.
- We show through experiments that our proposed TreeQA framework achieves 4–12 % performance gains over existing hybrid & agentic RAG baselines on benchmark datasets.

Paper organization. The remainder of this paper is structured as follows. Section 2 discusses the existing literature that is relevant to this work. Section 3 details the design and implementation of the TreeQA framework. Section 4 describes the experimental setup and results used to evaluate the performance of TreeQA. Finally, Section 5 concludes the paper and presents possible directions for future work.

2. Related work

2.1. Multi-hop question answering

Multi-Hop Question Answering (MHQA) [1,15] aims to answer natural language questions that require the synthesis of information across multiple pieces of evidence through several reasoning steps. Early methods for MHQA, predating the widespread adoption of LLMs, leveraged structured knowledge graphs or symbolic logic to construct reasoning paths, which exhibited inherent strengths in interpretability but suffered from severe limitations in coverage or flexibility. Specifically, the existing graph-based methods used the KG structure or constructed "text

graphs" from unstructured text for MHQA. Typical KG-based methods include GraftNet [16] and PullNet [17], which identify entities and relations in the question and then perform a path search or subgraph reasoning on the KG. Typical text graph-based methods include PathNet [18], which extracts entity chains across documents to form explicit reasoning paths. The primary advantage of these methods is the interpretability of the reasoning path. However, they are heavily dependent on the completeness and quality of the graph. Another line of methods is based on logic or symbolic reasoning. These approaches aimed to convert natural language questions into executable programs or logical expressions. Semantic parsers, such as Neural Symbolic Machines [19], have been used to generate LISP-style programs for MHQA. Methods for complex Web question answering [20] decomposed questions into a sequence of simpler, searchable sub-questions combined with symbolic operations. These methods also offered high interpretability and logical soundness but struggled with the ambiguity of natural language and often required high-quality labeled data or carefully designed operational primitives. In addition, advances in high-quality datasets such as HotpotQA [21], WebQSP [22], and 2WikiMultiHopQA [23] have also contributed to the development of MHQA methods. They are specifically designed to require multi-step reasoning and often provide supporting facts to facilitate explainable MHQA. They also serve as standard benchmarks to evaluate the performance of MHQA methods.

2.2. LLM-RAG for MHQA

The remarkable capabilities of LLMs in understanding and generating human language, coupled with their vast pre-trained knowledge, have led to their increasing application in MHQA tasks. However, LLMs do not perform well on MHQA tasks that require intricate reasoning over external knowledge. As such, RAG [9] has emerged as a dominant paradigm for grounding LLMs in factual data [24,25]. Nevertheless, existing LLM-RAG approaches still exhibit limitations in handling the complexities of multi-step inference and ensuring the reliability and interpretability of the reasoning process. Next, we summarize several prominent RAG methodologies for MHQA tasks, including text-based, graph-based, and hybrid & agentic RAG approaches.

Text-based RAG. Text-based RAG retrieves relevant text passages from a corpus to provide context for LLM generation [24]. Standard RAG methods typically use vector similarity search to find relevant text chunks. This approach is suitable for answering questions that require information contained within a single passage. However, its effectiveness decreases for multi-hop questions where answers require synthesizing information across multiple, potentially disparate, text segments. Vector similarity often cannot capture the underlying logical or temporal relationships that connect these segments [13]. Consequently, the performance heavily depends on the ability of the retriever to locate all necessary evidence fragments, which becomes increasingly unreliable as the complexity of the question increases.

Graph-based RAG. Graph-based RAG leverages the structure of knowledge graphs (KGs), using entities and relations (triples) to construct reasoning paths. Methods like RoG [26], ToG [27], and PoG [28] search the KG for relevant triple sequences. These approaches can achieve high accuracy, particularly when the KG contains the necessary information and the reasoning path is clear. However, they face significant challenges that limit their real-world applicability and reliability. First, many graph-based methods require pre-identified starting entities linked to the input query. This leads the system to overlook errors in entity linking. However, in practical scenarios, failures in entity linking can cause the entire reasoning path to deviate, resulting in incorrect answers. Second, their performance is susceptible to the completeness of KG. If a crucial triple is missing along the intended reasoning path, the path breaks, often leading to reasoning failure. Although some methods attempt to mitigate this by using the internal knowledge of LLMs to "bridge" these gaps in the KG path, this strategy raises additional concerns about trustworthiness. To more systematically address these challenges, recent works have explored deeper and more flexible integration strategies. Some frameworks like GraphRAG [29] focus on dynamically constructing KGs from unstructured text, thereby alleviating the dependency on pre-existing and often incomplete KGs. Other approaches improve the retrieval process itself. Instead of retrieving simple linear paths, methods like HippoRAG [30] retrieve entire relevant subgraphs in a single pass, providing the LLM with a richer, more interconnected context. Similarly, GRAG [31] is specifically designed to handle networked documents (such as citation networks or knowledge graphs) by retrieving entire textual subgraphs. It employs a divide-and-conquer strategy for efficient subgraph retrieval and integrates this information into the LLM through two complementary views: a "text view" and a "graph view." This allows the model to comprehend both content and topology, enhancing its multi-hop reasoning capabilities on complex data structures. These advanced methods aim to create a more robust and fault-tolerant reasoning process, better handling the complexities of real-world data and queries.

Hybrid & agentic RAG. Recent advancements have pushed beyond simple retrieval, leading to more dynamic frameworks often categorized as agentic RAG. These systems leverage LLMs as autonomous agents that can plan, use tools, and self-reflect to solve complex problems. In the context of MHQA, different agentic frameworks adopt distinct strategies, particularly in how they structure their reasoning plan and perform error correction. One common strategy involves a linear or forward-chaining reasoning process. For instance, the combination of **HippoRAG + IR-CoT** [30,32] features a reasoning agent that iteratively follows a Chain-of-Thought, delegating retrieval tasks to the powerful HippoRAG tool at each step. Similarly, **Chain-of-Knowledge (CoK)** [12] first plans a linear chain of rationales, then acts by retrieving evidence, and finally self-corrects by editing the current rationale. While powerful, these methods are susceptible to early-step errors, as their forward-chaining nature lacks a native mechanism for systematic backtracking. A similar limitation is seen in graph-based agents like **Think-on-Graph 2.0 (ToG-2)** [13], which explores paths on a knowledge graph but struggles to backtrack effectively from a failed reasoning avenue once a path proves unpromising [28]. Another approach focuses on **integrated, real-time reflection**. **Self-RAG** [33] exemplifies this by generating "reflection tokens" to self-critique evidence and its own statements during generation. This allows for fine-grained refinement, but its correction mechanism is primarily limited to selecting from parallel hypotheses rather than fundamentally re-planning a flawed reasoning structure. **Adaptive-RAG** [34] takes a "plan-then-execute" approach, where an agent makes an upfront decision on the reasoning pipeline. However, its agentic role is confined to this initial stage, lacking the ability to dynamically adapt if the chosen path leads to an error. In stark contrast to these approaches, which are constrained by linear plans or limited correction scopes, **TreeQA** proposes a different agentic architecture based on a **hierarchical logic tree**. This structures the agent's plan not as a linear sequence or a graph path, but as a branching tree of verifiable subgoals. This architectural choice enables a more powerful and flexible self-correction mechanism: **sub-tree reconstruction**. When evidence contradicts a hypothesis, the agent is not limited to editing a single step (as in CoK) or choosing an alternative path (as in Self-RAG). Instead, it can invalidate and regenerate an entire logical sub-tree. This provides a structured way to recover from errors by fundamentally altering the reasoning plan, directly addressing the lack of systematic backtracking that limits other forward-chaining agents.

2.3. Logic tree and structured reasoning paradigms

The concept of a "logic tree," often referred to as an "issue tree" or "problem tree" in the literature on strategy consulting and problem

solving, provides a structured approach to decomposing complex problems into smaller and more manageable components. Previous research [14] has highlighted that such structured approaches, akin to those used by top strategy consultants, help overcome common cognitive pitfalls, such as jumping to flawed solutions or ignoring conflicting evidence. They emphasize a step-by-step process of stating, structuring, and solving problems, which aligns with the hierarchical decomposition in logic trees. This structured decomposition ensures that all relevant aspects of a problem are systematically considered and addressed. Similarly, another work [35] discusses the application of issue trees to understand complex, ill-defined, and non-immediate problems. The authors argue that breaking down a complex challenge into smaller parts minimizes the risk of missing critical information, eliminates overlaps, and ensures that key drivers of a problem are identified.

The principle of structured decomposition and iterative refinement is not confined to human problem-solving but has emerged as a cornerstone for building robust and adaptive AI systems. In the field of personalized recommendation, for instance, advanced models implicitly adopt this paradigm to navigate complex information landscapes. The RAKCR framework [36] decomposes the recommendation task by first extracting sentiment from unstructured reviews and then aligning it with a structured knowledge graph. Similarly, GaGNN [37] breaks down the problem into parallel sub-tasks of modeling "group-level" and "individual-level" behaviors. These examples showcase the power of decomposing a complex inference into more focused, evidence-driven subprocesses.

Furthermore, the idea of iterative, evidence-based correction is crucial for systems operating in dynamic environments. The DT3OR framework [38], designed to handle out-of-distribution scenarios in recommendations, exemplifies this. It employs a test-time training mechanism that dynamically adapts the model by iteratively learning from new, unlabeled data. This mirrors TreeQA's self-correction loop, where the system iteratively validates hypotheses against retrieved evidence and refines its reasoning path. While DT3OR performs correction at the model parameter level to adapt to data distribution shifts, it shares the same fundamental goal as TreeQA: to build a system that can dynamically self-correct based on incoming evidence, thereby preventing the accumulation of errors.

In the context of MHQA, the principles underlying these logic/issue trees, as well as adaptive systems, are highly relevant. Decomposing a complex multi-hop question into a series of simpler, interconnected subquestions, which are organized hierarchically, mirrors the core idea of an issue tree. This allows for a more manageable reasoning process. Each node in such a tree can represent a sub-problem (a sub-question) that needs to be solved to contribute to the overall answer.

3. Methodology

Building upon the analysis of existing LLM-RAG methods and their challenges with reasoning structure, evidence integration, and dynamic error correction, this section details the TreeQA framework. TreeQA systematically answers multi-hop questions through three core stages. First, the logic tree generation module decomposes the input question into a hierarchical structure of simpler sub-questions and their corresponding hypotheses. Second, the knowledge retrieval module gathers relevant evidence from both structured (e.g., Wikidata KG) and unstructured (e.g., Wikipedia pages) sources for each step in the tree. Third, a self-correction mechanism iteratively validates the hypotheses against the retrieved evidence, dynamically correcting the reasoning path when inconsistencies arise, before generating the final answer. Fig. 2 illustrates the overall workflow of the TreeQA framework.

3.1. Logic tree generation

The foundation of the TreeQA framework lies in the generation of a logic tree, a hierarchical representation of the input question that guides the subsequent knowledge retrieval and reasoning processes. This initial phase focuses on transforming a multi-hop question into a structured tree, where each sub-node represents a simpler sub-question paired with a corresponding hypothesis. This structured decomposition is crucial for effectively navigating the complexities of multi-hop reasoning and enabling iterative refinement of the reasoning path. The goal is to develop a dynamic reasoning process that can adapt to complex, multi-hop questions. This is achieved through recursive decomposition and hypothesis generation, as described in the following section.

Recursive decomposition. Given an initial question (root node), the language understanding capabilities of the LLM are leveraged to break it down into several sub-questions. Each sub-question targets a specific aspect of the parent question, thereby reducing the complexity of the problem. This decomposition process is recursively applied to each newly generated sub-question until the LLM judges that a predefined termination condition has been reached. This condition signifies the target state where the sub-question is intended to be answerable by a single triple in the knowledge graph. It is important to note that the ability of the LLM to accurately reach this ideal level of granularity partly depends on the effectiveness of the prompt design.

Hypothesis generation. Accompanying each sub-question is a concrete hypothesis answer. This hypothesis answer is not an abstract guess, but rather a specific, verifiable answer generated by the LLM based on its internal knowledge base. This answer represents the best "guess" of the LLM, even if it may not be entirely correct. The hypothesis answer plays a crucial role in the subsequent retrieval and verification stages, providing a clear target for information retrieval and allowing the system to dynamically adjust the reasoning path.

Synergy of recursive decomposition and hypothesis generation. Recursive decomposition and hypothesis generation are mutually reinforcing and work in synergy. On the one hand, sub-questions provide context for hypothesis generation, enabling the LLM to predict answers more accurately. On the other hand, the hypothesis answer, in turn, guides the generation of sub-questions, as new sub-questions are often posed to verify or supplement the current hypothesis. This synergy is reflected in the construction of the logic tree, which adheres to the principles of Mutually Exclusive and Collectively Exhaustive (MECE). The detailed prompt designed for this approach is provided in Appendix A.1. The prompt for constructing the logic tree can be summarized by the following key points.

- **Summarization:** The hypothesis answer of a parent node should be a generalization and refinement of the information from all its child nodes. This means that before generating sub-questions, the LLM needs to first predict a possible answer based on the information of the current node (question or statement).
- **Support:** Sub-questions are posed to verify or supplement the hypothesis of its parent node. The LLM will generate more specific and granular sub-questions based on the hypothesis of the parent node.
- **MECE Principle:** During the decomposition process, TreeQA adheres to the MECE principle, ensuring that sub-questions at the same level are independent of each other and cover all relevant aspects as much as possible, avoiding repetition and omission.
- **Progression:** The generation of sub-questions follows a certain logical order, such as temporal sequence, causal relationship, etc., making the reasoning process more organized.

Recursive question decomposition and hypothesis generation are core components of the TreeQA framework. By progressively breaking down multi-hop questions into simpler sub-questions and generating concrete hypothesis answers for each sub-question, this can effectively guide the reasoning of the LLM and dynamically adjust the reasoning

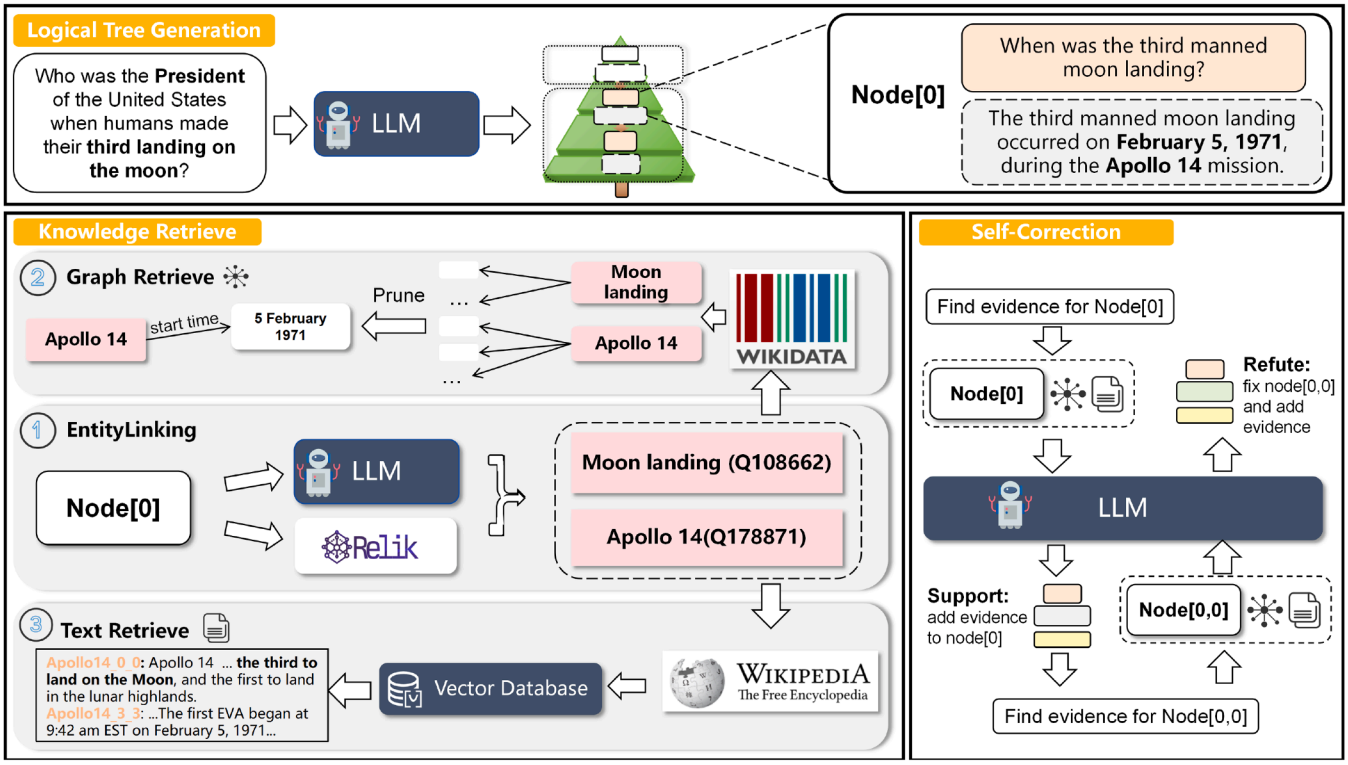


Fig. 2. Illustration of the workflow of the TreeQA framework that combines logic tree generation, knowledge retrieval, and self-correction.

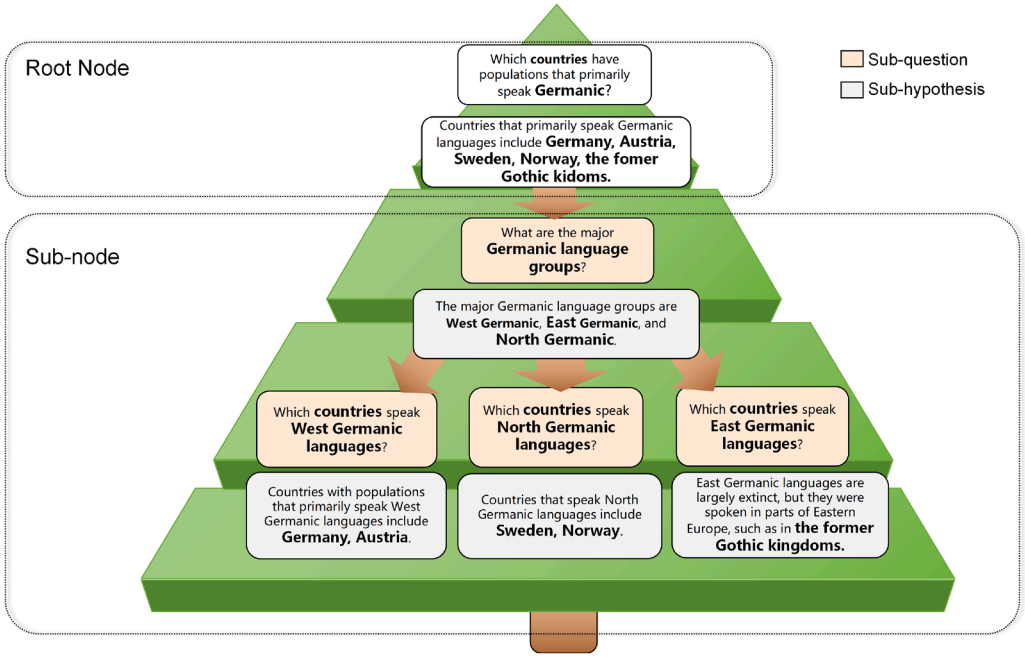


Fig. 3. Example for the logic tree structure used in TreeQA.

path during the subsequent retrieval and verification stages. This approach draws inspiration from the issue tree, making the reasoning process clearer and more organized, ultimately improving the accuracy and reliability of the question-answering system.

Fig. 3 provides a visual example of this process for the question "*Which countries have populations that primarily speak Germanic?*" The root node contains the original question and an initial, comprehensive hypothesis generated by the LLM *"Countries that primarily speak Germanic languages include Germany, Austria, Sweden, Norway, and the former Gothic kingdoms."* To verify or refine this broad hypothesis, the TreeQA framework breaks it down. The first sub-node poses a simpler, intermediate question "*What are the major Germanic language groups?*" and generates a corresponding hypothesis:"*West Germanic, East Germanic, and North Germanic*". This intermediate step breaks the problem down logically. Subsequently, this node is further decomposed based on the identified groups. Three parallel sub-nodes are created at the next level, each addressing a specific language group with the question: "*Which countries speak West/North/East Germanic languages?*" Each of these nodes, in

turn, has its own concrete hypothesis (e.g., "*Germany, Austria*" for "*West Germanic*"). This hierarchical structure demonstrates how the complex initial query is systematically broken down into manageable, verifiable steps, each with a specific question and a testable hypothesis, following a clear logical progression. The algorithm for logic tree node construction is presented in Algorithm 1.

Algorithm 1: Logic tree generation.

Input: Current question q, LLM $\mathcal{M}$
Output: Root node of the generated logic tree N
Function *BuildTree*$(q, \mathcal{M})$
  /* Initialize the current node */
  $N \leftarrow$ new Node()
  N.question $\leftarrow q$
  /* Generate hypothesis and sub-questions via LLM */
  N.hypothesis $\leftarrow$ GenerateHypothesis$(\mathcal{M}, q)$
  $\{q_1, \ldots, q_n\} \leftarrow$ DecomposeQuestion$(\mathcal{M}, q)$
  N.children $\leftarrow \emptyset$
  /* Recursively build the sub-tree for each sub-question */
  for $i \leftarrow 1$ **to** n **do**
    $child\_node \leftarrow$ BuildTree$(q_i, \mathcal{M})$
    Add $child\_node$ to N.children
  return N

3.2. Knowledge retrieval

Knowledge retrieval represents the core component in the interaction between TreeQA and external knowledge bases, directly determining the completeness of the information in subsequent verification processes. TreeQA employs an innovative dual-source retrieval strategy, utilizing both decomposed sub-questions and their corresponding hypothetical answers to retrieve information. This design is empirically validated, showing that even potentially incorrect hypotheses can provide valuable contextual clues and related terms that effectively supplement the semantic information of sub-questions. In contrast, existing approaches [26–28,39] typically rely solely on either question-path combinations or question-derived entity sets. By querying the knowledge bases with this synergistic information, TreeQA retrieves more comprehensive evidence, enabling multi-dimensional support for subsequent self-correction verification.

Knowledge sources. Wikidata[1] [40] serves as the KG in TreeQA. Specifically, we utilize the online version of Wikidata as our primary structured knowledge source. This allows TreeQA to access the most up-to-date information, which is crucial to answering questions that require timely knowledge. Wikipedia[2] [41] serves as the text corpus in TreeQA. TreeQA incorporates Wikipedia as a source of unstructured textual information to complement structured knowledge in Wikidata. This enables capturing a broader range of knowledge and addressing questions that may not be fully covered by the KG.

Graph retrieval from wikidata. The graph retrieval process leverages both *entity linking* and *relation linking* to accurately identify relevant information within Wikidata.

In contrast to existing methods [26–28] that typically rely on pre-annotated question entities for subsequent exploration, TreeQA automates the entity linking process by introducing an external model and optimizing the workflow by integrating LLMs with the Wikidata API. Although this method may have specific limitations in terms of runtime efficiency and answer accuracy, it compensates by significantly enhancing the versatility and usability of the system, making it more adaptable to a wider range of application scenarios. For each query, entity linking is performed to identify the most relevant entities in Wikidata. In TreeQA, a hybrid approach that combines an entity linking model with an LLM-based Named Entity Recognition (NER) system using the Wikidata API is employed. For each query Q, the entity linking model generates a set of top-k candidate entities $E_{\text{ELM}}(Q)$ from Wikidata as follows:

$$E_{\text{ELM}}(Q) = \{e_1, e_2, \ldots, e_k\}, \tag{1}$$

where each e_i is the ith ranked candidate entity based on their relevance to Q.

However, the entity linking model may have limitations in terms of timeliness and coverage, particularly for specialized domains. To address its limitations, an LLM-based NER system is also applied to extract entities from the text and query the Wikidata API to retrieve relevant entities, i.e.,

$$E_{\text{LLM}}(Q) = \{e'_1, e'_2, \ldots, e'_k\}, \tag{2}$$

where e'_i are the ith ranked candidate entity identified by the LLM and the Wikidata API. The candidates from both the entity linking model and the LLM-based NER system are combined and filtered using an LLM to select the top-3 most relevant entities, i.e.,

$$E(Q) = \text{LLM\_Filter}(E_{\text{ELM}}(Q) \cup E_{\text{LLM}}(Q)). \tag{3}$$

This filtering step ensures that only the most relevant entities are selected for further processing.

Next, we perform the relation linking process. Two primary strategies are considered for identifying and pruning relevant relations from a KG: (1) LLM-based linking and (2) dense retrieval-based linking.

The LLM-based linking method leverages the power of LLMs to enhance the process of identifying and filtering relevant relations. The core idea is to use an LLM to prune the search space and focus on the most promising connections between entities and the question. TreeQA first retrieves all one-hop relations of the entities identified in the question Q from Wikidata. Given $E(Q)$ returned by the entity linking process, Wikidata is queried to retrieve all relations where an entity in $E(Q)$ serves as a head or tail node. These relations are added to the set $R_{\text{rel}}(Q)$. For each extracted relation, the LLM-based linking method then employs an LLM to filter and identify the most relevant k (e.g., top-3) relations to the given query Q. This process can be formally represented as

$$G(Q) = \text{LLM\_Filter}(R_{\text{rel}}(Q), Q). \tag{4}$$

This pruning step ensures that only the most pertinent relationships are retained. The main strength of this method is its ability to nuance semantic understanding and contextual awareness, which generally translates to higher accuracy in relation identification. A potential drawback is the computational overhead associated with LLM inference for each query. Subsequent experimental evaluations, as detailed in Section 4.3, confirm that the LLM-based linking method typically achieves superior accuracy in relation linking, making it the default approach in TreeQA.

Alternatively, relation linking can also be performed using dense retrieval. This method requires a preliminary step: embedding and storing all relationship names from the KG. The subgraph induced by the entities identified in question Q is first retrieved from Wikidata. Using $E(Q)$, Wikidata is queried to retrieve a subgraph centered around these entities as $R_{\text{subgraph}}(Q)$. Embeddings for all Wikidata relations ($v(r) = \text{Embed}(r)$), along with their descriptions, are pre-computed and stored in a vector database D_{rel}. To retrieve the most relevant relations for a question Q, the question is transformed into a vector representation $v(Q)$, and a cosine similarity search is performed, restricting the results to the relations present in $R_{\text{subgraph}}(Q)$. Formally,

$$G(Q) = \underset{r \in R_{\text{subgraph}}(Q)}{\arg\max} \cos(v(Q), v(r)), \tag{5}$$

[1] https://www.wikidata.org
[2] https://www.wikipedia.org

where $\cos(v(Q), v(r))$ denotes the cosine similarity of $v(Q)$ and $v(r)$. The primary advantage of the dense retrieval-based linking method is its potential for lower latency, as it avoids the per-query computational cost of LLM-based pruning. However, its effectiveness is contingent on the quality of embeddings and the expressiveness of similarity metrics, which may not always capture the semantic complexity of relations as effectively as an LLM. As will be demonstrated in Section 4.3, this can result in a comparatively lower accuracy than the LLM-based linking method.

Text retrieval from wikipedia. In contrast to graph retrieval, the text retrieval process is relatively straightforward. Thanks to the open API,[3] Wikipedia pages can be conveniently split by sections. For sections longer than the predefined token limit, they are divided into different chunks. As such, each chunk is indexed in the following format: `ArticleName_SectionNumber_ChunkNumber`. The specific process for text retrieval is described as follows.

The process begins by utilizing entities obtained from the entity linking step. These entities are used to query the Wikipedia API to retrieve the corresponding pages related to the identified entities. For a given question Q, let $E(Q) = \{e_1, e_2, \ldots, e_k\}$ represent the set of entities obtained from entity linking. Each entity e_i corresponds to an article $A(e_i)$ in the Wikipedia corpus retrieved through the API. Then, the articles retrieved for Q are denoted as $A(Q) = \{A(e_1), A(e_2), \ldots, A(e_k)\}$.

Next, naive RAG techniques are applied to process the retrieved articles. Specifically, the articles in $A(Q)$ are first chunked into smaller segments. Let $C(A(e_i)) = \{c_1, c_2, \ldots, c_l\}$ denote the set of chunks for each article $A(e_i)$. Each chunk is then embedded using a pre-trained embedding model $\text{Embed}(\cdot)$, generating a set of vector embeddings for chunks, i.e.,

$$v(c_j) = \text{Embed}(c_j), \ \forall c_j \in C(A(e_i)). \tag{6}$$

These chunk embeddings are stored in a vector database $\mathcal{D}_{\text{chunk}}$, where each chunk embedding corresponds to a unique vector in the database, i.e.,

$$\mathcal{D}_{\text{chunk}} = \{v(c_1), v(c_2), \ldots, v(c_n)\}, \tag{7}$$

where n is the total number of chunks across all articles.

When a query Q is processed, it is embedded using the same pre-trained embedding model $\text{Embed}(\cdot)$ as

$$v(Q) = \text{Embed}(Q). \tag{8}$$

The database $\mathcal{D}_{\text{chunk}}$ is queried to retrieve the most similar text chunks to $v(Q)$ using cosine similarity search, i.e.,

$$T(Q) = \underset{c \in \mathcal{D}_{\text{chunks}}}{\arg\max} \cos(v(Q), v(c)). \tag{9}$$

Typically, we return the top-k text chunks $\{c_1, c_2, \ldots, c_k\}$ that are the most similar to $v(Q)$ from the database, which are then used for further processing.

3.3. Self-Correction

The robustness of the TreeQA framework is significantly enhanced by its self-correction mechanism. This mechanism traverses the logic tree to validate the generated sub-question and its corresponding hypothesis at each node against evidence retrieved from both the structured KG and unstructured text. If inconsistencies are detected, the reasoning path is dynamically corrected by reconstructing the affected sub-tree, thereby enabling iterative optimization and refinement. The traversal and validation process can be implemented using two distinct strategies: **serial** and **parallel**.

[3] https://github.com/martin-majlis/Wikipedia-API

- **Serial (DFS) Traversal:** The default approach is a serial validation process that follows a depth-first search (DFS) pattern. It recursively traverses the tree, fully validating one branch before moving to the next. While conceptually straightforward, the total inference time under this strategy is the sum of the validation times for all nodes.
- **Parallel (Level-by-Level) Traversal:** To improve efficiency, we also introduce a parallel validation strategy. This approach traverses the tree level by level, akin to a breadth-first search. All nodes at the same depth are processed concurrently. This parallel execution can significantly reduce overall latency, as the time taken for each level is determined by the slowest node in that level, rather than the sum of all nodes.

Regardless of the traversal strategy employed, the core logic applied at each node is divided into two stages: *hypothesis validation* and, if necessary, *sub-tree reconstruction*. Upon retrieving relevant information for a given node, TreeQA employs the LLM to assess the validity of the node's hypothesis. The LLM evaluates whether the retrieved evidence (G_{ret} and T_{ret}) supports, refutes, or is insufficient to judge the hypothesis in the context of its sub-question. This validation step, classifies the node's status into one of three categories:

Algorithm 2: Core validation and correction.

Input: Current node N, KB facts G_{ret}, text facts T_{ret}, LLM $\mathcal{M}$
Output: A tuple (N_{out}, status)

Function *VerifyAndCorrect*$(N, G_{ret}, T_{ret}, \mathcal{M})$
  `/* LLM classifies evidence as Supported, Refuted, or Unknown */`
  $verdict \leftarrow \mathcal{M}(N, G_{ret}, T_{ret})$
  if $verdict = Supported$ **then**
    $N_{out} \leftarrow \text{BindEvidence}(N, G_{ret}, T_{ret})$
    return $(N_{out}, \text{SUPPORTED})$
  else if $verdict = Refuted$ **then**
    $c \leftarrow \text{GetConflictReason}(\mathcal{M}, N, G_{ret}, T_{ret})$
    $N_{out} \leftarrow \text{ReconstructSubtree}(\mathcal{M}, N, c)$
    return $(N_{out}, \text{REFUTED})$
  else
    return $(N, \text{UNKNOWN})$

Algorithm 3: Node processing with retrieval and retry.

Input: Current node N, LLM $\mathcal{M}$, knowledge retriever $\mathcal{KR}$, max retries $K_{\max}$
Output: The final processed node N_{final}

Function *ProcessNode*$(N, \mathcal{M}, \mathcal{KR}, K_{\max})$
  `/* Perform initial retrieval */`
  $(G_{ret}, T_{ret}) \leftarrow \mathcal{KR}(N.\text{question})$
  for $k \leftarrow 1$ **to** $K_{\max}$ **do**
    `/* Call core validation logic (Algorithm 2) */`
    $(N_{out}, \text{status}) \leftarrow \text{VerifyAndCorrect}(N, G_{ret}, T_{ret}, \mathcal{M})$
    if $status = SUPPORTED$ **or** $status = REFUTED$ **then**
      return N_{out}
    if $k < K_{\max}$ **then**
      `/* Generate new clue and re-retrieve for retry */`
      $clue \leftarrow \text{GetNewClue}(\mathcal{M}, N, G_{ret}, T_{ret})$
      $(G_{ret}, T_{ret}) \leftarrow \mathcal{KR}(clue)$
  `/* Return original node if all retries fail */`
  return N

- **Supported:** The retrieved evidence aligns with and corroborates the node's hypothesis. This indicates that the current segment of the reasoning path is sound. The evidence is then associated with the node, enriching the logic tree with factual grounding.
- **Refuted:** The retrieved evidence contradicts the hypothesis answer or sub-question, indicating an error in the node's reasoning path. In this case, TreeQA triggers an error handling mechanism. First, the LLM is prompted to explain the reasons for the conflict. Then, using the current node as the root, the LLM reconstructs the sub-tree, generating a revised structure. TreeQA replaces the original sub-tree with this new sub-tree, achieving dynamic updating of the logic tree.
- **Unknown:** In some cases, no relevant information can be retrieved. To handle this, TreeQA prompts the model to suggest a new retrieval clue to optimize the search. Only after two consecutive retrieval attempts yield irrelevant information does the system cease modification, keeping the logic tree unchanged. This mechanism ensures that the system efficiently utilizes all potential information for reasoning.

Through this iterative cycle of hypothesis validation and conditional sub-tree reconstruction—applied node-by-node via either a serial or parallel traversal—TreeQA progressively refines the initial logic tree into a *golden tree*. This golden tree represents a validated and more reliable reasoning pathway, integrating insights from the LLM with factual information from the KG and text corpus, thereby forming a robust basis for the final answer generation. The core logic for this self-correction process is described in detail in Algorithms 2 and 3.

3.4. Final reasoning and answer generation

Upon completion of the depth-first traversal and the self-correction validation process across all nodes, the framework possesses a refined and verified reasoning structure, referred to as the "golden tree". This structure encapsulates not just the decomposed sub-questions but also their validated hypotheses and the supporting evidence retrieved from external knowledge sources. The final step involves leveraging this validated information to synthesize the definitive answer. The LLM is prompted, using the context derived from the golden tree, which may include the main question, key validated intermediate conclusions (hypotheses), and critical supporting facts, to generate the final response. Because the information pathways within the golden tree have undergone iterative scrutiny and correction against Wikidata facts and Wikipedia text, the foundation for this final generation step is significantly more robust. This grounding in verified evidence and corrected logical chains directly enhances the accuracy and reliability of the final answer, mitigating issues such as hallucination and error propagation common in less structured reasoning processes. The specific tree structure code format and more cases can be found in Appendix B.

4. Experiments

4.1. Experimental setup

In this section, we provide a thorough evaluation of TreeQA in addressing complex knowledge base question answering (KBQA) tasks, particularly those that require multi-hop reasoning and the integration of diverse knowledge sources.

Datasets and evaluation metrics. To evaluate the performance of TreeQA on KBQA tasks, we used the following four datasets: WebQSP [22], QALD10-en [42], AdvHotpotQA [43], and 2WikiMultiHopQA [23]. The statistics of these datasets can be found in Table 1.

For all these datasets, Hits@1 is adopted as the evaluation metric, consistent with previous studies. Furthermore, the SimpleQuestion dataset [44] is utilized specifically to evaluate the effectiveness of the entity and relation linking components, as detailed in the ablation studies.

Table 1
Statistics of datasets used in the experiments. We use * to denote that a random sample of 1000 questions is used for evaluation.

Dataset	Answer Format	Number of test samples
WebQSP	Entity/Number	1639
QALD-10	Entity/Number	333
AdvHotPotQA	Entity/Number	308
2WikimultiHopQA (*)	Entity/Number	1000
SimpleQuestion	Entity/Number	3000

Implementation details. For entity and relation linking, the top-3 entities and relations are retrieved. For the entity linking model, the latest version of the Relik series [45], i.e., `Relik-Entity-Linking-Large`, is used. The LLM-based strategy is utilized for relation linking by default, where the backbone model performs the identification and pruning of relations. The external knowledge sources include Wikidata and Wikipedia, with Nv-Embed-V2 [46] used as the vector retrieval model and ChromaDB[4] used as the vector database. For Wikipedia, articles are chunked into blocks of 300 tokens to manage the length of text efficiently. For each article corresponding to an entity, the top-3 relevant chunks are retrieved to assist the model in reasoning. To evaluate the effect of backbone LLMs on the performance of TreeQA, experiments are conducted with DeepSeek-V3 [47], GPT-3.5-turbo [48] Qwen2.5-14B-instruct [49] and GPT-4o-mini [50].

Baselines. To ensure a fair and comprehensive evaluation, we utilize GPT-3.5-turbo as the backbone model for all baselines, assessed under an unsupervised setting. The selection encompasses a range of methods representing distinct paradigms for MHQA. We begin with LLM-only methods, including **Direct Reasoning (DR)**, **Chain-of-Thought (CoT)** [32], and **Self-Consistency (SC)** [51], to establish the performance of the LLM's intrinsic reasoning. We then include vanilla RAG, which is retrieved from unstructured text, to benchmark standard retrieval augmentation. Finally, we compare against a suite of agentic RAG methods: **Adaptive-RAG** [34], which employs a classifier agent to strategically route queries based on complexity; **Self-RAG** [33], which enhances the LLM with self-reflection capabilities to critique and refine its own outputs; the combination of **HippoRAG and IRCoT** [30,32], which pairs an iterative reasoning agent with a powerful knowledge graph-based retrieval tool; **Chain-of-Knowledge (CoK)** [12], which performs iterative self-correction on a sequence of rationales; and **Think-on-Graph 2.0 (ToG-2)** [13], where an agent interactively explores a knowledge graph. This comprehensive set of baselines enables a nuanced comparison of various agentic strategies.

4.2. Experimental results

We compare the performance of each method on four benchmark datasets in terms of the Hits@1 score. The results are shown in Table 2. TreeQA demonstrates state-of-the-art performance across all benchmarks, achieving Hit@1 scores of 87.0 %, 57.2 %, 53.0 %, and 59.0 % on WebQSP, QALD10-en, AdvHotpotQA, and 2WikiMultiHopQA, respectively. These results represent significant improvements over a comprehensive suite of baselines, including several state-of-the-art agentic systems. The substantial gain compared to LLM-only methods (e.g., an improvement of 37.4 % over the best LLM-only baseline on WebQSP) underscores the critical benefit of grounding reasoning in external knowledge sources.

The superiority of TreeQA is most evident when compared against other advanced agentic systems on complex, multi-hop tasks. On the challenging AdvHotpotQA dataset, for instance, TreeQA achieves a Hit@1 score of 53.0 %, establishing a significant margin over strong competitors like Adaptive-RAG (47.0 %) and HippoRAG + IRCoT

[4] https://github.com/chroma-core/chroma

Table 2
Performance comparison of different methods on four benchmark datasets in terms of Hits@1 (%). The best and second-best results on each dataset are highlighted by **bold** and underlined fonts, respectively.

Type	Method	WebQSP	QALD10-en	AdvHotpotQA	2WikiMultiHopQA
LLM-only	DR	63.3	42.0	23.1	54.6
	CoT	62.2	42.9	30.9	52.1
	CoT-SC	61.1	45.3	34.4	53.2
Text-based RAG	Vanilla RAG	66.7	51.7	25.4	55.7
KG-based RAG	ToG	76.2	50.2	26.3	53.2
Hybrid & Agentic RAG	HippoRAG + IRCoT	–	–	45.7	47.7
	Self-RAG	–	–	39.0	30.6
	Adaptive-RAG	–	–	47.0	56.8
	CoK	77.6	47.1	35.4	54.2
	ToG-2	81.1	54.1	42.9	56.3
(* Ours)	TreeQA	**87.0**	**57.2**	**53.0**	**59.0**

Table 3
Performance comparison of direct reasoning (DR) and TreeQA with different backbone LLMs in terms of Hits@1 (%).

Dataset	Qwen2.5-14B-instruct		GPT-3.5-turbo		DeepSeek-V3		GPT-4o-mini	
	DR	TreeQA	DR	TreeQA	DR	TreeQA	DR	TreeQA
WebQSP	68.0	89.0 (30.9 %↑)	63.3	87.0 (37.5 %↑)	78.4	90.0 (14.8 %↑)	66.9	88.7 (32.6 %↑)
AdvHotpotQA	27.9	55.0 (97.1 %↑)	23.1	53.0 (129.4 %↑)	46.6	59.3 (27.3 %↑)	31.8	40.3 (26.7 %↑)
QALD10-en	38.2	59.7 (56.3 %↑)	42.0	57.2 (36.2 %↑)	45.5	63.2 (39.0 %↑)	60.1	63.4 (5.5 %↑)
2WikiMultiHopQA	43.2	51.5 (19.2 %↑)	54.6	59.0 (8.1 %↑)	61.1	83.3 (31.5 %↑)	50.7	68.4 (34.9 %↑)

(45.7 %). This performance advantage stems directly from TreeQA's core architectural design. Unlike the linear, forward-chaining reasoning of IRCoT or the pre-defined execution paths of Adaptive-RAG, which can struggle to recover from early-stage errors, TreeQA's ability to reconstruct entire reasoning sub-trees provides a more robust and flexible mechanism for error correction. This advantage is consistent across other datasets, such as on 2WikiMultiHopQA, where TreeQA (59.0 %) also outperforms the top-performing agentic baseline, Adaptive-RAG (56.8 %).

Furthermore, the fundamental benefits of TreeQA's hybrid, knowledge-grounded approach are clear when compared to single-source RAG methods. The significant advantage over text-based RAG (30.4 % gain on WebQSP, 108.7 % on AdvHotpotQA) highlights the necessity of integrating structured knowledge (Wikidata) alongside textual context (Wikipedia). Ultimately, it is the combination of a structured, verifiable decomposition into sub-problems and a powerful, evidence-based error-correction loop that allows TreeQA to effectively navigate complex information needs where other methods falter.

4.3. Ablation study

Effect of LLM backbones. We first assess the extent to which TreeQA enhances LLMs in terms of QA capability. An ablation experiment is conducted on three backbone LLMs, namely Qwen2.5-14B-instruct, GPT-3.5-turbo, DeepSeek-V3, and GPT-4o-mini, on the WebQSP, AdvHotpotQA, and QALD10-en datasets. The results are shown in Table 3. TreeQA generally improves the QA performance of all LLMs, but with distinct patterns. Specifically, weaker LLMs gain more drastic relative improvements (e.g., Qwen2.5-14B-instruct's 97.1 % & GPT-3.5-turbo's 129.4 % surges on AdvHotpotQA), effectively the bridging gaps to stronger models' direct reasoning baselines. Meanwhile, advanced LLMs like DeepSeek-V3 and GPT-4o-mini achieve smaller but critical absolute gains (e.g., 14.8–39.0 % for DeepSeek-V3 and 5.5–34.9 % for GPT-4o-mini), demonstrating the role of TreeQA as both a compensator for knowledge bottlenecks and a precision enhancer for complex reasoning.

Effect of entity linking methods. We aim to confirm whether using LLMs for NER + API search can achieve higher entity linking accuracy than the Relik entity linking model. To investigate this, we perform experiments on the SimpleQuestion dataset, a single-hop QA dataset originally built from Freebase but later aligned to Wikidata [52], with question-entity pairs annotated using Wikidata identifiers. DeepSeek-V3 is selected as the LLM for the NER and Wikidata API search and compared against the Relik entity linking model. In addition, a combined strategy that integrates the results of both approaches is also evaluated. The performance is measured by top-k recall scores when $k = 1, 3, 5$. The result is shown in Fig. 4. Relik achieves top-1, top-3, and top-5 recall scores of 70.13 %, 79.80 %, and 82.27 %, respectively. DeepSeek-V3 shows significantly lower recall scores of 60.40 %, 71.50 %, and 71.57 % when $k = 1, 3, 5$. Then, the combined strategy significantly outperforms both individual methods, achieving top-1, top-3, and top-5 recall scores of 82.60 %, 90.43 %, and 91.97 %. This demonstrates that combining the strengths of the Relik entity linking model with an LLM yields the best overall entity linking performance, particularly when multiple candidate entities need to be considered. Furthermore, entity linking models are often trained on relatively small datasets, leading to significantly lower accuracy on unseen entities compared to the search capabilities of LLMs. This difference in generalization ability is a contributing factor to the higher recall achieved by the combined approach.

Effect of relation linking methods. We then assess the effectiveness of the two linking methods we use. The result is shown in Fig. 5. The experiment is carried out on the SimpleQuestion dataset, also using the top-k recall scores with $k = 1, 3, 6$ for evaluation. We compare the performance between different LLMs, including DeepSeek-V3, GPT-3.5-turbo, Qwen2.5-14B-instruct, and Qwen2.5-7B-instruct, and dense retrieval models, including Nv-Embed-V2 (7.85B) [46], Stella (1.54B) [53], and all-MiniLM-L6-v2 (22.7M) [54]. Even the smallest LLM, Qwen2.5-7B-instruct, surpasses all dense retrieval models, while larger LLMs such as DeepSeek-V3 achieve top-1, top-3, and top-6 recall scores of 45.4 %, 94.0 %, 98.1 %, significantly outperforming the best dense retrieval model, Nv-Embed-V2, by a large margin. The performance advantage is attributed to LLMs' superior semantic understanding, contextual awareness, and generative capabilities, enabling more nuanced relationship capture than similarity-based matching. Thus, LLMs are the preferred choice for relation linking in most cases.

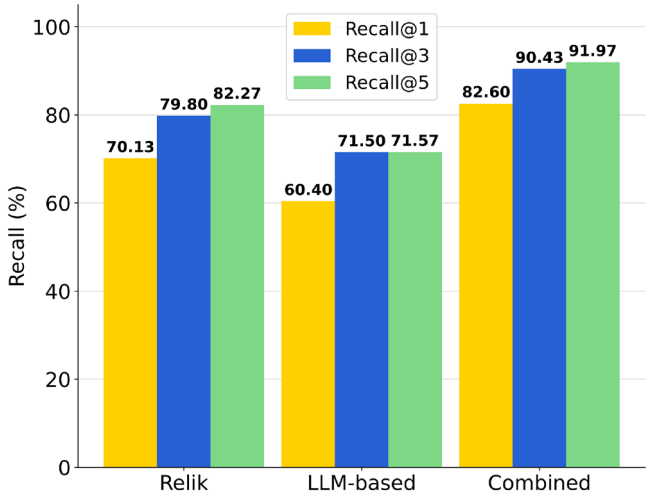


Fig. 4. Performance comparison of different entity linking methods.

Effect of logic tree decomposition. We compare the performance of our logical tree decomposition scheme against the Chain-of-Thought (CoT) strategy and direct reasoning (DR) for multi-hop reasoning. We conduct ablation experiments on the AdvHotpotQA dataset, and the performance of each method is evaluated using the Hit@1 metric. The three strategies were evaluated across three LLMs, including DeepSeek-V3, Qwen2.5-14B-instruct, and GPT-3.5-turbo, for comparative analysis. The results, presented in Table 4, show that for DeepSeek-v3 and Qwen2.5-14B-instruct, direct reasoning outperforms both CoT and the logical tree decomposition in TreeQA without validation and self-correction (TreeQA w/o Self-Correction), which is a counterintuitive result. It is conjectured that this effect may stem from the increased number of intermediate reasoning steps that can lead to error propagation. However, for GPT-3.5-turbo, logical tree decomposition shows better performance than DS and CoT even without self-correction. Notably, the TreeQA framework significantly outperformed all other methods for all three LLMs. This indicates that while simple logical tree decomposition might decrease reasoning performance due to error propagation, the validation and self-correction mechanisms of TreeQA effectively correct these errors, leading to a substantial improvement in overall performance.

Impact of retrieval quality on final accuracy. To precisely assess the respective contributions of our retrieval and generation modules, we conducted an ablation study to evaluate their performance independently. In this analysis, we randomly sampled 100 questions from the AdvHotpotQA dataset. Crucially, we utilized the ground-truth supporting facts provided in the dataset as the "gold facts" to serve as the gold standard for evaluating retrieval quality. We tested three distinct retrieval configurations: the standard `hybrid` approach (Wikidata + Wikipedia), a `graph-only` approach (retrieving exclusively from Wikidata), and a `text-only` approach (retrieving exclusively from Wikipedia). The backbone model is DeepSeek-V3, with all other settings maintained at their default values. For each configuration, we manually measured the Recall of gold facts within the retrieved context and the Accuracy (Acc) of the final answer. The results, presented in Table 6, clearly reveal the behavior of our framework. A key finding is the direct positive correlation between Recall and Accuracy across all configurations. This suggests that our generation module is highly robust: once the correct evidence is retrieved, the model consistently generates the correct final answer. This analysis, therefore, confirms that the primary bottleneck and source of error in the model's performance lie in the retrieval stage, rather than the generation stage. Moreover, the results strongly validate our choice of architecture. By integrating both structured and unstructured knowledge, the `hybrid` strategy achieves the highest Recall and Accuracy, outperforming all single-source methods. This outcome underscores the advantage of a more comprehensive knowledge base for tackling complex multi-hop reasoning tasks.

Table 4
Comparative analysis of logic tree decomposition with Chain-of-Thought (CoT) strategy and direct reasoning (DR).

Method	DeepSeek-V3	Qwen2.5-14B-instruct	GPT-3.5-turbo
DR	46.6	27.6	37.4
CoT	38.7	22.1	23.4
TreeQA w/o Self-Correction	44.2	22.4	40.9
TreeQA (full)	**59.3**	**55.0**	**53.0**

Table 5
Average inference time (in seconds), average token consumption, and accuracy (Hits@1, %) of different methods on the AdvHotpotQA dataset. The best result in each column is highlighted in **bold**.

Type	Method	Time (s) ↓	Tokens ↓	Hits@1 (%) ↑
LLM-only	DR	**1.8**	**113.2**	23.1
	CoT	4.1	520.7	30.9
	CoT-SC	11.6	2786.3	34.4
KG-based RAG	ToG	69.3	9606.4	26.3
Hybrid & Agentic RAG	CoK	30.1	2343.2	35.4
	ToG-2	27.3	6075.1	42.9
Our Method	TreeQA (Serial)	152.9	10459.1	53.3
	TreeQA (Parallel)	109.6	10192.1	**53.6**

Table 6
Comparison of golden fact Recall and final answer Accuracy for different retrieval strategies on a 100-question subset of AdvHotpotQA.

Retrieval strategy	Recall (%)	Accuracy (%)
Hybrid (Graph + Text)	68	68
Graph-only	60	60
Text-only	64	64

4.4. Computational cost analysis

We conducted an efficiency analysis of TreeQA on the AdvHotpotQA dataset, measuring the average inference time and token consumption compared to other methods. The results are presented in Table 5. Predictably, TreeQA is more resource-intensive than the baseline approaches. Its core strength, the iterative decomposition of questions and the dynamic evidence-based self-correction mechanism, requires multiple interactions with the LLM. This inherently leads to higher token consumption, which is the trade-off for its superior accuracy and reliability. To address the challenge of inference latency, we explored two execution strategies for the validation process. The first is a **serial** (DFS) strategy, where each node is processed sequentially. The second is a **parallel** strategy, where all nodes at the same level of the logic tree are validated concurrently. As the results show, employing a parallel execution model substantially reduces the average inference time (from 152.9 s to 109.6 s) without compromising, and in fact slightly improving, the final accuracy. This demonstrates that while TreeQA has a higher intrinsic cost, its latency can be effectively managed through parallelization.

4.5. Analysis of self-correction mechanism

To quantify the impact of TreeQA's dynamic validation and self-correction mechanism, the average number of updates per logic tree is recorded across four datasets of varying difficulty levels: WebQSP (easy), QALD10-en (normal), AdvHotpotQA (hard), and 2WikiMultiHopQA (hard). The analysis was performed using three LLMs: DeepSeek-V3, GPT-3.5-turbo, and Qwen2.5-14B-instruct. As shown in Table 7, the average number of tree updates generally increases with the

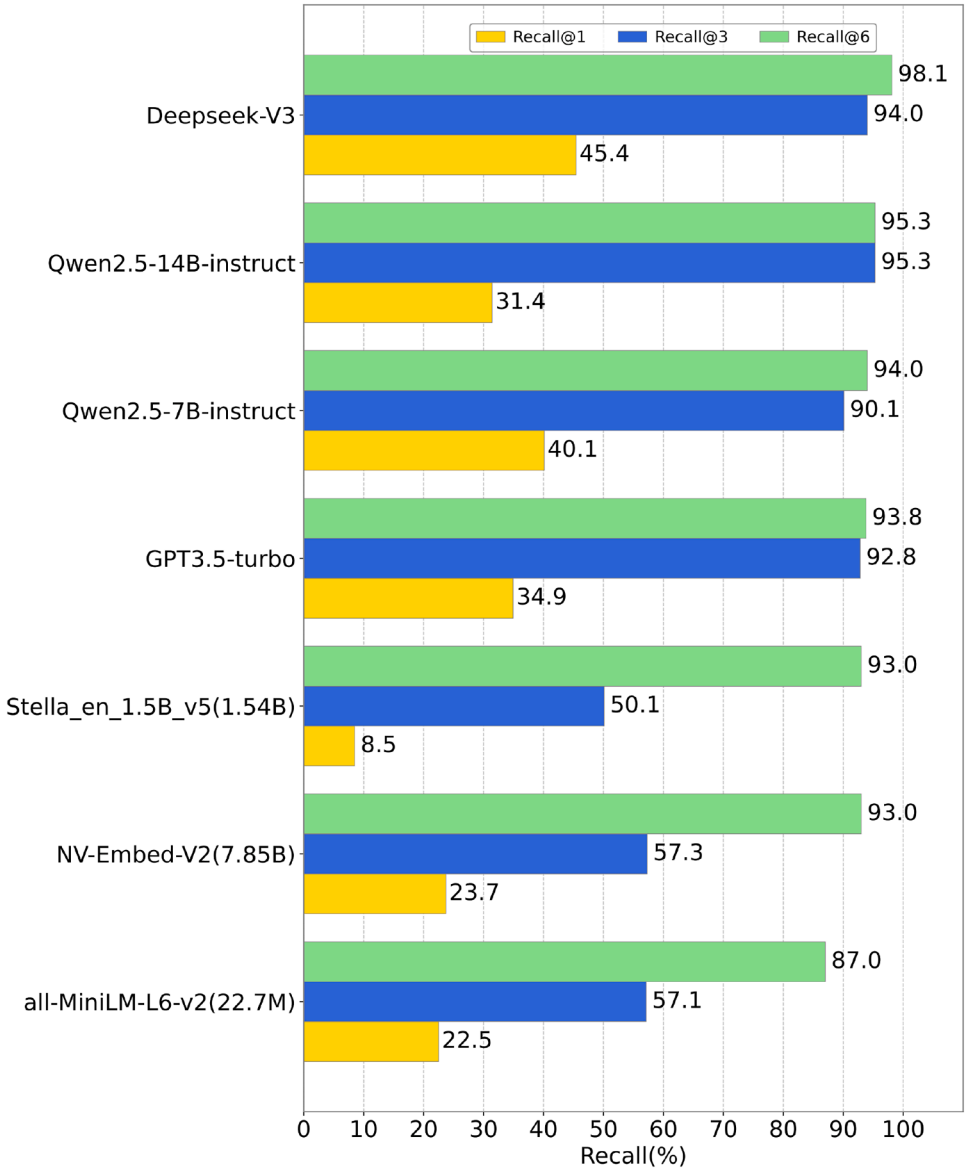


Fig. 5. Performance comparison of different relation linking methods.

difficulty of the dataset and decreases with the performance of the LLM, as expected. For instance, on the most challenging dataset, 2WikiMultiHopQA, Qwen2.5-14B-instruct requires an average of 1.83 updates per tree, while the more powerful DeepSeek-V3 requires only 1.04 updates. In contrast, on the easier WebQSP dataset, both models required fewer updates. This correlation between the difficulty of the dataset, the performance of the model, and the frequency of tree updates highlights the crucial role of the validation and self-correction mechanism in the ability of TreeQA to adaptively refine its reasoning and correct errors, particularly for more difficult questions.

4.6. Error analysis

Finally, a detailed error analysis is conducted on the AdvHotpotQA dataset to investigate the cases in which TreeQA cannot provide correct answers. Erroneous cases are manually reviewed and classified into four categories: retrieval errors, character matching errors, model-induced errors, and dataset ambiguity errors. The distribution of erroneous cases among the four categories is shown in Fig. 6.

Table 7
Average number of logic tree updates on each dataset for each LLM.

Model	WebQSP	QALD10-en	AdvHotpotQA	2WikiMultiHopQA
Qwen2.5-14B-instruct	0.47	1.24	1.54	1.83
GPT-3.5-turbo	0.45	0.79	1.45	1.38
DeepSeek-V3	0.56	0.41	0.52	1.04
Avg	0.49	0.81	1.17	1.42

Next, we analyze each type of error in more detail.

- **Retrieval Errors (41.7 %):** The primary source of errors in TreeQA stems from retrieval failures. These errors typically occur when the system fails to locate the correct entity, leading to retrieved information that is irrelevant to the question or completely missing. Since TreeQA heavily relies on retrieving knowledge from Wikidata and Wikipedia, any failure in linking entities or retrieving relevant passages directly impacts the final answer.

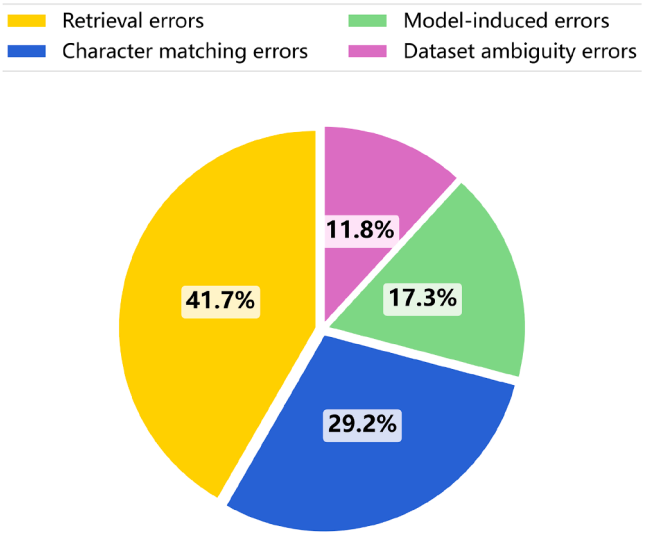


Fig. 6. Error distribution of TreeQA on the AdvHotpotQA dataset.

- **Character Matching Errors (29.2 %):** These errors are caused by the exact match evaluation metric, where semantically correct answers are mistakenly marked incorrect due to slight textual variations. Examples include cases such as *"Thomas Montgomery Newman"* being labeled as incorrect when TreeQA outputs *"Thomas Newman"* or *"Yale University"* vs. *"University Yale"*. While these errors do not indicate a fundamental issue in the reasoning of TreeQA, they highlight the limitations of rigid matching criteria.
- **Model-Induced Errors (17.3 %):** Some errors originate from the inherent limitations of the TreeQA reasoning mechanism. Specifically, cases were observed where the model incorrectly structured the logic tree by treating parent-child nodes as sibling nodes. This misalignment led to failures in updating conclusions when contradictions emerged, preventing error correction. Additionally, some of these errors were exacerbated by retrieval failures.
- **Dataset Ambiguity Errors (11.8 %):** A small fraction of errors arise due to ambiguities in the dataset itself. These questions often lack clear referents, making it difficult for any system to provide a definitive answer. For example, the question *"Jamell Anderson plays for the basketball team from what city?"* lacks a uniquely identifiable basketball team, leading to multiple valid interpretations.

5. Conclusion

In this paper, we proposed TreeQA, a novel framework designed to address the challenges of reliability and interpretability in multi-hop question answering for LLMs. By decomposing multi-hop questions into a hierarchical logic tree, leveraging hybrid knowledge retrieval from both structured and unstructured sources, and implementing a validation and self-correction mechanism, TreeQA demonstrates a significant improvement in reasoning accuracy over existing LLM-RAG methods across diverse QA benchmark datasets. The explicit tree structure provides a degree of interpretability in the reasoning process, while the iterative validation against external evidence effectively mitigates error propagation and model hallucination.

Despite its strengths, TreeQA still has several limitations. Its performance remains heavily dependent on the quality and coverage of the underlying knowledge sources, as well as the effectiveness of the initial entity/relation linking. Furthermore, the inherent sequential nature of the node-by-node validation process introduces significant latency, as the processing of each node must await the completion and validation of its predecessors. This dependency creates a bottleneck that limits the efficiency of the framework, particularly for deeper or more complex trees, making it less suitable for real-time applications. In future work, we aim to enhance the robustness of the knowledge retrieval phase. Exploring alternative decomposition strategies might yield benefits. In addition, addressing the efficiency bottleneck is also paramount. One potential avenue is exploring parallel validation strategies, with multiple nodes or sub-trees evaluated concurrently. Finally, developing more refined methods to handle conflicting evidence during validation can further enhance reasoning accuracy.

CRediT authorship contribution statement

Xiangrui Zhang: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization; **Fuyong Zhao:** Writing – review & editing, Validation, Investigation, Data curation; **Yutian Liu:** Writing – review & editing, Validation, Investigation, Data curation; **Panfeng Chen:** Writing – review & editing; **Yanhao Wang:** Writing – review & editing; **Xiaohua Wang:** Writing – review & editing; **Dan Ma:** Writing – review & editing; **Huarong Xu:** Writing – review & editing; **Mei Chen:** Writing – review & editing; **Hui Li:** Writing – review & editing, Supervision, Resources, Project administration, Funding acquisition, Conceptualization.

Declaration of generative AI in scientific writing

During the preparation of this work, the authors used AI-assisted technologies for language and grammar refinement to improve the readability and clarity of the manuscript. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

Data availability

Our code and data are publicly available at https://github.com/ACMISLab/TreeQA.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was funded by the Fund of National Natural Science Foundation of China (No. 61562010), National Key Research and Development Program of China (No. 2023YFC3341205), Guizhou Provincial Major Scientific and Technological Program (No. [2024]003), Guizhou Provincial Program on Commercialization of Scientific and Technological Achievements (Nos. [2025]008, [2023]010), Research Projects of the Science and Technology Plan of Guizhou Province (Nos. [2023]276, [2022]261, [2022]271).

Appendix A. Prompts for LLMs in TreeQA

A.1. Prompt for logic tree generation

The following prompt serves as a detailed instruction set for the LLM to perform the logic tree generation stage of the TreeQA framework. Its objective is to guide the LLM in systematically decomposing a complex, multi-hop input question into a hierarchical JSON structure. Each node in the tree contains a simpler sub-question and a concrete, verifiable hypothesis answer generated from the internal knowledge of LLM. The prompt emphasizes the adherence to construction principles like MECE and logical progression to ensure a coherent and granular breakdown. This structured decomposition, with its initial hypotheses, forms the foundation for subsequent knowledge retrieval and the iterative self-correction mechanism, ultimately aiming to produce a reliable and interpretable reasoning path for answering the original complex question.

```
You are an intelligent assistant who is good at analyzing and reasoning, and your task is to construct a logic tree to break down and reason step by step according to the complex questions posed by the user, and finally arrive at the answer.
**Rules for constructing a logic tree:**
1. **Root:** the user's main question.
2. **Sub-nodes:** Each sub-node contains a **Sub-question** and a **Specific Hypothesis Answer** based on your own knowledge.
    * **Sub-questions:** are used to guide reasoning and information retrieval, and each sub-question should address only a single entity or relationship.
    * **Concrete Hypothesis Answer:** is a **concrete answer** to a sub-question given by your own knowledge, rather than an abstract or broad hypothesis. This answer should be the one that the model considers most likely, even though it may not be correct, and the answer should also be richly detailed.
3. **Construction principles:** The construction of the logic tree follows the pyramid principle:
    * **Conclusion first:** The hypothetical answer of a parent node should be a generalization of the information of its children.
    * **Correspondence between top and bottom:** The information of a child node should support the hypothetical answer of the parent node.
    * **MECE Principle: ** Child nodes at the same level should be independent of each other and try to cover all possibilities.
    * **Logical progression:** The generation of child nodes should follow a certain logical order (e.g., time, cause and effect, etc.).
* **Reasoning steps:**
1. **Question Decomposition & Hypothesis Generation:** Based on the information (question or statement) in the current node, propose a simpler sub-question and give a **specific hypothetical answer** based on your own knowledge base.
2. **Answer Synthesis:** When the logic tree is constructed, synthesize and reason out the final answer based on the information from all nodes.
**Output Format:**
Please strictly follow the following JSON format for output:
{{
  "input_question":"<main_question>",
  "logic_tree": {{
      "children": [
        {{
          "sub_question":"<subquestion1>",
          "hypothesis_answer":"<specific hypothesis answer 1>",
          "children": [
            {{
              "sub_question":"<sub-sub-question 1.1>",
              "hypothesis_answer": "<specific hypothetical answer 1.1>",
              "children": [
                {{
                  "sub_question":"<children_subquestion 1.1.1>",
                  "hypothesis_answer":"<specific hypothetical answer 1.1.1>"
                }}
              ]
            }}
          ]
        }}
      ]
    }},
  "answer": "<final answer>"
}}
Caveats:
Each hypothetical answer must be a concrete, verifiable answer, not an abstract hypothesis.
Ensure that the logic tree construction process follows the pyramid principle.
The output should be concise and intuitive, with the lowest level question or hypothesis containing only the least granular information.
Please just output JSON format content, do not output any analysis.
Now the user question is: {query}
```

A.2. Prompt for validation

The following prompt is designed for the self-correction mechanism within the TreeQA framework, specifically for the hypothesis validation step. After the LLM generates a sub-question and a hypothesis answer in the logic tree, and relevant information is retrieved, this prompt instructs the LLM to evaluate the hypothesis against this retrieved evidence. This validation step is iterative and applied node-by-node. If a hypothesis is refuted ("isTrue": false), TreeQA triggers a sub-tree reconstruction, allowing it to dynamically correct the reasoning path and mitigate error propagation, ultimately leading to a more reliable "golden tree" for final answer generation. The strict JSON output format ensures that the validation result can be programmed to be interpreted by the TreeQA system to decide the next action.

```
Please verify whether the answer is correct based on the given Info.
- If no relevant Info is provided (e.g., ["No Information provided."] or []), set `"isTrue": "unknown"` and `"fact_sufficient": false`.
- If the answer is correct and no reason is needed, set `"isTrue": true`.
- If the answer is incorrect, set `"isTrue": false`, provide the reason for the error, and include the correct answer in the reason.
- Always include reference information (`ref`) when available, for both correct and incorrect answers.
### Reference Formatting Rules:
- If no Info is available or unrelated to the question, set `"fact_sufficient": false`, `"ref": "No Information provided."`, and `"isTrue": "unknown"`.
- If citing `textInfo`, provide only the Wikipedia article's title ID (omit full text).
- If citing `graphInfo`, provide up to 3 relevant triplets in the format: `entityLabel-relationLabel-Value` from Wikidata.
- Do not fabricate information-references must match the provided Info.
### Output Format (JSON only):

{{
    "isTrue": true/false/unknown,
    "fact_sufficient": true/false,
    "reason": "<None>/<reason>",
    "ref": {{
        "wikipedia": ["<textInfo id>"],
        "Wikidata": ["entityLabel-relationLabel-Value"]
    }}
}}
```

A.3. Prompt for CoT

The following prompt is designed to elicit Chain-of-Thought (CoT) reasoning from an LLM for answering multi-hop questions, as used in the baseline comparisons. Unlike TreeQA's structured logic tree generation, this prompt instructs the LLM to produce a more free-form sequential breakdown of its reasoning process. The CoT approach aims to improve reasoning by encouraging the model to "think step-by-step," but as the paper discusses, without external knowledge validation and correction as in TreeQA, it can still suffer from errors or hallucinations, especially in complex scenarios.

```
Given a multi-hop question, break it down step by step, explicitly stating intermediate reasoning before arriving at the final answer. Use the following format:

Example 1:
Q: What state is home to the university that is represented in sports by George Washington Colonials men's basketball?
A:First, the university represented by George Washington Colonials men's basketball is George Washington University.
Second, George Washington University is located in Washington, D.C..
The answer is {Washington, D.C.}.

Example 2:
Q: Who lists Pramatha Chaudhuri as an influence and wrote Jana Gana Mana?
A:First, Rabindranath Tagore wrote Jana Gana Mana.
Second, Rabindranath Tagore lists Pramatha Chaudhuri as an influence.
The answer is {Rabindranath Tagore}.

Example 3:
Q: Who was the artist nominated for an award for You Drive Me Crazy?
A:First, the artist associated with You Drive Me Crazy is Britney Spears.
Second, Britney Spears was nominated for an award for this song.
The answer is {Britney Spears}.

Example 4:
Q: What person born in Siegen influenced the work of Vincent Van Gogh?
A:First, Peter Paul Rubens, Claude Monet, and others influenced Vincent Van Gogh.
Second, Peter Paul Rubens was born in Siegen.
The answer is {Peter Paul Rubens}.

Example 5:
Q: What is the country close to Russia where Mikheil Saakashvili holds a government position?
A:First, countries close to Russia include China, Norway, Finland, Estonia, and Georgia.
Second, Mikheil Saakashvili held a government position in Georgia.
The answer is {Georgia}.

Example 6:
Q: What drug did the actor who portrayed the character Urethane Wheels Guy overdose on?
A:First, Mitch Hedberg portrayed the character Urethane Wheels Guy.
Second, Mitch Hedberg overdosed on heroin.
The answer is {Heroin}.
```

Appendix B. Examples for TreeQA output

The following text in JSON format demonstrates an example for the output of TreeQA: the input_question is decomposed into a logic_tree of simpler sub_questions, each with an LLM-generated hypothesis_answer. Crucially, these hypotheses are validated against retrieved references from Wikipedia and Wikidata. The final answer is then synthesized from this evidence-grounded reasoning chain, showcasing a structured and verifiable approach to multi-hop question answering.


```
{
    "input_question": "On which island is the Indonesian capital located?",
    "logic_tree": {
        "children": [
            {
                "sub_question": "What is the capital of Indonesia?",
                "hypothesis_answer": "The capital of Indonesia is Jakarta.",
                "children": [
                    {
                        "sub_question": "On which island is Jakarta located?",
                        "hypothesis_answer": "Jakarta is located on the island of Java.",
                        "ref": {
                            "Wikipedia": [
                                "Java_0_1||Indonesia's capital city, Jakarta, is on Java's
                                    northwestern coast. Many of the best-known events in
                                    Indonesian history took place on Java.",
                                "Jakarta_5_1||The Greater Jakarta area includes three
                                    bordering regencies (Bekasi Regency, Tangerang Regency
                                    and Bogor Regency) and five adjacent cities (Bogor,
                                    Depok, Bekasi, Tangerang and South Tangerang). Jakarta
                                    is situated on the northwest coast of Java, at the
                                    mouth of the Ciliwung River on Jakarta Bay, an inlet of
                                     the Java Sea. It is strategically located near the
                                    Sunda Strait."
                            ],
                            "Wikidata": [
                                "Jakarta-located in/on physical feature-Java"
                            ]
                        }
                    }
                ],
                "ref": {
                    "Wikipedia": [
                        "Capital of Indonesia_0_0||The capital of Indonesia, officially the
                             capital of the Unitary State of the Republic of Indonesia (
                            Indonesian: Ibukota Negara Kesatuan Republik Indonesia), is
                            Nusantara,. Jakarta(one of the oldest continuously inhabited
                            cities in Southeast Asia), previously known as Batavia, was the
                             de facto capital of the Dutch East Indies. In the early 20th
                            century, the Dutch East Indies government attempted to relocate
                             the capital from Batavia to Bandung."
                    ],
                    "Wikidata": [
                        "Jakarta-capital of-Indonesia"
                    ]
                }
            }
        ]
    },
    "answer": "The Indonesian capital, Jakarta, is located on the island of Java. This is
        supported by the information that Jakarta is situated on the northwest coast of
        Java, as referenced in the provided data."
}
```

```
{
    "input_question": "In the country where ghost in the shell is born, who is the current
        head of the state?",
    "logic_tree": {
        "children": [
            {
                "sub_question": "In which country was Ghost in the Shell created?",
                "hypothesis_answer": "Ghost in the Shell was created in Japan.",
                "children": [
                    {
                        "sub_question": "Who is the current head of state in Japan?",
                        "hypothesis_answer": "The current head of state in Japan is Emperor
                             Naruhito.",
                        "ref": {
                            "Wikipedia": [
                                "Naruhito_0_0||Naruhito (born 23 February 1960) is Emperor
                                    of Japan. He acceded to the Chrysanthemum Throne
                                    following his father's abdication on 1 May 2019,
                                    beginning the Reiwa era. He is the 126th monarch
                                    according to Japan's traditional order of succession.
                                    Naruhito is the elder son of Emperor Emeritus Akihito
                                    and Empress Emerita Michiko.",
                                "Japan_4_10||On May 1, 2019, after the historic abdication
                                    of Emperor Akihito, his son Naruhito became Emperor,
                                    beginning the Reiwa era."
                            ],
                            "Wikidata": [
                                "Japan-head of state-Naruhito",
                                "Naruhito-noble title-Emperor of Japan",
                                "Naruhito-position held-Emperor of Japan"
                            ]
                        }
                    }
                ],
                "ref": {
                    "Wikipedia": [
                        "Ghost in the Shell_0_0||Ghost in the Shell is a Japanese cyberpunk
                             media franchise based on the manga series of the same name
                            written and illustrated by Masamune Shirow. The manga, first
                            serialized between 1989 and 1991, is set in mid-21st century
                            Japan and tells the story of the fictional counter-
                            cyberterrorist organization Public Security Section 9, led by
                            protagonist Major Motoko Kusanagi. Animation studio Production
                            I.G has produced several anime adaptations of the series."
                    ],
                    "Wikidata": [
                        "Ghost in the Shell-country of origin-Japan"
                    ]
                }
            }
        ]
    },
    "answer": "The current head of state in Japan, where *Ghost in the Shell* was created,
        is **Emperor Naruhito**. He acceded to the Chrysanthemum Throne on May 1, 2019,
        following the abdication of his father, Emperor Akihito."
}
```


Appendix C. Analysis of failure cases

To gain a deeper insight into the root causes of errors, we have identified a specific class of problems that proves challenging for most existing methods, including our own. We term this category "list-intersection problems." These are multi-hop questions where the answer to an intermediate step is a long list (e.g., the cast of a movie, the discography of a singer). A subsequent hop then requires processing or verifying items from this list, significantly increasing the problem's complexity. Consider the example: "Who starred in Bird Box and also studied acting at the William Esper Studio?" As shown in Figs. C.7 and C.8, the CoK, despite its dynamic knowledge adaptation, often generates a rationale that fails to guide the retrieval process effectively. Consequently, the process may terminate prematurely at the second hop, retaining an incorrect answer. ToG-2 also exhibits a

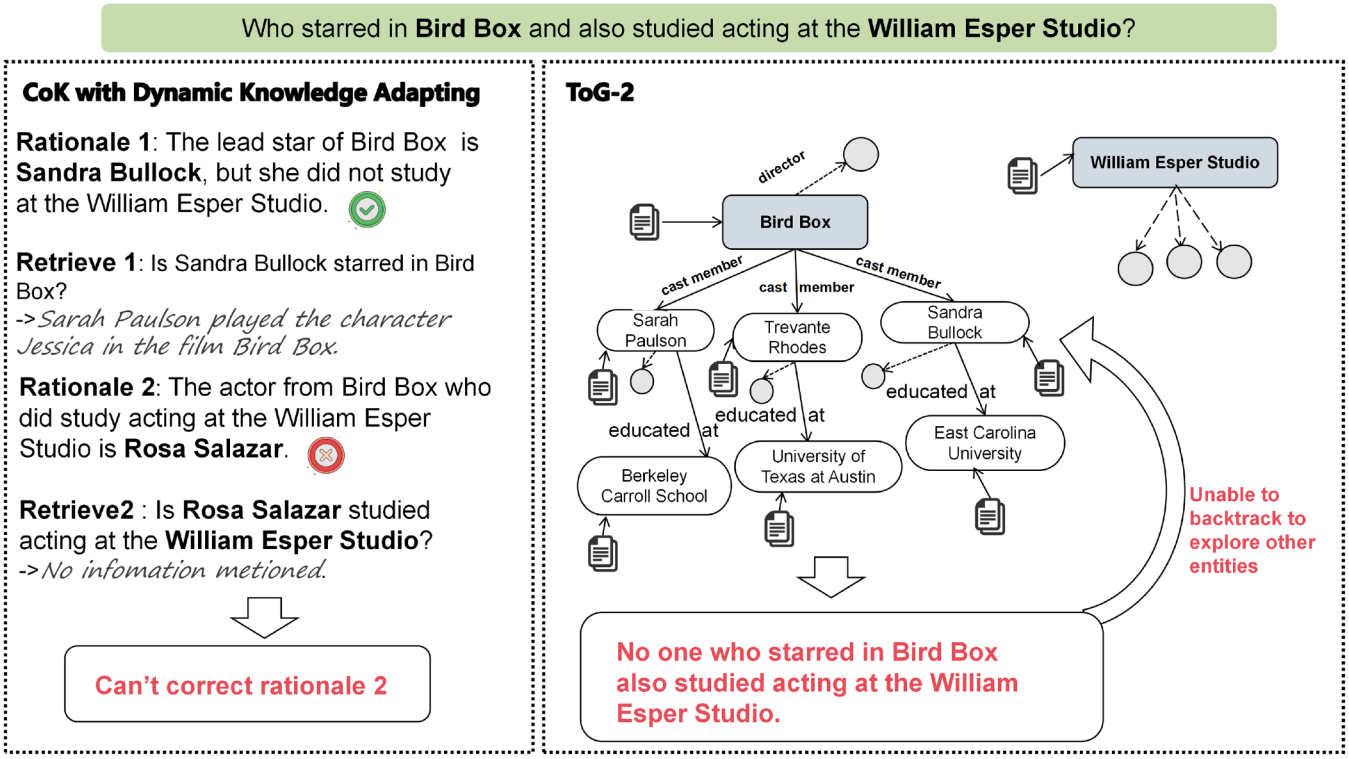


Fig. C.7. Failure Cases of CoK and ToG-2.

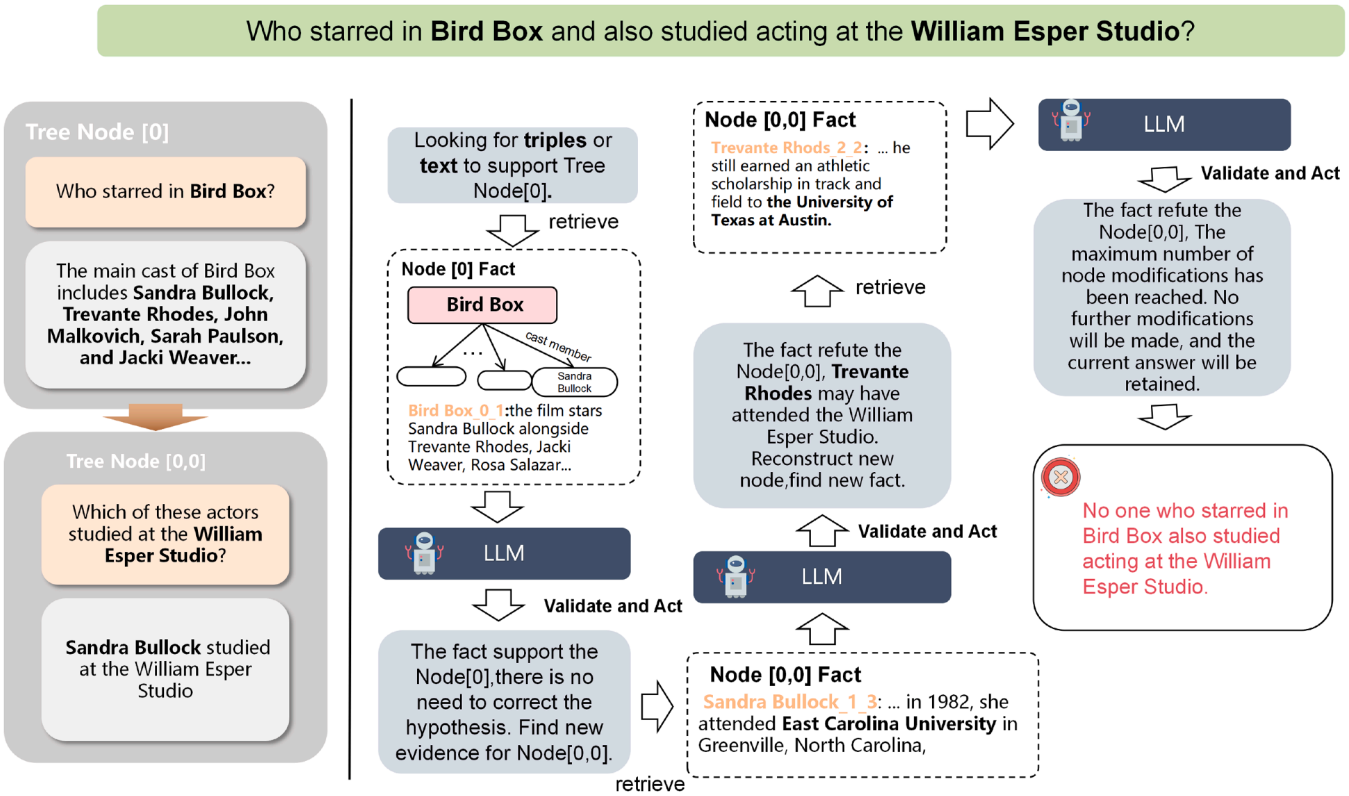


Fig. C.8. Failure Cases of Our Method (TreeQA).

significant limitation. Constrained by a fixed exploration depth and breadth, it halts exploration after examining just a few initial nodes (e.g., Sarah Paulson, Trevante Rhodes, Sandra Bullock). Even with an LLM for branch pruning, the model cannot backtrack to explore other entities once it reaches its predefined limits. This inability to backtrack is a critical flaw for this problem category. Our proposed method is also susceptible to a similar issue. In an extreme case, the model might incorrectly assume with high confidence that one entity (Sandra Bullock) satisfies the second constraint. It then pursues this single line of inquiry, but after two retrieval attempts yield no supporting evidence, the process halts and retains the incorrect conclusion. (In a more typical scenario, the model would create multiple branches to verify each actor from the list.)

References

[1] V. Mavi, A. Jangra, A. Jatowt, Multi-hop question answering, Found. Trends® Inf. Retr. 17 (5) (2024) 457–586. https://doi.org/10.1561/1500000102

[2] J. Feng, R. Xu, J. Hao, H. Sharma, Y. Shen, D. Zhao, W. Chen, Language models can be deductive solvers, in: Findings of the Association for Computational Linguistics: NAACL 2024, 2024, pp. 4026–4042. https://doi.org/10.18653/v1/2024.findings-naacl.254

[3] T. Kojima, S.S. Gu, M. Reid, Y. Matsuo, Y. Iwasawa, Large language models are zero-shot reasoners, Adv. Neural Inf. Process. Syst. 35 (2022) 22199–22213.

[4] H. Gu, K. Zhou, X. Han, N. Liu, R. Wang, X. Wang, PokeMQA: programmable knowledge editing for multi-hop question answering, in: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2024, pp. 8069–8083. https://doi.org/10.18653/V1/2024.ACL-LONG.438

[5] D. Chen, W.-T. Yih, Open-domain question answering, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts, 2020, pp. 34–37. https://doi.org/10.18653/v1/2020.acl-tutorials.8

[6] E. Kamalloo, N. Dziri, C. Clarke, D. Rafiei, Evaluating open-domain question answering in the era of large language models, in: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2023, pp. 5591–5606. https://doi.org/10.18653/v1/2023.acl-long.307

[7] J.-Y. Yao, K.-P. Ning, Z.-H. Liu, M.-N. Ning, Y.-Y. Liu, L. Yuan, LLM Lies: Hallucinations are not Bugs, but Features as Adversarial Examples, 2024. arXiv:2310.01469

[8] L. Huang, W. Yu, W. Ma, W. Zhong, Z. Feng, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, T. Liu, A survey on hallucination in large language models: principles, taxonomy, challenges, and open questions, ACM Trans. Inf. Syst. 43 (2) (2025) 42:1–42:55. https://doi.org/10.1145/3703155

[9] P. Zhao, H. Zhang, Q. Yu, Z. Wang, Y. Geng, F. Fu, L. Yang, W. Zhang, J. Jiang, B. Cui, Retrieval-Augmented Generation for AI-Generated Content: A Survey, 2024. arXiv:2402.19473

[10] Z. Jiang, M. Sun, L. Liang, Z. Zhang, Retrieve, summarize, plan: advancing multi-hop question answering with an iterative approach, in: Companion Proceedings of the ACM on Web Conference 2025, 2025, pp. 1677–1686. https://doi.org/10.1145/3701716.3716889

[11] A.O.M. Saleh, G. Tur, Y. Saygin, SG-RAG: multi-hop question answering with large language models through knowledge graphs, in: Proceedings of the 7th International Conference on Natural Language and Speech Processing, 2024, pp. 439–448.

[12] X. Li, R. Zhao, Y.K. Chia, B. Ding, S. Joty, S. Poria, L. Bing, Chain-of-knowledge: grounding large language models via dynamic knowledge adapting over heterogeneous sources, in: The Twelfth International Conference on Learning Representations, 2024. https://openreview.net/forum?id=cPgh4gWZlz.

[13] S. Ma, C. Xu, X. Jiang, M. Li, H. Qu, C. Yang, J. Mao, J. Guo, Think-on-Graph 2.0: Deep and Faithful Large Language Model Reasoning with Knowledge-guided Retrieval Augmented Generation, 2024. arXiv:2407.10805

[14] B. Garrette, C. Phelps, O. Sibony, B. Garrette, C. Phelps, O. Sibony, Structure the problem: pyramids and trees, in: Cracked it!, Springer, 2018, pp. 69–93.

[15] V. Mavi, A. Saparov, C. Zhao, Retrieval-augmented chain-of-thought in semi-structured domains, in: Proceedings of the Natural Legal Language Processing Workshop 2023, 2023, pp. 178–191. https://doi.org/10.18653/v1/2023.nllp-1.18

[16] H. Sun, B. Dhingra, M. Zaheer, K. Mazaitis, R. Salakhutdinov, W. Cohen, Open domain question answering using early fusion of knowledge bases and text, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018, pp. 4231–4242. https://doi.org/10.18653/v1/D18-1455

[17] H. Sun, T. Bedrax-Weiss, W. Cohen, PullNet: open domain question answering with iterative retrieval on knowledge bases and text, in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 2019, pp. 2380–2390. https://doi.org/10.18653/v1/D19-1242

[18] S. Kundu, T. Khot, A. Sabharwal, P. Clark, Exploiting explicit paths for multi-hop reading comprehension, in: Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, 2019, pp. 2737–2747. https://doi.org/10.18653/v1/P19-1263

[19] C. Liang, J. Berant, Q. Le, K.D. Forbus, N. Lao, Neural symbolic machines: learning semantic parsers on freebase with weak supervision, in: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2017, pp. 23–33. https://doi.org/10.18653/v1/P17-1003

[20] A. Talmor, J. Berant, The web as a knowledge-base for answering complex questions, in: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), 2018, pp. 641–651. https://doi.org/10.18653/v1/N18-1059

[21] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. Cohen, R. Salakhutdinov, C.D. Manning, HotpotQA: a dataset for diverse, explainable multi-hop question answering, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, 2018, pp. 2369–2380. https://doi.org/10.18653/v1/D18-1259

[22] W.-T. Yih, M. Richardson, C. Meek, M.-W. Chang, J. Suh, The value of semantic parse labeling for knowledge base question answering, in: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), 2016, pp. 201–206. https://doi.org/10.18653/v1/P16-2033

[23] X. Ho, A.-K. Duong Nguyen, S. Sugawara, A. Aizawa, Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps, in: Proceedings of the 28th International Conference on Computational Linguistics, 2020, pp. 6609–6625. https://doi.org/10.18653/v1/2020.coling-main.580

[24] W. Fan, Y. Ding, L. Ning, S. Wang, H. Li, D. Yin, T. Chua, Q. Li, A survey on RAG meeting LLMs: towards retrieval-augmented large language models, in: Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, 2024, pp. 6491–6501. https://doi.org/10.1145/3637528.3671470

[25] B. Peng, Y. Zhu, Y. Liu, X. Bo, H. Shi, C. Hong, Y. Zhang, S. Tang, Graph Retrieval-Augmented Generation: A Survey, 2024. arXiv:2408.08921

[26] L. Luo, Y. Li, G. Haffari, S. Pan, Reasoning on graphs: faithful and interpretable large language model reasoning, in: The Twelfth International Conference on Learning Representations, 2024. https://openreview.net/forum?id=ZGNWW7xZ6Q.

[27] J. Sun, C. Xu, L. Tang, S. Wang, C. Lin, Y. Gong, L.M. Ni, H. Shum, J. Guo, Think-on-graph: deep and responsible reasoning of large language model on knowledge graph, in: The Twelfth International Conference on Learning Representations, 2024. https://openreview.net/forum?id=nnVO1PvbTv.

[28] L. Chen, P. Tong, Z. Jin, Y. Sun, J. Ye, H. Xiong, Plan-on-graph: self-correcting adaptive planning of large language model on knowledge graphs, Adv. Neural Inf. Process. Syst. 37 (2024) 37665–37691.

[29] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R.O. Ness, J. Larson, From Local to Global: A Graph RAG Approach to Query-Focused Summarization, 2024. arXiv:2404.16130

[30] B.J. Gutiérrez, Y. Shu, Y. Gu, M. Yasunaga, Y. Su, HippoRAG: neurobiologically inspired long-term memory for large language models, Adv. Neural Inf. Process. Syst. 37 (2024) 59532–59569.

[31] Y. Hu, Z. Lei, Z. Zhang, B. Pan, C. Ling, L. Zhao, GRAG: graph retrieval-augmented generation, in: Findings of the Association for Computational Linguistics: NAACL 2025, 2025, pp. 4145–4157. https://doi.org/10.18653/V1/2025.FINDINGS-NAACL.232

[32] H. Trivedi, N. Balasubramanian, T. Khot, A. 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), 2023, pp. 10014–10037. https://doi.org/10.18653/V1/2023.ACL-LONG.557

[33] A. Asai, Z. Wu, Y. Wang, A. Sil, H. Hajishirzi, Self-RAG: learning to retrieve, generate, and critique through self-reflection, in: The Twelfth International Conference on Learning Representations, 2024. https://openreview.net/forum?id=hSyW5go0v8.

[34] S. Jeong, J. Baek, S. Cho, S.J. Hwang, J. Park, Adaptive-RAG: learning to adapt retrieval-augmented large language models through question complexity, in: Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), 2024, pp. 7036–7050. https://doi.org/10.18653/V1/2024.NAACL-LONG.389

[35] S. Holtel, How to crack complex, ill-defined, nonimmediate problems by issue trees: McKinsey on a shoestring: simple patterns for root cause analysis, in: Proceedings of the 28th European Conference on Pattern Languages of Programs, 2024, pp. 16:1–16:11. https://doi.org/10.1145/3628034.3628050

[36] Y. Cui, H. Yu, X. Guo, H. Cao, L. Wang, RAKCR: reviews sentiment-aware based knowledge graph convolutional networks for personalized recommendation, Expert Syst. Appl. 248 (2024) 123403.

[37] Z. Huang, Z. Sun, J. Liu, Y. Ye, Group-aware graph neural networks for sequential recommendation, Inf. Sci. 670 (2024) 120623. https://doi.org/10.1016/J.INS.2024.120623

[38] X. Yang, Y. Wang, J. Chen, W. Fan, X. Zhao, E. Zhu, X. Liu, D. Lian, Dual test-time training for out-of-distribution recommender system, IEEE Trans. Knowl. Data Eng. 37 (6) (2025) 3312–3326.

[39] L. Sun, X. Wang, Y. Li, Pyramid-Driven Alignment: Pyramid Principle Guided Integration of Large Language Models and Knowledge Graphs, 2024. arXiv:2410.12298

[40] D. Vrandecic, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Commun. ACM 57 (10) (2014) 78–85. https://doi.org/10.1145/2629489

[41] D.N. Milne, I.H. Witten, Learning to link with wikipedia, in: Proceedings of the 17th ACM Conference on Information and Knowledge Management, 2008, pp. 509–518. https://doi.org/10.1145/1458082.1458150

[42] A. Perevalov, D. Diefenbach, R. Usbeck, A. Both, QALD-9-plus: a multilingual dataset for question answering over DBpedia and wikidata translated by native speakers, in: 2022 IEEE 16th International Conference on Semantic Computing (ICSC), 2022, pp. 229–234. https://doi.org/10.1109/ICSC52841.2022.00045

[43] X. Ye, G. Durrett, The unreliability of explanations in few-shot prompting for textual reasoning, Adv. Neural Inf. Process. Syst. 35 (2022) 30378–30392.

[44] A. Bordes, N. Usunier, S. Chopra, J. Weston, Large-scale Simple Question Answering with Memory Networks, 2015. arXiv:1506.02075

[45] R. Orlando, P.H. Cabot, E. Barba, R. Navigli, ReLiK: retrieve and LinK, fast and accurate entity linking and relation extraction on an academic budget, in: Findings of the Association for Computational Linguistics, ACL 2024, 2024, pp. 14114–14132. https://doi.org/10.18653/V1/2024.FINDINGS-ACL.839
[46] C. Lee, R. Roy, M. Xu, J. Raiman, M. Shoeybi, B. Catanzaro, W. Ping, NV-embed: improved techniques for training LLMs as generalist embedding models, in: The Thirteenth International Conference on Learning Representations, 2025. https://openreview.net/forum?id=lgsyLSsDRe.
[47] DeepSeek-AI, DeepSeek-V3 Technical Report, 2024. arXiv:2412.19437
[48] OpenAI, GPT-3.5 and GPT-3.5-turbo models, 2023, https://platform.openai.com/docs/models/gpt-3-5.
[49] Qwen Team, Qwen2.5 Technical Report, 2024. arXiv:2412.15115
[50] OpenAI, GPT-4o mini: Advancing cost-efficient intelligence, 2025, https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence. Accessed: 2025-07-22.
[51] X. Wang, J. Wei, D. Schuurmans, Q.V. Le, E.H. Chi, S. Narang, A. Chowdhery, D. Zhou, Self-consistency improves chain of thought reasoning in language models, in: The Eleventh International Conference on Learning Representations, 2023. https://openreview.net/pdf?id=1PL1NIMMrw.
[52] D. Diefenbach, T.P. Tanon, K.D. Singh, P. Maret, Question answering benchmarks for wikidata, in: Proceedings of the ISWC 2017 Posters & Demonstrations and Industry Tracks co-located with 16th International Semantic Web Conference, 2017. https://ceur-ws.org/Vol-1963/paper555.pdf.
[53] D. Zhang, J. Li, Z. Zeng, F. Wang, Jasper and Stella: distillation of SOTA embedding models, 2025. arXiv:2412.19048
[54] N. Reimers, I. Gurevych, all-MiniLM-L6-v2: A Compact and Efficient Sentence Embedding Model, 2021, https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2.