StruQTO: Structure-aware Query Tree Optimization for Complex Logical Query over Knowledge Graphs

Yuyin Chen [1] and Guanfeng Li [1,2,*]

[1] School of Information Engineering, Ningxia University, Yinchuan, Ningxia, China
[2] Ningxia Key Laboratory of Artificial Intelligence and Information Security for Channeling Computing Resources from the East to the West, Yinchuan, Ningxia, China
Email: 12023131985@stu.nxu.edu.cn (Y.C.); Ligf@nxu.edu.cn (G.L.)
*Corresponding author

***Abstract*—Existing methods for complex logical query answering over knowledge graphs fall into two categories. The first category relies on explicitly engineered logical operators, which suffer from pronounced error accumulation when handling complex query structures. The second category directly delegates reasoning to Large Language Models (LLMs), often leading to disorganized reasoning trajectories and logical hallucinations. To address these issues, we propose Structure-aware Query Tree Optimization (StruQTO), a structure-centric approach that foregrounds query-structure dependencies for complex logical querying. StruQTO parses a First-Order Logic (FOL) query into a Query Computation Tree (QCT) and conducts local-independence analysis among substructures based on variable-dependency patterns, thereby constructing a Structured Chain-of-Thought (SCoT) that is strictly aligned with the original logical expression. The method explicitly distinguishes weakly coupled and strongly coupled substructures, guiding the LLM to execute reasoning in the query's true logical order while tightly constraining variable binding and set-theoretic operations throughout inference. Experiments on two public benchmarks, FB15K-237 and NELL-995, show that StruQTO achieves competitive or superior performance across a wide range of complex query patterns, with particularly notable improvements on structurally complex and negation-related queries. Ablation studies further validate the contributions of QCT construction and local-independence analysis to the overall performance gains.**

***Keywords*—knowledge graphs, complex logical query, Large Language Models (LLMs), query computation trees**

I. Introduction

Knowledge Graphs (KGs)—such as Freebase, YAGO, and Wikipedia—have become a foundational substrate for a wide range of intelligent systems, including information retrieval, question answering, recommender systems, knowledge reasoning, and decision support. A KG is typically represented as a collection of factual triples (h,r,t), where h, r, and t denote the head entity, relation, and tail entity, respectively. Owing to their structured representation of real-world facts, KGs have been extensively used in practical applications such as structured querying, dialogue generation, and machine reading comprehension. Complex logical query answering is a central problem in KG reasoning, aiming to infer the set of target entities that satisfy a query specified by compositional logical constraints over large-scale structured knowledge.

Complex logical querying stresses not only a model's ability to capture local KG semantics, but also its capacity to resolve deep logical dependencies spanning multiple entities and relations. As KGs are increasingly deployed in open-domain question answering, information extraction, and intelligent assistants, accurate reasoning over complex logical queries on graph-structured knowledge has emerged as a critical research focus. Conventional KG query models often rely on fixed embedding-space reasoning paradigms, and consequently exhibit limited capability in handling queries involving multi-hop dependencies, logical operators (e.g., intersection, union, and negation), existential quantification, and other compositional structures [1]. At the same time, Large Language Models (LLMs) have demonstrated strong performance in natural language understanding and reasoning, motivating growing interest in leveraging their linguistic reasoning ability to improve KG inference. However, complex logical queries are inherently highly structured and exhibit strong variable dependencies; naively delegating such reasoning to LLMs frequently yields disorganized reasoning trajectories, unclear inference paths, and limited interpretability [2].

Although embedding-based methods and Large Language Model (LLM) approaches have made notable progress on complex logical query tasks in recent years, both still exhibit substantial limitations when handling queries with multi-branch structures and shared-variable constraints. Specifically, embedding-based methods

Manuscript received March 25, 2026; revised May 19, 2026; accepted July 3, 2026; published September 24, 2026.

typically approximate discrete logical operations in continuous vector spaces. When multiple logical branches in a query share variable or must satisfy multiple constraints simultaneously, the semantic information from different branches is prone to becoming entangled during projection and aggregation, thereby weakening the model's ability to represent constraints accurately. In contrast, although LLM-based methods possess strong language understanding and reasoning capabilities, their reasoning process in scenarios involving multi-branch logical composition and variable binding relies heavily on implicit contextual modeling. As a result, they are susceptible to issues such as inconsistent branch-level reasoning and drift in variable reference, which in turn undermines the overall stability and logical consistency of the reasoning process.

To address the inconsistency and instability of reasoning in queries involving multi-branch structures and shared-variable constraints, recent studies have begun to explore structured chain-of-thought approaches to guide large language models through logical reasoning in a step-by-step manner [3], with the goal of explicitly decomposing complex queries into a series of controllable intermediate reasoning processes. However, existing structured chain-of-thought methods largely rely on predefined linear decomposition rules or manually designed reasoning templates. As a result, their reasoning chains typically assume that queries can be processed sequentially. When faced with complex queries involving multi-branch logical compositions and dependencies among multiple variables, these methods often fail to accurately capture the relationships among branches and their associated constraints, which in turn leads to insufficient logical consistency across reasoning steps and significantly degrades final reasoning performance. Therefore, a key challenge in improving model performance on complex logical queries is how to generate reasoning chains that are accurate, interpretable, and hierarchical in an adaptive manner based on the structural characteristics of the queries themselves [4].

In this study, inspired by the core idea of Query Computation Tree Optimization (QTO), which emphasizes the structural optimization of query computation trees, we propose a novel logical query decomposition method called StruQTO. Specifically, a complex logical query is represented as a computation tree, where the variable-sharing structure among subtrees determines the independence and composability of local reasoning processes. We introduce this structural perspective into the construction of reasoning chains: by analyzing the variable dependency relationships among subexpressions in a query, we identify subtrees that can be processed independently as well as strongly coupled chain-like structures, and then generate hierarchical reasoning steps that are strictly aligned with the underlying query logic. Experimental results show that the structured reasoning chain based on the QTO principle not only significantly improves the performance of language models on complex query tasks, but also demonstrates strong generalization ability across different model scales and different knowledge graph structures.

II. Literature Review

A. Complex Logical Query Answering Based on Knowledge Graph Embeddings

Before the advent of large language model-based approaches, complex logical query answering primarily relied on embedding-based neural reasoning frameworks. These methods represent queries and entities in a continuous vector space and realize logical operators—such as projection, intersection, union, and negation—via differentiable computations in that space, thereby enabling end-to-end learning for complex logical querying. Depending on how the embedding space is structured and how logical semantics are instantiated, existing approaches can be broadly categorized into three families: geometric-object methods, probabilistic-distribution methods, and fuzzy-logic methods. Despite their differences, all three paradigms share a common principle: mapping logical structures into an embedding space and performing differentiable reasoning, which constitutes the dominant technical route for non-LLM complex logical query answering.

1) Geometric-object methods

Geometric approaches treat a query as a spatial region in an embedding space, using geometric structures to capture multi-solution answer sets as well as logical operations and set-theoretic relations. Representative models have evolved progressively from points → boxes → higher-order geometric objects. Early geometric-object methods construct geometric regions to represent the query answer set, constituting one of the first systematic routes to differentiable logical operators. Point-based embeddings (e.g., GQE [5]) represent a query as a single vector and realize projection via geometric transformations; however, they struggle to model set-valued answers and have difficulty maintaining logical closure for disjunction and negation. To improve expressiveness, box-based embeddings (e.g., Query2box [6]) model a query region as a high-dimensional axis-aligned box, allowing operations such as intersection and union to be implemented naturally in geometric space. This substantially increases representational capacity for complex logical structures and has inspired a series of extensions—such as NewLook [7], Query2Geom [8], and RotatE-Box [9]—to support richer compositional logic and nonlinear mappings. More recently, higher-order geometric formulations (e.g., the cone representation in ConE [10] and the hyperbolic modeling in HypE [11]) aim to build a more closed algebra of logical operations within geometric space, and are among the first geometric methods to simultaneously support the three core operators $\wedge$, $\vee$, and $\neg$. Nonetheless, while high-dimensional geometric constructions enhance logical expressiveness, they also introduce practical challenges, including increased computational cost, sensitivity to negation, and

pronounced error accumulation when reasoning over complex query graphs.

2) Probabilistic-distribution methods

Probabilistic-distribution methods model the uncertainty of entities and queries using continuous probability distributions, transforming logical operations into parametric transformations of distributional representations (e.g., mean/variance or shape parameters). Early Gaussian-based models (e.g., GaussianPath [12] and PERM [13]) are effective for path reasoning and uncertainty modeling, yet they often struggle to guarantee the closure of logical operators. A major milestone in this line of work is BetaE [14], which exploits the natural closure properties of the Beta distribution over [0,1] to provide a fully differentiable realization of a complete operator set, including ∃, ∧, ∨, and ¬, making it one of the most logically comprehensive probabilistic approaches. Subsequent models such as GammaE [15] adopt the Gamma distribution to improve the efficiency of compositional operations, while NMP-QEM [16] employs mixture-of-Gaussians to capture multimodal answer sets and defines logical operators via neural networks, enabling complex logical querying to better accommodate the multi-peaked and irregular answer distributions commonly observed in real-world knowledge graphs. Overall, probabilistic methods offer strong expressive power and a natural treatment of ambiguity; however, they typically incur higher computational overhead, require carefully crafted operator designs, and may face scalability limitations on large-scale graphs.

3) Fuzzy-logic methods

Fuzzy-logic approaches draw on fuzzy set theory in its formal mathematical sense, treating a query as a membership function over entities and instantiating logical reasoning in the embedding space via operators such as T-norms, S-norms, and fuzzy negation. A representative model, EmQL [17], implements ∃, ∧, and ∨ using standard fuzzy operators, providing a differentiable formulation that captures the continuity of logical semantics. To alleviate limitations in expressive power, subsequent methods—including CQD [18], LogicE [19], and FuzzQE [20]—combine fuzzy logic with geometric operations, neural parameterizations, or link-prediction components, thereby enhancing interpretability and generalization of logical operators. More recent work (e.g., TFLEX [21], GNN-QE [22], ENeSy [23], and Var2Vec [24]) further extends fuzzy-logic reasoning to settings such as temporal/dynamic knowledge graphs, multimodal graphs, and large-scale query topologies, substantially improving the capacity to model complex structures. Despite these strengths, the performance of fuzzy-logic approaches remains highly sensitive to the choice of membership functions, and robust adaptation to complex topologies and temporal structures often requires additional enhancement modules.

Overall, geometric-object methods emphasize the geometric intuition of spatial regions and set operations; probabilistic-distribution methods highlight uncertainty modeling and probabilistic representations of soft boundaries; and fuzzy-logic methods inherit natural closure properties from formal logical systems. Together, these three families constitute the core landscape of non-LLM CLQA techniques, yet they also share common challenges, including complex operator instantiation, pronounced error accumulation under compositional reasoning, and limited engineering efficiency. These limitations, in turn, motivate and contextualize the emergence of LLM-driven approaches to complex logical query answering.

B. Complex Logical Query Answering with Large Language Models

Compared with traditional embedding-based approaches, Large Language Models (LLMs) have recently introduced a new paradigm for Complex Logical Query Answering (CLQA). Instead of explicitly implementing logical operators in a predefined embedding space—as in geometric or probabilistic frameworks—LLMs leverage large-scale pretraining to acquire strong language understanding and contextual reasoning capabilities, enabling complex logical queries to be reformulated as natural-language reasoning or structured semantic parsing problems. The central idea is to exploit LLMs' abilities in linguistic reasoning, rule abstraction, and multi-hop comprehension to dynamically execute logical operations such as projection, intersection, union, and negation in an end-to-end manner. Early studies indicate that GPT-style models can already handle multi-hop question answering and simple logical reasoning in a zero-shot setting, laying the groundwork for applying LLMs to knowledge graph querying.

Subsequent research has developed along several key directions, including structure alignment, explicit logical grounding, hallucination mitigation, and graph-knowledge injection. For example, methods such as ChatRule [25] and KG-Transformer [26] enhance controllable reasoning by synthesizing logical rules or adopting mask-based logic pretraining, so that logical structures are incorporated into the model in an explicit form. Approaches like ChatKBQA [27], SP-LLM [28], and UniOQA [29] adopt a generate–retrieve paradigm that couples LLM-based natural language generation with entity/relation retrieval modules, thereby improving both interpretability and structural consistency when mapping natural language into structured queries (e.g., SPARQL or CQL). Methods such as StructGPT [30] and ReasoningLM [31] introduce graph topology more directly—via graph serialization, graph-aware attention, or subgraph augmentation—so as to strengthen the model's ability to identify multi-hop relational paths. To alleviate hallucinations in open-domain reasoning, works including LGOT [32], LARK [33], and FiDeLiS [34] incorporate mechanisms such as logical-chain decomposition, subproblem verification, and KG-based deductive validation, using multi-round reasoning and path constraints to improve stability and logical consistency.

Overall, LLM-based methods, benefiting from semantic flexibility and contextual understanding, relax the constraints of traditional embedding approaches that stem from closed-world logical modeling and hand-crafted

operator design, and are particularly appealing for natural-language queries, multimodal integration, and open-domain scenarios. Nevertheless, current LLM-centric solutions for CLQA commonly suffer from insufficient alignment between the reasoning process and the underlying query structure, limited explicit modeling of variable-dependency constraints, and consequently restricted reasoning consistency and controllability—issues that constitute a major bottleneck for robust LLM reasoning on complex logical queries.

III. Methodology

A. Problem Formulation

Given a knowledge graph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{T})$,where $\mathcal{E} = \{e_1, e_2, \ldots, e_n\}$ denotes the entity set, $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ denotes the relation set, and $\mathcal{T} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{T}$ is the set of factual triples. Each triple can be written as (h, r, t),where $h \in \mathcal{E}$ is the head entity, $r \in \mathcal{R}$ is the relation (edge) connecting the two entities, and $t \in \mathcal{T}$ is the tail entity.

Complex logical queries aim to retrieve and manipulate data stored in knowledge graphs, with their theoretical foundation rooted in a subset of First-Order Logic (FOL). The core objective of such queries is to respond to complex requests containing logical operations, which are formally expressed as first-order logical expressions. These expressions may encompass operators such as existential quantifiers, relational projections, intersections, unions, and negations. The process of answering complex logical queries requires attempting to utilize combinations of queries to match and obtain results that satisfy the requirements, such as:

$$q[v_?] = \exists v: q_1 \wedge q_2 \wedge \ldots \wedge q_n \tag{1}$$

or,

$$q[v_?] = \exists v: q_1 \vee q_2 \vee \ldots \vee q_n \tag{2}$$

where q denotes an FOL query. Eq. (1) represents the Conjunctive Normal Form (CNF), while Eq. (2) represents the Disjunctive Normal Form (DNF). These two forms can be mutually converted through De Morgan's laws. The goal of this research is to predict the final set of answer entities.

B. Query Computation Tree Representation and Variable-Dependency Modeling

This study proposes StruQTO, a structured model for complex logical query answering that combines query-structure dependency analysis with the reasoning capabilities of Large Language Models (LLMs). When executing complex logical queries over incomplete knowledge graphs, conventional approaches often rely on manually crafted linear decomposition heuristics, which struggle to capture the deep structural dependencies encoded in logical expressions. In contrast, directly prompting LLMs to generate reasoning chains tends to produce disordered inference steps and may even induce logical hallucinations that are inconsistent with the underlying knowledge graph.

To overcome these limitations, StruQTO introduces a structured query decomposition mechanism based on local-independence analysis over a Query Computation Tree (QCT). The method maps a complex logical query into a hierarchical reasoning plan that can be executed step by step by an LLM. Specifically, upon receiving a complex logical query, StruQTO first converts it into a QCT, then analyzes variable dependencies across subtrees to identify independently solvable substructures versus strongly coupled chain-like structures, and finally automatically constructs a structure-aligned and semantically consistent multi-step reasoning chain (as illustrated in Fig. 1). By effectively integrating the structural expressiveness of computation trees with the natural-language reasoning strengths of LLMs, StruQTO substantially improves the logical consistency, interpretability, and accuracy of reasoning chains for complex logical query answering.

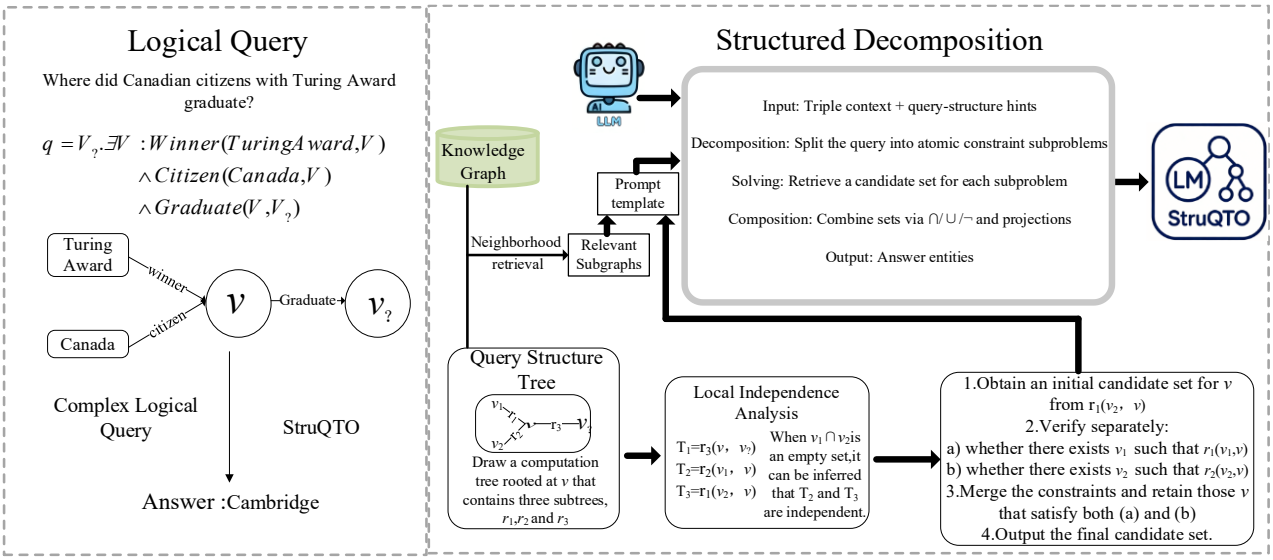

Fig. 1. Model overview.

To enable a structured decomposition of complex logical queries, we first parse the original First-Order Logic (FOL) query into a Query Computation Tree (QCT).

Our work is inspired by the query computation tree representation introduced in QTO. However, the role of the computation tree in StruQTO is fundamentally different from that in QTO. QTO employs the computation tree as an optimization structure for exact query answering in embedding-based reasoning. In contrast, StruQTO uses the computation tree as a structural representation for guiding LLM reasoning. Specifically, we inherit the query decomposition capability of QCT, while introducing (1) local-independence analysis for dependency-aware structural decomposition, and (2) structured chain-of-thought generation that converts query structures into executable reasoning sequences for LLMs. Therefore, the contribution of StruQTO lies not in proposing a new computation-tree representation, but in transforming query structures into reasoning guidance for LLM-based logical query answering.

The QCT is a hierarchical representation that is organized around variable-dependency structure and connected via logical operators, thereby providing a faithful structural abstraction of the query's semantic composition. Constructing the QCT constitutes the theoretical backbone of our approach: it establishes a unified, formal scaffold for query semantics, upon which subsequent local-independence analysis and structured reasoning-chain construction can be conducted in a principled and consistent manner.

To enable a structured decomposition of complex logical queries, we first parse the original first-order logic (FOL) query into a Query Computation Tree (QCT). The QCT is a hierarchical representation that is organized around variable-dependency structure and connected via logical operators, thereby providing a faithful structural abstraction of the query's semantic composition. Constructing the QCT constitutes the theoretical backbone of our approach: it establishes a unified, formal scaffold for query semantics, upon which subsequent local-independence analysis and structured reasoning-chain construction can be conducted in a principled and consistent manner.

Given a first-order logic query q, its basic semantic units include a variable set $\mathcal{V}$, an anchor entity set $\mathcal{A}$,and a relation set $\mathcal{R}$. Each relational operation can be represented as a binary predicate $r(u, v)$, where $u, v \in \mathcal{V} \cup \mathcal{A}$. Additionally, the query may contain logical connectives including conjunction (∧), disjunction (∨), and negation (¬).To achieve structured parsing, the raw query expression is first transformed into a semantic tree based on function composition, where nodes represent individual logical operations and edges represent dependency relationships.

Formally, the query computation tree is represented as a triple:

$$T = (V, E, O) \tag{3}$$

where V denotes the set of tree nodes, encompassing all variable nodes and anchor entity nodes; $E \subseteq V \times V$ represents the dependency edges between nodes; and O denotes the logical operation mapping applied to each node. For each node $v \in V$, its operation type $\mathrm{O}(v)$ can be one of the following four categories: relational projection operation with $\mathrm{O}(v) = r$; set intersection operation with $\mathrm{O}(v) = \bigcap$; set union operation with $\mathrm{O}(v) = \bigcup$; and negation operation with $\mathrm{O}(v) = \neg r$. Each QCT node possesses the following attributes: Type(v)= $\mathrm{O}(v)$ indicates the node type; vars(v) represents the set of variables appearing in the subexpression corresponding to this node; and out(v) denotes the candidate entity set computed by this node.

The semantics of QCT is defined using set operations (with the goal of returning an answer set). If v is a relation node with graph pattern $\mathcal{G} = (h, r, t)$, and h is a bound entity set, then:

$$\mathrm{out}(v) = \{\exists r \in R,\ (h, r, t) \in g\} \tag{4}$$

Intersection can be expressed as:

$$\mathrm{out}\big(\wedge\ (v_1, \ldots, v_k)\big) = \bigcap_{i=1}^{k} \mathrm{out}\ (v_i) \tag{5}$$

Union can be expressed as:

$$\mathrm{out}\big(\vee\ (v_1, \ldots, v_k)\big) = \bigcup_{i=1}^{k} \mathrm{out}\ (v_i) \tag{6}$$

Negation can be expressed as (taking the complement over the candidate universe Ω):

$$\mathrm{out}\big(\neg(v)\big) = \Omega \setminus \mathrm{out}(v) \tag{7}$$

where the value of Ω in implementation adopts the "upper-bound candidate set", such as the candidate set generated by upper-level parent nodes or the retrieval phase (avoiding the set explosion caused by taking the complement over the entire graph).

The variable set is defined as the collection of all logical variables appearing in the subtree rooted at this node. It is computed recursively as follows:

$$vars(v) = \begin{cases} \emptyset, & \mathrm{type}(v) = \mathrm{ENT} \\ \{x\}, & \mathrm{type}(v) = \mathrm{VAR} \\ \bigcup_{u \in \mathrm{child}(v)} \mathrm{vars}\ (u), & \text{otherwise} \end{cases} \tag{8}$$

where the ENT node is a leaf node in the QCT, representing a given starting entity or intermediate fixed entity in the query. The VAR node represents a logical variable in the query, whose value is unknown and needs to be determined progressively through relational constraints.

The tree construction process is accomplished through a top-down recursive parsing function. For a given query q, its highest-level logical structure can be expressed as:

$$q = f(q_1, q_2, \ldots, q_k) \tag{9}$$

where f denotes the outermost logical operation (such as relational projection, intersection, union, etc.), and $q_1, \ldots, q_k$ are its corresponding sub-queries. A tree node v_f is created for the operation f, and a subtree T_i is recursively constructed for each sub-query q_i. The root node of each subtree is then connected to v_f via a dependency edge. The resulting tree topology strictly corresponds to the semantic composition of the query.

For each relational operation $r(u, v)$, if u or v is a variable, the dependency relationship is recursively parsed; if it is an anchor entity, it is directly added to the tree as a leaf node. This parsing approach ensures that all semantic components involved in the query are uniquely mapped to the tree structure.

The resulting query computation tree possesses semantic completeness, ensuring that all relations, variables, and logical operations in the query have unique and explicit representations within the tree structure. Meanwhile, this structure is inherently decomposable—each subtree strictly corresponds to a logical subexpression in the query, enabling the query to be partitioned into multiple semantically self-consistent local reasoning units. Furthermore, the query computation tree explicitly presents dependencies and coupling relationships between variables through its topological structure, thereby providing precise grounds for analyzing the independence of substructures. More importantly, the hierarchical organization of the tree structure aligns strictly in sequence and semantics with the subsequently generated structured chain-of-thought, offering a clear and stable structural template for natural language multi-step reasoning processes. Therefore, the construction of QCT is not only a necessary step for formal representation but also a critical foundation for ensuring that logical queries can be accurately decomposed and reliably executed by large language models. Relying on this structure, it becomes possible to further conduct local independence analysis and reasoning chain generation, thereby establishing a complete and consistent structured logical query reasoning framework.

C. Local-Independence Analysis

After constructing the query computation tree, the next critical task is to identify the semantic dependency strength between different substructures within the tree and determine the unfolding order for reasoning chain generation accordingly. Therefore, local independence analysis is performed based on variable sharing relationships to establish the expansion and merging order of the reasoning chain: if two subtrees do not share intermediate variables, they are semantically independent and can be reasoned separately; conversely, if multiple subtrees share critical variables, they constitute a strongly coupled structure that requires sequential processing following a logical chain order. To eliminate ambiguity, this study adopts the following operator precedence and scope conventions: negation ($\neg$) > conjunction ($\wedge$) > disjunction ($\vee$), with operators of the same precedence associating from left to right.

Given two child subtree root nodes u and v sharing the same parent node p in the QCT, their variable intersection is defined as:

$$I(u, v) = \text{vars}(I) \cap \text{vars}(v) \tag{10}$$

If $I(u, v) = \emptyset$, then subtrees u and v are said to be variable-level independent, which can be solved in parallel and subsequently merged at the parent node (through intersection or union). If $I(u, v) \neq \emptyset$, then they are termed coupled, requiring ordered resolution according to the binding relationships of shared variables.

During this resolution process, if multiple subtrees depend solely on the same upper-level variable from their parent node, while their internal variable sets are mutually disjoint, then these subtrees can be regarded as weakly coupled branches, whose semantic constraints are logically "parallel." Such subtrees can be presented in the reasoning chain in the form of "considering separately" or "reasoning simultaneously," enabling the language model to process sub-constraints in a parallelized manner. In contrast, when two subtrees share a critical variable—for instance, when multiple relations jointly constrain the value of the same variable—such structures are considered strongly coupled chains, requiring strict sequential unfolding of reasoning according to the depth order of the tree to ensure that variable binding relationships remain consistent in subsequent steps.

The purpose of local independence analysis lies not only in determining a reasonable order for the reasoning chain but also in providing the language model with highly readable, logically consistent structured reasoning templates. By explicitly identifying which substructures require sequential execution and which can be solved independently, the logical rhythm of reasoning chain generation can be explicitly controlled, making the reasoning steps more aligned with human reasoning patterns and further reducing the risk of hallucinations or skipped steps by large language models. Ultimately, this independence analysis establishes the semantic foundation for constructing structured reasoning chains, enabling the entire reasoning process to unfold naturally under variable consistency constraints while maintaining precise expressiveness for complex logical structures.

D. Structured Reasoning-Chain Construction

After completing the query computation tree construction and local independence analysis, StruQTO further maps the query structure into a structured reasoning chain executable by large language models. This reasoning chain represents the solution order of complex logical queries, variable binding relationships, and the composition manner between substructures in natural language form, thereby providing structural constraints for subsequent reasoning execution.

The structured data is transformed into natural language prompts, denoted as SCoT (Structured Chain-of-Thought), and fed into the large language model step by step.

Specifically, retrieved subgraphs are converted into context prompts, and each decomposed sub-query corresponds to the generation of corresponding reasoning prompts. Intermediate results generated by the large language model are stored in cache for reuse in subsequent reasoning steps. During prompt construction, the system employs placeholder markers to insert intermediate answers into subsequent prompts, ensuring the integrity and consistency of the reasoning chain.

The generation of the reasoning chain strictly follows the topological structure of the query computation tree. For each node in the computation tree, a corresponding reasoning step description is generated according to its operation type: relational projection nodes are transformed into relational constraint operations based on given entity sets; intersection and union nodes correspond to set merging of multiple sub-constraint results; negation nodes filter candidate entities through complement constraints. On this basis, the expansion order of reasoning steps is determined utilizing local independence analysis results. For subtree structures with non-overlapping variable sets, their corresponding reasoning steps can be regarded as mutually independent and presented in parallel form within the reasoning chain; for strongly coupled substructures sharing critical variables, chain-style reasoning steps are generated according to variable dependency order to ensure consistency of variable binding.

By recursively traversing the query computation tree and generating reasoning descriptions layer by layer, a structured reasoning chain strictly aligned with the original logical query structure is ultimately obtained. This reasoning chain maintains consistency with the query computation tree in both sequence and semantics, enabling large language models to execute complex logical queries step-by-step under explicit structural guidance. Based on the above reasoning chain generation results, the specific construction approach is illustrated in Fig. 2 and will be utilized for structured fine-tuning and reasoning input construction in the next section.

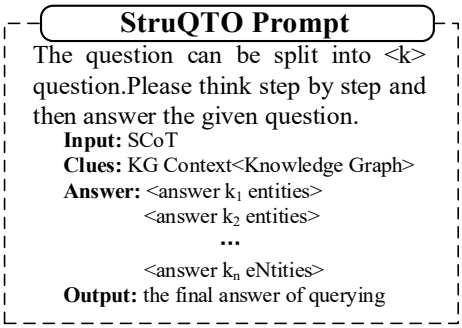

Fig. 2. StruQTO's prompt build.

E. LLM-Based Structured Fine-Tuning Framework

In StruQTO, complex logical query reasoning is uniformly modeled as a conditional text generation task. Given a structured complex logical query q and its corresponding knowledge graph context, the large language model is required to generate the final answer entity set under structural constraints. Formally, each training sample is represented as an input-output pair (S, A) ,where the input sequence S consists of three components:

$$S = D \oplus X \oplus T(q) \tag{11}$$

where D denotes the natural language or symbolic description of the query; X represents the set of knowledge graph context triples relevant to the query; and $\mathcal{T}(q)$ denotes the structured reasoning chain generated from the Query Computation Tree (QCT).

The output sequence A is the textual representation of the final answer entity set:

$$A = F(V_q) \tag{12}$$

where V_q denotes the set of target entities satisfying the complex logical query q.

To effectively constrain the reasoning space of large language models, StruQTO explicitly introduces knowledge graph context information in both fine-tuning and inference stages. Starting from anchor entities and intermediate variables involved in the query, the relevant triple set X is retrieved through a neighborhood traversal strategy with restricted depth, and concatenated to the model input in text form. This process ensures that the model's reasoning results are always anchored to the real knowledge graph structure, effectively alleviating the problem of hallucinations produced by large language models in complex reasoning scenarios.

Unlike directly requiring the model to generate answers, StruQTO employs the structured reasoning chain $\mathcal{T}(q)$ generated from the query computation tree as an explicit supervision signal for model fine-tuning. Complex logical queries are first parsed into query computation trees, and further decomposed through local independence analysis into a series of sub-reasoning steps with explicit variable binding orders. This decomposition result is linearized into a natural language-form structured reasoning chain, which guides the model to execute reasoning step-by-step according to the true logical structure of the query.

F. Reasoning Module

In the inference stage, given an unseen complex logical query q , StruQTO adopts a structured execution pipeline consistent with the training stage. First, the query is parsed into a query computation tree, and the structured reasoning chain $\mathcal{T}(q)$ is generated through local independence analysis. Subsequently, starting from anchor entities involved in the query, relevant knowledge graph triples are retrieved to construct the model input context. During the model decoding process, the large language model executes logical operations step-by-step under the constraints of the structured reasoning chain, including relational projection, set intersection and union, and negation constraints, while dynamically maintaining candidate entity sets for variables. For substructures with local independence, the model can generate intermediate reasoning results in parallel and merge them at upper-level

nodes; for strongly coupled chain structures, reasoning steps are executed strictly in sequence according to variable dependency order.

Finally, the model generates the candidate entity set for target variables based on all logical constraints and outputs the final answer. Since the entire reasoning process strictly aligns with the query computation tree structure, StruQTO maintains high logical consistency and reasoning interpretability in complex logical query scenarios.

IV. Result and Analysis

This section evaluates the performance of the proposed StruQTO on two benchmark datasets. We first briefly introduce the experimental setup, including experimental datasets, baseline methods, and evaluation metrics. Subsequently, the effectiveness of the proposed method is verified through performance comparison and analysis.

A. Training Settings

The experiments were conducted using 8 NVIDIA A100 Tensor Core GPUs for training. The open-source large model LLaMA was employed as the base model for fine-tuning. For the fine-tuning configuration, the AdamW optimizer was used to train the model for 1 epoch with a batch size of 128. The experimental environment was built upon PyTorch, a unified deep learning training framework, with Python as the development language.

1) Datasets

The model was evaluated on the benchmark datasets provided in BetaE [14], which comprise 14 logical query types on FB15k-237 [35] and NELL-995 [36]. The naming convention for query structure types in the BetaE datasets is as follows: *p* denotes relational projection, *i* denotes conjunction (intersection), *n* denotes negation (complement), and *u* denotes disjunction (union). Ten query structures were used for training and evaluation: 1*p*, 2*p*, 3*p*, 2*i*, 3*i*, 2*in*, 3*in*, *inp*, *pni*, and *pin*. To test the model's generalization capability, four query structures (*ip*, *pi*, 2*u*, *up*) were excluded from the training set and used for evaluation only. The dataset statistics are shown in Table I.

TABLE I. Datasets

Datasets	Entities	Relations	Training Edges	Validation Edges	Test Edges
NELL-995	63361	200	114213	14324	14267
FB15K-237	14505	237	272115	17526	20438

2) Baselines

To validate the performance of the proposed model, comparisons were conducted from the following two perspectives: (1) Embedding-based complex logical query methods, which define logical operators through various formulation approaches to perform queries, such as GQE [5], Query2box [6], and QTO [37]; and (2) Large language model-based complex logical query methods, which leverage the rich semantic and contextual information embedded in massive corpora to capture complex linguistic structures and achieve complex logical query answering, such as LARK [33], SILR [38], and PPMT [39].

GQE:Embeds graph queries into vector spaces by constructing embedding representations of query graphs, utilizing differentiable operations (such as intersection and projection) to simulate logical query paths, enabling the model to handle complex logical reasoning tasks in knowledge graphs.

Query2box: This model embeds queries as hyper-rectangles and answer entities as points within these rectangles.

QTO: Constructs query computation trees and performs forward-backward exact optimization, enabling efficient solving of complex logical queries on knowledge graphs without training while obtaining interpretable intermediate results.

LARK: Decomposes KG queries into reusable logical chains and performs sequential reasoning on abstract subgraphs through LLMs, significantly improving cross-graph and cross-task generalization capabilities on complex logical queries.

SILR: Linearly encodes logical structures and employs structural prompts with geometric operation modeling, enabling Pre-trained Language Models (PLMs) to generalize to complex logical queries with unseen entities/relations in inductive scenarios.

PPMT: Eliminates linguistic noise in serialized queries through a prefix memory correction mechanism, and enhances PLMs' modeling capability for complex logical semantics by combining a progressive conditional gradient update strategy.

3) Evaluation metrics

In this model, Mean Reciprocal Rank (MRR) is adopted as the primary evaluation metric to measure model performance. MRR is a metric used for ranking tasks, representing the average of the reciprocals of the ranks of correct answers in the query results. MRR measures whether the answer returned by the model is contained within the top-ranked answers. A good model will place the correct answer at higher ranks; the higher the MRR, the better the model performance. The specific calculation formula is:

$$\mathrm{MRR} = \frac{1}{Q}\sum_{i=1}^{|Q|}\frac{1}{\mathrm{rank}_i} \tag{13}$$

B. Experimental Results and Analysis

In the experiments, we first compare the proposed StruQTO with several baselines on both datasets. Subsequently, a set of ablation studies is conducted to investigate the impact of different model components on reasoning performance.

1) Main result

The proposed StruQTO was compared with baseline models on two publicly available KG datasets. Tables II and III present the performance comparison results between the proposed StruQTO method and baseline models on the two datasets, respectively.

TABLE II. THE MRR COMPARISON EXPERIMENTAL RESULTS OF VARIOUS MODELS ON DIFFERENT QUERY TYPES OF FB15K-237

Models	1p	2p	3p	2i	3i	ip	pi	2u	up	2in	3in	inp	pin	pni	Avg_e	Avg_n
GQE	35.0	7.2	5.3	23.3	34.6	10.7	16.5	8.2	5.7	-	-	-	-	-	16.3	-
Q2B	40.6	9.4	6.8	29.5	42.3	12.6	21.2	11.3	7.6	-	-	-	-	-	20.1	-
SILR	43.2	28.8	24.0	28.2	41.4	10.0	19.5	25.3	16.7	-	-	-	-	-	26.3	-
PPMT	43.5	30.5	25.2	32.2	43.6	15.2	23.5	29.7	20.2	-	-	-	-	-	29.3	-
QTO	49.0	21.4	21.2	43.1	56.8	28.0	38.1	22.7	21.4	16.8	26.7	15.1	13.6	5.4	33.5	15.5
LARK	**70.0**	36.9	24.5	44.3	43.1	23.2	**45.6**	**56.6**	25.4	7.0	4.1	**23.9**	**16.8**	3.5	41.1	11.1
StruQTO	62.4	**38.7**	**32.6**	**45.6**	**58.3**	**28.8**	43.3	42.9	**29.2**	**17.2**	25.3	20.5	15.2	**6.2**	**42.4**	**16.9**

TABLE III. THE MRR COMPARISON EXPERIMENTAL RESULTS OF VARIOUS MODELS ON DIFFERENT QUERY TYPES OF NELL-995

Models	1p	2p	3p	2i	3i	ip	pi	2u	up	2in	3in	inp	pin	pni	Avg_e	Avg_n
GQE	32.8	11.9	9.6	27.5	35.2	14.4	18.4	8.5	8.8	-	-	-	-	-	18.6	-
Q2B	42.2	14.0	11.2	33.3	44.5	16.8	22.4	11.3	10.3	-	-	-	-	-	22.9	-
SILR	52.1	28.9	31.2	33.0	47.8	8.0	19.1	33.8	18.7		-	-	-	-	29.9	-
PPMT	56.2	32.2	32.3	35.0	49.6	16.7	24.0	40.3	21.9	-	-	-	-	-	34.2	-
QTO	60.7	24.1	21.6	42.5	50.6	26.5	31.3	20.4	17.9	13.8	17.9	16.9	9.9	5.9	32.9	12.9
LARK	**83.2**	**42.3**	31.0	**49.9**	48.7	23.1	23.0	20.1	**37.2**	10.4	6.6	**25.4**	**13.6**	7.6	39.8	12.7
StruQTO	74.3	34.6	**33.3**	43.1	**50.3**	**27.4**	**32.9**	**39.6**	26.5	**14.3**	**18.5**	18.6	13.2	**7.9**	**40.2**	**14.5**

To better validate the performance of the proposed model, datasets with varying complexity, namely FB15K-237 and NELL-995, are introduced. Through comparative experiments across datasets with different levels of structural complexity, StruQTO demonstrates strong overall performance and remains competitive with state-of-the-art methods across a diverse range of query structures. Specifically, on the FB15K-237 dataset, StruQTO achieves improvements of 3.2% and 26.6% in MRR for EPFO queries (∃,∧,∨) over LARK and QTO, respectively, and an 18.1% improvement in MRR for queries with negation (¬) over QTO, demonstrating significant performance gains. On the NELL-995 dataset, StruQTO achieves improvements of 1.0% and 22.2% in MRR over LARK and QTO, respectively, along with a 12.4% improvement in MRR for queries with negation (¬) over QTO. The experiments conducted on these two distinct datasets collectively verify that the proposed StruQTO outperforms existing models on complex logical query answering tasks over knowledge graphs.

Based on the above comparative analysis, it can be observed that embedding-based complex logical query models, despite continuous improvements in query operator representation, still exhibit performance deficiencies. Among embedding methods, QTO achieves the best performance; its core idea involves constructing query computation trees and performing forward-backward exact optimization, enabling efficient solving of complex logical queries on knowledge graphs without training while obtaining interpretable intermediate results. Regarding large language model-based complex logical query approaches, this direction has emerged with the rapid development of pre-trained language models and large language models. Research explorations in this field primarily focus on how to combine the strong semantic capabilities of natural language understanding with the structured reasoning mechanisms of knowledge graphs to enhance the accuracy and generalization of complex logical reasoning. Although embedding-based complex logical query methods demonstrate excellent performance in efficiency and interpretability, they still exhibit certain limitations when handling high-dimensional complex logical queries and dynamic knowledge graphs. Therefore, the novel complex logical query structured approach proposed in this study, based on query structure dependency analysis and large language model reasoning capabilities, can better perform query tasks.

It is worth noting that the performance advantages of StruQTO are not uniformly distributed across all query structures. Compared with LARK, StruQTO exhibits lower performance on several relatively simple query types, including 1*p* and 2*u* on FB15K-237, as well as 1*p*, 2*p*, 2*i*, and *up* on NELL-995. These query structures are characterized by shallow reasoning depth, limited variable interactions, or relatively weak dependency constraints. In such scenarios, LARK's logical-chain decomposition strategy is already sufficient to capture the required reasoning process, and the additional structural constraints introduced by StruQTO may provide limited benefit.

In contrast, StruQTO demonstrates more substantial advantages on query structures involving deeper reasoning paths, stronger variable-sharing dependencies, and more complex logical compositions. Examples include 3*p*, 3*i*, *ip*, 2*in*, 3*in*, *pin*, and *pni*, where reasoning requires maintaining consistent variable bindings across multiple dependent branches or handling negation-related constraints. The Query Computation Tree and Local-Independence Analysis mechanisms explicitly model these structural dependencies and guide the reasoning process according to the underlying logical organization of the query, thereby reducing reasoning drift and improving logical consistency. These observations suggest that the primary strength of StruQTO lies in dependency-intensive and structurally complex reasoning tasks rather than simple projection-oriented queries. Therefore, the proposed framework should be viewed as a structure-aware reasoning method whose advantages become increasingly evident as query complexity and variable dependency increase.

2) Ablation study

To investigate the impact of different model components on StruQTO's performance, this section

conducts a set of ablation experiments on NELL-995. First, by removing the query computation tree construction and local independence analysis, and directly using LLM for querying, we explore their influence; the model abbreviation in this case is StruQTO-LLM. Then, by removing the structured decomposition in query computation tree construction and directly using the initial computation tree, denoted as StruQTO-T. Furthermore, to validate the effect of the independence analysis component, this part is ablated, with the model name StruQTO-TR. Table IV shows the performance comparison results of the ablation experiments.

TABLE IV. ABLATION EXPERIMENT RESULTS

Method	1p	2p	3p	2i	3i	ip	pi	2u	up	2in	3in	inp	pin	pni
StruQTO-LLM	60.1	20.3	21.8	39.7	43.5	24.8	28.5	21.4	18.3	10.3	12.5	13.4	7.8	5.4
StruQTO-T	62.4	23.1	22.3	42.7	47.6	25.3	31.4	22.5	20.3	12.2	16.7	16.4	10.8	6.5
StruQTO-TR	66.2	28.5	28.4	43.0	49.2	26.2	31.9	29.8	22.4	13.4	17.4	16.9	11.4	6.9
StruQTO	74.3	34.6	33.3	43.1	50.3	27.4	32.9	39.6	26.5	14.3	18.5	18.6	13.2	7.9

By ablating certain components in StruQTO, we found that each component in the model contributes to performance, demonstrating that all parts of the model are effective. On the NELL-995 dataset, by removing all pre-query components, the average performance across all query types dropped by 24.5% in terms of MRR. When these components were ablated and the model directly used large language models for querying, the overall model performance degraded. This proves that computation tree construction can better represent and implement queries, while also providing more accurate representations for query tasks. When ablating the structured decomposition and independence analysis components in query computation tree construction, the ablation experiments revealed that MRR performance on NELL-995 dropped by 17.1% and 9.6%, respectively. Compared to the independence analysis component, the query computation tree construction mechanism exhibits stronger influence, proving that although all components of the model contribute to some extent, the query computation tree construction module can better enhance the overall model performance. In summary, ablation experiments verify that each part of the model contributes to the final performance, demonstrating that the model components are not redundant or unnecessary.

3) Scalability and cross-model generalization

In complex logical query tasks, query structural complexity and reasoning context length often grow significantly with the number of logical operations and variable dependency relationships. Although StruQTO effectively constrains the reasoning order through query computation trees and structured reasoning chains, the context capacity of large language models may still become an important factor affecting overall performance.

To analyze the impact of model capacity on structured reasoning effectiveness, under the same StruQTO framework, language models of different scales were adopted as reasoning execution modules to evaluate complex logical queries. In the experiments, the query computation tree construction, local independence analysis, and reasoning chain generation methods remained unchanged; only the underlying LLM was replaced. In the experiments, the query computation tree construction, local independence analysis, and reasoning chain generation methods remained unchanged; only the underlying LLM was replaced.

As shown in Table V, experimental results indicate that with the improvement of model scale and context processing capability, StruQTO's performance on high-complexity query structures (such as multi-hop paths and queries containing negation) significantly increases, particularly in structures like 3*i*, 3*in*, and *pni*. This phenomenon demonstrates that structured reasoning chains can fully leverage the context modeling capabilities of stronger models, while current performance bottlenecks are primarily constrained by underlying model capacity rather than the method itself.

TABLE V. THE MRR RESULTS OF STRUQTO WITH LLAMA2-7B AND LLAMA2-13B AS BASE MODELS

Method	1p	2p	3p	2i	3i	ip	pi	2u	up	2in	3in	inp	pin	pni
LLaMA2-7B	51.2	30.1	21.2	38.4	39.3	18.4	33.7	28.6	20.1	10.2	12.3	13.9	7.4	4.6
Qwen2.5-7B	53.6	32.8	25.9	40.7	46.1	22.7	37.4	32.1	23.9	12.5	14.9	15.8	9.8	6.2
LLaMA2-13	**58.4**	**38.7**	**32.6**	**45.6**	**58.3**	**28.8**	**43.3**	**38.9**	**29.2**	**14.3**	**18.5**	**18.6**	**13.2**	**7.9**

To further investigate whether the effectiveness of StruQTO depends on a specific model family, we additionally evaluate the framework using Qwen2.5-7B under the same experimental setting. The results show that Qwen2.5-7B consistently outperforms LLaMA2-7B across most query structures, while exhibiting performance trends similar to those observed in the LLaMA family. In particular, the improvements are more pronounced on structurally complex queries such as 3i, ip, 3in, and pin, where maintaining dependency consistency and handling compositional reasoning are especially important. This observation suggests that the benefits of StruQTO are not tied to a particular language model architecture. Instead, the proposed Query Computation Tree and Local-Independence Analysis mechanisms provide a general structural reasoning framework that can be effectively utilized by different large language model families.

Overall, the results across both model scales and model families demonstrate that explicitly modeling query structures and constraining reasoning order can consistently improve logical reasoning performance.

These findings provide stronger evidence that the gains achieved by StruQTO originate from its structure-aware reasoning mechanism rather than characteristics specific to a particular language model, while also indicating promising potential for future integration with more advanced long-context and large-scale foundation models.

As shown in Fig. 3, a generation case of the StruQTO model is presented. By leveraging the knowledge retrieved from the knowledge graph and the complete logical chain for reasoning and querying, the model fully utilizes the provided relevant information and derives the correct answer, effectively reducing factual errors during the query process and thereby significantly enhancing reasoning capability.

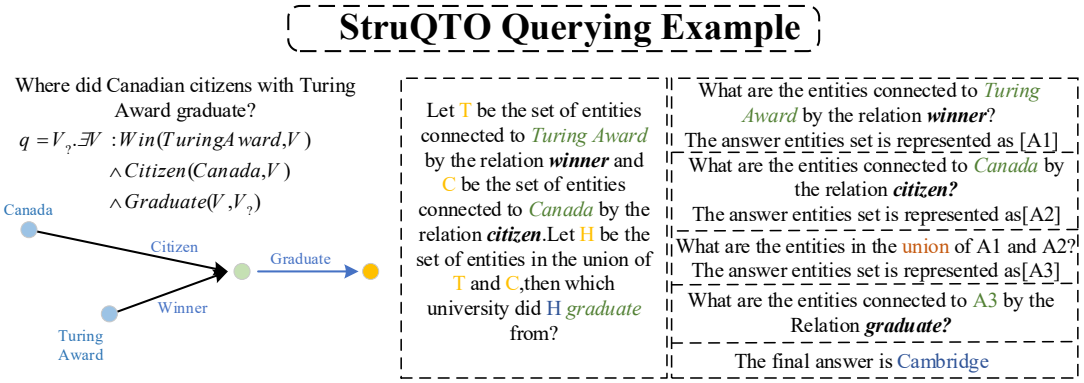

Fig. 3 Case study.

V. Conclusion

This paper proposes a structured reasoning method for complex logical queries, StruQTO, which achieves favorable results and performance on two benchmark datasets. The experiments validate that explicitly modeling query structure dependencies and generating structured reasoning chains can effectively enhance the reasoning capability of large language models in knowledge graph complex logical query tasks. This method employs query computation trees for structured representation of complex logical queries, explicitly characterizing dependency relationships between variables, thereby providing clear and consistent structural constraints for subsequent reasoning processes. Meanwhile, through local independence analysis, queries are decomposed into substructures that can be executed independently or sequentially, making the reasoning process more aligned with the true logical semantics of queries and effectively reducing structural confusion and logical deviation during reasoning. In comparison with various embedding-based and large language model-based methods, StruQTO demonstrates superior reasoning performance across multiple query structures, exhibiting particularly stronger stability in complex queries involving multi-hop reasoning and negation operators. Ablation studies further validate the effectiveness of query computation tree construction, local independence analysis, and structured reasoning chain generation modules in improving overall performance, proving the important roles of each model component in complex logical reasoning tasks.

Although the proposed method achieves promising experimental results in complex logical query tasks, there remains room for further research. Future work will focus on more refined query structure modeling approaches, optimization of structured reasoning chain generation and execution strategies, and further improvement of reasoning robustness in scenarios with sparse or incomplete knowledge graphs.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Yuyin Chen: Conceptualization, methodology,software, validation, formal analysis, investigation,writing. Guanfeng Li: Writing review and editing, supervision, funding acquisition, project administration, Supervision and guidance. Both authors had approved the final version.

Funding

This work was supported in part by the Key Research and Development Program of Ningxia under Grant 2023BSB03066; and in part by the National Science Foundation of China under Grant 62066038; and in part by the Natural Science Foundation of Ningxia under Grant 2026AAC03098.

References

[1] T. Feng, W. Li, Q. Guo, G. Wang, Y. Zhang, and Z. Qiao, "Overview of knowledge graph question answering enhanced by large language models," *Journal of Frontiers of Computer Science and Technology*, vol. 18, no. 11, pp. 2887–2900, 2024.

[2] K. Liang, L. Meng, M. Liu *et al.*, "A survey of knowledge graph reasoning on graph types: Static, dynamic, and multi-modal," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 46, no. 12, pp. 9456–9478, 2024.

[3] J. Li, G. Li, Y. Li *et al.*, "Structured chain-of-thought prompting for code generation," *ACM Trans. Softw. Eng. Methodol.*, vol. 34, no. 2, pp. 1–23, 2025.

[4] H. Yuan, H. Yu, S. Gui *et al.*, "Explainability in graph neural networks: A taxonomic survey," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 45, no. 5, pp. 5782–5799, 2022.

[5] W. Hamilton, P. Bajaj, M. Zitnik *et al*., "Embedding logical queries on knowledge graphs," *Adv. Neural Inf. Process. Syst*., vol. 31, 2018.
[6] H. Ren, W. Hu, and J. Leskovec, "Query2box: Reasoning over knowledge graphs in vector space using box embeddings," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2020.
[7] L. Liu, B. Du, H. Ji *et al*., "Neural-answering logical queries on knowledge graphs," in *Proc. 27th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, 2021, pp. 1087–1097.
[8] J. Sardina, C. Sardina, J. D. Kelleher *et al.*, "Analysis of attention mechanisms in box-embedding systems," in *Proc. Irish Conf. Artif. Intell. Cognitive Sci.*, Cham, Switzerland: Springer, 2022, pp. 68–80.
[9] V. Adlakha, P. Shah, and S. J. Bedathur, "Regex queries over incomplete knowledge bases," in *Proc. Automated Knowl. Base Construction*, 2021.
[10] Z. Zhang, J. Wang, J. Chen *et al.*, "Cone: Cone embeddings for multi-hop reasoning over knowledge graphs," *Adv. Neural Inf. Process. Syst*., vol. 34, pp. 19172–19183, 2021.
[11] N. Choudhary, N. Rao, S. Katariya *et al*., "Self-supervised hyperboloid representations from logical queries over knowledge graphs," in *Proc. Web Conf. 2021 (WWW)*, 2021, pp. 1373–1384.
[12] G. Wan and B. Du, "Gaussianpath: A Bayesian multi-hop reasoning framework for knowledge graph reasoning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 5, 2021, pp. 4393–4401.
[13] N. Choudhary, N. Rao, S. Katariya *et al.*, "Probabilistic entity representation model for reasoning over knowledge graphs," *Adv. Neural Inf. Process. Syst*., vol. 34, pp. 23440–23451, 2021.
[14] H. Ren and J. Leskovec, "Beta embeddings for multi-hop logical reasoning in knowledge graphs," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 19716–19726, 2020.
[15] D. Yang, P. Qing, Y. Li *et al*., "GammaE: Gamma embeddings for logical queries on knowledge graphs," in *Proc. 2022 Conf. Empirical Methods Natural Language Process. (EMNLP)*, 2022.
[16] X. Long, L. Zhuang, A. Li *et al*., "Neural-based mixture probabilistic query embedding for answering FOL queries on knowledge graphs," in *Proc. 2022 Conf. Empirical Methods Natural Language Process. (EMNLP)*, 2022, pp. 3001–3013.
[17] H. Sun, A. Arnold, T. Bedrax-Weiss *et al*., "Faithful embeddings for knowledge base queries," *Adv. Neural Inf. Process. Syst*., vol. 33, pp. 22505–22516, 2020.
[18] E. Arakelyan, D. Daza, P. Minervini *et al.*, "Complex query answering with neural link predictors," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2021.
[19] F. P. Luus, P. Sen, R. N. Riegel *et al.*, "Logic embeddings for complex query answering," U.S. Patent 17/488,226, Mar. 30, 2023.
[20] X. Chen, Z. Hu, and Y. Sun, "Fuzzy logic based logical query answering on knowledge graphs," in *Proc. AAAI Conf. Artif. Intell.*, 2022, vol. 36, no. 4, pp. 3939–3948.
[21] X. Lin, C. Xu, G. Zhou *et al*., "TFLEX: Temporal feature-logic embedding framework for complex reasoning over temporal knowledge graph," *Adv. Neural Inf. Process. Syst*., vol. 36, 2024.
[22] Z. Zhu, M. Galkin, Z. Zhang *et al.*, "Neural-symbolic models for logical queries on knowledge graphs," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2022, pp. 27454–27478.
[23] Z. Xu, W. Zhang, P. Ye *et al.*, "Neural-symbolic entangled framework for complex query answering," *Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 1806–1819, 2022.
[24] D. Wang, Y. Chen, and B. C. Grau, "Efficient embeddings of logical variables for query answering over incomplete knowledge graphs," in *Proc. AAAI Conf. Artif. Intell*., 2023, vol. 37, no. 4, pp. 4652–4659.
[25] L. Luo, J. Ju, B. Xiong *et al*., "Chatrule: Mining logical rules with large language models for knowledge graph reasoning," in *Proc. Pacific-Asia Conf. Knowl. Discovery Data Mining (PAKDD)*, Singapore: Springer, 2025, pp. 314–325.
[26] X. Liu, S. Zhao, K. Su *et al.*, "Mask and reason: Pre-training knowledge graph transformers for complex logical queries," in *Proc. 28th ACM SIGKDD Conf. Knowl. Discovery Data Mining (KDD)*, 2022, pp. 1120–1130.
[27] H. Luo, E. Haihong, Z. Tang *et al.*, "ChatKBQA: A generate-then-retrieve framework for knowledge base question answering with fine-tuned large language models," in *Proc. Findings Assoc. Comput. Linguistics: ACL 2024*, 2024, pp. 2039–2056.
[28] L. Liu, Z. Wang, R. Qiu *et al.*, "Logic query of thoughts: Guiding large language models to answer complex logic queries with knowledge graphs," arXiv preprint arXiv:2404.04264, 2024.
[29] Z. Li, L. Deng, H. Liu *et al.*, "UnioQA: A unified framework for knowledge graph question answering with large language models," arXiv preprint arXiv:2406.02110, 2024.
[30] J. Jiang, K. Zhou, K. M. Ye *et al.*, "StructGPT: A general framework for large language model to reason over structured data," in *Proc. 2023 Conf. Empirical Methods Natural Language Process. (EMNLP)*, 2023.
[31] J. Jiang, K. Zhou, W. X. Zhao *et al*., "ReasoningLM: Enabling structural subgraph reasoning in pre-trained language models for question answering over knowledge graph," in *Proc. 2023 Conf. Empirical Methods Natural Language Process. (EMNLP)*, 2023, pp. 3721–3735.
[32] L. Liu, Z. Wang, R. Qiu *et al.*, "Logic query of thoughts: Guiding large language models to answer complex logic queries with knowledge graphs," in *Proc. AAAI Conf. Artif. Intell.*, 2025, pp. 12881–12889.
[33] N. Choudhary and C. K. Reddy, "Complex logical reasoning over knowledge graphs using large language models," arXiv preprint arXiv:2305.01157, 2023.
[34] Y. Sui, Y. He, N. Liu *et al.*, "Fidelis: Faithful reasoning in large language models for knowledge graph question answering," in *Proc. Findings Assoc. Comput. Linguistics: ACL 2025*, 2025, pp. 8315–8330.
[35] K. Toutanova and D. Chen, "Observed versus latent features for knowledge base and text inference," in *Proc. 3rd Workshop Continuous Vector Space Models Compositionality*, 2015, pp. 57–66.
[36] W. Xiong, T. Hoang, and W. Y. Wang, "DeepPath: A reinforcement learning method for knowledge graph reasoning," in *Proc. 2017 Conf. Empirical Methods Natural Language Process. (EMNLP)*, Assoc. Comput. Linguistics, 2017.
[37] Y. Bai, X. Lv, J. Li *et al.*, "Answering complex logical queries on knowledge graphs via query computation tree optimization," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2023, pp. 1472–1491.
[38] S. Wang, Z. Wei, M. Han *et al.*, "Query structure modeling for inductive logical reasoning over knowledge graphs," in *Proc. 61st Annu. Meeting Assoc. Comput. Linguistics (ACL)*, vol. 1, 2023, pp. 4706–4718.
[39] X. Zhuo, S. Pan, J. Wang *et al*., "Progressive prefix-memory tuning for complex logical query answering on knowledge graphs," in *Proc. 34th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2025, pp. 3716–3724.