GCLQR: Global Context-Aware Logical Query Reasoning over Knowledge Graphs

Pengwei Pan*, Jun Ma†, Mingzhe Wang*, Xinyi Xia*, Wenxuan Xie*, Yanmei Kang†‡
* Soochow University, Suzhou, China
{pwpen, MingzheWang, XinyiXia, WenxuanXie}@stu.suda.edu.cn
† University of International Relations, Beijing, China
majun@uir.edu.cn, kymei@163.com
‡ Corresponding author

***Abstract*—Logical query reasoning over incomplete knowledge graphs remains challenging for complex queries involving long reasoning chains and semantic constraints. Existing stepwise reasoning methods often fail to capture global logical dependencies, particularly across sequential operations and negation. To address these issues, we propose GCLQR, a global context-aware logical query reasoning framework with three key components: (1) a tree-based serialization strategy that combines depth-first and breadth-first traversals to preserve global structural information; (2) an instruction-context fusion module based on pretrained language models to generate contextualized semantic embeddings; and (3) a transformer-based long-chain query encoder with neural intersection operators that supports full first-order logical reasoning.Experiments on FB15k, FB15k-237, and NELL995 demonstrate that GCLQR achieves the best average performance across datasets and delivers strong improvements on challenging query types, especially long-chain, negation, and intersection queries.**

***Index Terms*—Knowledge Graph,Complex Logical Query Answering, Global Context Awareness**

I. Introduction

Knowledge Graphs (KGs) organize entities, relations, and logical dependencies, providing a fundamental basis for intelligent reasoning systems. However, real-world KGs are often incomplete, motivating Knowledge Graph Reasoning (KGR) to infer missing facts and support downstream tasks such as question answering and recommendation [1], [2].

In KGR, complex logical queries typically involve multiple reasoning steps. Recent query encoding methods model a logical query as a sequence of first-order logic operations, recursively applying predefined operators from anchor nodes toward target variables according to the query structure. At each step, only a single local operation is executed, and the resulting embedding is progressively updated to infer the target entity set via similarity-based retrieval. As illustrated in Figure 1, such logical queries naturally form tree-structured dependency graphs, where multiple subqueries converge on a shared target entity set $E_?$. These tree-structured queries are prevalent in knowledge graph logical reasoning and constitute the primary focus of this work.

Despite their effectiveness, stepwise query encoding strategies suffer from an inherent limitation: by operating solely on local logical information at each step, they fail to capture global logical dependencies spanning the entire query. This lack of global and future-aware reasoning prevents models from leveraging downstream semantic constraints early in the reasoning process, leading to ambiguity accumulation and error propagation, particularly in long-chain queries, negation, and intersection operations. Moreover, the hierarchical structure of query trees further restricts access to global logical context, constraining the model's receptive field and undermining its reasoning capability.

To address these challenges, we propose **Global Context-aware Logical Query Reasoning (GCLQR)**, a framework explicitly designed to capture and exploit global logical context in complex knowledge graph queries. GCLQR serializes computation trees using a combination of depth-first and breadth-first traversal strategies, enabling comprehensive preservation of global query structure. The serialized representations are then encoded using pretrained language models to capture rich contextual semantics. In addition, an instruction–context fusion module integrates entities, relations, and logical operators to generate query-adaptive embeddings. To support complex reasoning patterns, GCLQR incorporates a Transformer-based long-chain query encoder and a neural intersection operator, enabling complete first-order logical reasoning over intricate query structures.

Our main contributions are summarized as follows:

- We propose GCLQR, a global context-aware logical query reasoning framework that unifies computation tree serialization, pretrained Transformer encoding, and instruction–context fusion to enable comprehensive reasoning over complex logical queries.
- We design a neural intersection operator that effectively models intersecting logical constraints, supporting complete first-order logic inference.
- Extensive experiments on widely used benchmarks demonstrate that GCLQR consistently outperforms existing methods across diverse query types, including long chains, negation, and complex intersections.

II. Related Work

Recent research on Knowledge Graph Reasoning (KGR) can be broadly categorized into three paradigms: embedding-based query encoding, symbolic–neural fusion, and Transformer-based query serialization approaches [3].

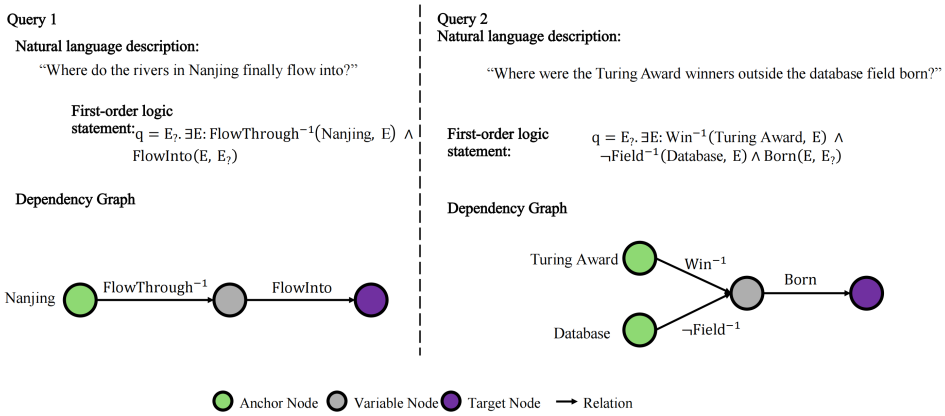

Fig. 1. Illustration of Global Context's Guiding Role in Knowledge Graph Logical Query Reasoning.

Embedding-based methods project entities and logical queries into continuous spaces and perform reasoning via neural set operators. Representative models include GQE [4], which operates in Euclidean space, Q2B [5] using box embeddings, and BetaE [6] and GammaE [7], which model uncertainty and negation via probabilistic distributions. Although effective for structured queries, these approaches rely on stepwise reasoning and are inherently limited in capturing global logical dependencies across long reasoning chains.

Symbolic–neural fusion methods aim to alleviate this limitation by integrating symbolic constraints with neural representations. CQD [9] formulates query answering as constrained optimization, while CQDA [10] extends it with adaptive negation modeling. LMPNN [11] propagates logical messages along the query graph using neural message passing. Despite improved interpretability, these methods often incur high computational cost and scale poorly to complex or long-chain queries.

More recently, serialization-based approaches linearize query structures into sequences and leverage pretrained Transformers to capture global context. BIQE [12] encodes decomposed query paths, KG-Transformer [13] applies random-walk-based serialization, and SILR [14] introduces execution-aware instructions within a BERT framework. While these methods better exploit global contextual information, they generally lack explicit support for complete first-order logic, particularly for negation and complex intersections.

Unlike prior serialization-based methods such as BIQE, KG-Transformer, and SILR, GCLQR is designed to encode the global logical context of an entire query computation tree rather than only path-level or execution-local information. The key difference is twofold. First, GCLQR combines depth-first and breadth-first serialization to preserve both nested structural dependencies and layer-wise execution semantics. Second, GCLQR introduces an instruction-context fusion mechanism that generates query-adaptive embeddings for entities, relations, and logical operators, allowing downstream reasoning to be guided by global structural constraints throughout the encoding process.

III. Methodology

Our proposed framework, GCLQR, aims to capture global logical context in complex knowledge graph queries. Given a first-order logic query, GCLQR first linearizes its computation tree using two complementary traversal strategies—depth-first and breadth-first—to preserve hierarchical and global structural information. The resulting instruction sequences are encoded by a pretrained Transformer to model long-range logical dependencies. An instruction–context fusion module then generates adaptive embeddings for entities, relations, and logical operators. Finally, a Transformer-based chain query encoder supports long-chain reasoning, while a neural intersection symbol module enables complete first-order logical inference over intersecting subqueries.

As illustrated in Figure 2, the GCLQR framework consists of three tightly integrated components. First, the Instruction Generation module linearizes an input query's computation graph into two serialized instruction sequences: Depth-First Serialization (DFS) and Breadth-First Serialization (BFS), which together preserve both local hierarchical structure and global logical dependencies (Figure 2(a)). The original computation graph is shown in Figure 2(b). Second, the Instruction Encoding module employs a pretrained language model (PLM) to obtain dense representations for both instruction sequences (Figure 2(c)). These representations are refined through an instruction–context fusion layer, which generates query-adaptive embeddings conditioned on entities, relations, and negation operators. Finally, the Query Encoding module (Figure 2(d)) follows the execution order of the computation graph, applying the adaptive instruction embeddings with an atomic query encoder to produce a unified query representation. This final embedding is used to retrieve answer entities via nearest-neighbor search.

A. Serialization Instruction Generation

To preserve the hierarchical and global structure of complex logical queries, we employ dual serialization strategies based on depth-first and breadth-first traversals. Given a query computation tree, the Instruction Generation module produces two

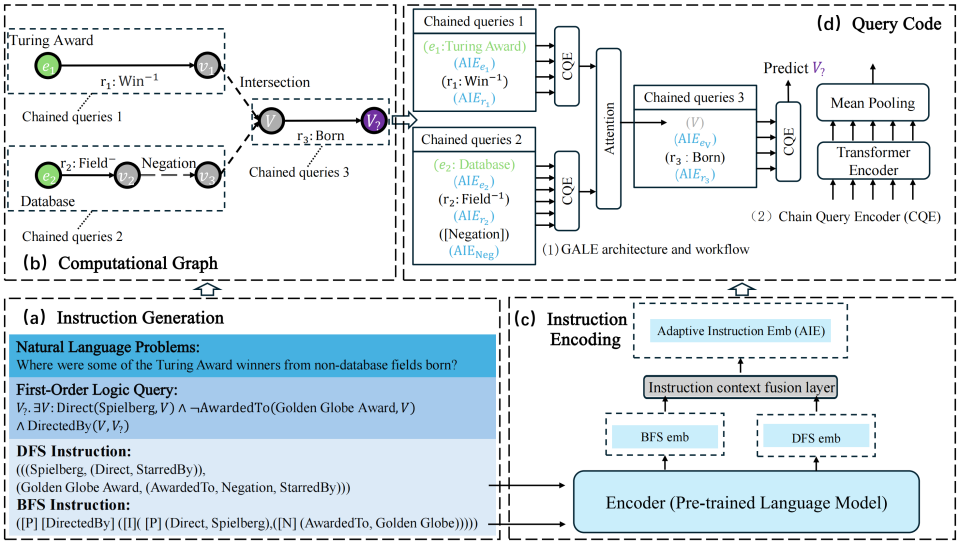

Fig. 2. Framework Diagram of Global Context-Aware Logical Query Reasoning.

corresponding instruction sequences, which together encode both local compositional structure and global logical dependencies, as illustrated in Figure 2(a).

We adopt depth-first serialization (DFS) to explicitly represent the nested structure of logical operations. In DFS, parenthetical brackets are used to delimit subtree boundaries, yielding a compact and expressive representation of hierarchical query semantics. Table I summarizes the DFS instruction formats for representative first-order logic query patterns.

In contrast, breadth-first serialization (BFS) provides a complementary global view by organizing logical operators according to their execution layers. BFS introduces control tokens to denote projection, intersection, and negation operations, enabling the model to perceive high-level logical composition early in the encoding process. Together, DFS and BFS offer complementary structural perspectives that jointly facilitate effective global context modeling.

For brevity, we explicitly enumerate DFS instruction formats, while BFS serialization is defined at the operator and structural level, which is sufficient for reproducibility and implementation.

B. Instruction Encoding

1) Instruction Encoder: We adopt a BERT-based Instruction Encoder to model contextual dependencies within serialized instruction sequences. Given an instruction sequence $X = \{x_t\}_{t=1}^{T}$, the encoder maps it to contextualized token representations via:

$$\mathbf{X} = \mathit{INST}_{\Theta}(X), \tag{1}$$

where $\mathit{INST}_{\Theta}(\cdot)$ denotes a pretrained Transformer encoder with parameters Θ, and each output token embedding lies in $\mathbb{R}^{D_1}$.

Applying the encoder to the depth-first and breadth-first serialized instructions yields:

$$\mathit{DFS}_{\text{emb}} = \mathit{INST}_{\Theta}(\{\mathit{DFS}_i\}_{i=1}^{n}), \quad \mathit{BFS}_{\text{emb}} = \mathit{INST}_{\Theta}(\{\mathit{BFS}_j\}_{j=1}^{m}), \tag{2}$$

where n and m denote the sequence lengths, and both outputs lie in $\mathbb{R}^d$.

2) Instruction Context Fusion Layer: To capture element-specific contextual relevance, we introduce an attention-based instruction–context fusion mechanism that integrates DFS and BFS encodings. For a query element x (entity, relation, or negation operator) with embedding $\mathbf{x} \in \mathbb{R}^d$, we compute attention-weighted context representations over both serialized sequences:

$$\mathbf{c}_{\text{dfs}} = \sum_{i=1}^{n_d} \alpha_i^{dfs} \mathbf{dfs}_i, \quad \mathbf{c}_{\text{bfs}} = \sum_{j=1}^{n_b} \alpha_j^{bfs} \mathbf{bfs}_j, \tag{3}$$

where α^{dfs} and α^{bfs} are obtained via softmax-normalized dot-product attention.

The two context vectors are concatenated to preserve complementary structural perspectives:

$$\mathbf{c}_{\text{fusion}} = [\mathbf{c}_{\text{dfs}}; \mathbf{c}_{\text{bfs}}] \in \mathbb{R}^{2d}. \tag{4}$$

3) Adaptive Embedding Generation: The fused context is projected back to the embedding space through a two-layer MLP:

$$\mathbf{AIE}_x = \mathbf{MLP}_{\text{ins}}(\mathbf{c}_{\text{fusion}}), \tag{5}$$

yielding an adaptive instruction embedding $\mathbf{AIE}_x \in \mathbb{R}^d$ that captures both semantic content and global logical context.

C. Query Encoding

We decompose each logical query into a disjunction of conjunctive chain queries under Disjunctive Normal Form (DNF). Each chain query consists of a sequence of atomic operations (projections with optional negation), forming a linear computation path.

TABLE I
DEPTH-FIRST SERIALIZATION INSTRUCTION FORMAT

Query	First-Order Logic Query Format	Depth-First Serialization Instruction Format
1p	$E_T : r_1(e_1, E_T)$	(e1)
2p	$E_T : \exists E_1, E_2 : r_1(e_1, E_1) \wedge r_2(E_1, E_T)$	(e1, (r1, r2))
3p	$E_T : \exists E_1, E_2, E_3 : r_1(e_1, E_1) \wedge r_2(E_1, E_2) \wedge r_3(E_2, E_T)$	(e1, (r1, r2, r3))
2i	$E_T : r_1(e_1, E_1) \wedge r_2(e_2, E_T)$	((e1, r1), (e2, r2))
2in	$E_T : r_1(e_1, E_1) \wedge \neg r_2(e_2, E_T)$	((e1, r1), (e2, negative))
3i	$E_T : r_1(e_1, E_1) \wedge r_2(e_2, E_2) \wedge r_3(e_3, E_T)$	((e1, r1), (e2, r2), (e3, r3))
3in	$E_T : r_1(e_1, E_1) \wedge r_2(e_2, E_2) \wedge \neg r_3(e_3, E_T)$	((e1, r1), (e2, r2), (e3, negative))
ip	$E_T : r_1(e_1, E_1) \wedge r_2(E_1, E_2) \wedge r_3(E_2, E_T)$	(((e1, r1), r2), r3)
inp	$E_T : r_1(e_1, E_1) \wedge \neg r_2(E_1, E_2) \wedge r_3(E_2, E_T)$	(((e1, r1), negative), r3)

1) Chain Query Encoder: Each chain query is encoded using a Transformer-based Chain Query Encoder. Given a chain of length m, we construct an input embedding sequence:

$$\mathbf{E}_{\text{in}} = \{\mathbf{E}_1, \mathbf{E}_2, \ldots, \mathbf{E}_m\}, \tag{6}$$

where each element corresponds to an entity, relation, negation token, or its associated adaptive instruction embedding. The sequence alternates between base embeddings and their adaptive counterparts:

$$\mathbf{E}_{\text{in}} = \{\mathbf{E}_e, \mathbf{AIE}_e, \mathbf{E}_r, \mathbf{AIE}_r, \ldots\}. \tag{7}$$

The resulting sequence is processed by a k_{CQE}-layer Transformer encoder, followed by mean pooling to obtain the final chain representation:

$$\mathbf{E}_{\text{cq}} = \text{MeanPooling}\big(\text{TrmE}_{k_{\text{CQE}}}(\mathbf{E}_{\text{in}})\big). \tag{8}$$

2) Intersection Operator: Beyond chain-computable queries, complex queries may contain branching conjunctive structures. Following DNF-based reasoning [6], unions are handled by merging subquery answer sets, and we focus on modeling the intersection operator.

We adopt a self-attention–based intersection mechanism [17]. Given entity embeddings $S = \{\mathbf{e}_1, \ldots, \mathbf{e}_n\} \in \mathbb{R}^{n\times d}$, we augment each entity with its adaptive instruction embedding (AIE), yielding $\mathbf{S}_{\text{AIE}} \in \mathbb{R}^{2n\times d}$:

$$\mathbf{S}_{\text{AIE}} = \{\mathbf{e}_1, \mathbf{AIE}_{e_1}, \ldots, \mathbf{e}_n, \mathbf{AIE}_{e_n}\}. \tag{9}$$

The intersection embedding $\mathbf{e}_{\text{inter}} \in \mathbb{R}^d$ is computed as a weighted aggregation:

$$\mathbf{e}_{\text{inter}} = \sum_i a_i \odot \mathbf{e}_i, \tag{10}$$

where the attention weights are defined as

$$a_i = \frac{\exp(\text{MLP}_{\text{inter}}(\mathbf{S}_{\text{AIE}i}))}{\sum_j \exp(\text{MLP}_{\text{inter}}(\mathbf{S}_{\text{AIE}j}))}. \tag{11}$$

Here, $\text{MLP}_{\text{inter}}(\cdot)$ is a multilayer perceptron and $\odot$ denotes element-wise multiplication.

Compared with the intersection mechanisms in BetaE and Q2B, our design differs in that the aggregation is conditioned not only on subquery embeddings themselves but also on their associated adaptive instruction embeddings. In other words, the proposed operator is explicitly guided by the serialized global query context, rather than relying solely on geometric overlap or distributional parameter aggregation. This enables the intersection module to better capture context-dependent logical constraints in complex queries.

D. Training Objective

After executing the computation graph, GCLQR produces a final query representation $\mathbf{v}_q \in \mathbb{R}^d$. Each entity in the knowledge graph is associated with an embedding $\mathbf{v}_e \in \mathbb{R}^d$. Reasoning is then performed by measuring the distance between $\mathbf{v}_q$ and candidate entity embeddings.

1) Distance Function: We embed both queries and entities in a d-dimensional Euclidean space [4], [18]. Given a query embedding $\mathbf{v}_q$ and an entity embedding $\mathbf{v}_e$, their distance is measured by the L_1 norm:

$$\text{Dist}(\mathbf{e}, \mathbf{q}) = \|\mathbf{v}_e - \mathbf{v}_q\|_1. \tag{12}$$

2) Training Objective: We adopt a margin-based negative sampling loss [19]:

$$\mathcal{L} = -\log \sigma(\gamma - \text{Dist}(\mathbf{e}, \mathbf{q})) - \frac{1}{u}\sum_{j=1}^{u} \log \sigma(\text{Dist}(\mathbf{e}'_j, \mathbf{q}) - \gamma), \tag{13}$$

where $\mathbf{e}$ and $\mathbf{e}'_j$ denote positive and negative entities, respectively, γ is the margin, and u is the number of negative samples.

3) Reasoning: During inference, a query is encoded via the Instruction Encoder, Chain Query Encoder, and the neural intersection operator according to its computation graph, yielding $\mathbf{v}_q$. Candidate entities are ranked by $\text{Dist}(\mathbf{e}, \mathbf{q})$, and approximate nearest-neighbor search is performed using Locality-Sensitive Hashing [20].

IV. EXPERIMENTS

A. Experimental Settings

1) Hyperparameter Configuration.: In our experiments, we adopt the "bert-base-cased" model as the backbone PLM and train each variant on two NVIDIA A100 GPUs with 120 GB of memory each. Recognizing that different layers of BERT capture contextual information at varying granularities [21], we configure $\text{INST}_\Theta(\cdot)$ to focus on the hierarchical structure of X rather than solely on its semantic embedding. Consequently, the trainable parameters Θ include only the

token embedding layer and a subset of the lower Transformer layers of BERT; the exact number of fine-tuned layers is explored in Section IV-C. By default, the Instruction Encoder is assigned one Transformer layer, and the number of attention heads H is set to 4.

All experiments in this chapter are implemented in PyTorch. For both the main experiments and ablation studies, we configure the model with the following hyperparameters: embedding dimension $d = 768$; learning rate $= 1 \times 10^{-4}$; negative sample size $= 128$; batch size $= 512$; margin $\gamma = 24$; total training steps $= 450{,}000$; number of Transformer layers in the Chain Query Encoder $k_{\mathrm{CQE}} = 6$; dropout rate $= 0.1$; and Transformer feed-forward network dimension $d_{\mathrm{ffn}} = 4d$.

2) Benchmark Methods.: We select several competitive methods for Knowledge Graph Logical Query Reasoning:

- **GQE** [4]: Models queries and entities as points in Euclidean space, using a single-layer feed-forward network or TransE-style edge translations for projections. Answers are retrieved by minimizing Euclidean distance to the query.
- **Q2B** [5]: Represents entities as points and queries as axis-aligned hyper-rectangles. Projections correspond to edges, and intersections to overlapping regions. Answers lie within the query box.
- **BetaE** [6]: Embeds queries and entities as Beta distributions, using MLPs for projections and Kullback–Leibler divergence to identify the closest distribution.
- **MLP** [22]: Applies multi-layer perceptrons to project and intersect query and entity embeddings, similar to GQE.
- **ConE** [8]: Models queries as conic-sector intervals, handling negation for the first time in geometry-based query encoding.
- **FuzzQE** [23]: Uses t-norm fuzzy logic for neural encoding of conjunction, disjunction, and negation.
- **GammaE** [7]: Uses Gamma embeddings to encode complex logical queries.
- **LMPNN** [11]: Aggregates logical information from a pre-trained KG completion model via MLP to answer complex queries efficiently.

B. Experimental Results

Table II summarizes the mean reciprocal rank (MRR) of GCLQR and representative query encoding baselines on the BetaE benchmark. Overall, GCLQR achieves the best average MRR on FB15k, FB15k-237, and NELL995, demonstrating the effectiveness of incorporating global logical context into query reasoning. More importantly, the largest gains are observed on structurally challenging query types, especially long-chain queries (e.g., *ip*, *inp*, and *pni*) and negation-related queries (e.g., *2in*, *3in*, and *inp*). These improvements are consistent with our design motivation: the dual serialization strategy provides a global view of the computation tree, while the instruction-context fusion module injects query-level constraints into each reasoning step. For simpler query types, GCLQR remains competitive and in several cases matches the strongest baseline.

C. Experimental Analysis

1) Ablation Study: This section evaluates the impact of our serialized instruction encoding within GCLQR. We compare four variants of GCLQR, with results reported in Table III: (1) a Baseline without the Instruction Encoding module; (2) GCLQR – DFS, which omits the DFS instructions; (3) GCLQR – BFS, which omits the BFS instructions; and (4) GCLQR – PT-BERT, which forgoes pretrained BERT and instead trains a randomly initialized BERT from scratch.

All four variants perform significantly worse than the full GCLQR model. First, regardless of whether DFS or BFS is used alone, both outperform the Baseline, demonstrating that each serialization strategy effectively encodes the query graph's contextual semantics. Their combination in the full model yields further gains, confirming the complementary strengths of the two traversals. Second, DFS alone yields better results than BFS, likely because depth-first traversal more faithfully preserves long-chain contextual dependencies, thereby enhancing reasoning capability. Finally, training the instruction encoder from scratch—even though GCLQR uses only a single BERT layer—incurs a substantial learning challenge for modeling logical semantics, whereas leveraging pretrained parameters provides a superior initialization and enables more efficient context modeling. Consequently, while the PT-BERT variant's compute cost remains relatively low compared to the Baseline (due to its single-layer design), it still lags behind the fully pretrained GCLQR.

2) Effectiveness of Future Context in the Chain Query Encoder: To assess the impact of bidirectional attention and, in particular, the utilization of future context within the Chain Query Encoder, we construct a variant of GCLQR by introducing masked self-attention. In this variant, the query encoding process is constrained to rely solely on current and past context, thereby isolating the effect of future information on model performance. We conduct a comparative experiment on the FB15k-237 dataset using a single-layer Transformer encoder, evaluating this masked variant against the standard GCLQR framework. As shown in Table IV, the absence of future context significantly degrades GCLQR's performance. These results demonstrate that incorporating future information via bidirectional self-attention is essential for capturing the implicit, complex dependencies among different parts of the query. Overall, this experiment further validates the effectiveness of bidirectional attention in modeling contextual dependencies for logical query reasoning.

3) Role of Position Encoding in the Query Encoder: In a chain query, each element must strictly follow a predefined execution order. To effectively capture this sequential information, we examine the encoding mechanisms within the Transformer architecture. Extensive prior work has demonstrated their efficacy in sequence modeling tasks, which underpins their widespread adoption. In addition, we explore the use of Relative Position Encoding (RPE) [24] to further enhance the model's sensitivity to internal positional relationships within the query sequence. We evaluate different position encoding

TABLE II
GCLQR MRR RESULTS (%) ON THE BETAE FRAMEWORK

Method	1p	2p	3p	2i	3i	ip	pi	2u	up	2in	3in	inp	pin	pni
FB15k														
GQE	54.6	15.3	10.8	39.7	51.4	19.1	27.6	22.1	11.6	–	–	–	–	–
Q2B	68.0	21.0	14.2	55.1	66.5	26.1	39.4	35.1	16.7	–	–	–	–	–
BetaE	65.1	25.7	24.7	55.8	66.5	28.1	43.9	40.1	25.2	14.3	14.7	11.5	6.5	12.4
ConE	73.3	33.8	29.2	64.4	73.3	35.7	50.9	55.7	37.1	17.9	18.7	12.5	9.8	15.1
MLP	67.1	31.2	27.2	57.1	66.9	33.9	45.7	38.0	28.0	16.0	17.3	13.2	8.3	14.6
FuzzQE	77.2	33.1	25.3	58.7	69.0	32.3	47.9	40.1	29.4	18.0	16.7	13.4	8.8	13.4
GammaE	76.5	36.9	**31.4**	65.4	75.1	39.7	**53.9**	53.5	30.9	20.1	20.5	13.5	11.8	**17.1**
LMPNN	85.0	39.9	28.6	**68.2**	**76.5**	43.0	46.7	36.7	31.4	**29.1**	29.4	14.9	10.2	16.4
GCLQR(Ours)	**85.0**	**40.5**	29.6	68.1	75.7	**51.0**	43.6	39.3	**32.4**	28.0	**29.5**	**16.4**	**11.0**	16.1
FB15k-237														
GQE	35.0	7.2	5.3	23.3	34.6	10.7	16.5	8.2	5.7	–	–	–	–	–
Q2B	40.6	9.4	6.8	29.5	42.3	12.6	21.2	11.3	7.6	–	–	–	–	–
BetaE	39.2	10.7	9.8	29.0	42.5	11.9	22.1	12.6	9.7	5.1	7.9	7.4	3.5	3.4
ConE	42.4	12.6	10.8	32.5	46.8	13.6	25.1	14.1	10.6	5.4	8.6	7.8	4.0	3.6
MLP	42.7	12.4	10.6	33.0	43.9	14.9	24.2	13.7	9.7	6.4	10.6	8.0	4.6	4.4
FuzzQE	44.3	13.8	10.2	33.0	47.4	18.8	26.3	15.6	10.8	8.5	11.6	7.8	5.2	5.9
GammaE	43.2	13.2	11.0	33.5	47.9	15.9	27.2	13.9	10.3	6.7	9.4	8.6	4.8	4.4
LMPNN	45.9	13.1	10.3	34.8	48.9	17.6	22.7	13.5	10.3	8.7	12.9	7.7	4.6	5.2
GCLQR(Ours)	**47.0**	**14.7**	**12.8**	**36.0**	**49.3**	**19.0**	**28.7**	**16.9**	**11.6**	**8.0**	**13.3**	**10.0**	**6.3**	**6.4**
NELL995														
GQE	32.8	11.9	9.6	27.5	35.2	14.4	18.4	8.5	8.8	–	–	–	–	–
Q2B	42.2	14.0	11.2	33.3	44.5	16.8	22.4	11.3	10.3	–	–	–	–	–
BetaE	53.0	13.0	11.4	37.6	47.5	14.3	24.1	12.2	8.5	5.1	7.8	10.0	3.1	3.5
ConE	53.1	16.1	13.9	40.0	50.8	17.5	26.3	15.3	11.3	5.7	8.1	10.8	3.5	3.9
MLP	55.2	16.4	14.7	36.4	48.0	18.2	22.7	14.7	11.3	5.1	8.0	10.0	3.6	3.6
FuzzQE	57.6	19.2	15.3	39.8	50.3	21.8	28.1	17.3	13.7	7.8	9.8	11.1	4.9	5.5
GammaE	55.1	17.3	14.2	41.9	51.1	18.3	26.9	15.1	11.2	6.3	8.7	11.4	4.0	4.5
LMPNN	**60.6**	**22.1**	17.5	40.1	50.3	**24.9**	28.4	17.2	**15.7**	8.5	10.8	12.2	3.9	4.8
GCLQR(Ours)	59.0	20.1	**17.8**	**43.4**	**53.5**	22.7	**28.8**	**17.3**	13.7	**7.9**	**11.5**	**13.8**	**7.3**	**6.3**

TABLE III
MRR RESULTS (%) USING DIFFERENT SERIALIZATION METHODS

Dataset	MRR (%)		
	FB15k-237	FB15k	NELL995
Baseline	15.4	34.7	16.8
+GCLQR	**17.5**	**35.9**	**20.0**
w/o BFS	16.8	35.3	18.7
w/o DFS	16.4	35.2	18.1
w/o PT-BERT	16.6	35.5	19.8

TABLE IV
AVERAGE MRR RESULTS (%) ON FB15K-237 WITH AND WITHOUT FUTURE CONTEXT INFORMATION

Model	**avg MRR (%)**
GCLQR(No future context)	16.6
GCLQR	17.5

strategies on the FB15k-237 dataset, with results presented in Table V. Here, "w/o PE" denotes the omission of any position encoding, while "w/ RPE" indicates the adoption of relative position encodings. Compared to the variant without position encoding, introducing positional information yields a modest but consistent improvement. Although the gap is relatively small on FB15k-237, the result suggests that positional cues still provide useful inductive bias for modeling execution order in chain queries. Relative position encoding achieves the best score, while absolute position encoding offers a favorable balance between accuracy and computational overhead, and is therefore used as the default setting. Although RPE achieves slightly higher accuracy than absolute position encoding, we ultimately select absolute position encoding as the default configuration to balance model performance against computational overhead.

4) Impact of BERT Layer Depth in the Instruction Encoder: As illustrated in Figure 3, we examine the effect of varying the number of Transformer layers (from 1 to 4) in the BERT-based Instruction Encoder. The results demonstrate a

TABLE V
AVERAGE MRR RESULTS (%) ON FB15K-237 FOR DIFFERENT POSITION ENCODING STRATEGIES

Model	avg MRR (%)
GCLQR	17.5
GCLQR w/o PE	17.4
GCLQR w/RPE	17.6

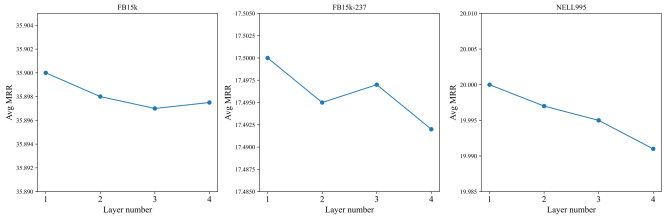

Fig. 3. MRR Results (%) for Different Numbers of BERT Layers in the Instruction Encoder

consistent decline in overall performance on all three datasets as layer depth increases. This trend likely arises from the hierarchical nature of BERT's contextual representations [21]. Lower layers predominantly capture syntactic structures and phrase-level dependencies—features crucial for representing the logical context of queries—whereas higher layers progressively abstract toward broader semantic concepts. Since our task hinges on accurately modeling the logical and hierarchical relationships among query variables, it benefits more from the syntactic and structural cues encoded in BERT's shallow layers. This reliance on shallow representations explains the observed performance degradation when employing deeper encoder configurations.

5) Impact of Transformer Depth in the Chain Query Encoder: We analyze the effect of varying the number of Transformer layers within the Chain Query Encoder on overall performance, evaluating on the FB15k-237 dataset. Specifically, we instantiate encoders with different depths and report the results in Table VI. The experiments show that model performance steadily improves as the number of Transformer layers increases, indicating that deeper Transformer architectures more effectively capture the implicit, complex semantic dependencies among the components of a chain query [21]. However, there exists a trade-off between performance gains and computational cost. Consequently, we select a 6-layer Transformer as our optimal configuration, striking a balance between resource consumption and accuracy. As shown in Table II, this setup significantly outperforms existing baselines.

TABLE VI
MRR RESULTS (%) FOR DIFFERENT NUMBERS OF TRANSFORMER LAYERS IN THE CHAIN QUERY ENCODER

Model	GCLQR(1)	GCLQR(2)	GCLQR(4)	GCLQR(6)	GCLQR(8)
avg	16.5	17.0	17.4	17.5	17.7

V. CONCLUSION

We present GCLQR, a Global Context-Aware Logical Query Reasoning framework designed to overcome the limitations of existing query-encoding approaches in capturing global logical dependencies. By serializing a query's computation tree via both DFS and BFS, we encode its full structural context and inject it into a pretrained Transformer to enhance the semantic representation of the query's global constraints. An instruction-context fusion module then adaptively integrates entity, relation, and operator information by applying attention over the DFS and BFS embeddings, yielding customized global directives for each query element. Building on this enriched representation, our Transformer-based Chain Query Encoder excels at long-chain reasoning, while a neural intersection symbol module supports full first-order logic inference across branching and intersecting substructures. Empirical evaluations on multiple public benchmarks demonstrate that GCLQR consistently outperforms state-of-the-art baselines, validating its effectiveness in complex logical query reasoning.

REFERENCES

[1] Das, R., Godbole, A., Naik, A., Tower, E., Zaheer, M., Hajishirzi, H., Jia, R., McCallum, A.: Knowledge base question answering by case-based reasoning over subgraphs. In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 4777–4793 (2022)

[2] Wu, L., Zhou, Y., Zhou, D.: Towards high-order complementary recommendation via logical reasoning network. In: *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, pp. 1227–1232 (2022)

[3] Tian, L., Zhou, X., Wu, Y.P., Zhou, W.T., Zhang, J.H., Zhang, T.S.: Knowledge graph and knowledge reasoning: A systematic review. *Journal of Electronic Science and Technology* **20**(2), 100159 (2022)

[4] Hamilton, W., Bajaj, P., Zitnik, M., Jurafsky, D., Leskovec, J.: Embedding logical queries on knowledge graphs. *Advances in Neural Information Processing Systems* **31** (2018)

[5] Ren, H., Hu, W., Leskovec, J.: Query2box: Reasoning over knowledge graphs in vector space using box embeddings. *arXiv preprint* arXiv:2002.05969 (2020)

[6] Ren, H., Leskovec, J.: Beta embeddings for multi-hop logical reasoning in knowledge graphs. *Advances in Neural Information Processing Systems* **33**, 19716–19726 (2020)

[7] Yang, D., Qing, P., Li, Y., Lu, H., Lin, X.: GammaE: Gamma embeddings for logical queries on knowledge graphs. *arXiv preprint* arXiv:2210.15578 (2022)

[8] Zhang, Z., Wang, J., Chen, J., Ji, S., Wu, F.: Cone: Cone embeddings for multi-hop reasoning over knowledge graphs. *Advances in Neural Information Processing Systems* **34**, 19172–19183 (2021)

[9] Minervini, P., Arakelyan, E., Daza, D., Cochez, M.: Complex query answering with neural link predictors. In: *Proceedings of the 31st International Joint Conference on Artificial Intelligence (IJCAI 2022)*, pp. 5309–5313 (2022)

[10] Arakelyan, E., Minervini, P., Daza, D., Cochez, M., Augenstein, I.: Adapting neural link predictors for data-efficient complex query answering. *Advances in Neural Information Processing Systems* **36**, 27079–27091 (2023)

[11] Wang, Z., Song, Y., Wong, G.Y., See, S.: Logical message passing networks with one-hop inference on atomic formulas. *arXiv preprint* arXiv:2301.08859 (2023)

[12] Kotnis, B., Lawrence, C., Niepert, M.: Answering complex queries in knowledge graphs with bidirectional sequence encoders. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, **35**(6), 4968–4977 (2021)

[13] Liu, X., Zhao, S., Su, K., Cen, Y., Qiu, J., Zhang, M., Wu, W., Dong, Y., Tang, J.: Mask and reason: Pre-training knowledge graph transformers for complex logical queries. In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1120–1130 (2022)

[14] Wang, S., Wei, Z., Han, M., Fan, Z., Shan, H., Zhang, Q., Huang, X.: Query structure modeling for inductive logical reasoning over knowledge graphs. *arXiv preprint* arXiv:2305.13585 (2023)

[15] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 4171–4186 (2019)

[16] Alon, U., Sadaka, R., Levy, O., Yahav, E.: Structural language models of code. In: *Proceedings of the International Conference on Machine Learning (ICML)*, pp. 245–256 (2020)

[17] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. *Advances in Neural Information Processing Systems* **30** (2017)

[18] Bai, J., Wang, Z., Zhang, H., Song, Y.: Query2Particles: Knowledge graph reasoning with particle embeddings. *arXiv preprint* arXiv:2204.12847 (2022)

[19] Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. *arXiv preprint* arXiv:1301.3781 (2013)

[20] Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 604–613 (1998)

[21] Jawahar, G., Sagot, B., Seddah, D.: What does BERT learn about the structure of language? In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, (2019)

[22] Amayuelas, A., Zhang, S., Rao, S.X., Zhang, C.: Neural methods for logical reasoning over knowledge graphs. *arXiv preprint* arXiv:2209.14464 (2022)

[23] Chen, X., Hu, Z., Sun, Y.: Fuzzy logic-based logical query answering on knowledge graphs. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, **36**(4), 3939–3948 (2022)

[24] Shaw, P., Uszkoreit, J., Vaswani, A.: Self-attention with relative position representations. *arXiv preprint* arXiv:1803.02155 (2018)