Efficient and Scalable Neural Symbolic Search for Knowledge Graph Complex Query Answering

Weizhi Fei
Department of Mathematical Sciences
Tsinghua University
Beijing, China
fwz220@mails.tsinghua.edu.cn

Zihao Wang
Department of CSE
HKUST
Hong Kong, China
zwanggc@cse.ust.hk

Hang Yin
Department of Mathematical Sciences
Tsinghua University
Beijing, China
h-yin20@mails.tsinghua.edu.cnn

Shukai Zhao
Department of Computer Sciences
University of Rochester
Rochester, New York
szhao27@ur.rochester.edu

Wei Zhang
Department of Mathematical Sciences
Tsinghua University
Beijing, China
zhangwei.2020@tsinghua.org.cn

Yangqiu Song
Department of CSE
HKUST
Hong Kong, China
zwanggc@cse.ust.hk

Abstract

Complex Query Answering (CQA) aims to retrieve answer sets for complex logical formulas from incomplete knowledge graphs, which is a crucial yet challenging task in knowledge graph reasoning. While neuro-symbolic search utilized neural link predictions achieve superior accuracy, they encounter significant complexity bottlenecks: (i) Data complexity typically scales quadratically with the number of entities in the knowledge graph, and (ii) Query complexity becomes NP-hard for cyclic queries. Consequently, these approaches struggle to effectively scale to larger knowledge graphs and more complex queries. To address these challenges, we propose an efficient and scalable symbolic search framework. First, we propose two constraint strategies to compute neural logical indices to reduce the domain of variables, thereby decreasing the data complexity of symbolic search. Additionally, we introduce an approximate algorithm based on local search to tackle the NP query complexity of cyclic queries. Experiments on various CQA benchmarks demonstrate that our framework reduces the computational load of symbolic methods by 90% while maintaining nearly the same performance, thus alleviating both efficiency and scalability issues.

1 Introduction

Knowledge Graphs (KGs) are knowledge bases that represent relational facts in graph form. Although KGs have an interpretable structure supporting many real-world applications (Ji et al., 2021), they often suffer from incompleteness (Safavi and Koutra, 2020; Hu et al., 2020). Recently, complex query answering (CQA) (Ren et al., 2023; Wang et al., 2022) on knowledge graphs has attracted significant interest because this practical task performs logical reasoning with new knowledge inferred from observed knowledge graphs. CQA aims to retrieve the whole answer set of the logical query

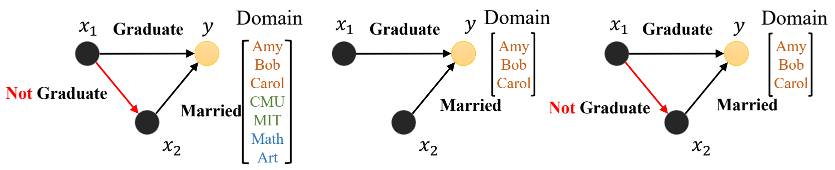

Figure 1: **The left** is the cyclic query graph of formula $\exists x_1, x_2.\neg\text{Graduate}(x_1, x_2) \wedge \text{Graduate}(x_1, y) \wedge \text{Married}(x_2, y)$. It can be interpreted as "Find someone who is married to a person who graduated from a different institution". Find all the answers of this formula from KG is $O(N^3)$, where N is number of entities within KG. Notably, complex query answering is NP hard since it can be reduced as constraint satisfaction problem. **The middle** presents the constraints used in the local strategy for the free variable y, while **the right** shows the constraints employed in the global strategy for the same free variable y. By the constraints, we can reduce the domain of free variable to accelerate search.

expressed with first-order logic (Ren and Leskovec, 2020; Yin et al., 2023) from the KGs. Due to the incompleteness of KGs, many answers are overlooked in direct traversal searching and require machine learning models for discovery.

There are primarily two lines of research to address the challenge of CQA. One is query embedding methods which represent the answer set of query using representations like vector, box, beta distribution of low dimensional space (Hamilton et al., 2018; Ren et al., 2020; Ren and Leskovec, 2020). In this approach, logical operations are transformed into set operations within an operator tree (Wang et al., 2021; Ren et al., 2023), modeled by neural networks in alignment with their semantics in the low-dimensional space. Although the representational capabilities have been thoroughly explored (Zhang et al., 2021; Choudhary et al., 2021), current query embedding methods still face limitations in both performance and expressiveness (Yin et al., 2024). The second line of research, neural-symbolic search methods (Arakelyan et al., 2020; Zhu et al., 2022; Bai et al., 2023b; Yin et al., 2024), utilizes knowledge graph completion methods (Bordes et al., 2013; Sun et al., 2018) as a backbone to predict missing facts and model logical operations using fuzzy logic inference. Though symbolic search methods usually have both strong performance and interpretability, they typically suffer from high complexity and lack scalability. The detailed related work can refer to Appendix A.

We provide a detailed analysis of the complexity of "precise"[1] symbolic search methods with (e.g., QTO (Bai et al., 2023b), FIT (Yin et al., 2024)) and discuss their scalability issues. For simple acyclic queries, the complexity of these search methods is $O(n|\mathcal{E}|^2)$, where $|\mathcal{E}|$ is the size of entities within KG and n is the size of query. From the perspective of data complexity, this complexity grows quadratically with the size of the entities, making it challenging to scale to large-scale graphs (Ren et al., 2022). Considering the cyclic queries, the complexity of precise searching is even $O(|\mathcal{E}|^n)$. From the perspective of query complexity, this complexity is NP-hard and increases exponentially with the size of query. This complexity bottleneck limits the efficiency in handling cyclic queries (Yin et al., 2024). To address theses complexity issues and improve the scalability of symbolic search, we propose an Neural Logical Indices for Search Approximately (NLISA) framework.

Inspired from the arc consistency (Chen et al., 2011) in constraint satisfaction problem (Gottlob et al., 2000), we find it is unnecessary to use all entities of the KG as the search domain for symbolic search methods. Our first technical contribution is the proposed Neural Logical Indices (NLI) to narrow the search domain for variables involved in logical queries. To fast compute the NLI, we provide two constraints strategies and corresponding methods. The **local constraints** strategy uses the relations directly connected with the variable, with a relation tail prediction task formulated to assist in computing the local constraints. The **global constraints** strategy considers broader constraints across the entire query to further reduce the search domain. Further details are presented in Section 3.

The second technical contribution is that we propose an approximately search algorithm to address the cyclic query in quadratic complexity $O(n|\mathcal{E}|^2)$. Our approximate search autoregressively seeks the most likely entity assignments and employs local search to avoid exponential growth of the search space. Notably, this algorithm supports parallel execution and reduced domains, significantly enhancing search efficiency. Furthermore, this algorithm can be integrated with the existing precise

[1]The precise search method means that these method can get the exact answers with perfect knowledge base.

search framework, maintaining precise in tree-form queries similar to QTO (Bai et al., 2023b) and FIT (Yin et al., 2024).

By combining the approximate search framework with neural logical indices, our method efficiently addresses general Existential First-Order Logic queries with a single free variable (EFO1 queries), particularly including cyclic queries that are NP-hard. To validate the effectiveness of our framework, we conduct comprehensive experiments on three knowledge graphs using two typical benchmarks: the classical BetaE benchmark and the extended Real EFO1 benchmark. By reducing the search space for variables by 90%, our method achieves significant efficiency gains with a minimal performance loss. On the FB15K-237 (KG) from the two benchmarks, our method with global constraints averagely achieves $13\times$ runtime speedup while maintaining 97% accuracy , compared to the state-of-the-art search method FIT (Yin et al., 2024). Additionally, we demonstrate that our framework can extended to KG with 400,000 entities,highlighting the scalable nature of our approach.

2 Background

2.1 Knowledge Graph and Logical Queries

We conceptualize the knowledge graph as a first-order logic knowledge base, subsequently defining the logical query and answer set based on this framework.

Definition 2.1 (Knowledge Graph) *Let $\mathcal{E}$ and $\mathcal{R}$ be the finite set of entities and relations, a knowledge graph is a collection of factual triples $\mathcal{G} = \{(s_i, r_i, o_i)\}$, where s_i and o_i are entity objects, and r_i is a relation predicate.*

Definition 2.2 ((EFO1 query)) *The existential first order logic queries is defined as :*

$$\psi(y; x_1, \cdots, x_n) = \exists x_1, \cdots x_n.(c_1^1 \wedge \cdots \wedge c_{n_1}^1) \vee \cdots \vee (c_1^k \wedge \cdots \wedge c_{n_k}^k), \qquad (1)$$

where c_j^i is the atomic formula $r(h,t)$ or its negation $\neg r(h,t)$, r is a relation predict from $\mathcal{R}$, h and t are entity belong to $\mathcal{E}$ or a variable ranging from $\mathcal{E}$.

Definition 2.3 (Answer set) *Given an EFO1 query $\psi(y)$, the answer set is defined by*

$$\mathcal{A}[\psi(y)] = \{s \in \mathcal{E} | \psi(s/y) = \text{TRUE}\}. \qquad (2)$$

We utilize the query graph to represent logical queries due to its expressive power. With the Disjunctive Normal Form (DNF), we can consider the conjunctive query ϕ in a primary manner.

2.2 Neural Symbolic Search with Knowledge Graph Embedding

To address the incompleteness of KGs, Knowledge Graph Embedding (KGE) models were proposed to predict the trueness of involved facts. Given an atomic formula $r(s, o)$, its truth value $P_r(s, o) \in [0, 1]$ represents the trueness of the fact. With fuzzy logic (Mendel, 1995) to generalize logical operations, we then introduce the truth value function $T(\cdot)$ as follows:

Definition 2.4 (Truth value function) *Let ϕ and ψ be existential formulas, $\top$ and $\perp$ are t-norms and t-conorms, and $r \in \mathcal{R}$, $a, b \in \mathcal{E}$, with $P_r(a, b)$ representing the truth value of $r(a, b)$. The truth value function T, whose range is $[0, 1]$, is defined as follows:*

(i) $T(r(a, b)) = P_r(a, b)$,
(ii) $T(\neg\phi) = 1 - T(\phi)$
(iii) $T(\phi \wedge \psi) = T(\phi) \top T(\psi)$, $x \top_P y = x * y$
(iv) $T(\phi \vee \psi) = T(\phi) \perp T(\psi)$, $x \perp y : 1 - (1 - x) \top (1 - y)$
(iv) $T(\exists x.\phi(x)) = \max_{a \in \mathcal{E}} T(\phi(a))$

Then we can rewrite the logical query answering as the following combination optimization:

$$T(\phi(s)) = \max \quad T(\exists x_1, \cdots x_n.c_1(s) \wedge \cdots \wedge c_{n_1}(s)) \quad \text{s.t.} \quad x_i \in \mathcal{E}, 1 \leq i \leq n. \qquad (3)$$

Neural symbolic search methods solve this problem by continually pruning the query graph.

Definition 2.5 (Query Graph) *The query graph of ϕ is defined as a set of quadruples, $G_\phi = \{(h_i, r_i, t_i, \textsc{Neg}_i)\}$. The h and t are nodes, where we refer to entities as constant nodes and to variables as variable nodes. The edge is defined by a quadruple, along with two attributes: r, which denotes the relation, and $\textsc{Neg}_i$, which is the bool variable indicating whether the atom is positive.*

Definition 2.6 (Fuzzy vector) *Given the domain $\mathcal{D}$ of variable and a membership function $\mu : \mathcal{D} \to [0,1]$, the fuzzy vector of $\mathcal{D}$ is defined with $D_i = \mu(s, \mathcal{D})$, where $s \in \mathcal{D}$.*

Proposition 2.7 (RemoveConstNode) *The constant node in G_ϕ can be removed in $O(|\mathcal{E}|)$.*

Proposition 2.8 (RemoveLeafNode) *A leaf node in is a variable node that connects to only one other variable node. The leaf node in G_ϕ can be removed in $O(|\mathcal{E}|^2)$.*

The symbolic search method assigns each variable a fuzzy vector and utilizes edge removal to simplify constraints while updating the fuzzy vectors. During execution, all constant nodes are removed first, followed by a stepwise removal of leaf nodes. These operations effectively solve acyclic queries in $O(n|\mathcal{E}|^2)$, where $|\mathcal{E}|$ is the size of entities within KG and n is the size of query. However, for cyclic queries, FIT enumerates one remaining variable, leading to NP hardness $O(|\mathcal{E}|^n)$.

3 Neural Logical Indices Reduces the domain

We define the search domain $\mathcal{D}$ for each variable as the set of candidate entities for symbolic search methods. We denote $\mathcal{D}_x$ and $\mathcal{D}_y$ as the domain for x and y, respectively. We argue treating the entire entity set $\mathcal{E}$, as done by previous search algorithms (Bai et al., 2023b; Yin et al., 2024), is unnecessary because each variable must maintain consistency with its surrounding constraints (Chen et al., 2011).

3.1 Neural Logical Indices

Given a query with the variable set $V = \{x_1, \cdots, x_n, y\}$ over KG, neural logical indices are defined as a mapping from a variable in V to a subset of the entity set $\mathcal{E}$: $\mathcal{I}(x) : V \to 2^{\mathcal{E}}$. To compute the $\mathcal{I}(x)$ for each variable, we conceptually define the constraints as the subgraph pattern G_x^S from the query graph G_ϕ. Then we propose the strategy to apply neural embedding models h to identify the entities that satisfy the constraints. Let $h(G_x^S)$ denote the ranking of the entities, we determine $\mathcal{I}(x)$ by selecting the top k entities from this ranking: $\mathcal{I}(x) = \textbf{Topk}(h(G_x^S), k)$. $\mathcal{I}(x)$ serves as our reduced search domain $\mathcal{D}_x$ accelerating symbolic search algorithms.

By using NLI as the search domain for variables, we can narrow down the search space, thereby effectively simplifying the optimization in Equation 3 as follows:

$$T(\phi(s)) = \max \quad T(\exists x_1, \cdots x_n . c_1(s) \wedge \cdots \wedge c_{n_1}(s)) \quad \text{s.t.} \quad x_i \in \mathcal{D}_i, 1 \le i \le n, \tag{4}$$

where $s \in \mathcal{D}_y$ and $\mathcal{D}x_i$ is simply denoted by $\mathcal{D}_i$.

For computing neural logical indices, there is a trade-off between accuracy and efficiency. We propose two strategies: the local strategy prefers efficiency, while the global strategy emphasizes accuracy. An example in Fig. 1 illustrates the subgraph used by two constraint strategies. We define the process as the CutDomain function, which can be easily integrated with symbolic search algorithms.

3.2 Local Constraints Strategy

The local constraints strategy only involve the constraints within the following neighbor subgraph:

Definition 3.1 (Neighbor Subgraph) *Let G_ϕ be a conjunctive query graph and x be the variable in G_ϕ, the edges of neighbor subgraph for x_i is $\mathcal{N}_e(x_i, G_\phi) = \{(h_i, r_i, t_i, \textsc{Neg}_i) \in G_\phi | h_i = x \text{ or } t_i = x\}$, formed from the neighbor constraints of x_i. The $\mathcal{N}_n(x_i, G_\phi)$ is the corresponding node set.*

For universality, we only consider the information from relations within neighbor subgraph. For the logical query "$\exists x_1, x_2. \neg Graduate(x_1, x_2) \wedge Graduate(x_1, y) \wedge Married(x_2, y)$" shown in Fig. 1, it is evident that y must be the tail of the "Graduate" relation and the head of the "Married" relation. Searching the variable y within these entities is equivalent to searching the entire space $\mathcal{E}$.

To utilize the relations to prune the search domain, we propose the task that predicates the tail only given the relation. [2] This task is similar to predict missing facts of KG. To improve the generalization and training efficiency, we propose an simple but effectively hypernet to adapt the KG embedding models (Trouillon et al., 2016; Chen et al., 2021) to address this task. Then the likelihood of relation tail (r, t) can be computed by $g_s(\texttt{HN}(r), \texttt{HN}(t))$, where $\texttt{HN}$ is the hyper-network and g_s is the adapted scoring function. The details of this task can refers to Appendix E and G.

Then we can employ fuzzy logic to combine the constraints to compute the final ranking. For a possible entity e_o in the Figure 1, we employ the T-norm to calculate the scores as follows: $g_s((\texttt{HN}(\text{Married}), \texttt{HN}(e_o))\top g_s(\texttt{HN}(\text{Graduate}), \texttt{HN}(e_o)).$

3.3 Global Constraints Strategy

The global constraints extend the constraints to encompass the entire query graph, which means that $G_y^S = G_\phi$ for y. Although problem formulation has become more complex, we can leverage the ability of query embedding h to directly address the problem. For further details, see Appendix C. Following this, the two-stage coarse-to-fine ranking process is implemented, similar to the coarse-to-fine ranking used in information retrieval (Liu et al., 2019).

4 Efficient Search Framework

In this section, we introduce the Neural Logical Indices for Search Approximately (NLISA) framework. Our framework first compute the neural logical indices and use them to improve the efficiency of the node removal operation in previous methods (Bai et al., 2023a; Yin et al., 2024). Additionally, we propose a parallel local optimization operation with quadratic complexity to replace enumeration for handling cyclic queries. Visualizations of our methods addressing cyclic queries can be found in Appendix B.

4.1 Search with Neural Logic Indices

With neural logic indices to reduce search domain, we can effectively reduce the complexity for two edge removal operations. The reduced complexity is $O(|\mathcal{D}_x| + |\mathcal{D}_y|)$ and $O(|\mathcal{D}_x|^2 + |\mathcal{D}_y| \times |\mathcal{D}_y|)$ for constant node and leaf node, respectively. It is noteworthy that our improvement even exhibits quadratic growth in relation to $\frac{|\mathcal{D}|}{|\mathcal{E}|}$ for leaf variable nodes. The implementations of REMOVECONSTNODE and REMOVELEAFNODE functions, can refer to Appendix D.

4.2 Approximate Search for Cyclic Queries

For cyclic queries, after removing the leaf and constant nodes, at least one loop will still remain. FIT performs enumeration to break this loop (Yin et al., 2024), which has exponential complexity. We propose approximate search to find the assignments of variables for each candidate within domain $\mathcal{D}_y$. For the answers to the query, the truth values of the optimal assignments will be close to 1. To reduce the large search space, we utilize local search to explore the neighboring subgraph.

Without loss of generality, let $\mathcal{X}_r = \{x_1, \cdots, x_n\}$ be arranged from closest to farthest in terms of distance from y, with n being the number of remaining variables. We aim to find the corresponding $\{x_1^s, \cdots, x_n^s\}$ autoregressively over the $\mathcal{X}_r$ for each candidate entity $s \in \mathcal{D}_y$. For the truth value of each s, we use t-norms to integrate the truth values of the logical query based on the obtained assignments. At step i, we define the current query graph as $G_{\phi(o;x_1^o,\cdots,x_{i-1}^o,x_i,\cdots,x_n)}$ and let $\mathcal{N}^i$ denote the neighbor graph of x_i within current query graph. Then we optimize the most likely assignment of x_i over its local constraints:

$$x_i^o = \arg\max_{x_i}[\top_{e_i \in \mathcal{N}_e^i} T(e_i)]\top[\top_{x \in \mathcal{N}_n^i} \mu(x, C_x)]. \tag{5}$$

The complexity of the local search at each step is $O(|\mathcal{D}_x|^2 + |\mathcal{D}_x||\mathcal{D}_y|)$, which is still quadratic in complexity and can be processed in parallel. Realization of the above approximate search induces a function LOCALOPTIMIZE. We present an example of LOCALOPTIMIZE in Appendix B.

[2]Predicting the head of relation r_+ can be modeled as the tail of reverse relation r_-.

Algorithm 1 Neural logical Indices enhanced Search Approximately (NLISA)

```
Require: Input query graph $G_\phi$ and initial fuzzy vectors for existential variables and free variable as $C_x$ and $C_y$, the size of the reduced domain $|\mathcal{D}_x|$ and $|\mathcal{D}_y|$.
Ensure: Output answer vector $T(G_\phi, \{C_x\}, C_y)$
  $(\{\mathcal{D}_x\}, \mathcal{D}_y) \leftarrow$ CutDomain$(G_\phi, |\mathcal{D}_x|, |\mathcal{D}_y|)$
  RemoveConstNode$(G_\phi, \{C_x\}, C_y)$
  while $G_\phi$ contains a leaf node $x_i$ do
    RemoveLeafNode$(x_i, G_\phi, \{C_x\}, C_y)$
  end while
  for each remaining node $x_j$ in $G_\phi$ do
    LocalOptimize$(x_j, G_\phi, \{C_x\}, C_y)$
  end for
  Return $T((G_\phi, \{C_x\}, C_y))$
```

4.3 Algorithm and Complexity Analysis

Finally, we present the complete procedure of our method, as shown in Algorithm 1. Our objective is to propose an efficient symbolic search method for **general EFO1 queries**. The key aspect is that we can flexibly **reduces the search domain**. Additionally, our appropriate search reduces the complexity of answering cyclic queries from exponential to **quadratic with respect to the search domain**.

The space complexity of our method is $\mathcal{O}((|\mathcal{E}| + |\mathcal{R}| + d_h) * d)$, where d is the embedding dimension of the KG embedding and d_h is the hidden dimension of hyper-network. This complexity is the same as the knowledge graph embedding and scales linearly with the sizes of entities and relations.

The time complexity is given by $\mathcal{O}((|\mathcal{D}_x||\mathcal{D}_y| + |\mathcal{E}_x|^2 + |\mathcal{D}_x| + |\mathcal{D}_y|)d)$, where $|\mathcal{D}_x|$ and $|\mathcal{D}_y|$ are the size of search domain of the existential and free variable, respectively. It is evident that our algorithm exhibits quadratic complexity for any EFO1 query and can flexibly reduce the complexity by adjusting the size of the search domain.

5 Experiments Setting

In this section, we conduct a comprehensive evaluation of our method across diverse tasks to investigate its effectiveness and efficiency. In terms of query structure, we consider tree-form queries and general EFO1 queries. Regarding the scale of the knowledge graph, we consider graphs with 15,000, 60,000, and 400,000 entities.

5.1 Benchmarks

The BetaE benchmark (Ren and Leskovec, 2020) is the standard benchmark for complex question answering (CQA), primarily containing tree-form queries. The benchmark is comprised of three knowledge graphs (KGs): FB15k (Bordes et al., 2013), FB15k-237 (Toutanova et al., 2015), and NELL995 (Xiong et al., 2022). Specifically, the BetaE benchmark contains 14 distinct query types, with 5 types involving negation operations. It is important to note that the "pni" query type in BetaE is a universal first-order logic query, and is therefore excluded from the evaluation.

The Real EFO1 benchmark (Yin et al., 2024) proposes 10 new query types beyond the tree-from queries, with the same KGs as BetaE. In particular, the Real EFO1 benchmark introduces new patterns, including multi-graph, and cyclic graphs. The visualization of these query structures of BetaE and Real EFO1 benchmark is presented in Appendix F.

The Smore benchmark (Ren et al., 2022) considers the same tree-form queries as BetaE benchmark, but the queries are sampled from a much larger-scale KG. Since only the FB400K dataset with 40,0000 entities has been released, we select this KG as the large-scale benchmark.

5.2 Evaluation Protocol

To evaluate the effectiveness over incomplete knowledge graphs (KGs), we adopt the evaluation scheme from (Ren and Leskovec, 2020), distinguishing the answers to each query into easy and hard sets. Hard queries are defined as non-trivial queries that cannot be answered by direct traversal along the edges of the KG and require predicting at least one missing link in the test and validation splits. We assess the CQA models on these non-trivial queries by calculating the rank r for each hard answer against non-answers, and we compute the Mean Reciprocal Rank (MRR) and HIT@k.

To evaluate the efficiency, we experimentally measure the speed and running memory required to answer complex queries. We sample 100 queries for different query types in each benchmark and calculate the average queries per second (QPS). We record both the model memory and the maximum running memory during evaluations. For a fair comparison between models that support batching and those that do not, the running memory is calculated by excluding the model memory from the total running memory and then dividing the result by the batch size.

5.3 Baselines

We consider various state-of-the-art CQA methods as baselines. In particular, we compare our approach with strong baselines from symbolic search methods, including CQD-CO (Arakelyan et al., 2020), CQD-Beam (Arakelyan et al., 2020),QTO (Bai et al., 2023b), and FIT (Yin et al., 2023). Additionally, we include ConE and LMPNN as baselines for comparison with query embedding methods. We also examine the GNN-QE method, which combines graph neural networks with symbolic methods. To ensure fairness, we use the same checkpoint for those methods requiring a pre-trained neural link predictor, including CQA, QTO, FIT, and our method. For the FB15K, FB15K-237, and NELL datasets, we utilize the checkpoints provided by CQD-CO (Arakelyan et al., 2020). For FB400K, the details for the pre-trained checkpoint are provided in Appendix G. Since FIT is equivalent to QTO in tree-form queries (Yin et al., 2024), we don't distinguish them in this case.

Table 1: MRR results(%) of the Tree-Form queries on BetaE benchmark (Ren and Leskovec, 2020). The scores of the baselines are taken from their papers. We highlight the best results in red and the second-best results in blue.

Method	1p	2p	3p	2i	3i	ip	pi	2u	up	AVG.(P)	2in	3in	inp	pin	AVG.(N)
FB15K-237															
CQD-CO	**46.7**	9.6	6.2	31.2	40.6	16.0	23.6	14.5	8.2	21.9					
ConE	41.8	12.8	11.0	32.6	47.3	25.5	14.0	14.5	10.8	23.4	5.4	8.6	7.8	4.0	5.9
LMPNN	45.9	13.1	10.3	34.8	48.9	17.6	22.7	13.5	10.3	24.1	8.7	12.9	7.7	4.6	8.5
CQD-Beam	**46.7**	13.3	7.9	34.9	48.6	20.4	27.1	17.6	11.5	25.3					
GNN-QE	42.8	**14.7**	11.8	**38.3**	**54.1**	18.9	**31.1**	16.2	**13.4**	26.8	10.0	16.8	9.3	7.2	8.5
FIT/QTO	**46.7**	**14.6**	**12.8**	**37.5**	**51.6**	**21.9**	30.1	**18.0**	**13.1**	**27.4**	**14.0**	**20.0**	**10.2**	**9.5**	**13.4**
NLISA(Local)	46.6	13.8	12.2	33.1	45.8	20.2	27.1	16.9	11.4	25.2	12.8	17.3	**9.5**	9.0	12.2
NLISA(Global)	46.6	14.1	12.3	37.2	51.3	**21.3**	**30.2**	**17.6**	12.2	**27.0**	**13.8**	**20.1**	**9.5**	**9.3**	**13.2**
FB15K															
CQD-CO	89.2	25.6	13.6	77.4	78.3	44.2	33.2	41.7	22.1	46.9					
ConE	75.3	33.8	29.2	64.4	73.7	50.9	35.7	55.7	31.4	49.8	17.9	18.7	12.5	9.8	14.8
LMPNN	85.0	39.3	28.6	68.2	76.5	43.0	46.7	36.7	31.4	50.6	29.1	29.4	14.9	10.2	20.9
CQD-Beam	89.2	65.3	29.7	77.1	80.6	**71.6**	70.6	72.3	**59.4**	68.5					
GNN-QE	88.5	**69.3**	**58.7**	**79.7**	**83.5**	70.4	69.9	**74.1**	**61.0**	**72.8**	**44.7**	**41.7**	**42.0**	30.1	**39.6**
FIT/QTO	**89.4**	**65.6**	**56.9**	**79.1**	**83.5**	**71.8**	**73.1**	**73.9**	59.0	**72.5**	40.2	38.9	34.8	28.1	35.5
NLISA(local)	87.2	61	53.1	67.7	75.4	66.2	65	56.3	46.7	64.3	44.8	39.9	36.8	**33.0**	38.6
NLISA(Global)	**89.4**	64.7	55.4	76.4	82.3	71.1	**71.1**	71.9	57.5	71.1	**48.3**	**44.4**	**38.0**	**34.5**	**41.3**
NELL															
CQD-CO	60.4	17.8	12.8	39.3	46.6	22.1	30.1	17.3	13.2	28.8					
ConE	53.1	16.1	13.9	40.0	50.8	26.3	17.5	15.3	11.3	27.2	5.7	8.1	10.8	3.5	6.4
LMPNN	60.6	22.1	17.5	40.1	50.3	24.9	28.4	17.2	15.7	30.8	8.5	10.8	12.2	3.9	8.9
CQD-Beam	60.4	22.6	13.6	43.6	53.0	25.6	31.2	19.9	16.7	31.8					
GNN-QE	53.3	18.9	14.9	42.4	52.5	18.9	30.8	15.9	12.6	28.9	9.9	14.6	11.4	6.3	10.6
FIT/QTO	**60.8**	**23.8**	**21.2**	**44.3**	**54.1**	26.6	31.7	**20.3**	**17.6**	**33.4**	**12.6**	**16.4**	**15.3**	**8.3**	**13.2**
NLISA(Local)	60.4	23.5	**20.1**	43.7	**54.1**	**27.0**	**32.8**	**20.3**	17.3	33.2	12.7	**16.5**	**15.2**	**8.3**	**13.2**
NLISA(Global)	**60.8**	**23.5**	20.0	**44.4**	**54.4**	**27.2**	**33.1**	**20.3**	**17.4**	**33.5**	**12.5**	**16.5**	**15.3**	**8.3**	**13.2**

Table 2: Efficiency results of the Tree-Form queries on BetaE benchmark (Ren and Leskovec, 2020).

KG	Metric	ConE	LMPNN	CQD-CO	FIT	NLISA(local)	NLISA(global)
FB15k-237	Queries per Second ↑	100	105	14	31	**115**	86
	CUDA Memory of Running (M) ↓	458	355	**176**	1208	232	337
FB15K	Queries per Second ↑	97	101	10.1	20	**109**	79
	CUDA Memory of Running (M) ↓	338	365	**189**	2068	255	381
NELL	Queries per Second ↑	23	24	7	3	**35**	20
	CUDA Memory of Running (M) ↓	1635	1955	760	15324	**774**	953

6 Experiments Results

Our experimental results reveal two key insights. First, the search domain can be significantly reduced with minimal performance loss, thereby alleviating efficiency and scalability issues. Second, in addressing cyclic queries, our proposed approximate search exhibits quadratic complexity with respect to the search domain and achieves performance comparable to that of precise symbolic search methods. To simplify the analysis, we set $|\mathcal{D}_x| = |\mathcal{D}_y|$ as approximately 10% of the corresponding entity size. Specifically, we set $\mathcal{D}_x$ as 2000, 2000, and 6000 for FB15k-237, FB15k, and NELL, respectively. We present the results of three benchmarks in the following three sections. The first insight is demonstrated across all three benchmarks, while the second insight is illustrated on the Real EFO1 benchmark in Section 6.2. In the implementation, both NLISA (Local) and NLISA (Global) utilize the local constraints strategy for existential variables. NLISA (Local) apply the local strategy for free variables, while NLISA (Global) solves the global constraints by LMPNN (Wang et al., 2023b).

6.1 Tree Form Queries: BetaE benchmark

We present the results of performance and efficiency on BetaE benchmark in Table 1 and Table 2, respectively. Table 1 demonstrates that NLISA (Global), with a 10% reduction in the search domain, achieves nearly lossless performance on average across three datasets when compared to the state-of-the-art symbolic search method, FIT (Yin et al., 2024). Notably, the average results of NLISA (Global) on negative queries do not decline. The results indicate that the search domain of symbolic method can be significantly reduced, validating the effectiveness of our proposed neural logic indices. Although NLISA (Local) produces lower results, it offers a higher QSP compared to NLISA (Global), making it practical since it does not require training query embedding models. The strong performance of NLISA (Global) underscores the need for the development of efficient and effective query embedding models. Table 2 shows that symbolic search methods, including CQD-CO and FIF, exhibit remarkably low QSP, while FIT suffers from rapid CUDA memory usage as the size of the KG increases. Our methods, with the reduced search domain, enhance efficiency by improving QSP and decreasing CUDA memory usage.

6.2 General EFO1 queires: real EFO1 benchmark

Table 3: MRR results(%) of the queries on the real EFO1 benchmark (Yin et al., 2024). The scores of the baselines are directly taken from Yin et al. (2024). We highlight the best results in red and the second-best results in blue.

Method	pni	2il	3il	2m	2nm	3mp	3pm	im	3c	3cm	AVG.
					FB15K-237						
CQD-CO	7.7	29.6	46.1	6.0	1.7	6.8	3.3	12.3	25.9	23.8	16.3
ConE	10.8	27.6	43.9	9.6	7.0	9.3	7.3	14.0	28.2	24.9	18.3
LMPNN	10.7	28.7	42.1	9.4	4.2	9.8	7.2	15.4	25.3	22.2	17.5
QTO	12.1	28.9	47.9	8.5	10.7	11.4	6.5	17.9	38.3	35.4	21.8
FIT	**14.9**	**34.2**	**51.4**	**9.9**	**12.7**	**11.9**	**7.7**	**19.6**	**39.4**	**37.3**	**23.9**
NLISA(Local)	12.5	34.0	51.8	9.2	**9.8**	10.3	7.4	17.8	34.9	34.5	22.2
NLISA(Global)	**14.2**	**34.3**	**52.7**	**9.5**	9.2	**10.7**	**7.4**	**18.1**	**36.0**	**35.5**	**22.8**
					FB15K						
CQD-CO	7.7	29.6	46.1	6.0	1.7	6.8	3.3	12.3	25.9	23.8	16.3
ConE	37.0	40.1	57.3	33.3	11.5	23.9	27.6	38.7	35.0	36.3	34.1
LMPNN	38.7	43.2	57.8	40.3	7.9	24.0	30.5	48.4	32.2	30.9	35.4
QTO	48.2	49.5	68.2	64.6	19.4	48.5	53.7	73.9	53.3	54.9	53.4
FIT	**57.9**	**70.4**	**77.6**	**73.5**	**39.1**	**57.3**	**64.0**	**79.4**	**63.8**	**65.4**	**64.8**
NLISA(local)	55.9	69.1	76.0	66.0	**38.3**	49.1	55.2	70.5	57.8	59.7	59.8
NLISA(Global)	**60.7**	**70.4**	**78.0**	**68.8**	36.2	**51.0**	**57.0**	**76.1**	**60.3**	**61.5**	**62.0**
					NELL						
CQD-CO	7.9	48.7	68.0	31.7	1.5	12.9	13.8	33.9	38.8	35.9	29.3
ConE	10.3	42.1	65.8	32.4	7.0	12.6	16.8	34.4	40.2	38.2	30.0
LMPNN	11.6	43.9	62.3	35.6	6.2	15.9	19.3	38.3	39.1	34.4	30.7
QTO	12.3	48.5	68.2	38.8	12.3	22.8	19.3	41.1	45.4	43.9	35.3
FIT	**14.4**	**53.3**	69.5	**42.1**	**12.5**	**24.0**	22.8	**41.5**	**47.5**	**45.3**	**37.3**
NLISA(local)	**14.0**	53.1	**71.7**	23.0	**11.8**	**23.0**	**23.0**	41.1	**45.6**	**44.4**	**36.8**
NLISA(Global)	**14.0**	**53.4**	**72.2**	**40.9**	10.6	22.6	**23.1**	**41.5**	**46.0**	**44.1**	**36.8**

Here, we present our results in Table 3 on the real EFO1 benchmark beyond tree-form queries. For baselines QTO and ConE relied on the operator tree, the new queries cannot be represented by an operator tree and can only be **syntactically approximated**. Table 3 shows that our methods achieve nearly 95% of the performance of symbolic search methods, outperforming all other baselines. This result indicates that our reduction framework is superior to the tree-structure approximation used by QTO. Regarding cyclic queries, including "3c" and "3cm", our approximate search framework achieves an average performance of almost 95% compared to FIT, validating our second insight. Our method even outperforms over "2il" and "3il", which we conjecture is due to the half-precision used in FIT to reduce memory usage. As shown in Fig. **??**, our method demonstrates significantly higher QSP compared to FIT.

6.3 Large scale Tree-Form Queries: Smore benchmark

Table 4: Averaged MRR results(%) on large KGs with different methods. The results of GQE, Q2B, and BetaE are taken from Ren et al. (2022). The CQD-CO and NLISA(Local) use the same backbone of knowledge graph embedding.

	GQE	Q2B	BetaE	CQD-CO	FIT/QTO	NLISA(Local)
FB400K	36.0	51.7	50.5	43.3	OOM	**58.9**

Since training query embeddings on large-scale knowledge graphs (KGs) is challenging, we can only consider existing query embedding methods and symbolic search using knowledge graph embeddings as baselines. For the FB400K dataset, which contains 400,000 entities, we set $|\mathcal{D}_x| = |\mathcal{D}_y| = 8000$, representing only 2% of the total. Table 4 demonstrates that precise symbolic search methods face out-of-memory issues due to their quadratic complexity with respect to $|\mathcal{E}|$. This can be anticipated based on the rapid increase in CUDA memory usage as the number of entities grows, as shown in Table2. Our method, which employs a local constraints strategy, outperforms both the query embedding methods and CQD-CO. This underscores the scalability challenges encountered by precise symbolic search methods and highlights the importance of effective pruning techniques over the search domain.

7 Conclusion

Complex query answering over knowledge graphs is a crucial multi-hop reasoning task aimed at addressing first-order logical queries over incomplete KG. Although symbolic search methods exhibit strong performance, they face efficiency challenge that hinders further development and application. Query embedding models offer fast speed but often provide generic performance. In this paper, we integrate their advantage together in a mutually beneficial way and propose an approximate search framework combined with flexible neural logical indices to address these efficiency issues. The neural logical indices can be computed rapidly using embedding methods, significantly reducing the search domain of symbolic methods. In particular, the approximate search framework can handle cyclic queries well with a quadratic complexity. Our approximate search is precise for acyclic queries. Experiments on various benchmarks show that, with a 10% reduced search domain, our method achieves 90% performance, including for cyclic queries, while the QSP assess efficiency is improved to be comparable to that of query embedding methods. Additionally, we demonstrate that our framework can execute on a KG with an order of magnitude more entities than before, highlighting the scalable nature of our approach.

References

Arakelyan, E., Daza, D., Minervini, P., and Cochez, M. (2020). Complex Query Answering with Neural Link Predictors. In *International Conference on Learning Representations*.

Arakelyan, E., Minervini, P., and Augenstein, I. (2023). Adapting Neural Link Predictors for Complex Query Answering. arXiv:2301.12313 [cs].

Bai, J., Liu, X., Wang, W., Luo, C., and Song, Y. (2023a). Complex Query Answering on Eventuality Knowledge Graph with Implicit Logical Constraints. arXiv:2305.19068 [cs].

Bai, J., Wang, Z., Zhang, H., and Song, Y. (2022). Query2Particles: Knowledge Graph Reasoning with Particle Embeddings. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 2703–2714.

Bai, Y., Lv, X., Li, J., and Hou, L. (2023b). Answering Complex Logical Queries on Knowledge Graphs via Query Computation Tree Optimization. In *Proceedings of the 40th International Conference on Machine Learning*, pages 1472–1491. PMLR. ISSN: 2640-3498.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Chen, H., Dalmau, V., and Grußien, B. (2011). Arc consistency and friends. *J. Log. Comput.*, 23:87–108.

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

Chen, Y., Minervini, P., Riedel, S., and Stenetorp, P. (2021). Relation prediction as an auxiliary training objective for improving multi-relational graph representations. In *3rd Conference on Automated Knowledge Base Construction*.

Choudhary, N., Rao, N., Katariya, S., Subbian, K., and Reddy, C. (2021). Probabilistic entity representation model for reasoning over knowledge graphs. *Advances in Neural Information Processing Systems*, 34:23440–23451.

Gottlob, G., Leone, N., and Scarcello, F. (2000). A comparison of structural CSP decomposition methods. *Artificial Intelligence*, 124(2):243–282.

Hamilton, W., Bajaj, P., Zitnik, M., Jurafsky, D., and Leskovec, J. (2018). Embedding logical queries on knowledge graphs. *Advances in neural information processing systems*, 31.

Hu, W., Fey, M., Zitnik, M., Dong, Y., Ren, H., Liu, B., Catasta, M., and Leskovec, J. (2020). Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133.

Ji, S., Pan, S., Cambria, E., Marttinen, P., and Philip, S. Y. (2021). A survey on knowledge graphs: Representation, acquisition, and applications. *IEEE transactions on neural networks and learning systems*, 33(2):494–514.

Liu, H., Shi, S., and Huang, H. (2019). Coarse-to-fine document ranking for multi-document reading comprehension with answer-completion. *2019 International Conference on Asian Language Processing (IALP)*, pages 407–412.

Liu, L., Du, B., Ji, H., Zhai, C., and Tong, H. (2021). Neural-answering logical queries on knowledge graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 1087–1097, New York, NY, USA. Association for Computing Machinery.

Mendel, J. M. (1995). Fuzzy logic systems for engineering: a tutorial. *Proceedings of the IEEE*, 83(3):345–377.

Ren, H., Dai, H., Dai, B., Chen, X., Zhou, D., Leskovec, J., and Schuurmans, D. (2022). Smore: Knowledge graph completion and multi-hop reasoning in massive knowledge graphs. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1472–1482. arXiv:2110.14890 [cs].

Ren, H., Galkin, M., Cochez, M., Zhu, Z., and Leskovec, J. (2023). Neural Graph Reasoning: Complex Logical Query Answering Meets Graph Databases. arXiv:2303.14617 [cs].

Ren, H., Hu, W., and Leskovec, J. (2020). Query2box: Reasoning Over Knowledge Graphs In Vector Space Using Box Embeddings. In *International Conference on Learning Representations (ICLR)*.

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

Safavi, T. and Koutra, D. (2020). Codex: A comprehensive knowledge graph completion benchmark. *arXiv preprint arXiv:2009.07810*.

Sun, Z., Deng, Z.-H., Nie, J.-Y., and Tang, J. (2018). RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. In *International Conference on Learning Representations*.

Toutanova, K., Chen, D., Pantel, P., Poon, H., Choudhury, P., and Gamon, M. (2015). Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1499–1509.

Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., and Bouchard, G. (2016). Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR.

Wang, Z., Fei, W., Yin, H., Song, Y., Wong, G. Y., and See, S. (2023a). Wasserstein-Fisher-Rao Embedding: Logical Query Embeddings with Local Comparison and Global Transport. *arXiv preprint arXiv:2305.04034*.

Wang, Z., Song, Y., Wong, G., and See, S. (2023b). Logical Message Passing Networks with One-hop Inference on Atomic Formulas. In *The Eleventh International Conference on Learning Representations*.

Wang, Z., Yin, H., and Song, Y. (2021). Benchmarking the Combinatorial Generalizability of Complex Query Answering on Knowledge Graphs. *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 1.

Wang, Z., Yin, H., and Song, Y. (2022). Logical Queries on Knowledge Graphs: Emerging Interface of Incomplete Relational Data. *Data Engineering*, page 3.

Xiong, B., Potyka, N., Tran, T.-K., Nayyeri, M., and Staab, S. (2022). Faithful Embeddings for \$\$\$\backslash\$mathcal \${\$e\$}\$\$\backslash\$mathcal \${\$l\$}\$^\${\$++\$}\$ \$\$ Knowledge Bases. In *International Semantic Web Conference*, pages 22–38. Springer.

Yang, D., Qing, P., Li, Y., Lu, H., and Lin, X. (2022). GammaE: Gamma Embeddings for Logical Queries on Knowledge Graphs. arXiv:2210.15578 [cs].

Yin, H., Wang, Z., and Song, Y. (2023). On Existential First Order Queries Inference on Knowledge Graphs. arXiv:2304.07063 [cs].

Yin, H., Wang, Z., and Song, Y. (2024). Rethinking existential first order queries and their inference on knowledge graphs. In *The Twelfth International Conference on Learning Representations*.

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

Zhu, Z., Galkin, M., Zhang, Z., and Tang, J. (2022). Neural-Symbolic Models for Logical Queries on Knowledge Graphs. *arXiv preprint arXiv:2205.10128*.

A Related Work

A.1 Query Embedding methods

Answering complex logical queries over knowledge graphs is naturally extended from link prediction and aims to handle queries with complex conditions beyond simple link queries. This task gradually grows by extending the scope of complex logical queries, ranging from conjunctive queries (Hamilton et al., 2018) to Existential Positive First-Order (EPFO) queries (Ren et al., 2020), Existential First-Order (EFO) queries (Ren and Leskovec, 2020), real Existential First-Order queries (Yin et al., 2024). The primary method is query embedding, which maps queries and entities to a low-dimensional space. The form of embedding has been well investigated, such as vectors (Hamilton et al., 2018; Chen et al., 2022; Bai et al., 2022), geometric regions (Ren et al., 2020; Zhang et al., 2021; Liu et al., 2021), and probabilistic distributions (Ren and Leskovec, 2020; Choudhary et al., 2021; Yang et al., 2022; Wang et al., 2023a). These methods not only explore knowledge graphs embedding but also leverage neural logical operators to generate the embedding of complex logical queries.

A.2 Symbolic methods

Neural-symbolic CQA models represent variables as fuzzy sets with vector forms $\mu \in [0,1]^{\mathcal{E}}$ and apply fuzzy logic to model the logic operations naturally. Built on the operator tree, GNN-QE (Zhu et al., 2022) represents the intermediate variable as a fuzzy vector, and simultaneously adapts graph neural network from KG completion to execute relation projection and models the logical operations with fuzzy logic. In particular, the neural symbolic search method combining the knowledge graph embedding with symbolic search is of particular interest. CQD-beam (Arakelyan et al., 2020, 2023) uses beam search during the execution of operator tree, maintaining only a beam width of entities for intermediate variables. CQD-CO uses gradient optimization to estimate the embedding of existential variables (Arakelyan et al., 2020). Unlike previous approximate search methods, QTO is a precise symbolic search method because it finds that the tree-form queries can be solved on $O(|\mathcal{E}|^2)$. FIT (Yin et al., 2024) extends its scope to general EFO1 by using the enumeration to handle cyclic queries. However, the complexity is $|\mathcal{E}|^n$ (Yin et al., 2024), where n is the variable number of query. In general, the above two precise methods constantly remove nodes and preserve results with fuzzy vectors corresponding to variables.While symbolic methods demonstrate good performance and strong interpretability, they struggle with high computational complexity.

There are many other models and datasets proposed to enable answering queries with good performance and additional features, see the comprehensive survey Ren et al. (2023).

B Details of symbolic search pipelines

We present a visualization of the pipelines, including our proposed NLISA and FIT (Yin et al., 2024), in Figure 2. Both pipelines are designed to address the given cyclic query. Figure 2 illustrates that our proposed NLISA implements an efficient node removal operation using neural logic indices. Additionally, Figure 2 demonstrates the local optimization of our method, which aims to achieve parallel optimization for the best assignments for each candidate of free variables.

C Details of constraints strategies

For local constraints, when the variable involves logical disjunction, we use the T-conorm to handle it.

For global constraints, the computation of existential variables should treat them as free variables, while the original free variables are converted into existential variables. This allows us to use query embedding to compute the global constraints for the existential variables.

D Details of functions

$\textsc{RemovConstNode}(G_\phi, |\mathcal{D}_x|, |\mathcal{D}_y|)$ removes the constant nodes based on the given domain of variables. We consider the simplest case where $(s, r, e, \textsc{Neg} = \textsc{False})$, with s as the constant node,

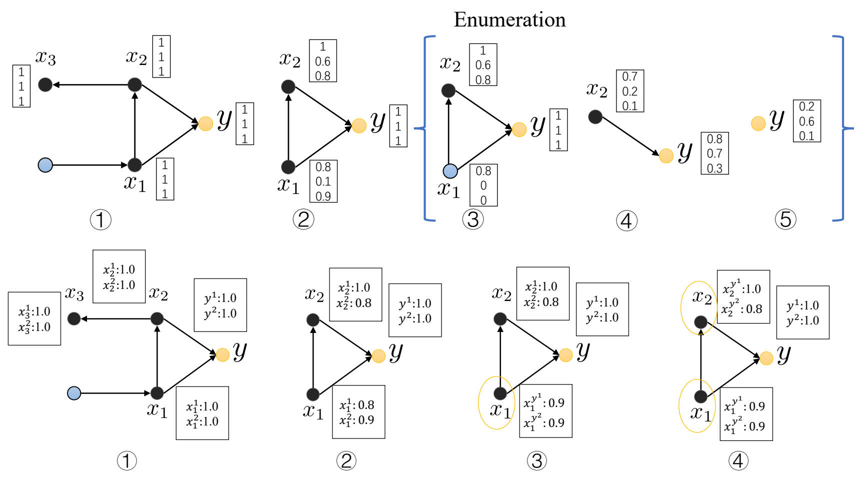
Figure 2: The top section depicts the pipeline of FIT (Yin et al., 2024) for addressing a given cyclic query, while the bottom section illustrates the pipeline of NLISA tackling the same cyclic query. NLISA maintains a smaller fuzzy vector by preserving only the indices within the relevant domains, resulting in greater flexibility. Additionally, NLISA employs Local Optimization to search for assignments for each candidate of free variables, which can be executed in parallel and operates with quadratic complexity.

r as the relation, e as the existential variable, and NEG $=$ FALSE indicating that this edge is positive. We first construct the symbolic representation of this edge: $\mathbf{s} = [P_r(s, e^i)] \in \mathcal{R}^{|\mathcal{D}_e|}, e^i \in \mathcal{D}_e$. We then update this representation into the fuzzy vector of the existential variable e by t-norm: $\mu(e, \mathcal{D}) = \mu(e, \mathcal{D}) \top \mathbf{s}$. Consequently, we can remove this edge while preserving its constraints in the fuzzy vector of e. The above method can be generalized to other cases, such as when the variable is a free variable y and the edge is negative. A similar update approach can be used, and specific details can be found in Yin et al. (2024).

REMOVCONSTNODE$(G_\phi, |\mathcal{D}_x|, |\mathcal{D}_y|)$ removes one leaf node based on the given domain of variables. We consider the most difficult case where $(e_1, r, e_2, \text{NEG} = \text{FALSE})$, with s as the constant node, r as the relation, e as the existential variable, and NEG $=$ FALSE indicating that this edge is positive. The symbolic representation of this edge results in a matrix: $\mathcal{S} = [P_r(e_1^i, e_2^j)] \in \mathcal{R}^{|\mathcal{D}_{e_1}| \times |\mathcal{D}_{e_2}|}, e_1^i \in \mathcal{D}_{e_1}, e_2^i \in \mathcal{D}_{e_2}$. We then update this representation into the fuzzy vector of the existential variable e by t-norm and max operation: $\mu(e, \mathcal{D}) = \mu(e, \mathcal{D}) \top \max(\mathcal{S}, axis = 0)$. The proof of the effectiveness of the above update can be found in Yin et al. (2024), and this update can be easily extended to other cases, such as when the variable is a free variable y and the edge is negative.

E Details of knowledge graph embedding and truth values

E.1 Knowledge graph embedding and HyperNet

Knowledge graph embedding first embeds the entities and relations into a continuous space. Given the query $(s, r, ?)$, we first compute its embedding and compare it with the entities within the embedding spce. The estimated embedding of $(s, r, ?)$ is first computed by $f_t(e_s, e_r)$, where $f(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ is the transformation function. Then, the likelihood of $r(s, o)$ is computed by the scoring function $f_s(f_t(e_r, e_s), e_o)$, where $f_s(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to [-\infty, +\infty]$ is the scoring function related to the embedding space. The scoring function is usually a distance function or a similar metric in the embedding space.

For relation tail prediction task, we adapt existing KG embedding models using a hypernet. The hypernet modifies the embeddings of entities and relations, while we employ the same scoring function as the original embedding models. Given the query $(r, ?)$ and candidate entity t, the score

$g_s(\texttt{HN}(r), \texttt{HN}(t))$, where HN represents the hypernet and $g_s = \sigma(f_s)$ with $\dot{\sigma()}$ being the sigmoid activation function.

E.2 Truth values

The symbolic representation in the previous symbolic search method QTO and FIT is constructing the set of truth values matrix for the whole knowledge graph To convert real number scores computed by knowledge graph embedding modes to truth value that falls into $[0, 1]$, QTO/FIT use the softmax function: $P^{\star}_{r,a}(b) = \frac{exp(s(a,r,b))}{\Sigma_{c\in\mathcal{E}} exp(s(a,r,c))}$. Next, QTO and FIT scale the results of the softmax function using a factor based on the observed edges in the training graph since softmax outputs a vector that sums to 1: $\mathcal{G}_o$.

$$Q_{a,b} = \begin{cases} \frac{|\{d|(a,r,d)\in\mathcal{G}_o\}|}{\Sigma_{c\in\{d|(a,r,d)\in\mathcal{G}_o\}} P^{\star}_{r,a}(c)}, & \text{if } |\{d|(a,r,d) \in \mathcal{G}_o\}| > 0 \\ 1, & \text{if } |\{d|(a,r,d) \in \mathcal{G}_o\}| = 0 \end{cases} \tag{6}$$

where $E_{a,r} = \{b \mid (a, r, b) \in \mathcal{G}_o\}$ represents the set of observed edges. Then the a-th row of r-th matrix is got by clamping the value for each element:

$$P_r(a, b) = min(1, P^{\star}_{r,a}(b) \times Q_{a,b}) \tag{7}$$

They then mark the observed edges and set the truth value for these edges to 1. The scaling and marking operations are performed on a case-by-case basis for each fact, which cannot be parallelized.

We demonstrate that these scaling operations can be parallelized through caching. For cases where $|\{d|(a, r, d) \in \mathcal{G}_o\}| > 0$: the truth value is computed by the following normalization:

$$Q_{a,b} = \frac{exp(f_r(a, b))}{\sum_{\{d|(a,r,d)\in\mathcal{G}_o\}} exp(f_r(a, d))}. \tag{8}$$

To simplify the calculations, we use log-scale operations

$$log(Q_{a,b}) = f_r(a, b) - log(\sum_{\{d|(a,r,d)\in\mathcal{G}_o\}} exp(f_r(a, d))). \tag{9}$$

If we cache $S^r_a = log(\sum_{\{d|(a,r,d)\in\mathcal{G}_o\}} exp(f_r(a, d)))$ with $|\mathcal{E}| \times |\mathcal{R}|$ size, we can parallel index the cached values to optimize the computation of Equation 9. For cases where $|\{d|(a, r, d) \in \mathcal{G}_o\}| > 0$, we have:

$$S^r_a = log(\sum_{\{d\in\mathcal{E}\}} exp(f_r(a, d))), \tag{10}$$

which allows the scaling operation to yield the softmax result. In general: the cached values $S \in R^{\mathcal{E}|\times|\mathcal{R}}$ are defined as follows:

$$S^r_a = \begin{cases} log(\sum_{\{d|(a,r,d)\in\mathcal{G}_o\}} exp(f_r(a, d))), & \text{if } |\{d|(a, r, d) \in \mathcal{G}_o\}| > 0 \\ log(\sum_{\{d\in\mathcal{E}\}} exp(f_r(a, d))), & \text{if } |\{d|(a, r, d) \in \mathcal{G}_o\}| = 0 \end{cases} \tag{11}$$

This caching mechanism grows linearly with the size of the knowledge graph. By utilizing this caching strategy, we can perform parallel computations for the scaling operations, facilitating efficient dynamic symbolic representation construction. However, when it comes to listing the marked training facts, our symbolic representation may exhibit slightly lower performance compared to the original construction.

However, the mask created based on the training facts cannot be completed in parallel. Therefore, we rely on previous steps by caching. This is a key issue that may lead to a drop in our method's performance in experiments.

F Details of query structures

We present the visualization of query types BetaE benchmark (Ren and Leskovec, 2020) and Real EFO1 benchmark (Yin et al., 2023) in Fig. 3 and Fig. 4, where the visualization of BetaE benchmark is taken from Wang et al. (2023a) and the visualization of Real EFO1 benchmark is taken from Yin et al. (2024).

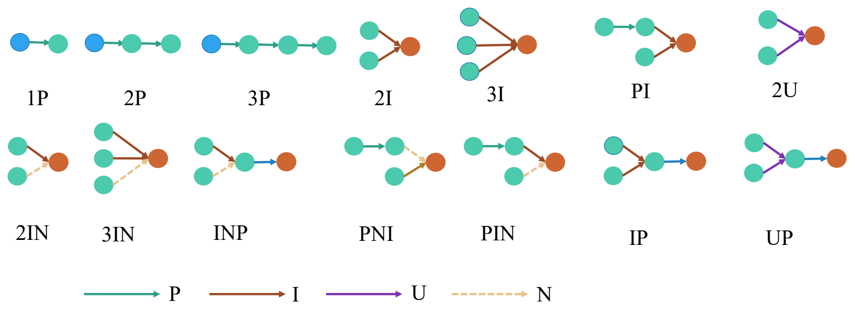

Figure 3: 14 query types propsed in BetaE benchmark (Ren and Leskovec, 2020). These query types are modeled by the operator tree.

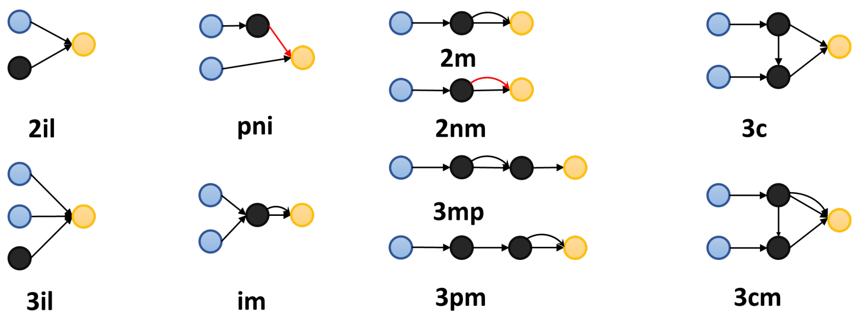

Figure 4: 10 query types propsed in Real EFO1 benchmark (Yin et al., 2024). These query types are modeled by query graph.

G Details of implementation

G.1 Knowledge graph completion

We reproduce the results from previous work (Chen et al., 2021) to train the embedding and hypernetwork. For the FB400K dataset, we search the hyperparameters with the following settings: learning rates of $[1 \times 10^{-1}, 1 \times 10^{-2}]$, embedding dimensions of $[100, 200, 400]$, and λ values of $[0.0005, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 0]$. We utilize the ComplEx model with the N3 regularizer. The embedding initialization is set to 1×10^{-3}, and we employ the Adagrad optimizer.

G.2 Relation tail prediction

Why Use Hypernet?

Relation tail prediction is fundamentally similar to knowledge graph completion (KGC) tasks, as both involve inferring missing tails from a knowledge graph. In KGC, the query is presented as $(h, r, ?)$, to predict the tail t. In contrast, relation tail prediction aims to predict the tail t given the relation r. This similarity makes the embedding methods used in KGC equally applicable to relation tail prediction. Hypernet is employed to generate weights dynamically, allowing the model to automatically adjust representations for new tasks. This facilitates the sharing and adaptation of weights across different tasks. By adopting the embeddings from KGC, Hypernet enhances the utilization of existing knowledge and enables quicker adaptation to relation tail prediction. Additionally, Hypernet requires only a few parameters, improving performance while reducing training costs and storage needs.

How Is Hypernet Trained?

The training process is similar to that of KGC, where each triple (h,r,t)(h,r,t) is replaced with (r,t)(r,t). This requires only minor modifications compared to the existing KGC framework.

What Is the Performance of Hypernet?

Table 5: MRR results for the trained Hypernet.

	FB15K-237	FB15K	NELL	FB400K
MRR	1.0	0.998	0.99	1.0

For the hypernet used for relation tail prediction, we set the learning rate to 1×10^{-2} and search the hidden dimensions $[100, 200, 400]$ also using the ComplEx model with the N3 regularizer. The embedding initialization remains at 1×10^{-3}, and the optimizer is Adagrad. The performance of our method is quite impressive, as demonstrated in Table 5. The results indicate a strong capability of the hypernet in addressing relation tail prediction tasks.

H t-norm introduction

Definition H.1 (t-norm) *A t-norm $\top$ is a function: [0,1] x [0,1] $\rightarrow$ [0,1] that satisfies the following properties:*

(i) Communitavity: $a \top b = b \top a$

(ii) Monotonicity: $(a \top b) \leq (c \top b)$ *if* $a \leq c$

(iii) Associativity: $(a \top b) \top c = a \top (b \top c)$

(iv) Neutrality: $a \top 1 = a$

Then the t-conorm $\perp$ is directly defined by $a \perp b = 1 - (1 - a) \top (1 - b)$, which follows the De Morgan's law.

Finally, we introduce some common t-norms which are of interest:

(i) Godel: $a \top_G b = \min(a, b)$
(ii) Product: $a \top_P b = a * b$
(iii) Łukasiewicz: $a \top_{LK} b = \max(a + b - 1, 0)$

In the main paper, we mainly focus on the Godel and Product t-norm.