Transformers for Complex Query Answering over Knowledge Hypergraphs

Anonymous ACL submission

Abstract

Complex Query Answering (CQA) has been extensively studied in recent years. In order to model data that is closer to real-world distribution, knowledge graphs with different modalities have been introduced. Triple KGs, as the classic KGs composed of entities and relations of binary, have limited representation of real-world facts. Real-world data is more sophisticated. While hyper-relational graphs have been introduced, there are limitations in representing relationships of varying arity that contain entities with equal contributions. To address this gap, we sampled new CQA datasets: JF17k-HCQA and M-FB15k-HCQA. Each dataset contains various query types that include logical operations such as projection, negation, conjunction, and disjunction. In order to answer knowledge hypergraph (KHG) existential first-order queries, we propose a two-stage transformer model, the Logical Knowledge Hypergraph Transformer (LKHGT), which consists of a Projection Encoder for atomic projection and a Logical Encoder for complex logical operations. Both encoders are equipped with Type Aware Bias (TAB) for capturing token interactions. Experimental results on CQA datasets show that LKHGT is a state-of-the-art CQA method over KHG and is able to generalize to out-of-distribution query types.

1 Introduction

Knowledge graphs (KG), comprising facts with entities as nodes and relations as edges, have gained much attention recently. Triplets (head, relation, tail) represent binary relational facts in KG. Under the Open World Assumption (OWA) (Ji et al., 2022), most existing graphs are considered incomplete. Both embedding methods and message passing approaches have been proposed to learn the underlying representations of entities and relations in KG, aiming to mine undiscovered facts.

Complex Query Answering (CQA) in KG is an important task in Neural Graph Database (NGDB) (Ren et al., 2023). In NGDB, KG serves as the data source, where CQA involves performing logical queries over incomplete KG via parameterization of entities, relations, and logical operators like projection, negation, conjunction, and disjunction. Various neural query approaches have been developed to infer missing links in KG and enrich the answer set.

Existentially quantified First Order queries with a single variable (EFO-1) are the primary type of complex query. While classic KG representations (Trouillon et al., 2016) excel in one-hop inference, they fall short for complex queries. Different approaches have been proposed for answering EFO-1 queries. Embedding models (Chen et al., 2022; Ren and Leskovec, 2020; Zhang et al., 2021) model entities and relations in vector spaces, transforming queries into operator tree formats, with logical operators interacting with entity embeddings. Other methods, like CQD (Arakelyan et al., 2021) and LMPNN (Wang et al., 2023), utilize pretrained KG embeddings to tackle EFO-1 queries. CQD formulates logical inference as an optimization problem, while LMPNN treats EFO-1 queries as query graphs, instantiating nodes with pretrained embeddings and executing message passing to predict answer embeddings.

However, less attention has been given to complex queries beyond binary relations. Most current models focus on ordinary KG, despite the abundance of n-ary facts in modern large KG. Hypergraphs and hyper-relation graphs are suitable for constructing KG to accommodate n-ary relations. StarQE (Alivanistos et al., 2022) introduces hyper-relational graphs in WD50K-QE, where each triple is built from qualified statements (head, relation, tail, qp), with qp as contextual information. However, hyper-relational graphs differ from hypergraphs; they treat n-ary facts as triples with supporting facts, focusing on query answering. Hypergraphs better represent entities with equal contribu-

tions in relations. For instance, the relation coauthor(authorA,authorB,authorC) can be correctly depicted by a hyperedge in a hypergraph but misrepresented in a hyper-relational graph. Efforts have been made to create Knowledge Hypergraph (KHG) embeddings in M-FB15k (Fatemi et al., 2020) and JF17k (Wen et al., 2016), primarily for link prediction. However, little work has focused on CQA in KHG, with LSGT (Bai et al., 2024) being the only study investigating CQA with ordered hyperedges, using binary relations for intermediate variable nodes.

In this paper, we propose a novel two-stage transformer model, Logical Knowledge Hypergraph Transformer (LKHGT), a transformer-based approach for reasoning over KHGs, extending CQA from binary to n-ary relations. LKHGT consists of two encoders: the Projection Encoder, which predicts answers for variables in atomic hyperedges, and the Logical Encoder, which derives intermediate variable embeddings for logical operations. Both encoders are equipped with Type Aware Bias (TAB), which helps identify each input token type and capture their interactions. We define the logical query in EFO-1 Disjunctive Normal Form and construct ordered query hypergraphs. Each ordered hyperedge represents an atomic formula, including a predicate and potentially a negation operator.

We investigate LKHGT's performance against baseline models. To ensure fair experiments, we included multiple methods capable of reasoning on hypergraphs and extended LMPNN (Wang et al., 2023) to relational ordered knowledge hypergraphs, utilizing pretrained KHG embeddings with logical inference capabilities.

Our experiments show that LKHGT outperforms other n-ary CQA models in ordered hyperedge settings. The model generalizes from edges of binary to arity N and achieves promising results. After learning basic logical operations, our model also demonstrates good performance on out-of-distribution queries. By comparing the Logical Encoder and fuzzy logic (Zadeh, 1988) for logical operations, we show that our transformer-based encoder outperforms neural symbolic methods using fuzzy logic. Thus, our approach bridges the gap between CQA and Knowledge Hypergraphs.

2 Related Works

2.1 Knowledge Hypergraphs Embedding

BoxE (Abboud et al., 2020), HypE, ReAlE (Fatemi et al., 2021, 2020), n-Tucker (Liu et al., 2020), m-TransH (Wen et al., 2016) and RAE (Zhang et al., 2018) are Knowledge Hypergraphs Embeddings that considered n-ary facts in the form of r(e1, e2 . . .). These methods aims to perform link prediction by encoding entities and relations into embeddings continuous spaces. These models are equivalently performing projection on atomic hyperedge. Although they can perform well in atomic query projection task, they lack the ability to perform multi-hop query, where performance degrades as projection continues on sub-queries. To enable logical operations on these embeddings, fuzzy logic (Zadeh, 1988; Chen et al., 2022) or message passing methods (Wang et al., 2023) can be applied.

2.2 Complex Query Answering

Complex Query Answering (CQA) leverages various approaches, including transformers, query embeddings, neural-symbolic methods, and message passing techniques. The introduction of Logical Message Passing Neural Networks (LMPNN) (Wang et al., 2023) has facilitated the use of different pretrained Knowledge Graph (KG) embeddings, which serve as initial entity and relation embeddings. Final answer embeddings are obtained through message passing on query graphs, utilizing embeddings such as RESCAL (Nickel et al., 2011), TransE (Bordes et al., 2013), DistMult, ComplEx (Trouillon et al., 2016), ConvE (Dettmers et al., 2018), and RotatE(Sun et al., 2019).

For query embeddings, GQE (Hamilton et al., 2019) addresses queries with existential quantifiers and conjunctions, while FuzzQE (Chen et al., 2022) employs fuzzy logic to define logical operators. In the neural-symbolic domain, methods like BetaE (Ren and Leskovec, 2020), ConE (Zhang et al., 2021), QUERY2BOX (Ren et al., 2020) and ENeSy (Xu et al., 2022), project symbols into continuous spaces. MLPMix (Amayuelas et al., 2022) and NewLook (Liu et al., 2021) utilize MLPs and attention mechanisms for CQA. Additionally, QTO (Bai et al., 2023b), CQD (Arakelyan et al., 2021) and FIT (Yin et al., 2024) apply combinatorial optimization to query computation trees.

Since the introduction of Transformers (Vaswani et al., 2023), several methods have applied this architecture to CQA. SQE (Bai et al., 2023a) lin-

earizes EFO-1 queries for sequence encoding, PathFormer (Zhang et al., 2024) recursively process EFO-1 queries tree like NQE (Luo et al., 2023) and Tree-LSTM did (Tai et al., 2015). While QTP (Xu et al., 2023) separates simple and complex queries, employing distinct neural link predictors and query encoders.

Additionally, HAN (Wang et al., 2021b) implements transformers for link prediction in knowledge graphs, and KnowFormer (Liu et al., 2024a) tailors the transformer for simple query answering. TEGA (Zheng et al., 2025) applies inductive biases based on token type interactions for EFO-syntax queries.

Previous studies on CQA have primarily focused on knowledge graphs, with extensive research on ordinary knowledge graphs addressing ordinary query answering (OWA) by predicting unseen triples. EFO-1 queries can be effectively answered when the arity equals two. However, few attempts have been made to extend these approaches to general n-ary facts in hyperedge or hyper-relational formats.

2.3 N-ary Graph Reasoning

First N-ary reasoning problem introduced in the field of CQA is performed on hyper-relational knowledge graph. StarQE (Alivanistos et al., 2022) utilize StarE (Galkin et al., 2020) as graph encoder for StarQE, equip it with message passing to have the ability to deal with multi-hop queries. TransEQ (Liu et al., 2024b) is an query embedding models that generalize star expansion (Agarwal et al., 2006) for hyper-edges to hyper-relational graphs, then use encoder-decoder to capture structural information and semantic information for hyper-relational knowledge graph completion task. NeuInfer (Guan et al., 2020) chose to represent n-ary fact as a primary triple coupled with a set of its auxiliary descriptive attribute-value pair(s) and use neural network to perform knowledge inference. NQE (Luo et al., 2023) use dual-heterogeneous Transformer encoder and fuzzy logic (Zadeh, 1988) to recursively process hyper-relational query tree. Hyper-relational edges encode entities with possibly different relation in a single triples, which does not exhibit same characteristics as hyperedges. The encoder input format for NQE generalize n-ary inputs, thus it can be naturally extend to be used in hyper-edges query answering. SessionCQA (Bai et al., 2024), The first CQA model incorporates the concept of hyper-edges, encoding user sessions as hyper-edges for item recommendations. However, its query type does not follow a fully hyper-edge setting; only the initial first hop is represented as a hyper-edge, while subsequent hops use binary relation edges. The query input format for SessionCQA can also naturally extend for N-ary facts.

3 PRELIMINARIES

3.1 Knowledge Hypergraphs

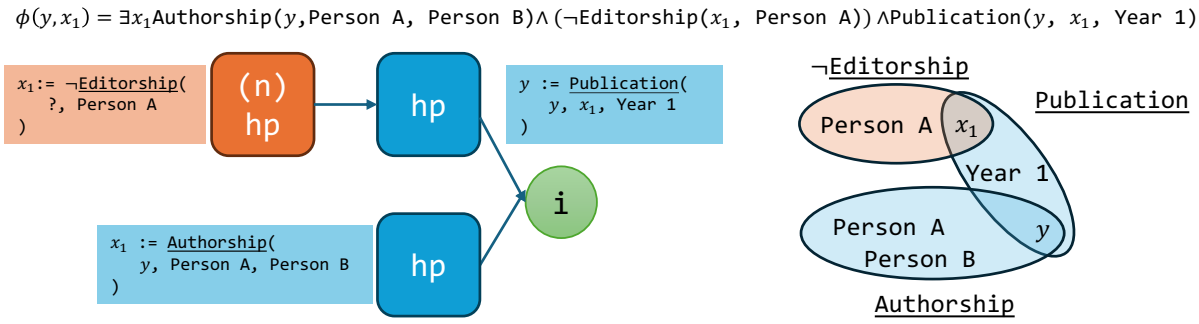
A knowledge hypergraph $\mathcal{G} = (\mathcal{E}, \mathcal{R}, \mathcal{H})$, where $\mathcal{E}$ is the set of entities e in the knowledge hypergraph, $\mathcal{R}$ is the set of relations r, and $\mathcal{H}$ is the set of ordered hyperedges $h = r(e_1, \ldots, e_k) \in \mathcal{H}$, where $e_1, \ldots, e_k \in \mathcal{E}$ and $r \in \mathcal{R}$. The arity of a hyperedge h is defined as $k = ar(r)$, where each relation type has a fixed arity size. Each position in a relation type has a specific semantic meaning that constructs the ordered hyperedge. Although through star expansion (Agarwal et al., 2006) of hyperedge, an hyperedge can be converted to homogeneous graph, however structural information will be loss in process.

3.2 EFO-1 Query in Hypergraphs

In this paper, we will focus on Existential First Order queries with a single free variable with logical formulas (EFO-1) under the disjunctive normal form in hypergraph settings. In the following discussions, we will refer to hyper-projection simply as projection without explicitly specifying it. We aim to emphasize the unique properties of projections within hypergraph structures.

An atomic formula a is composed of term, relation and variable. A simplest atomic formula can be intuitively represented by an ordered hyperedge $h = r_1(e_1, ..., v_k)$, where e_1 is a term, v_k is an variable at position k and r_1 is the relation. Let $ar(r)$ is the arity of the given relation r. A knowledge graph is a relational knowledge hypergraph where for all $r \in \mathcal{R}, ar(r) = 2$. An atomic formula can be negated by adding $\neg$ to form $\neg r(e_1, ..., v_k)$. A first order formula can be iteratively constructed by atomic formula a using connectives conjunction $\wedge$ and disjunction $\vee$. Quantifiers can be added to variables in a using quantifiers like $\exists$ and $\forall$. Variables without quantifiers is considered as free.

Given a knowledge hypergraph $\mathcal{G}$, an EFO-1 query q is defined as a first-order formula in the below Disjunctive Normal Form (DNF),



(a) Query Operator Tree with (Hyper) Projection

(b) Query Hypergraph

Figure 1: Example of LKHGT processing query tree of ip type

$$q(y, x_1, ..., x_n) = \exists x_1, \ldots, \exists x_m[(a_{11} \wedge a_{12} \wedge \cdots \wedge a_{1n_1}) \vee \cdots \vee (a_{p1} \wedge a_{p2} \wedge \cdots \wedge a_{pn_p})] \quad (1)$$

where y is the only free variable, and x_i for $1 \leq i \leq m$ are existential variables. Each brackets represent an complex query where each a_{ij} is an atomic formula with constants y, x_i and variables that can be either negated or not. To address the EFO-1 queries, one must determine the answer set A$[Q, KG]$ such that for each a $\in$ A$[Q, KG]$, $q(y = a, x_1, ..., x_n)$ returns true.

Considering a query in the form of

$$r_1(y_1, e_1, e_2) \wedge \neg r_2(x_1, e_3) \wedge r_3(y_1, x_1, e_4)$$

there are 2 ways to represent EFO-1 queries in hypergraphs, which are query graphs and operator tree.

Operator tree. In operator tree, draws from existential quantifiers, allowing us to transform first-order logic into corresponding set logic operations. Each node itself is an operator node that corresponds to set logical operations like, projection, conjunction, disjunction, negation. In knowledge hypergraph settings, each projection node consists of set of entities for itself and a variable nodes in random arity position. As Figure 1 shown, each atomic query is represented by the projection nodes. Then, two projection nodes pointed to the intersection node, and the intersection node represent the conjunction of answer sets from its pointers. Intersection node use the intersected variable to obtain the final answer.

Query Hypergraph. In Figure1, it shows an alternative way to represent EFO-1 query following the style of query graph in LMPNN (Wang et al., 2023). Each hyper edge representing an atomic formula containing edge information like relation types and negation information. The nodes involved are either constant symbol, free variable or existential variable.

In this paper, each Knowledge Hypergraph EFO-1 query is presented in the form operator tree for LKHGT. As we can see that in order to allow transformer to perform message passing alike operation through the self-attention mechanism, we have to encode whole query in a single pass. When transformer based model encode whole query graphs as a sequence of tokens, it becomes more challenging for transformers to focus on solving each atomic query in the multihop-query. Thus, operator tree is more suitable for our choice of modeling.

4 Logical Knowledge Hyper Graph Transformer

In this section, we will describe the how operator tree is used to represent complex query, and explain how the 2 stage encoder, Logical Knowledge Hyper Graph Transformer (LKHGT), works given a query tree input. In order to allow precise estimation of answer set, we propose to use 2 stage encoder to process the query tree iteratively. For example, refer to Figure 2, we can see that each projection node at bottom level first receive necessary information like, negation, relation, entities and position of variable node inside the hyperedge. Then these information is fed into projection encoder for each projection. For second hop of the atomic query, previous projected variable embedding is fed as one of the input. Projection is performed using the answer set embeddings output from projection encoder. Finally, intersection is then performed with logical encoder, which is done by receive all variable embeddings output from each projection encoder. Although using iterative method to pro-

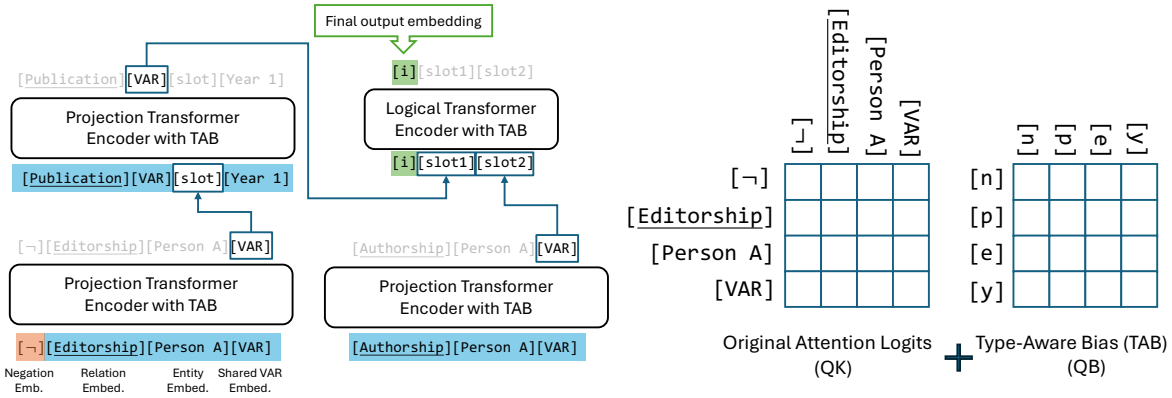


Figure 2: Example of LKHGT processing query tree

cess query tree is slower comparing to encoding whole query graph at once, we maximize the correctness of answer set embeddings output for each atomic formula.

4.1 Tokens Input and Output

Operators type included in operator tree are projection (p), negation (¬), intersection (∧), union (∨). As defined, each complex query is composed of atomic formula which is made of relaiton (r), subject (s), variable (e). Overally speaking, there are 8 types of token. Relation, existential variable, free variable, entity and negation type of tokens will be fed into projection encoder. Projection, intersection, union types of token will be fed into logical encoder.

Projection Encoder Given the fact that negation can only applied to atomic formula which is represented as projection operation, we chose to process negation operation in projection encoder. Projection Encoder projects tokens into continuous space according to their token type. If negation is involved, the input tokens will be,

$$\text{tokens} = T_p = [n, r_1, e_1, x_2, ...] \tag{2}$$

The corresponding projected sequence will be,

$$X = W_p T_p = [W_n n, W_r r_1, W_e e_1, ...] \in R^{n \times d} \tag{3}$$

where $W_p \in R^{d \times d}$is the linear projection weight, where $p \in \{n, r, x, e, y\}$. Since projection encoder is performing projection over ordered hyperedge, we will also add absolute positional encoding to entities node and variable node. Let the set of nodes be N for an hyperedge, then the input embeddings will be.

$$X_{v \in N} = X_{v \in N} + PE_{pos} \tag{4}$$

Logical Encoder Logical encoder will focus in dealing with logical task that involves more than 1 projection node, which is intersection (conjunction) and union (disjunction). These nodes will receive pointers from more than one operators node, we use transformer logical encoder to encode all projected variable, the input tokens for logical encoder will be,

$$\text{tokens} = T_l = [i/u, p_1, p_2, p_3, ...] \tag{5}$$

The corresponding projected sequence will be,

$$X = W_l T_l = [W_{i/u}\{i/u\}, W_p p_1, ...] \in R^{n \times d} \tag{6}$$

where $W_l \in R^{d \times d}$is the linear projection weight, where $l \in \{i, u, p\}$. Both projected sequence is then further process with modified version of (Vaswani et al., 2023). Finally, Projection Encoder will output embeddings of variable node in hyperedge and Logical Encoder will output the embeddings of the logical operator tokens.

4.2 Type Aware Bias (TAB) for Projection Encoder and Logical Encoder

Inspired by the idea of introducing inductive bias for meta paths in (Wang et al., 2021b), LKHGT also introduced modified inductive bias tailored for CQA task. With the given 8 token types, we can observe there are types of token interaction bias to be added into self attention matrix. For Projection Encoder there are P_2^5 edge types for token types $[n, r, x, e, y]$. For Logical Encoder, there are P_2^3 edge types for token types. In order to cater for these edge types differences, we construct the attention matrix with Type Aware Bias (TAB)

as below:

$$e_{ij} = \frac{x_i W_o^Q (x_j W_o^k + B_{ij})^T}{\sqrt{d}}, a_{ij} = \text{softmax}(e_{ij}) \tag{7}$$

$$z_i = \sum_{j=1}^{n} a_{ij}(x_j W_o^V) \tag{8}$$

Where (W^Q, W^K, W^V) is the linear projection matrix $\in R^{d\times d}$ and $\{o, i, j\}$ can be the eight token types $t \in \{n, r, x, e, y, i, u, p\}$, B_{ij} is the Type Aware Bias (TAB) vector in token interaction bias matrix $B \in R^{|t|\times|t|\times d}$. These bias able to differentiate the token types interaction and help facilitates the capture of nuanced interactions among token types, allowing the transformer to adaptively weigh the importance of each token based on its role in the context.

In our case, we can treat Transformer as a special case of GNN. For Projection encoder, each hyperedge is an fully connected graph input in the perspective of GNN. We are doing aggregation with attention on this fully connected graph and predict the embeddings of the projection node using transformer self-attention mechanism. This aggregation process not only enriches the embeddings of the projection node but also ensures that the model can dynamically adjust to the significance of different tokens based on the context. The result is a robust embedding that encapsulates the rich relational structure inherent in the input data.

4.3 Transformer as the replacement for Fuzzy Logic

Transformers serve as a compelling alternative to fuzzy logic in our CQA models due to their superior representation capabilities and scalability. Unlike fuzzy logic-based methods (Bai et al., 2023b; Yin et al., 2024) that requires pre-computing adjacency matrices, which is impractical due to computational costs of $O(N^2)$ for binary relations and $O(N^k)$ for k-ary relations in our case. Transformers efficiently use attention mechanisms to generate detailed low-dimensional embeddings. This aligns with approaches that use low-dimensional embeddings with fuzzy operators, like NQE, but provides a more adaptable framework, offering a better contextual understanding of projected nodes' embeddings through TAB. Thus, within the constraints of low-dimensional vector representations, transformers provide a more powerful and adaptable framework for processing and representing information in CQA models.

4.4 Training LKHGT

After obtaining the output embeddings $f \in R^d$, we can compute the output label of the entity by a simple MLP decoder and have output $\hat{y} \in R^{|\mathcal{E}|}$. Given a the query embedding $\hat{y}$, we can first use a softmax function to obtain the probability scores of query embedding between all entity,

$$\sigma(\hat{y}) = \frac{e^{\hat{y}_i}}{\sum_j e^{\hat{y}_j}} \tag{9}$$

Then we can construct the cross-entropy loss to maximize the log probability of $\hat{y}$ matching the answer a,

$$L = -\frac{1}{N}\sum_i \log(\sigma(\hat{y})) \tag{10}$$

where N is the batch size.

5 Experiments

In this section, we will describe the experiment set up including dataset, baselines and evaluation methods.

5.1 Dataset

Since Knowledge Hypergraph CQA is a subfield of CQA problem that has not been investigated before, thus we created custom Dataset JF17k-HCQA and M-FB15k-HCQA. Following the rationale from binary CQA sampling methods (Wang et al., 2021c), we modified the sampling methods into hypergraph version (Appendix B), created datasets consists of query types (1P 2P 3P 2I 3I PI IP 2U UP 2IN 3IN INP PIN PNI), total 14 types, including all logical operation. Statistics for the number of examples sampled for each query type are listed in Table 4.

5.2 Baselines

For baseline models, we chose NQE (Luo et al., 2023) and LSGT (Bai et al., 2024) as our baseline models. NQE is a method that encode hyper-relational facts from HKG into embeddings. LSGT encode query graph information like, nodes ids, graph structure and logical structure using transformer.As NQE (Luo et al., 2023) uses hyper-relational edges as input for its encoder. So, we followed the implementation of NQE in (Bai et al., 2024) and modified our data format to feed it into NQE. For LSGT, we transformed our data into unified id format for input. For fair comparison, we replace all models encoder with simple basic

Table 1: MRR results of different CQA models on two KHGs. AP represents the average score of EPFO queries and AN represents the average score of queries with negation. The boldface indicates the best results for each KG.

KHG	Model	1P	2P	3P	2I	3I	PI	IP	2U	UP	2IN	3IN	INP	PIN	PNI	AP	AN
JF17k	HLMPNN	45.66	6.58	12.24	46.03	47.27	15.52	7.91	25.45	12.75	10.43	14.49	10.83	7.26	8.29	24.38	10.26
	LSGT	49.51	21.73	13.10	50.46	34.85	32.07	23.00	22.27	20.33	16.05	13.45	8.44	10.10	12.39	29.70	12.09
	NQE	56.19	38.29	34.72	64.88	**70.40**	53.95	38.24	29.11	38.39	22.51	**27.80**	15.04	17.84	21.83	47.13	21.00
	LKHGT	**58.19**	**41.87**	**38.76**	**68.47**	42.71	**57.48**	**42.01**	**33.38**	**41.70**	**26.28**	18.26	**20.66**	**18.32**	**24.87**	**47.17**	**21.68**
M-FB15k	HLMPNN	**47.86**	28.04	23.02	**45.78**	45.65	32.89	26.12	25.35	26.27	17.32	22.58	21.98	9.47	15.85	33.44	17.44
	LSGT	44.15	30.35	9.87	39.47	31.19	30.01	28.57	16.46	27.28	16.55	10.10	10.59	11.37	19.29	28.59	13.58
	NQE	46.43	33.43	29.80	43.75	**48.84**	34.94	32.09	16.83	30.20	22.26	**26.28**	20.19	**17.51**	24.95	35.15	**22.24**
	LKHGT	46.81	**35.04**	**30.66**	45.38	40.31	**38.65**	**32.63**	**27.14**	**31.37**	**22.85**	19.65	**26.50**	16.21	**24.96**	**36.44**	22.03

Table 2: MRR results for LKHGT in fuzzy settings, excluding absolute positional encoding and under a variable cardinality configuration.

KHG	Model	1P	2P	3P	2I	3I	PI	IP	2U	UP	2IN	3IN	INP	PIN	PNI	AP	AN
JF17k	LKHGT	**58.19**	**41.87**	**38.76**	**68.47**	42.71	**57.48**	**42.01**	**33.38**	**41.70**	**26.28**	18.26	20.66	18.32	**24.87**	47.17	21.68
	LKHGT w/ fuzzy.	57.26	40.62	37.95	65.48	**71.04**	54.46	40.25	29.87	40.51	23.00	**27.58**	18.56	**19.65**	22.80	48.60	22.32
	LKHGT w/o abs.	57.40	41.01	37.51	67.87	41.94	57.02	41.63	31.56	40.72	26.19	17.75	**20.92**	18.53	25.00	46.30	21.68
	LKHGT w/ var cardinality	57.55	41.31	38.35	67.13	41.61	56.68	41.23	32.02	40.79	25.79	17.69	21.04	18.58	24.81	46.30	21.59
	LKHGT w/ full training set	58.43	43.89	42.75	69.11	75.19	58.19	43.16	32.05	42.51	26.26	31.62	23.18	18.74	25.35	51.70	25.03
	LKHGT w/ full train & var card.	58.54	44.40	43.58	69.20	74.46	58.56	43.56	32.91	42.90	26.48	31.51	24.06	19.30	25.59	52.01	25.39

transformer layer without modifying the other implementation details. Transformer embedding size is set to be 400 for iterative model like NQE and LKHGT. For LSGT, its embedding size is set to be 1024 in order to have it converge. Aside from transformer based model, we also implemented HLMPNN (Appendix A), a hypergraph version of LMPNN (Wang et al., 2023), which is set to have embedding size of 200. All models are trained on single 3090 GPU with 400 epochs.

5.3 Evaluation

In all experiments, we follow the common practice, sampled query from Knowledge Hypergrpahs $\mathcal{G}_{train}, \mathcal{G}_{val}, \mathcal{G}_{test}$, where $\mathcal{E}_{train} \subset \mathcal{E}_{val}$ and $\mathcal{E}_{val} \subset \mathcal{E}_{test}$. For all models, we will train on all query types except [3p, 3in, 3i, inp], in order to test the generalization of logical operations. We select the mean reciprocal rank (MRR),

$$MRR(q) = \frac{1}{||v||} \sum_{v_i \in v} \frac{1}{rank(v_i)} \qquad (11)$$

as the evaluation metric to evaluate the ranking. For each query instance, we initially rank all entities, excluding those identified as easy answers, by their cosine similarity to the query variable embedding. The rankings of the hard answers are then used to calculate the Mean Reciprocal Rank (MRR) for that particular query instance. Finally, we compute the average of the metrics across all query instances. In this paper, we report and compare the MRR results.

5.4 Results

Table 1 presents the MRR results of LKHGT and other baseline models for answering EFO-1 queries across the two CQA datasets. It demonstrates that LKHGT achieves state-of-the-art performance on average for both EPFO and negation queries for n-ary hypergraph queries. When comparing LKHGT with transformer-based and message-passing methods, we observe that the message-passing methods underperform in complex query types. Additionally, when comparing the Iterative Model (LKHGT & NQE) to the Sequential Model (LSGT), the Iterative Model shows superior performance. This disadvantage may be attributed to their approach of encoding the entire query at once, which can hinder performance on complex queries. In contrast, the LKHGT model shows notable improvements with a logical encoder instead of fuzzy set operation, enhancing its ability to handle conjunctions and disjunctions effectively.LKHGT surpasses NQE on negation queries by integrating negation directly into the Projection Encoder, rather than using fuzzy logic. This suggests that fuzzy logic may not be sufficient for addressing hypergraph complex query answering tasks. The 1p, 2p, and 3p performance of LKHGT outperforming NQE is evidence that Type Aware Bias is contributing, as these query types do not involve the use of logical encoder. This distinction from the NQE model suggests that this bias is better at capturing the different interactions of token inputs. Overall, these findings confirm that the two-stage architecture of LKHGT successfully enhances performance in complex query scenarios, demonstrating its effectiveness and robustness in the context of query answering.

However, based on the performance of the LKHGT model in 3i and 3in, we observe that it is signif-

Table 3: MRR results for CQA models under full training set.

KHG	Model	1P	2P	3P	2I	3I	PI	IP	2U	UP	2IN	3IN	INP	PIN	PNI	AP	AN
JF17k	LKHGT	58.43	43.89	42.75	69.11	75.19	58.19	43.16	32.05	42.51	26.26	31.62	23.18	18.74	25.35	**51.70**	**25.03**
	LSGT	50.56	21.85	17.75	53.34	53.71	32.06	20.78	22.49	18.33	14.00	16.40	6.05	7.44	8.59	32.32	10.50
	HLMPNN	45.26	7.26	13.38	46.08	48.36	16.01	8.54	24.99	13.28	10.35	15.49	11.48	7.32	8.24	24.80	10.58
	NQE	56.37	39.18	39.21	65.41	72.37	55.36	39.22	29.05	39.36	21.68	28.83	20.25	16.21	22.13	48.39	21.82

Table 4: Statistics for each query types

Dataset	1P	Others
Train	60,000	20,000
Valid	10,000	10,000
Test	10,000	10,000

icantly lower than NQE. This discrepancy arises because the logical encoder did not encounter the input combinations generated by the output embedding of the projection encoder and the logical encoder itself, preventing it from effectively process the input. To provide a fair comparison of the effectiveness of LKHGT, we trained another instance using the full training set. The results are presented in Tables 2 and 3, where its performance in 3i and 3in shows significant improvement.

5.5 Ablation Study

In the ablation study, we conduct further experiments to justify the effects of the logical encoder and the positional encoding of LKHGT. We also examine how input cardinality affects the logical encoder, comparing processing embeddings in pairs versus handling multiple embeddings simultaneously for operations like intersection and union. All experiments are performed on queries over JF17k with same settings as before. Table 2 presents the results of the ablation study. From the results, we observe that without the positional encoding, the model's performance degrades, indicating the importance of this bias for identifying the position of each nodes to enhance performance. This may be due to the fact that each position contains a different semantic meaning within an hyperedge of ordered knowledge hypergraph. Such position information helps identify the entity properties in hyperedges. When using LKHGT with fuzzy logic, the results are still slightly better than NQE except for 3i and 3in due to the reason stated before, suggesting that the transformer-based projection encoder improve performance in negation and projection, and for logical encoder, it able to replace fuzzy logic operation, given that all types of input combination has been trained beforehand. For the Operator Cardinality, interestingly, for the logical encoder, processing all inputs simultaneously outperforms handling them in pairs.

6 Conclusion

In this paper, we present the Knowledge Hypergraph CQA dataset to bridge the current gap in complex question answering (CQA) within hypergraph settings. We propose LKHGT to answer complex queries over Knowledge Hypergraphs, particularly EFO-1 queries. LKHGT achieves strong performance through its two-stage architecture with Type Aware Bias (TAB). In the ablation study, we demonstrate that the use of absolute positional encoding further enhances performance. Our research effectively addresses the gap in EFO-1 query answering tasks in hypergraph settings while emphasizing the advantages of transformer-based approaches for logical operations. The experiments show that, given sufficient training on various query types, LKHGT is the state-of-the-art model for current CQA tasks involving Knowledge Hypergraphs.

7 Limitation

A key limitation of our model is the time complexity linked to the projection and logical encoder components. The shift from fuzzy logic to a transformer-based logical encoder has increased complexity due to a larger parameter count and attention mechanisms. Additionally, the inductive bias requires training all combinations of token interactions, which can introduce noise if not properly managed, resulting in suboptimal performance and longer training times.

Potential Risks The increased time complexity may lead to longer training and inference times, limiting practical applications. If the inductive bias is not addressed effectively, it might mislead the model toward noise rather than meaningful patterns. Furthermore, reliance on transformer architectures may create scalability challenges when dealing with larger datasets or complex tasks. Addressing these limitations and risks is crucial for the model's effectiveness.

References

Ralph Abboud, İsmail İlkan Ceylan, Thomas Lukasiewicz, and Tommaso Salvatori. 2020. BoxE: A Box Embedding Model for Knowledge Base Completion. *arXiv preprint*. ArXiv:2007.06267 [cs].

Sameer Agarwal, Kristin Branson, and Serge Belongie. 2006. Higher order learning with graphs. In *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pages 17–24, Pittsburgh, Pennsylvania. ACM Press.

Dimitrios Alivanistos, Max Berrendorf, Michael Cochez, and Mikhail Galkin. 2022. Query Embedding on Hyper-relational Knowledge Graphs. *arXiv preprint*. ArXiv:2106.08166 [cs].

Alfonso Amayuelas, Shuai Zhang, Susie Xi Rao, and Ce Zhang. 2022. Neural Methods for Logical Reasoning Over Knowledge Graphs. *arXiv preprint*. ArXiv:2209.14464 [cs].

Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. 2021. Complex Query Answering with Neural Link Predictors. *arXiv preprint*. ArXiv:2011.03459 [cs].

Jiaxin Bai, Chen Luo, Zheng Li, Qingyu Yin, and Yangqiu Song. 2024. Understanding Inter-Session Intentions via Complex Logical Reasoning. *arXiv preprint*. ArXiv:2312.13866 [cs].

Jiaxin Bai, Tianshi Zheng, and Yangqiu Song. 2023a. Sequential Query Encoding For Complex Query Answering on Knowledge Graphs. *arXiv preprint*. ArXiv:2302.13114 [cs].

Yushi Bai, Xin Lv, Juanzi Li, and Lei Hou. 2023b. Answering Complex Logical Queries on Knowledge Graphs via Query Computation Tree Optimization. *arXiv preprint*. ArXiv:2212.09567 [cs].

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

Xuelu Chen, Ziniu Hu, and Yizhou Sun. 2022. Fuzzy Logic Based Logical Query Answering on Knowledge Graphs. *arXiv preprint*. ArXiv:2108.02390 [cs].

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2D Knowledge Graph Embeddings. *arXiv preprint*. ArXiv:1707.01476 [cs].

Bahare Fatemi, Perouz Taslakian, David Vazquez, and David Poole. 2020. Knowledge Hypergraphs: Prediction Beyond Binary Relations. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 2191–2197, Yokohama, Japan. International Joint Conferences on Artificial Intelligence Organization.

Bahare Fatemi, Perouz Taslakian, David Vazquez, and David Poole. 2021. Knowledge Hypergraph Embedding Meets Relational Algebra. *arXiv preprint*. ArXiv:2102.09557 [cs].

Mikhail Galkin, Priyansh Trivedi, Gaurav Maheshwari, Ricardo Usbeck, and Jens Lehmann. 2020. Message Passing for Hyper-Relational Knowledge Graphs. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7346–7359, Online. Association for Computational Linguistics.

Saiping Guan, Xiaolong Jin, Jiafeng Guo, Yuanzhuo Wang, and Xueqi Cheng. 2020. NeuInfer: Knowledge Inference on N-ary Facts. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6141–6151, Online. Association for Computational Linguistics.

William L. Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. 2019. Embedding Logical Queries on Knowledge Graphs. *arXiv preprint*. ArXiv:1806.01445 [cs].

Xingyue Huang, Miguel Romero Orth, Pablo Barceló, Michael M. Bronstein, and İsmail İlkan Ceylan. 2024. Link Prediction with Relational Hypergraphs. *arXiv preprint*. ArXiv:2402.04062 [cs].

Shaoxiong Ji, Shirui Pan, Erik Cambria, Pekka Marttinen, and Philip S. Yu. 2022. A Survey on Knowledge Graphs: Representation, Acquisition, and Applications. *IEEE Transactions on Neural Networks and Learning Systems*, 33(2):494–514. Conference Name: IEEE Transactions on Neural Networks and Learning Systems.

Junnan Liu, Qianren Mao, Weifeng Jiang, and Jianxin Li. 2024a. KnowFormer: Revisiting Transformers for Knowledge Graph Reasoning. *arXiv preprint*. ArXiv:2409.12865 [cs].

Lihui Liu, Boxin Du, Heng Ji, ChengXiang Zhai, and Hanghang Tong. 2021. Neural-Answering Logical Queries on Knowledge Graphs. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1087–1097, Virtual Event Singapore. ACM.

Yu Liu, Shu Yang, Jingtao Ding, Quanming Yao, and Yong Li. 2024b. Generalizing Hyperedge Expansion for Hyper-relational Knowledge Graph Modeling. *arXiv preprint*. ArXiv:2411.06191 [cs].

Yu Liu, Quanming Yao, and Yong Li. 2020. Generalizing Tensor Decomposition for N-ary Relational Knowledge Bases. *arXiv preprint*. ArXiv:2007.03988 [cs].

Haoran Luo, Haihong E, Yuhao Yang, Gengxian Zhou, Yikai Guo, Tianyu Yao, Zichen Tang, Xueyuan Lin, and Kaiyang Wan. 2023. NQE: N-ary Query Embedding for Complex Query Answering over Hyper-Relational Knowledge Graphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(4):4543–4551. ArXiv:2211.13469 [cs].

Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. 2011. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 809–816, Madison, WI, USA. Omnipress.

Hongyu Ren, Mikhail Galkin, Michael Cochez, Zhaocheng Zhu, and Jure Leskovec. 2023. Neural Graph Reasoning: Complex Logical Query Answering Meets Graph Databases. *arXiv preprint*. ArXiv:2303.14617 [cs].

Hongyu Ren, Weihua Hu, and Jure Leskovec. 2020. Query2box: Reasoning over Knowledge Graphs in Vector Space using Box Embeddings. *arXiv preprint*. ArXiv:2002.05969 [cs].

Hongyu Ren and Jure Leskovec. 2020. Beta Embeddings for Multi-Hop Logical Reasoning in Knowledge Graphs. In *Advances in Neural Information Processing Systems*, volume 33, pages 19716–19726. Curran Associates, Inc.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. RotatE: Knowledge Graph Embedding by Relational Rotation in Complex Space. *arXiv preprint*. ArXiv:1902.10197 [cs].

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. *arXiv preprint*. ArXiv:1503.00075 [cs].

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex Embeddings for Simple Link Prediction. *arXiv preprint*. ArXiv:1606.06357 [cs].

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. *arXiv preprint*. ArXiv:1706.03762 [cs].

Quan Wang, Haifeng Wang, Yajuan Lyu, and Yong Zhu. 2021a. Link Prediction on N-ary Relational Facts: A Graph-based Approach. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 396–407, Online. Association for Computational Linguistics.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Peng Cui, P. Yu, and Yanfang Ye. 2021b. Heterogeneous Graph Attention Network. *arXiv preprint*. ArXiv:1903.07293 [cs].

Zihao Wang, Yangqiu Song, Ginny Y. Wong, and Simon See. 2023. Logical Message Passing Networks with One-hop Inference on Atomic Formulas. *arXiv preprint*. ArXiv:2301.08859 [cs].

Zihao Wang, Hang Yin, and Yangqiu Song. 2021c. Benchmarking the Combinatorial Generalizability of Complex Query Answering on Knowledge Graphs. *arXiv preprint*. ArXiv:2109.08925 [cs].

Jianfeng Wen, Jianxin Li, Yongyi Mao, Shini Chen, and Richong Zhang. 2016. On the representation and embedding of knowledge bases beyond binary relations. *arXiv preprint*. ArXiv:1604.08642 [cs].

Yao Xu, Shizhu He, Cunguang Wang, Li Cai, Kang Liu, and Jun Zhao. 2023. Query2Triple: Unified Query Encoding for Answering Diverse Complex Queries over Knowledge Graphs. *arXiv preprint*. ArXiv:2310.11246 [cs].

Zezhong Xu, Wen Zhang, Peng Ye, Hui Chen, and Huajun Chen. 2022. Neural-Symbolic Entangled Framework for Complex Query Answering. *arXiv preprint*. ArXiv:2209.08779 [cs].

Naganand Yadati. 2020. Neural Message Passing for Multi-Relational Ordered and Recursive Hypergraphs. In *Advances in Neural Information Processing Systems*, volume 33, pages 3275–3289. Curran Associates, Inc.

Naganand Yadati, Madhav Nimishakavi, Prateek Yadav, Vikram Nitin, Anand Louis, and Partha Talukdar. 2019. HyperGCN: A New Method of Training Graph Convolutional Networks on Hypergraphs. *arXiv preprint*. ArXiv:1809.02589 [cs].

Hang Yin, Zihao Wang, and Yangqiu Song. 2024. Rethinking Complex Queries on Knowledge Graphs with Neural Link Predictors. *arXiv preprint*. ArXiv:2304.07063 [cs].

L.A. Zadeh. 1988. Fuzzy logic. *Computer*, 21(4):83–93. Conference Name: Computer.

Chongzhi Zhang, Zhiping Peng, Junhao Zheng, Linghao Wang, Ruifeng Shi, and Qianli Ma. 2024. Pathformer: Recursive Path Query Encoding for Complex Logical Query Answering. *arXiv preprint*. ArXiv:2406.14880 [cs].

Richong Zhang, Junpeng Li, Jiajie Mei, and Yongyi Mao. 2018. Scalable Instance Reconstruction in Knowledge Bases via Relatedness Affiliated Embedding. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, pages 1185–1194, Lyon, France. ACM Press.

Zhanqiu Zhang, Jie Wang, Jiajun Chen, Shuiwang Ji, and Feng Wu. 2021. ConE: Cone Embeddings for Multi-Hop Reasoning over Knowledge Graphs. *arXiv preprint*. ArXiv:2110.13715 [cs].

Tianshi Zheng, Jiazheng Wang, Zihao Wang, Jiaxin Bai, Hang Yin, Zheye Deng, Yangqiu Song, and Jianxin Li. 2025. Enhancing Transformers for Generalizable First-Order Logical Entailment. *arXiv preprint*. ArXiv:2501.00759 [cs].

Supplimentary Materials

A Appendix A

A.1 Inspiration from LMPNN

Many previous works emphasize hypergraph message passing (Agarwal et al., 2006; Yadati et al., 2019; Yadati, 2020), with some tailored for hyper-relational graphs. Notable attempts include StarE (Galkin et al., 2020) and GRAN (Wang et al., 2021a), which address one-hop queries on hyper-relational KGs. HR-MPNN (Huang et al., 2024) established the General Relational Ordered Hypergraphs Message Passing Framework. Our baseline combines HR-MPNN (Huang et al., 2024) with pretrained embeddings, drawing inspiration from LMPNN (Wang et al., 2023).

A.2 A Natrual Extension of Logical Message Passing Nueral Network to Answer Hypergraph Queries

- **Statement 1**: Prove $\rho(\{e_i|(e,i) \in N_j(h)\}, r, j, 0)$ is a generalization of $\rho(h, r, h2t, 0)$.
  Where $N_i(h)$ as the positional neighborhood of a hyperedge h.

  Let ρ be the hypergraph logical mesage encoding function of four input parameters, including neighboring entity embedding, relation embedding, query node position, and logical negation information.

  Suppose that each edge with head at position 0 and tail at position 1, in the format of $(h, 0) \rightarrow (t, 1)$ for edge with arity = 2. Then with the definition of the ordered hyperedge, there are 2 cases representing normal binary edge in hyperedge message encoding function. Their equivalent representation according to our definition:

  1. $\rho(h, r, h2t, 0) = \rho(h, r, 1, 0)$
  2. $\rho(t, r, t2h, 0) = \rho(t, r, 0, 0)$

  We simply replaced h2t and t2h flag with positional information.
  For example, suppose a particular edge with arity = 6, in the form of $r(e_1, e_2, e_3, e_4, e_5, x)$, we can use the message encoding function express in the form of $\rho(\{e_1, e_2, e_3, e_4, e_5\}, r, 6, 0)$.

- **Statement 2**: Suppose $\rho(e_0, r, 1, 0) = \rho(h, r, h2t, 0)$ and $\rho(h, r, h2t, 0) = f(h, r)$ are true, where f is a binary function (e.g. elementwise multiplication).
  Prove $\rho(\{e_i|(e,i) \in N_j(e)\}, r, j, 0) = f(g(\{N_j(e)\}), r)$ and is a generalization of KG message encoding function, where $g(\{N_j(e)\}) = f(f(e_1, ...(f(e_{n-1}, e_n))))$ is a function that recursively apply f.

  As in LMPNN (Wang et al., 2023), there are 2 types of KG embeddings, characterized by their scoring functions, which are the inner-product-based scoring function and distance-based scoring function. The proof for closed-form logical messages for KHG representation is the same. Due to the fact that the properties of recursive function g does not affect the proving in LMPNN(Wang et al., 2023), so as long as we can we prove our recursive function g can obtain the same equation for normal KG closed-form logical messages with arity = 2, we can prove that:

  $$\rho(\{e_i|(e,i) \in N_j(h)\}, r, j, 0) = f(g(\{N_j(e)\}), r)$$

  Suppose we have an hyperedge q and j = 1, with arity = 2.

  $$N_j(q) = h$$

  $$g(N_j(q)) = f(h, r)$$

  $$\rho(h, r, h2t, 0) = \rho(\{e_i \mid (e,i) \in N_j(e_0)\}, r, j, 0)$$

  $$\rho(\{e_i \mid (e,i) \in N_j(e_0)\}, r, j, 0) = f(h, r)$$

  So for any edge with arity $\geq k$

  $$\rho(\{e_i \mid (e,i) \in N_j(e_0)\}, r, j, 0) = f(g(N_j(e), r)$$

A.3 Closed-form logical messages for KHG representation

Table 5 is a table of KHG embeddings that can be express in the form of $\rho(\{e_i|(e,i) \in N_j(e)\}, r, j, 0)$

B Sampling Algorithm for Knowledge Hypergraph Query

In this section, we introduce the algorithm for sampling EFO-1 queries from a Knowledge Graph of any arity, detailed in Algorithm 1. We adopt the general sampling approach for knowledge graphs

KHG Embedding	f(h, r)	Estimate Function
HypE	$e_i \otimes e_j$	$\otimes(r, g(\{N_j(e)\}))$
m-DistMult	$e_i \otimes e_j$	$\otimes(r, g(\{N_j(e)\}))$
m-CP	$e_i \otimes e_j$	$\otimes(r, g(\{N_j(e)\}))$
HSimplE	$e_i \otimes \text{shift}(e_j, \text{len}(e_j)/\alpha)$	$\otimes(r, g(\{N_j(e)\}))$

Table 5: Closed form foward estimation function f for KHG representations.

from (Wang et al., 2021c). Given a graph *G* and query type *t*, we randomly select a node as the root answer. From there, we sample a hyperedge to determine the relation type and its neighbors. If the operation is projection, we randomly choose a neighboring node as the answer for the subsequent query and recursively sample based on the next operation. The key distinction from ordinary sampling is that the position of the sampled neighbor may differ between sub-queries a_1 and a_2; for example, in $r_1(e_1, e_2, x)$ and $r2(x, e_3, e_4, y)$, the varaible x can occupy different positions. To accommodate these differences, position information is stored for both backward and forward passes.

C Complexity and Runtime Analysis

As shown in Table 6, if we consider the inference time for any transformer model to be approximately 0.5 seconds per batch, it is evident that the computational time for the iterative model grows with the number of nodes in the Query Operator Tree.
Let P represent the number of projection nodes and L the number of logical nodes in the query operator tree. The total number of nodes in the query operator tree can be expressed as:

$$n = P + L,$$

where n denotes the total number of nodes.
Time Complexity:

- **Single Pass Model:** In the single-pass model, only one transformer inference is required for all queries, which can be represented as $O(1)$.
- **LKHGT Time Complexity:** In the LKHGT framework, the time complexity primarily stems from the number of transformer inferences required. Each query type necessitates processing through the transformer for each node in the operator tree. Therefore, for LKHGT, the time complexity is:

$$O(n).$$

Algorithm 1 Ground Query Type

Require: G is KG with arity >= 2.
function SAMPLEQUERY(T, q)
T is an arbitrary node of the Knowledge graph
q is the query structure
if q.operation is projection **then**
Sample edge to obtain relation *r* and neighbor set *n*
RelationType ← *r*
NextAnswer ← random select *n*
NeighborSet ← $n - \{T, NextAnswer\}$
Store current position of T in previous edge (if any) and current sampled edge
Subquery ← SampleQuery(*NextAnswer*, q)
return (T.op, RelationType, NeighborSet, SubQuery)
else if q.operation is negation **then**
return (T.op, SubQuery)
else if q.operation is union or intersection **then** OperationResult ← []
for T.subquery_structure **do**
OperationResult.append(SampleQuery(T.subquery_structure, q))
end for
return (T.op, OperationResult)
end if
end function

Query Type(/s)	Single Pass (LSGT)	Iterative Model (LKHGT)
1p	0.5187	0.7516
2p	0.5449	1.5486
2i	0.5901	1.7755
pi	0.5096	2.4583
ip	0.5480	2.4869
2in	0.5375	1.8202
pin	0.5310	2.5221
pni	0.5752	2.5202
2u	0.5634	1.7913
up	0.5366	2.4643

Table 6: Query Type Process Time (in seconds) for 1024 Queries per Batch

Space Complexity: Assume there are a total of 8 token types for both the Logical Encoder and the Projection Encoder. Each token type has a unique set of Query, Key, and Value matrices. Thus, the total number of parameters required for self-attention in LKHGT is:

$$O(8d^2),$$

where d is the dimensionality of the embeddings. In contrast, the native self-attention mechanism has a space complexity of:

$$O(d^2).$$

The space complexity for other components of the model remains consistent with the original transformer architecture.