Sequential Query Encoding For Complex Query Answering on Knowledge Graphs

Anonymous authors
Paper under double-blind review

Abstract

Complex Query Answering (CQA) is an important and fundamental task for knowledge graph (KG) reasoning. Query encoding (QE) is proposed as a fast and robust solution to CQA. In the encoding process, most existing QE methods first parse the logical query into an executable computational direct-acyclic graph (DAG), then use neural networks to parameterize the operators, and finally recursively execute these neuralized operators. However, the parameterization-and-execution paradigm may be potentially over-complicated, as it can be structurally simplified by a single neural network encoder. Meanwhile, sequence encoders, like LSTM and Transformer, proved to be effective for encoding semantic graphs in related tasks. Motivated by this, we propose sequential query encoding (SQE) as an alternative to encode queries for CQA. Instead of parameterizing and executing the computational graph, SQE first uses a search-based algorithm to linearize the computational graph to a sequence of tokens and then uses a sequence encoder to compute its vector representation. Then this vector representation is used as a query embedding to retrieve answers from the embedding space according to similarity scores. Despite its simplicity, SQE demonstrates state-of-the-art neural query encoding performance on FB15k, FB15k-237, and NELL on an extended benchmark including twenty-nine types of in-distribution queries. Further experiment shows that SQE also demonstrates comparable knowledge inference capability on out-of-distribution queries, whose query types are not observed during the training process.

1 Introduction

Complex query answering (CQA) is a fundamental and important task in knowledge graph (KG) reasoning. CQA can also be used for solving downstream tasks like knowledge-base question answering (KBQA) (Sun et al., 2020). The complex queries on KG are defined in first-order logical form, and they can express complex semantics with the help of logical operators like *conjunction* $\wedge$, *disjunction* $\vee$, and *negation* $\neg$. In Figure 1 for example, given the logical query q_1, we want to find all the entities $V_?$ such that there exists certain protein V that either associate with *Alzheimer's Disease* or *Mad Cow Disease*.

CQA is challenging from the following two aspects. First, real-world KGs are always incomplete. To overcome this incompleteness issue of KGs, the task of CQA proposes to evaluate the knowledge inference capability of a query-answering system, so that the system can still effectively answer the queries even if some of the required information needs to be implicitly inferred from the KG. The method of sub-graph matching is sensitive to the missing edges from the KG and thus unable to conduct such implicit knowledge inference (Hamilton et al., 2018; Ren et al., 2020). Second, the complexity of conducting brute-force matching is exponential to the number of variables in the complex queries. Because of these two reasons, matching algorithms cannot be directly applied to the problem of CQA (Ren et al., 2020).

Query Encoding (QE) is proposed as a fast and robust solution for CQA (Hamilton et al., 2018). Most QE systems first parse the complex query into a computational graph. The computational graph describes how to find the answers to the query by using projection and set operations. For an example shown in Figure 2 (B), suppose we want to find the substances that interact with the proteins associated with Alzheimer's or Mad Cow disease as in Figure 2 (A). In this case, we need to first find the proteins that are associated with

Complex Queries	Interpretations
$q_1 = V_? . \exists V: Interact(V_?, V)$ $\wedge (Assoc(V, Alzheimer) \vee Assoc(V, MadCow))$	Find the substances that interact with the proteins associated with Alzheimer's or Mad cow disease.
$q_2 = V_? . \exists V: LocatedIn(V, America)$ $\wedge \neg Hold(V, WorldCup) \wedge IsPresident(V_?, V)$	Listing the presidents of American countries that so far, they have not held the World Cup.
$q_3 = V_? : HasProfession(V_?, Fictionist)$ $\wedge \neg IsResident(V_?, Europe) \wedge Win(V_?, HugoAward)$	Find the non-European fictionist that are the Hugo Award winners.

Figure 1: Three complex query examples and corresponding interpretations expressed in natural language.

Mad Cow disease and Alzheimer's disease respectively, then use a union operation to compute the union set of them, and finally, find what substances can interact with any of the proteins in this set.

Existing QE methods first use different neural networks to parameterize operators like *projection* and *union*, and then recursively execute them to encode a query into a query embedding (Hamilton et al., 2018). For example, they first use the embedding of *Alzheimer's Disease* and *Mad Cow Disease* and relational projection operators to compute the embeddings of two sets of proteins that are associated with them respectively. Then a *union* operator is used to compute the embedding of their union set. After this, another relation projection is used to compute the embedding of the substances that can interact with these proteins. After the encoding process is finished, they use the embedding of the answer set as the query embedding to retrieve the entities from the embedding space according to similarity scores between embeddings. In the learning process, the parameters of entity embeddings and the neural operators are jointly optimized.

Various QE methods are proposed following this paradigm, and their main focus is on using better embedding structures to encode the set of answers (Sun et al., 2020; Liu et al., 2021). For example, Ren et al. (2020); Zhang et al. (2021) propose to use geometric structures like rectangles and cones in the hyperspace to encode the entities. Bai et al. (2022) propose to use multiple vectors to encode the query to address the diversity of the answer entities. Meanwhile, probabilistic distributions can also be used for query encoding (Choudhary et al., 2021a;b), like Beta Embedding (Ren & Leskovec, 2020) and Gamma Embedding (Yang et al., 2022).

However, the procedures of first parameterizing and then executing the operators in the graph might be potentially over-complicated, because they can be structurally simplified by using a single neural network to encode the whole computational graph. The computational graph, on the other hand, can be regarded as a special type of semantic graph telling the meaning of how to execute (Yin & Neubig, 2018; Ren et al., 2020; Ren & Leskovec, 2020). Moreover, the sequence encoders, like LSTM (Hochreiter & Schmidhuber, 1997) and transformer (Vaswani et al., 2017), achieve high performance on tasks involving semantic graph encoding, such as Graph-to-Text generation (Konstas et al., 2017; Ribeiro et al., 2021).

Inspired by this, we propose sequential query encoding (SQE) for complex query encoding. In SQE, instead of parameterizing the operators and executing the computational graph, we use a search-based algorithm to linearize the graph into a sequence of tokens. After this, SQE uses a sequence encoder, like LSTM (Hochreiter & Schmidhuber, 1997) and Transformer (Vaswani et al., 2017), to encode this sequence of tokens. Its output sequence embedding is used as the query embedding. Similarly, SQE computes the similarity scores between the query embedding and entity embeddings to retrieve answers from the embedding space.

Despite its simplicity, SQE demonstrates better faithfulness and inference capability than state-of-the-art neural query encoders over FB15k (Bollacker et al., 2008; Bordes et al., 2013), FB15k-237 (Toutanova & Chen, 2015), and NELL (Carlson et al., 2010), under an extended benchmark, which includes twenty-nine types of logical queries (Wang et al., 2021). Moreover, we further evaluate the compositional generalization (Fodor & Lepore, 2002) of the SQE to see whether it can also generalize to the out-of-distribution query types that are unobserved during the training process. Again, the SQE method with an LSTM backbone demonstrates comparable inference capability to state-of-the-art neural query encoders on the out-of-distribution queries. [1] The main contributions of this paper can be summarized as follows:

[1] We will release all experiment source code and data after this paper is accepted.

(A) $q_1 = V_? . \exists V : Interact(V_?, V) \wedge (Assoc(V, Alzheimer) \vee Assoc(V, MadCow))$

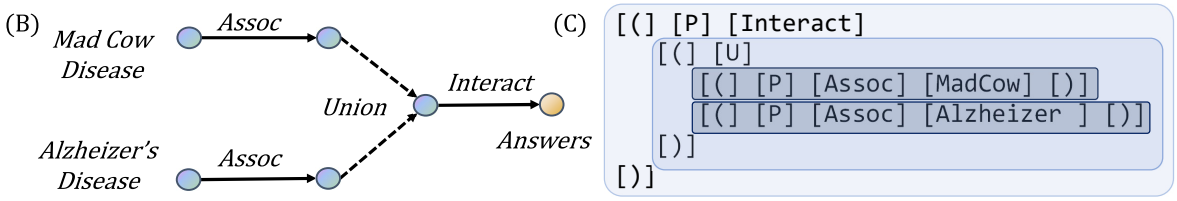

Figure 2: (A) The example complex query with the meaning of *Finding the substances that interact with the proteins associated with Alzheimer's or Mad cow disease.* (B) The computational graph of the complex query; (C) The linearized computational graph with proper indentations as a sequence of tokens, and the brake lines and indents are for better display. All the terms enclosed by square brackets `[.]` are tokens. The token of `[(]` and `[)]` are used to indicate the original graph structure. The token of `[P]` and `[U]` are tokens representing the *projection* and *union* operators. `[Interact]` and `[Assoc]` are the tokens representing relations. `[MadCow]` and `[Alzheizer]` are tokens representing the entities. All the tokens of brackets, operations, relations, and entities are treated in the same way by a sequence encoder.

- We propose sequential query encoding (SQE), which is the first method that uses sequence encoders to encode linearized computational graphs for answering first-order logic queries on KG.
- We conduct extensive experiments to demonstrate that SQE is the current state-of-the-art neural query encoding method for encoding in-distribution queries. Meanwhile, SQE demonstrates comparable inference capability on the out-of-distribution queries.
- Further analysis shows that even using the same neural structure, compared with executing following the computational graph, sequential encoding demonstrates better performance in encoding in-distribution queries on both faithfulness and knowledge inference capability.

2 Problem Definition

2.1 Logical Query

CQA is conducted on a knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{R})$. The $\mathcal{V}$ is the set of vertices v, and the $\mathcal{R}$ is the set of relation r. To describe the relations in logical expressions, the relations are defined in functional forms. Each relation r is defined as a function, and it has two arguments, which represent two entities v and v'. The value of function $r(v, v') = 1$ if and only if there is a relation between the entities v and v'.

The queries are defined in the first-order logical (FOL) forms. In a first-order logical expression, there are logical operations such as existential quantifiers $\exists$, conjunctions $\wedge$, disjunctions $\vee$, and negations $\neg$. In such a logical query, there are anchor entities $V_a \in \mathcal{V}$, existential quantified variables $V_1, V_2, ...V_k \in \mathcal{V}$, and a target variable $V_? \in \mathcal{V}$. The knowledge graph query is written to find the answer entities $V_? \in \mathcal{V}$, such that there exist $V_1, V_2, ...V_k \in \mathcal{V}$ satisfying the logical expression in the query. For each query, it can be converted to a disjunctive normal form, where the query is expressed as a disjunction of several conjunctive expressions:

$$q[V_?] = V_?.\exists V_1, ..., V_k : c_1 \vee c_2 \vee ... \vee c_n, \tag{1}$$

$$c_i = e_{i1} \wedge e_{i2} \wedge ... \wedge e_{im}. \tag{2}$$

Each c_i represents a conjunctive expression of literals e_{ij}, and each e_{ij} is an atomic or the negation of an atomic expression in any of the following forms: $e_{ij} = r(v_a, V)$, $e_{ij} = \neg r(v_a, V)$, $e_{ij} = r(V, V')$, or $e_{ij} = \neg r(V, V')$. Here $v_a \in V_a$ is one of the anchor entities, and $V, V' \in \{V_1, V_2, ..., V_k, V_?\}$ are distinct variables satisfying $V \neq V'$. When the query is an EPFO query, then there is no negation in the expressions.

Algorithm 1 Linearization of the computational graph of a query into a sequence of tokens.

Require: G is the computational graph of a certain query.
Require: $Tokenize()$ is the function that converts relations and entities to their token ids
function LINEARIZE(T)
T is the target node of the computational graph.
if $T.operation = projection$ **then**
`Rel` $\leftarrow Tokenize(T.relation)$
`SubQueryTokens` $\leftarrow$ LINEARIZE($T.prev$)
return `[(][P] + Rel + SubQueryTokens + [)]`
else if $T.operation = intersection$ or $T.operation = union$ **then**
`QueryTokens` $\leftarrow$ `[(]`
if $T.operation = intersection$ **then**
`QueryTokens` $\leftarrow$ `QueryTokens` + `[I]`
else
`QueryTokens` $\leftarrow$ `QueryTokens` + `[U]`
end if
for $prev \in T.prevs$ **do**
`SubQueryTokens` $\leftarrow$ LINEARIZE($prev$)
`QueryTokens` $\leftarrow$ `QueryTokens + SubQueryTokens`
end for
return `QueryTokens + [)]`
else if $T.operation = negation$ **then**
$SubQueryTokens \leftarrow$ LINEARIZE($T.prev$)
return `[(][N] + SubQueryTokens + [)]`
else if $T.operation = e$ **then return** $Tokenize(T.entity)$
end if
end function

2.2 Computational Graph

As shown in Figure 2 (B), there is a corresponding computational graph for each FOL logical query. The computational graph is defined in a directional acyclic graph (DAG) structure. The nodes and edges in the graph represent the intermediate states and operations respectively. The operations are used to encode the sub-queries following the computational graph recursively, they implicitly model the set operations of the intermediate query results. The set operations are as follows:

- *Relational Projection*: Given a set of entities A and a relation $r \in R$, this operation returns all entities holding relation r with at least one entity $e \in A$. Namely, $P_r(A) = \{v \in \mathcal{V} | \exists v' \in A, r(v', v) = 1\}$;
- *Intersection*: Given sets of entities $A_1, ...A_n \subset \mathcal{V}$, this operation computes their intersection $\cap_{i=1}^{n} A_i$;
- *Union*: Given several sets of entities $A_1, ...A_n \subset \mathcal{V}$, this operation calculates their union $\cup_{i=1}^{n} A_i$;
- *Complement/Negation*: Given a set of entities A, it calculates the absolute complement $\mathcal{V} - A$.

3 Sequential Query Encoding

In this paper, we propose an alternative way to encode the complex query by first linearizing a computational graph into a sequence of tokens, and then using a sequence encoder to compute its sequence embedding as the corresponding query embedding. For example in Figure 2, we equivalently convert a computational graph (B) into a sequence of tokens shown in (C) by using the Algorithm 1. Then SQE uses sequence encoders, such as LSTM and transformer, to encode the token sequence in Figure (C). Finally, SQE uses the output sequence embedding as query embedding to retrieve answers from the entity embedding space.

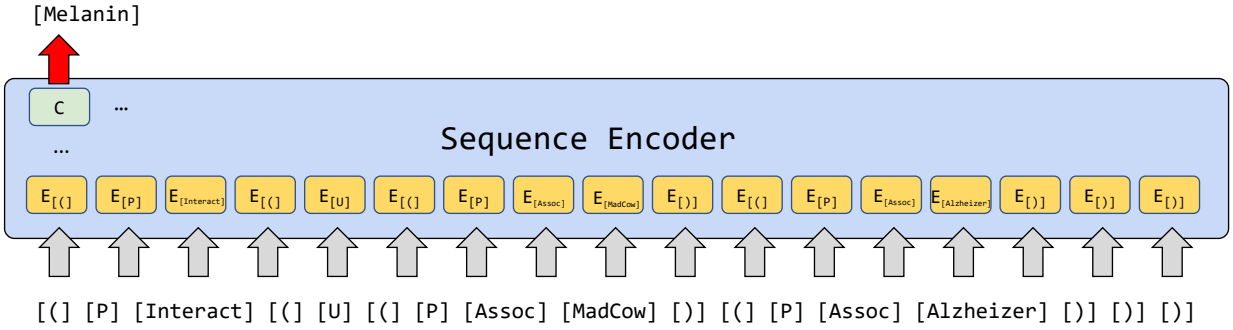

Figure 3: The illustration of using a sequence encoder to encode the linearized computational graph. The sequence encoder uses the representation of the first output token as the sequence representation.

3.1 Linearizing Computational Graph

A directed acyclic computational graph is first linearized to a sequence of tokens Our linearizing algorithm as shown in Algorithm 1, starts from its target node T. First, we try to find the last operation in this graph for T. It could be either a relational *projection*, *intersection*, *union*, or *negation*. Then the first two tokens of the answer are determined as `[(][P]`, `[(][I]`, `[(][U]`, and `[(][N]` correspondingly, and the last token is determined as `[)]`. If the operation type is *projection*, we will additionally add the third token indicating the relation type like `[Interact]` in Figure 2 (C). After this, we will find the nodes in the DAG that have an outward edge pointing to the target node T, and these nodes are called previous nodes. If the operation is *projection* or *negation*, there is always only one such node. If the operation is *intersection* and *union*, there might be two or more such nodes. Regardless of the operation type, the linearizing algorithm is recursively called on the previous nodes until the base case is reached. The base case of this recursion is the previous node is a given anchor entity, such as Mad Cow disease in Figure 2 (C). Then the algorithm will use a unique token to represent this entity like `[MadCow]`. During the recursions, the output tokens from the previous nodes are put between the square bracket tokens `[(]` and `[)]` determined by the target node.

3.2 Encoding Linearized Computational Graph

After the computational graph is linearized to a sequence of tokens, SQE uses a sequence encoder to encode them. All the tokens in the sequence, including the brackets `[(][)]`, operations `[P][I][N][U]`, relations `[Assoc][Interact]`, and entities `[Alzheizer]` are assigned with unique and unified ids respectively. Then a unified embedding table is created to hold the token embeddings corresponding to all these tokens.

As shown in Figure 3, the input tokens are first converted to a sequence of embedding vectors, and then these embeddings are used as input to sequence encoders. The sequence encoder will compute the contextualized representation for each token, and we use the embedding of the first token as the sequence representation. This sequence representation is used as the query embedding to retrieve answers from the entity embedding space. Suppose the entity embedding of the entity v is e_v, and the sequence embedding of query q is e_q. We use the inner product between e_v and e_q as the measurement of the similarity between query q and entity v. To train the SQE model, we compute the normalized probability of the entity v being the correct answer of query q by using the `softmax` function on all similarity scores,

$$p(q, v) = \frac{< e_q, e_v >}{\sum_{v' \in V} < e_q, e_{v'} >}. \tag{3}$$

Then we construct a cross-entropy loss to maximize the log probabilities of all correct query-answer pairs:

$$L = -\frac{1}{N} \sum_i \log p(q^{(i)}, v^{(i)}). \tag{4}$$

Each $(q^{(i)}, v^{(i)})$ denotes one of the positive query-answer pairs, and there are N pairs in the training set.

4 Experiment

In this section, we first explain the evaluation settings and metrics for query encoding. Then we discuss the knowledge graphs and the benchmark datasets for evaluation. Then we briefly present the neural query encoding baseline methods that we directly compared with. After this, we disclose the implementation details for SQE. Finally, we discuss the experiment results and conduct further analysis on SQE.

4.1 Evaluations

In our experiment, following previous work, we also use the following three knowledge graphs: FB15k (Bollacker et al., 2008; Bordes et al., 2013), FB15k-237 (Toutanova & Chen, 2015), and NELL (Carlson et al., 2010). As shown in Table 7, the edges in each knowledge graph are separated into training, validation, and testing with a ratio of 8:1:1 respectively. Training graph $\mathcal{G}_{train}$, validation graph $\mathcal{G}_{val}$, and test graph $\mathcal{G}_{test}$ are constructed by training edges, training+validation edges, and training+validation+testing edges respectively following previous setting (Ren et al., 2020). All QE models have evaluated the following three aspects: faithfulness, knowledge inference capability, and compositional generalization.

4.1.1 Faithfulness and Knowledge Inference Capability

The most important aspect of the query encoding model is its capability of knowledge `inference`. To robustly deal with the incompleteness of knowledge graphs, the query encoding model is required to answer queries with answers that need to be implicitly inferred from existing facts in the KG. On the other hand, the capability of `entailment` is also measured to evaluate whether a QE model can faithfully answer the queries that are explicitly shown on the training graph (Sun et al., 2020).

To precisely describe the metrics, we use the q to represent a testing query and $\mathcal{G}_{val}$, $\mathcal{G}_{test}$ to represent the validation and the testing knowledge graph. Here we use $[q]_{val}$ and $[q]_{test}$ to represent the answers of query q on the validation graph $\mathcal{G}_{val}$ and testing graph $\mathcal{G}_{test}$ respectively. Equation 5 and Equation 6 describe how to compute the `Inference` and `Entailment` metrics respectively. When the evaluation metric is Hit@K, the $m(r)$ is defined as $m(r) = \mathbf{1}[r \leq K]$. In other words, $m(r) = 1$ if $r \leq K$, otherwise $m(r) = 0$. Meanwhile, if the evaluation metric is mean reciprocal ranking (MRR), then the $m(r)$ is defined as $m(r) = \frac{1}{r}$.

$$\texttt{Inference}(q) = \frac{\sum_{v \in [q]_{test}/[q]_{val}} m(\texttt{rank}(v))}{|[q]_{test}/[q]_{val}|}. \tag{5}$$

$$\texttt{Entailment}(q) = \frac{\sum_{v \in [q]_{train}} m(\texttt{rank}(v))}{|[q]_{train}|}. \tag{6}$$

During the training process, the testing graph $\mathcal{G}_{test}$ is unobserved. In the hyper-parameters selection process, we are still using the metrics in Equation 5 but replacing graphs $\mathcal{G}_{test}/\mathcal{G}_{val}$ by $\mathcal{G}_{val}/\mathcal{G}_{train}$ respectively.

4.1.2 Compositional Generalization

In addition to these two aspects, the compositional generalizability of the query encoding models should also be systematically studied. Compositionality is the idea that the meaning of a complex expression can be constructed from its less complex sub-expressions (Fodor & Lepore, 2002). The compositional generalizability describes the capability of a system that, when it is given some primitive examples and their simple combinations, the system can deal with the examples with unseen combinations. Such ability is commonly evaluated on the problem of language or visual reasoning (Johnson et al., 2017; Finegan-Dollak et al., 2018; Loula et al., 2018; Lake & Baroni, 2018; Keysers et al., 2020). In this paper, we extend the evaluation of compositional generalizability toward the problem of complex query answering.

Compositional generalizability is critical to query encoding. As the total number of query types grows exponentially with the number of variables inside a complex query, it is infeasible to enumerate all query types for training. So the QE models are expected to have compositional generalizability to enhance their performance on the unseen/out-of-distribution query types. For example, suppose the query when the QE

Table 1: The comparison between different benchmark datasets on the number of query types. The in-distribution query types refer to the types that the QE model is trained on during the training process. The out-of-distribution query types refer to the types that are unobserved in the training process, but are required to be tested in the evaluation process.

Datasets	In-distribution Types	Out-of-distribution Types	Total
Q2B (Ren et al., 2020)	5	4	9
BetaE (Ren & Leskovec, 2020)	10	4	14
SMORE (Ren et al., 2022)	10	4	14
Our Benchmark	**29**	**29**	**58**

methods are trained on the query types of `(p,(e))`, such as *What substance can interact with Prion Protein?* and `(u,(p,(e)), (p,(e)))`, like *What protein are associated with Alzheimer's or Mad Cow disease?*, we expect it can also perform well on `(p,(u,(p,(e)), (p,(e))))`, like *What are the substances that interact with the proteins associated with Alzheimer's or Mad cow disease*, which is composed of the previous two.

4.2 Benchmarks

Most existing QE models use the benchmark from Ren & Leskovec (2020) to evaluate their performance. However, this benchmark has two major drawbacks. First, its number of query types is limited. In total, it includes fourteen types of queries. Because of this, it is insufficient to describe the complex structures of general logical queries. Second, it cannot be used for evaluating compositional generalizability because all the query types that are evaluated are observed during the training process. Recently, SMORE (Ren et al., 2022) scales up the size of the knowledge graph and the number of samples but keeps the same query types, leaving these two problems unsolved. Meanwhile, Wang et al. (2021) discusses the logic forms of different query types and scales up the query types. However, the experiments and discussions on compositional generalization are still limited to five in-distribution query types and two out-of-distribution query types. Because of these reasons, we choose to construct our own benchmark dataset.

To effectively evaluate the compositional generalizability, we use the queries with two anchors with a maximum depth of three as the training queries (Wang et al., 2021). Meanwhile, we additionally sample the same number of query types with three anchor entities and a maximum depth of three as unseen query types for the evaluation of compositional generalizability. As a result, we obtain twenty-nine types of in-distribution training queries and additional twenty-nine types of unseen/out-of-distribution evaluation queries. The detailed structures of query types are listed in Table 9. Meanwhile, we use the conjunctive query types from these queries to evaluate the query encoding model that does not intrinsically support *negation* and *union* operators, and their details are shown in Table 8. Here, we sample the knowledge graph queries according to the query types by using the Algorithm 2. The training queries are sampled from the training graph. The testing queries are sampled from the testing graph, while their training, validation, and testing answers are searched from the training, validation, and testing graphs respectively. Unlike previous benchmarks (Ren et al., 2020; Ren & Leskovec, 2020; Wang et al., 2021), we do not filter out queries with large answer sets, because the motivation for removing such queries in previous work is not well justified, and the removal potentially creates a distributional bias that the sampled queries could be less likely to include nodes that have higher degrees. Finally, the statistics of the queries sampled are shown in Table 1 and Table 2. Compared with previous benchmarks, we scaled up the number of query types from fourteen to fifty-eight types, while keeping the same amount of queries on each of the query types.

4.3 Baseline Models

We briefly introduce the baseline query encoding models that use various neural networks to parameterize the operators in the computational graph to recursively encode the query into various embedding structures.

- GQE (Hamilton et al., 2018) uses vectors to encode complex queries.

Table 2: Number of queries used for each query type.

Knowledge Graph	Training		Validation	Testing
	(p,(e))	Other Types	All Types	All Types
FB15k	273,710	821,130	8,000	8,000
FB15k-237	149,689	449,067	5,000	5,000
NELL-995	107,982	323,946	4,000	4,000

- Q2B (Ren et al., 2020) uses hyper-rectangles to encode complex queries.
- HYPE (Choudhary et al., 2021b) encodes the queries in a hyperbolic space.
- BetaE (Ren & Leskovec, 2020) uses Beta distributions to encode queries.
- ConE (Zhang et al., 2021) uses Cone Embeddings to handle negations.
- Q2P (Bai et al., 2022) uses multiple vectors to encode the queries.
- Neural MLP (Mixer) (Amayuelas et al., 2022) use MLP and MLP-Mixer to as the operators.

The performance of GQE, Q2B, and Hype are shown in Table 3. Because they do not support *negation* and *union* operators during the query encoding process, we train and evaluate them separately on the conjunctive queries, and details are shown in Table 8. For the rest of the QE models, we evaluate them by using the queries shown in Table 9, and their results are shown in Table 4. All the baseline query encoding structures are implemented using the same latent space size of three hundred. There are also neural-symbolic query encoding methods proposed (Sun et al., 2020; Xu et al., 2022; Zhu et al., 2022). In this line of research, their query encoders refer back to the training knowledge graph to obtain symbolic information from the graph. Because of this, the query encoder is not purely learned from the training queries. As their contribution is orthogonal to our discussion on pure neural query encoders, we do not conduct direct comparisons.

4.4 Implementation Details for SQE

Sequential query encoding (SQE) is implemented with the backbones of established sequence encoding models. The previous dominant method for sequence encoding is recurrent neural networks, such as GRU, and LSTM (Hochreiter & Schmidhuber, 1997; Chung et al., 2014). The recurrent models are all used in a bi-directional way, and they are stacked for three layers. Meanwhile, SQE also uses the encoder part of the Transformer (Vaswani et al., 2017) as its sequence encoding backbone. The three layers of the transformer encoder structures are used, and each of them is implemented with sixteen attention heads. The hidden sizes of both recurrent models and transformer are set to be three hundred to fairly compared with the baselines. All the SQE and previous QE models are trained with the same batch size of 1024. All the experiments are conducted on the Nvidia GeForce RTX 3090 graphics cards.

4.5 Experiment Results

Performance on In-distribution Types Table 3 and 4 show the performance of the in-distribution query whose query types are used for training the QE models during the training process. The SQE models with LSTM and Transformer backbones constantly outperform previous QE methods on the evaluation of `Entailment` and `Inference` on both Conjunctive Queries and FOL queries. This means that, when dealing with the query types that have been used for training, the sequential encoding models are better at faithfully encoding the information in the knowledge graph. Meanwhile, they have better knowledge inference capability to answer queries that require implicit knowledge inference from the KG.

Table 3: The MRR results on answering conjunctive queries. The `Entailment` and `Inference` metrics are the higher the better. The best and second-best performances are marked in Bold and underlined.

Datasets	Models	In-distribution Queries		Out-of-distribution Queries		Average	
		Entailment	Inference	Entailment	Inference	Entailment	Inference
FB15K	GQE	21.57	17.31	15.58	15.52	18.58	16.42
	Q2B	24.40	17.08	19.59	16.63	22.00	16.86
	HYPE	28.70	22.50	<u>27.76</u>	<u>26.08</u>	28.23	24.29
	Q2P	30.18	25.97	23.65	23.08	26.92	<u>24.53</u>
	SQE + LSTM	<u>47.80</u>	<u>29.65</u>	**42.93**	**30.74**	**45.37**	**30.20**
	SQE + Transformer	**55.08**	**36.21**	14.12	12.64	<u>34.60</u>	24.43
FB15K-237	GQE	23.88	9.87	15.77	10.04	19.83	9.96
	Q2B	25.04	8.80	19.25	10.91	22.15	9.86
	HYPE	29.40	11.97	<u>28.19</u>	<u>15.81</u>	28.80	<u>13.89</u>
	Q2P	38.05	12.59	25.27	14.65	31.66	13.62
	SQE + LSTM	<u>62.51</u>	<u>14.53</u>	**50.75**	**19.34**	**56.63**	**16.94**
	SQE + Transformer	**67.13**	**15.48**	16.36	9.64	<u>41.75</u>	12.56
NELL	GQE	51.65	9.11	37.87	10.43	44.76	9.77
	Q2B	55.29	8.37	47.34	11.89	51.32	10.13
	HYPE	53.87	12.15	<u>50.53</u>	<u>16.09</u>	52.20	<u>14.12</u>
	Q2P	71.20	11.75	19.79	8.67	45.50	10.21
	SQE + LSTM	<u>86.99</u>	<u>12.92</u>	**85.24**	**18.08**	**86.12**	**15.50**
	SQE + Transformer	**90.19**	**14.25**	34.76	11.51	<u>62.48</u>	12.88

Performance on Out-of-distribution Query Types The out-of-distribution queries are used for measuring the compositional generalizability of QE models on both `Entailment` and `Inference`. As shown in Table 4, when SQE is used for the query types that have not been used for training, it performs worse on `Entailment` than previous models. This implies that SQE models are less compositional generalizable on faithfulness. However, SQE performs comparably to previous QE methods in the `Inference` metric on out-of-distribution query types. This indicates SQE is comparably compositional and generalizable to the previous QE model on the knowledge inference capability.

5 Discussion

There are two major differences between the previous QE methods and SQE: (1) The previous query encoding methods encode the query recursively following the computational graph, but the sequence encoder directly uses special input tokens of `[(]` and `[)]` to indicate and represent the graph structure as model input; (2) The previous query encoding contains the inductive bias stemming from the understanding of logical and set operations in parameterizations and encoding structures, while the sequence encoder also regards logical operations as some special tokens, whose meaning and functionality are purely learned from the empirical data. To further investigate the effects of these factors on the performance differences, we further propose to use another semantic graph encoding model to control the variables. Tree-LSTM (Tai et al., 2015) is another model that is widely used in NLP tasks to encode semantic and syntactic parsing graphs. As shown in Figure 4, Tree-LSTM can also encode the computational graph in CQA recursively following the computational graph, which is the same as previous QE methods. On the other hand, Tree-LSTM treated all operations, entities, and relations as tokens. Thus, similar to SQE, Tree-LSTM does not have prior knowledge and related structural inductive bias about logic and set operations.

5.1 The importance of using sequential query encoding

Both Tree-LSTM and SQE+LSTM models use exactly the same neural network structure of LSTM cells, and their only difference is whether using a linearized computational graph to conduct sequence encod-

Table 4: The MRR results on answering FOL queries. Note that the `Entailment` and `Inference` metrics are the higher the better. The best and second-best performances are marked in Bold and underlined.

Datasets	Models	In-distribution Queries		Out-of-distribution Queries		Average	
		Entailment	Inference	Entailment	Inference	Entailment	Inference
FB15K	ConE	30.14	20.58	23.01	**16.04**	26.58	18.31
	BetaE	34.91	19.01	25.84	14.84	30.38	16.93
	Q2P	43.00	<u>23.34</u>	27.47	15.51	35.24	**19.43**
	Neural MLP	44.18	21.62	**31.37**	<u>15.78</u>	37.78	18.70
	+ MLP Mixer	38.17	20.45	27.81	15.11	32.99	17.78
	SQE + CNN	44.74	22.49	24.61	13.36	34.68	17.93
	SQE + GRU	48.64	22.67	<u>29.36</u>	15.11	<u>39.00</u>	18.89
	SQE + LSTM	**49.17**	23.17	29.32	14.90	**39.25**	<u>19.04</u>
	SQE + Transformer	<u>49.13</u>	**25.83**	13.39	8.00	31.26	16.92
FB15K-237	ConE	36.69	9.75	28.13	**8.82**	32.41	9.29
	BetaE	32.48	8.30	22.96	7.29	27.72	7.80
	Q2P	52.33	10.17	32.70	8.62	42.52	9.40
	Neural MLP	51.09	10.03	**36.85**	<u>8.75</u>	<u>43.97</u>	9.39
	+ MLP Mixer	45.19	10.07	33.03	8.66	39.11	9.37
	SQE + CNN	52.09	10.14	28.21	7.65	40.15	8.90
	SQE + GRU	55.46	10.59	32.25	8.34	43.86	<u>9.47</u>
	SQE + LSTM	<u>56.02</u>	<u>10.62</u>	<u>33.41</u>	8.62	**44.72**	**9.62**
	SQE + Transformer	**59.15**	**11.30**	15.06	4.98	37.11	8.14
NELL	ConE	58.36	8.55	45.07	7.92	51.72	8.24
	BetaE	48.13	7.06	34.63	6.65	41.38	6.86
	Q2P	79.79	10.29	56.18	<u>8.45</u>	67.99	<u>9.37</u>
	Neural MLP	80.42	9.98	**63.12**	8.20	<u>71.77</u>	9.09
	+ MLP Mixer	78.08	10.05	58.67	8.26	68.38	9.16
	SQE + CNN	82.10	9.99	51.80	7.30	66.95	8.65
	SQE + GRU	85.36	10.30	57.42	8.21	71.39	9.26
	SQE + LSTM	<u>85.42</u>	<u>10.37</u>	<u>60.06</u>	**8.59**	**72.74**	**9.48**
	SQE + Transformer	**85.52**	**10.95**	22.24	4.97	53.88	7.96

ing. SQE+LSTM performs consistently better than the Tree-LSTM on three datasets on `Entailment` and `Inference` over queries on the in-distribution queries. This indicates that when using the same encoding structure, the sequence encoding used in SQE is better than encoding recursively following the computational graph on in-distribution queries. Meanwhile, for the out-of-distribution queries, the faithfulness of Tree-LSTM is better than SQE+LSTM, but they have comparable knowledge inference capabilities.

5.2 The importance of neural network designs in query encoding

To further explore the reason why Tree-LSTM is able to achieve comparable and even better performance than the previous query encoding methods, we conduct ablations on its structure by removing its memory cell vector c_i in the LSTM cell. The performance of Tree-LSTM drastically drops when the memory cells are removed. The memory cells in LSTM cells are mainly designed to prevent gradient vanishing during optimization. The performance drop also suggests potential optimization issues for the recursive query encoders. This is because previous QE methods are designed based on the understanding of logic and set operations. However, the design of the previous query encoding structures neglects whether the structures proposed can be effectively optimized without experiencing optimization issues. It is also possible that, although a QE has a good inductive bias for encoding logic and sets, it may not be effectively optimized due to gradient vanishing or gradient explosion. The SQE model, on the other hand, uses established sequence encoders as backbones and is less likely to suffer from technical problems in optimization.

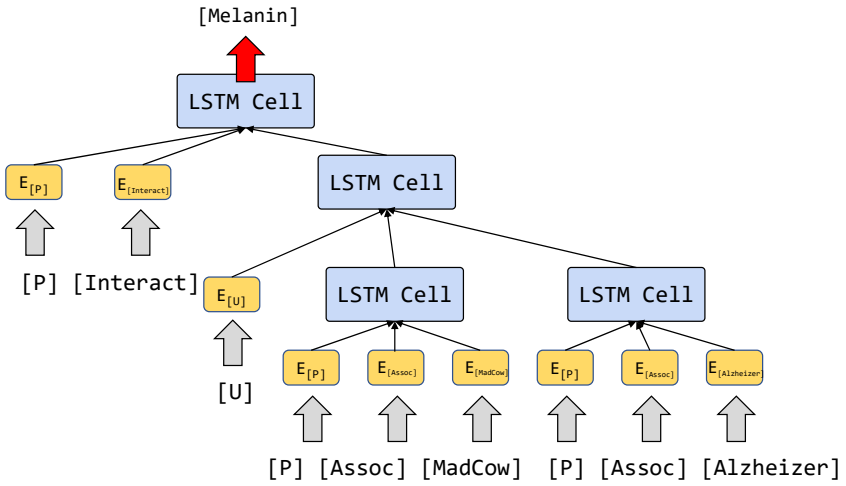

Figure 4: A Tree-LSTM encoder (Tai et al., 2015) is used to encode the computational DAG. All the operations, relations, and entities are converted to corresponding token embeddings.

Table 5: Comparison between Tree-LSTM query encoder and SQE+LSTM encoder. The Memory cell removed version LSTM cell suffers vanishing gradients.

Datasets	Models	In-distribution Queries		Out-of-distribution Queries		Average	
		Entailment	Inference	Entailment	Inference	Entailment	Inference
FB15K	SQE + LSTM	**49.17**	**23.17**	29.32	14.90	**39.25**	**19.04**
	Tree LSTM	46.42	21.50	**31.89**	**15.71**	39.16	18.61
	- Memory Cell	33.78	17.26	20.25	11.75	27.02	14.51
FB15K-237	SQE + LSTM	**56.02**	**10.62**	33.41	**8.62**	44.72	**9.62**
	Tree LSTM	53.15	9.63	**36.89**	8.61	**45.02**	9.12
	- Memory Cell	39.17	8.63	21.80	7.27	30.49	7.95
NELL	SQE + LSTM	**85.42**	**10.37**	60.06	**8.59**	72.74	**9.48**
	Tree LSTM	85.39	9.79	**68.52**	8.28	**76.96**	9.04
	- Memory Cell	67.33	8.03	28.89	5.58	48.11	6.81

5.3 The importance of query representation

Previous research on QE proposed various ways to represent complex queries. For example, Ren et al. (2020) propose to use hyper-rectangles, and Bai et al. (2022) propose to use particle embeddings to represent the complex queries. Meanwhile, vector embeddings are generally perceived as insufficient to represent the answers to complex queries. However, in the SQE method, each query is simply represented as a vector embedding. In addition to this, Tree-LSTM also uses a single vector to achieve comparable results to previous QE methods. This indicates that, with the proper design of neural network structures and effective optimization, vector embedding is also able to achieve comparable or better performance.

5.4 Why SQE-Transformer bad at compositional generalization?

We notice that the compositional generalizability of the SQE+Transformer encoding is low. So we conduct further experiments and analysis on why it has such a performance drop. We compare the sequence encoder Transformer and another Transformer-based encoding method. BiQE (Kotnis et al., 2021) also proposes to use a transformer to encode complex queries. Differently, they use special positional encoding schemes instead of tokens to represent the graph structure and operations. Because of its specially designed positional

Table 6: The performance comparison between BiQE and SQE+Transformer.

Datasets	Models	In-distribution Queries		Out-of-distribution Queries		Average	
		Entailment	Inference	Entailment	Inference	Entailment	Inference
FB15K	BiQE	52.73	35.64	**25.53**	**21.41**	**39.13**	**28.53**
	SQE + Transformer	**55.08**	**36.21**	14.12	12.64	34.60	24.43
FB15K-237	BiQE	64.35	14.86	**39.36**	**16.21**	**51.86**	**15.54**
	SQE + Transformer	**67.13**	**15.48**	16.36	9.64	41.75	12.56
NELL	BiQE	88.87	**14.29**	**37.79**	10.83	**63.33**	12.56
	SQE + Transformer	**90.19**	14.25	34.76	**11.51**	62.48	**12.88**

encoding scheme, BiQE is only applicable to conjunctive queries and is unable to deal with *union* and *negation* operators. According to Table 6, BiQE performs better on out-of-distribution queries and worse on in-distribution queries than the SQE+Transformer model, both on `Entailment` and `Inference`. This indicates that the original positional encoding of the Transformer is the main reason for the poor compositional generalizability on complex query answering. It is also possible to design a better positional encoding scheme for the transformer that improves the compositional generalization while can also deal with the `Union` and `Negation` operation. We leave this as a direction for future research.

6 Related Work

Complex query answering is a deductive knowledge graph reasoning task, in which a model or system is required to answer the logical query on an incomplete knowledge graph. Query encoding is a fast and robust method for dealing with complex query answering. Recently, there is also new progress on query encoding that is orthogonal to this paper, which puts a focus on the neural encoders for complex queries. Xu et al. (2022) propose a neural-symbolic entangled method, ENeSy, for query encoding. Wang et al. (2023) propose to use pre-trained knowledge graph embeddings and one-hop message passing to conduct complex query answering. Recently, Yang et al. (2022) propose to use Gamma Embeddings to encode complex logical queries. Liu et al. (2022) propose to use pre-training on the knowledge graph with kg-transformer and then conduct fine-tuning on the complex query answering.

Meanwhile, query decomposition (Arakelyan et al., 2021) is another way to deal with the problem of complex query answering. In this method, the probabilities of atomic queries are first computed by a link predictor, and then continuous optimization or beam search is used to conduct inference time optimization. Moreover, Wang et al. (2023) propose an alternative to query encoding and query decomposition, in which they conduct message passing one the one-hop atomics to conduct complex query answering. Xi et al. (2022) propose ROMA, a framework to answer complex logical queries on multi-view knowledge graphs. Theorem proving is another deductive reasoning task on knowledge graphs. Neural theorem proving (Rocktäschel & Riedel, 2017; Minervini et al., 2020; 2021) methods are proposed to deal with the incompleteness of existing knowledge graphs to conduct inference on the missing information by using embeddings.

7 Conclusions

In this paper, we present sequential query encoding for complex query answering. We evaluate the faithfulness and inference capability for various query encoding methods on both in-distribution and out-of-distribution queries. Experiments show that, despite its simplicity, SQE has better faithfulness and inference capability than existing neural query encoding methods on in-distribution query types. Meanwhile, SQE achieves comparable inference capability to previous QE methods. Further experiments address the importance of the structural design of neural network structures, meanwhile demonstrating that the existing design of positional encoding in Transformer obstructs its compositional generalization to out-of-distribution queries.

References

Alfonso Amayuelas, Shuai Zhang, Susie Xi Rao, and Ce Zhang. Neural methods for logical reasoning over knowledge graphs. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=tgcAoUVHRIB.

Erik Arakelyan, Daniel Daza, Pasquale Minervini, and Michael Cochez. Complex query answering with neural link predictors. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=Mos9F9kDwkz.

Jiaxin Bai, Zihao Wang, Hongming Zhang, and Yangqiu Song. Query2Particles: Knowledge graph reasoning with particle embeddings. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pp. 2703–2714, Seattle, United States, July 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-naacl.207. URL https://aclanthology.org/2022.findings-naacl.207.

Kurt D. Bollacker, Colin Evans, Praveen K. Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In Jason Tsong-Li Wang (ed.), *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, pp. 1247–1250. ACM, 2008. doi: 10.1145/1376616.1376746. URL https://doi.org/10.1145/1376616.1376746.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pp. 2787–2795, 2013. URL https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. Toward an architecture for never-ending language learning. In Maria Fox and David Poole (eds.), *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1879.

Nurendra Choudhary, Nikhil Rao, Sumeet Katariya, Karthik Subbian, and Chandan K. Reddy. Probabilistic entity representation model for reasoning over knowledge graphs. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 23440–23451, 2021a. URL https://proceedings.neurips.cc/paper/2021/hash/c4d2ce3f3ebb5393a77c33c0cd95dc93-Abstract.html.

Nurendra Choudhary, Nikhil Rao, Sumeet Katariya, Karthik Subbian, and Chandan K. Reddy. Self-supervised hyperboloid representations from logical queries over knowledge graphs. In Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (eds.), *WWW '21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19-23, 2021*, pp. 1373–1384. ACM / IW3C2, 2021b. doi: 10.1145/3442381.3449974. URL https://doi.org/10.1145/3442381.3449974.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir R. Radev. Improving text-to-sql evaluation methodology. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pp. 351–360. Association for Computational Linguistics, 2018. doi: 10.18653/v1/P18-1033. URL https://aclanthology.org/P18-1033/.

Jerry A Fodor and Ernest Lepore. *The compositionality papers*. Oxford University Press, 2002.

William L. Hamilton, Payal Bajaj, Marinka Zitnik, Dan Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 2030–2041, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/ef50c335cca9f340bde656363ebd02fd-Abstract.html.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross B. Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 1988–1997. IEEE Computer Society, 2017. doi: 10.1109/CVPR.2017.215. URL https://doi.org/10.1109/CVPR.2017.215.

Daniel Keysers, Nathanael Schärli, Nathan Scales, Hylke Buisman, Daniel Furrer, Sergii Kashubin, Nikola Momchev, Danila Sinopalnikov, Lukasz Stafiniak, Tibor Tihon, Dmitry Tsarkov, Xiao Wang, Marc van Zee, and Olivier Bousquet. Measuring compositional generalization: A comprehensive method on realistic data. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=SygcCnNKwr.

Ioannis Konstas, Srinivasan Iyer, Mark Yatskar, Yejin Choi, and Luke Zettlemoyer. Neural AMR: Sequence-to-sequence models for parsing and generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 146–157, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1014. URL https://aclanthology.org/P17-1014.

Bhushan Kotnis, Carolin Lawrence, and Mathias Niepert. Answering complex queries in knowledge graphs with bidirectional sequence encoders. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pp. 4968–4977. AAAI Press, 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/16630.

Brenden M. Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In Jennifer G. Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2879–2888. PMLR, 2018. URL http://proceedings.mlr.press/v80/lake18a.html.

Lihui Liu, Boxin Du, Heng Ji, ChengXiang Zhai, and Hanghang Tong. Neural-answering logical queries on knowledge graphs. In Feida Zhu, Beng Chin Ooi, and Chunyan Miao (eds.), *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14-18, 2021*, pp. 1087–1097. ACM, 2021. doi: 10.1145/3447548.3467375. URL https://doi.org/10.1145/3447548.3467375.

Xiao Liu, Shiyu Zhao, Kai Su, Yukuo Cen, Jiezhong Qiu, Mengdi Zhang, Wei Wu, Yuxiao Dong, and Jie Tang. Mask and reason: Pre-training knowledge graph transformers for complex logical queries. In Aidong Zhang and Huzefa Rangwala (eds.), *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, pp. 1120–1130. ACM, 2022. doi: 10.1145/3534678.3539472. URL https://doi.org/10.1145/3534678.3539472.

João Loula, Marco Baroni, and Brenden M. Lake. Rearranging the familiar: Testing compositional generalization in recurrent networks. In Tal Linzen, Grzegorz Chrupala, and Afra Alishahi (eds.), *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018,*

Brussels, Belgium, November 1, 2018, pp. 108–114. Association for Computational Linguistics, 2018. doi: 10.18653/v1/w18-5413. URL https://doi.org/10.18653/v1/w18-5413.

Pasquale Minervini, Matko Bosnjak, Tim Rocktäschel, Sebastian Riedel, and Edward Grefenstette. Differentiable reasoning on large knowledge bases and natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 5182–5190. AAAI Press, 2020. URL https://ojs.aaai.org/index.php/AAAI/article/view/5962.

Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving. In Pascal Hitzler and Md. Kamruzzaman Sarker (eds.), *Neuro-Symbolic Artificial Intelligence: The State of the Art*, volume 342 of *Frontiers in Artificial Intelligence and Applications*, pp. 280–293. IOS Press, 2021. doi: 10.3233/FAIA210359. URL https://doi.org/10.3233/FAIA210359.

Hongyu Ren and Jure Leskovec. Beta embeddings for multi-hop logical reasoning in knowledge graphs. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e43739bba7cdb577e9e3e4e42447f5a5-Abstract.html.

Hongyu Ren, Weihua Hu, and Jure Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BJgr4kSFDS.

Hongyu Ren, Hanjun Dai, Bo Dai, Xinyun Chen, Denny Zhou, Jure Leskovec, and Dale Schuurmans. SMORE: knowledge graph completion and multi-hop reasoning in massive knowledge graphs. In Aidong Zhang and Huzefa Rangwala (eds.), *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*, pp. 1472–1482. ACM, 2022. doi: 10.1145/3534678.3539405. URL https://doi.org/10.1145/3534678.3539405.

Leonardo F. R. Ribeiro, Martin Schmitt, Hinrich Schütze, and Iryna Gurevych. Investigating pretrained language models for graph-to-text generation. In *Proceedings of the 3rd Workshop on Natural Language Processing for Conversational AI*, pp. 211–227, Online, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.nlp4convai-1.20. URL https://aclanthology.org/2021.nlp4convai-1.20.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 3788–3800, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/b2ab001909a8a6f04b51920306046ce5-Abstract.html.

Haitian Sun, Andrew O. Arnold, Tania Bedrax-Weiss, Fernando Pereira, and William W. Cohen. Faithful embeddings for knowledge base queries. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/fe74074593f21197b7b7be3c08678616-Abstract.html.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pp. 1556–1566. The Association for Computer Linguistics, 2015. doi: 10.3115/v1/p15-1150. URL https://doi.org/10.3115/v1/p15-1150.

Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In Alexandre Allauzen, Edward Grefenstette, Karl Moritz Hermann, Hugo Larochelle, and Scott Wen-tau Yih (eds.), *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality, CVSC 2015, Beijing, China, July 26-31, 2015*, pp. 57–66. Association for Computational Linguistics, 2015. doi: 10.18653/v1/W15-4007. URL https://doi.org/10.18653/v1/W15-4007.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Zihao Wang, Hang Yin, and Yangqiu Song. Benchmarking the combinatorial generalizability of complex query answering on knowledge graphs. In Joaquin Vanschoren and Sai-Kit Yeung (eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/7eabe3a1649ffa2b3ff8c02ebfd5659f-Abstract-round2.html.

Zihao Wang, Yangqiu Song, Ginny Y. Wong, and Simon See. Logical message passing networks with one-hop inference on atomic formulas. *CoRR*, abs/2301.08859, 2023. doi: 10.48550/arXiv.2301.08859. URL https://doi.org/10.48550/arXiv.2301.08859.

Zhaohan Xi, Ren Pang, Changjiang Li, Tianyu Du, Shouling Ji, Fenglong Ma, and Ting Wang. Reasoning over multi-view knowledge graphs. *CoRR*, abs/2209.13702, 2022. doi: 10.48550/arXiv.2209.13702. URL https://doi.org/10.48550/arXiv.2209.13702.

Zezhong Xu, Wen Zhang, Peng Ye, Hui Chen, and Huajun Chen. Neural-symbolic entangled framework for complex query answering. *CoRR*, abs/2209.08779, 2022. doi: 10.48550/arXiv.2209.08779. URL https://doi.org/10.48550/arXiv.2209.08779.

Dong Yang, Peijun Qing, Yang Li, Haonan Lu, and Xiaodong Lin. Gammae: Gamma embeddings for logical queries on knowledge graphs. *CoRR*, abs/2210.15578, 2022. doi: 10.48550/arXiv.2210.15578. URL https://doi.org/10.48550/arXiv.2210.15578.

Pengcheng Yin and Graham Neubig. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In Eduardo Blanco and Wei Lu (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pp. 7–12. Association for Computational Linguistics, 2018. doi: 10.18653/v1/d18-2002. URL https://doi.org/10.18653/v1/d18-2002.

Zhanqiu Zhang, Jie Wang, Jiajun Chen, Shuiwang Ji, and Feng Wu. Cone: Cone embeddings for multi-hop reasoning over knowledge graphs. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 19172–19183, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/a0160709701140704575d499c997b6ca-Abstract.html.

Zhaocheng Zhu, Mikhail Galkin, Zuobai Zhang, and Jian Tang. Neural-symbolic models for logical queries on knowledge graphs. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 27454–27478, 2022.

A Sampling Algorithm

In this section, we introduce the algorithm used for sampling the complex queries from a given knowledge graph. The detailed algorithm is described in Algorithm 2. For a given knowledge Graph G and a query

type t, we start with a random node v to reversely find a query that has answer v with the corresponding structure t. Basically, this process is conducted in a recursion process. In this recursion, we first look at the last operation in this query. If the operation is *projection*, we randomly select one of its predecessors u that holds the corresponding relation to v as the answer of its sub-query. Then we call the recursion on node u and the sub-query type of t again. Similarly, for *intersection* and *union*, we will apply recursion on their sub-queries on the same node v. The recursion will stop when the current node contains an anchor entity.

Algorithm 2 Ground Query Type

Require: G is a knowledge graph.
  function GROUNDTYPE(T, v)
    T is an arbitrary node of the computation graph.
    v is an arbitrary knowledge graph vertex
    if $T.operation = p$ **then**
      $u \leftarrow$ SAMPLE($\{u|(u, v)$is an edge in $G\}$)
      $RelType \leftarrow$ type of (u, v) in G
      $ProjectionType \leftarrow p$
      $SubQuery \leftarrow$ GROUNDTYPE($T.child, u$)
      return $(ProjectionType, RelType, SubQuery)$
    else if $T.operation = i$ **then**
      $IntersectionResult \leftarrow (i)$
      for $child \in T.Children$ **do**
        $SubQuery \leftarrow$ GROUNDTYPE($T.child, v$)
        $IntersectionResult$.PUSHBACK($child, v$)
      end for
      return $IntersectionResult$
    else if $T.operation = u$ **then**
      $UnionResult \leftarrow (u)$
      for $child \in T.Children$ **do**
        if $UnionResult.length > 2$ **then**
          $v \leftarrow$ SAMPLE(G)
        end if
        $SubQuery \leftarrow$ GROUNDTYPE($T.child, v$)
        $UnionResult$.PUSHBACK($child, v$)
      end for
      return $UnionResult$
    else if $T.operation = e$ **then**
      return $(e, T.value)$
    end if
  end function

Table 7: The basic information about the three knowledge graphs used for the experiments, and their standard training, validation, and testing edges separation according to Ren & Leskovec (2020).

Dataset	Relations	Entities	Training	Validation	Testing	All Edges
FB15k	1,345	14,951	483,142	50,000	59,071	592,213
FB15k-237	237	14,505	272,115	17,526	20,438	310,079
NELL995	200	63,361	114,213	14,324	14,267	142,804

Table 8: Seen and Unseen Query Types in Conjunctive Queries.

Conjunctive Queries	**Number of Types**	**Query Formula**	**Query Depth**
In-Distribution	12	(p,(e))	1
		(p,(p,(e)))	2
		(p,(p,(p,(e))))	3
		(p,(i,(p,(e)),(p,(e))))	2
		(p,(i,(p,(e)),(p,(p,(e)))))	3
		(p,(i,(p,(p,(e))),(p,(p,(e)))))	3
		(i,(p,(e)),(p,(e)))	1
		(i,(p,(e)),(p,(p,(e))))	2
		(i,(p,(e)),(p,(p,(p,(e)))))	3
		(i,(p,(p,(e))),(p,(p,(e))))	2
		(i,(p,(p,(e))),(p,(p,(p,(e)))))	3
		(i,(p,(p,(p,(e)))),(p,(p,(p,(e)))))	3
Out-of-Distribution	3	(i,(i,(p,(e)),(p,(p,(p,(e))))),(p,(p,(e))))	3
		(i,(i,(p,(e)),(p,(p,(e)))),(p,(p,(p,(e)))))	3
		(i,(i,(p,(p,(e))),(p,(p,(p,(e))))),(p,(p,(e))))	3

Table 9: Seen and Unseen Query Types in First-Order Logic Queries.

FOL Queries	Number of Types	Query Formula	Query Depth
In-Distribution	29	(p,(e))	1
		(p,(p,(e)))	2
		(p,(p,(p,(e))))	3
		(p,(i,(p,(e)),(p,(e))))	2
		(p,(i,(p,(e)),(p,(p,(e)))))	3
		(p,(i,(n,(p,(e))),(p,(e))))	2
		(p,(i,(p,(p,(e))),(p,(p,(e)))))	3
		(p,(i,(n,(p,(e))),(p,(p,(e)))))	3
		(p,(u,(p,(e)),(p,(e))))	2
		(p,(u,(p,(e)),(p,(p,(e)))))	3
		(p,(u,(p,(p,(e))),(p,(p,(e)))))	3
		(i,(p,(e)),(p,(e)))	1
		(i,(p,(e)),(p,(p,(e))))	2
		(i,(p,(e)),(p,(p,(p,(e)))))	3
		(i,(n,(p,(e))),(p,(e)))	1
		(i,(n,(p,(p,(e)))),(p,(e)))	2
		(i,(p,(p,(e))),(p,(p,(e))))	2
		(i,(p,(p,(e))),(p,(p,(p,(e)))))	3
		(i,(n,(p,(e))),(p,(p,(e))))	2
		(i,(n,(p,(p,(e)))),(p,(p,(e))))	2
		(i,(p,(p,(p,(e)))),(p,(p,(p,(e)))))	3
		(i,(n,(p,(e))),(p,(p,(p,(e)))))	3
		(i,(n,(p,(p,(e)))),(p,(p,(p,(e)))))	3
		(u,(p,(e)),(p,(e)))	1
		(u,(p,(e)),(p,(p,(e))))	2
		(u,(p,(e)),(p,(p,(p,(e)))))	3
		(u,(p,(p,(e))),(p,(p,(e))))	2
		(u,(p,(p,(e))),(p,(p,(p,(e)))))	3
		(u,(p,(p,(p,(e)))),(p,(p,(p,(e)))))	3
Out-of-Distribution	29	(i,(i,(p,(e)),(p,(p,(p,(e))))),(p,(p,(e))))	3
		(u,(p,(e)),(p,(i,(n,(p,(e))),(p,(e)))))	2
		(p,(u,(i,(n,(p,(e))),(p,(e))),(p,(e))))	2
		(i,(n,(p,(e))),(p,(u,(p,(e)),(p,(p,(e))))))	3
		(p,(i,(p,(e)),(u,(p,(p,(e))),(p,(p,(e))))))	3
		(i,(p,(p,(p,(e)))),(p,(u,(p,(e)),(p,(p,(e))))))	3
		(u,(i,(n,(p,(e))),(p,(p,(p,(e))))),(p,(p,(p,(e)))))	3
		(i,(i,(p,(e)),(p,(p,(e)))),(p,(p,(p,(e)))))	3
		(i,(n,(u,(p,(e)),(p,(e)))),(p,(p,(e))))	2
		(u,(i,(p,(e)),(p,(p,(e)))),(p,(p,(e))))	2
		(i,(p,(e)),(u,(p,(p,(p,(e)))),(p,(p,(p,(e))))))	3
		(p,(i,(i,(n,(p,(e))),(p,(p,(e)))),(n,(p,(e)))))	3
		(u,(i,(p,(p,(e))),(p,(p,(p,(e))))),(p,(p,(p,(e)))))	3
		(p,(u,(i,(p,(e)),(p,(p,(e)))),(p,(p,(e)))))	3
		(i,(i,(p,(p,(e))),(p,(p,(p,(e))))),(p,(p,(e))))	3
		(i,(n,(p,(p,(e)))),(p,(i,(p,(e)),(p,(e)))))	2
		(i,(p,(p,(e))),(u,(p,(e)),(p,(e))))	2
		(i,(p,(e)),(u,(p,(p,(e))),(p,(p,(p,(e))))))	3
		(u,(i,(n,(p,(e))),(p,(p,(p,(e))))),(p,(p,(e))))	3
		(i,(i,(p,(e)),(p,(p,(p,(e))))),(n,(p,(p,(e)))))	3
		(u,(i,(p,(p,(e))),(p,(p,(e)))),(p,(p,(p,(e)))))	3
		(i,(i,(p,(p,(e))),(p,(p,(p,(e))))),(n,(p,(p,(e)))))	3
		(u,(i,(p,(e)),(p,(e))),(p,(p,(e))))	2
		(u,(p,(i,(n,(p,(e))),(p,(p,(e))))),(p,(p,(e))))	3
		(i,(p,(e)),(p,(i,(n,(p,(e))),(p,(p,(e))))))	3
		(u,(p,(p,(e))),(p,(u,(p,(p,(e))),(p,(p,(e))))))	3
		(i,(n,(p,(e))),(u,(p,(p,(e))),(p,(p,(e)))))	2
		(p,(i,(n,(p,(e))),(u,(p,(e)),(p,(p,(e))))))	3
		(i,(n,(i,(n,(p,(e))),(p,(e)))),(p,(p,(p,(e)))))	3