Latent Execution-Guided Reasoning for Multi-Hop Question Answering on Knowledge Graphs

Hongyu Ren[1], **Hanjun Dai**[2], **Bo Dai**[2], **Xinyun Chen**[3], **Jure Leskovec**[1], **Denny Zhou**[2]
[1] Stanford University, {hyren,jure}@cs.stanford.edu
[2] Google Research, Brain Team, {hadai,bodai,dennyzhou}@google.com
[3] UC Berkeley, xinyun.chen@berkeley.edu

Abstract

Answering complex questions on knowledge graphs (KGQA) is a challenging task. It requires reasoning with both the input natural language questions as well as a massive, incomplete heterogeneous KG. Prior methods obtain an abstract structured query graph/tree from the input question and traverse the KG for answers following the query tree. However, they inherently cannot deal with missing links in the KG. Here we present LEGO, a Latent Execution-Guided reasOning framework with key insights: *execution-guided query synthesis* and *embedding-based query execution*. Our framework iteratively (1) synthesizes a reasoning action and grows the query tree, *e.g.*, extend one branch by a relation traversal or take intersections of multiple branches; and (2) executes the new reasoning action in the latent embedding space. The query synthesis adaptively infers the new interpretable reasoning action, guided by the past execution of the query tree on the KG; and the embedding-based query execution is naturally robust to incomplete KG. Experimental results on the MetaQA benchmark demonstrates the effectiveness of our framework compared with previous state of the art.

1 Introduction

Knowledge graphs (KGs) capture and encode knowledge/facts as triples, e.g., (Sacramento, Capital, California). Prominent examples include Freebase [1], Yago [2], NELL [3], ConceptNet [4]. These real-world KGs are massive, noisy and contain missing links. Answering complex, multi-hop natural language questions on these incomplete KGs (KGQA) is a challenging and fundamental task in artificial intelligence [5, 6, 7, 8, 9, 10]. It requires learning a structured representation to bridge the gap between natural language and entities/relations on a KG, as well as a robust multi-hop reasoning algorithm to efficiently locate the answer entities under missing links and data.

Previous KGQA models [11, 12, 13, 14, 15] first synthesize a complete tree-structured query by parsing the questions and then execute the query tree, which traverses the KG for answers. However, their performance is hindered by three major challenges. The first challenge is that the synthesis process and the execution process are separate and disjoint, leading to a large search space for query synthesis, currently addressed by KG context-free beam search [11, 12]. Another challenge is the scale of KG. Real-world KGs often have millions of entities and multi-hop traversal on a KG leads to an exponential growth in computation time and space. Finally, since KGs are often noisy and incomplete, executing even the ground truth query tree may still not return the full answer set.

Recent knowledge graph embeddings [16, 17, 18] have shown promise in handling missing links, *i.e.*, one-hop reasoning, where they embed entities and relations in a vector space so that missing links can be imputed. Another line of work [19, 20, 21] further proposes to embed complex logical queries by designing neural logical operators, which allows for multi-hop reasoning and execution of logical queries in the embedding space. This line of work enables a scalable and robust reasoning/execution

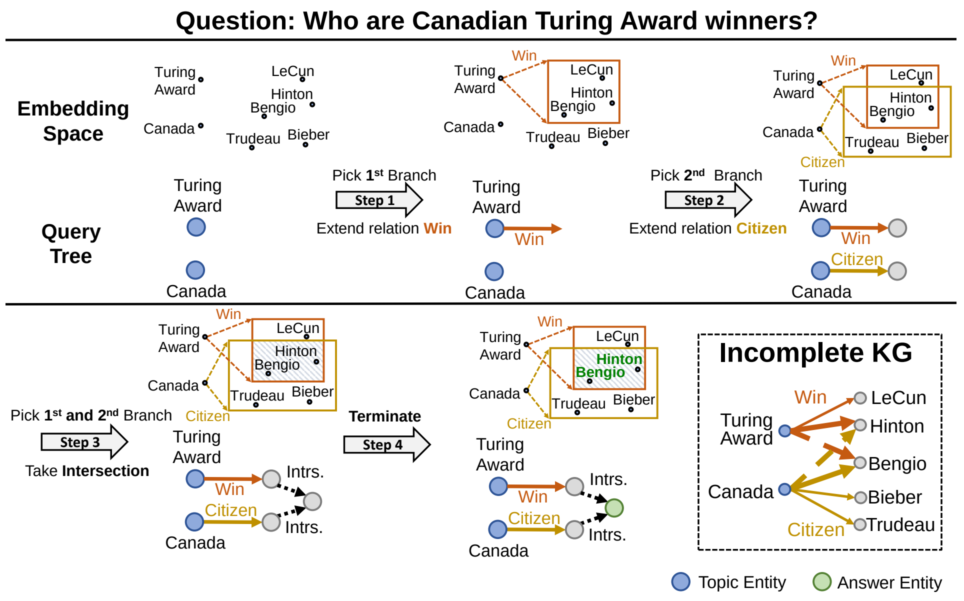

Figure 1: LEGO answers a question by iteratively perform execution-guided query synthesis and embedding-based query execution. LEGO is robust to incomplete KGs, whereas direct traversal will not return all the answers given the incomplete KG (dashed lines denote missing links).

paradigm where answering a structured query is reduced to a K-nearest neighbor search of entities that are close to the query embedding in the vector space. However, such execution relies on the presence of structured logical formulas and how to generalize this line of work to take *natural language question* as input remains unexplored.

Here we propose LEGO, a KGQA framework that synthesizes a query tree for a given question and executes the structured query in a latent embedding space. Our framework consists of a query synthesis module and a KG embedding module, which we build upon Query2box (Q2B) [20]. In Q2B, a query is represented as a hyper-rectangle (box) in the Euclidean space. The key insight of reasoning in our framework is that we iteratively synthesize the query tree and execute it in the embedding space, and the two processes are mutually dependent, *i.e.*, *execution-guided query synthesis* as well as *embedding-based query execution*. Concretely, given a question, we start with the topic entities (initial query tree) and the entity embeddings (initial query embedding). At each step, the synthesis module infers the next reasoning action based on the question embedding (obtained by pretrained language models) as well as the current query embedding; then the KG embedding module executes this new reasoning action in the embedding space and updates the query embedding as well as the query tree accordingly. Our framework naturally addresses the three challenges, where the synthesis step is guided by the execution/embedding of the current query tree; it is scalable, with linear computation complexity with respect to the size of the query tree, as well as robust against missing edges in a KG.

As an example shown in Figure 1, if we aim to answer the question “Who are Canadian Turing Award winners?”, we start with the topic entities {“Canada”, “Turing Award”} and initialize the query embedding with the embedding of the topic entities. The query synthesis module will take as input the question embedding as well as the query embedding, and infer the reasoning action, *e.g.*, pick branch “Turing Award” and traverse by relation “Win”. Then the KG embedding module will update the embedding of the first branch by performing relation projection in the embedding space using Q2B. The process is iteratively executed until the query synthesis module outputs “Terminate”, marking the end of the reasoning process. Finally, the answers are those enclosed in the final box embedding of the query.

We evaluate LEGO on the MetaQA benchmark [22], and we demonstrate the effectiveness and scalability of our method in answering questions on a massive, incomplete KG. The framework also achieves explainability and interpretability by providing a query tree, representing the reasoning process.

2 Preliminaries

A knowledge graph (KG) $\mathcal{G}$ consists of a set of entities $\mathcal{V}$ and a set of relations $\mathcal{R}$. Each relation $r \in \mathcal{R}$ is a binary function $r : \mathcal{V} \times \mathcal{V} \to \{\texttt{True}, \texttt{False}\}$ that indicates (directed) edges of the relation r between pairs of entities. Given a question q, we aim to extract its answers by reasoning on $\mathcal{G}$. We assume that the topic entities of the question, *e.g.*, "Canada" and "Turing Award" in Figure 1, are given. For each question q, there exists an underlying query tree corresponding to it, where the leaves are the topic entities, the root represents the answer, and each edge belongs to a relation traversal or a logical operation, *e.g.*, conjunction. We emphasize that in our task, we only have access to a training dataset of (question, answer) pairs *without knowledge of the ground truth query tree*.

3 Framework

Our framework consists of a knowledge embedding module and a query synthesis module, which perform embedding-based execution and execution-guided synthesis respectively. Given an input question q and its topic entities $[e_1, \ldots, e_n]$, we use a pretrained language model [23] to obtain the representation $\mathbf{q}$, and our framework adopts a bottom-up strategy that synthesizes the query tree from the topic entities (leaves in the tree).

3.1 Knowledge Embedding Module

We build upon the Query2box (Q2B) model [20], which embeds a query into a hyper-rectangle (box) in the Euclidean space. Q2B represents each entity e as a point (box with zero offset), and provides two logical operators $\mathcal{P}$ and $\mathcal{I}$ to perform relation projection/traversal and box intersection in the embedding space respectively. See Figure 1 and [20] for more details.

The goal of the knowledge embedding module is to execute a reasoning action given the current query embedding in a vector space. Given a question q and the box embedding of its query tree $\mathbf{g_t} = [\mathbf{b_1^t}, \ldots, \mathbf{b_n^t}]$ at step t where $\mathbf{b_i^t}$ represents the embedding of branch b_i^t of the query tree, the knowledge embedding module takes as input a reasoning action, which is the output of the query synthesis module (detailed in Sec. 3.2). Valid reasoning action includes: (1) relation traversal r of one branch b_i, we perform one relation projection and update the query embedding: $\mathbf{b_i^{t+1}} = \mathcal{P}(\mathbf{b_i^t}, r)$; (2) conjunction of multiple branches $\mathbf{B} \subseteq \{\mathbf{b_i}\}_{i=1}^n$, we use the intersection operator $\mathcal{I}$, merges all the branches in $\mathbf{B}$, and replace it with the output $\mathcal{I}(\mathbf{B})$; (3) termination of the process.

3.2 Query Synthesis Module

Branch(es) Selection. Here we introduce how we model stepwise query synthesis. Given a question q and the box embedding of its query tree $\mathbf{g_t} = [\mathbf{b_1^t}, \ldots, \mathbf{b_n^t}]$ at step t, the goal of query synthesis module is to infer a reasoning action, which first requires selecting one element from the powerset[1] of the branches $\texttt{pset}_t = [\varnothing, \{\mathbf{b_1^t}\}, \ldots, \{\mathbf{b_1^t}, \mathbf{b_2^t}\}, \ldots, \{\mathbf{b_1^t}, \ldots, \mathbf{b_n^t}\}]$: $\varnothing$ represents termination (note only when the query tree has one branch will the model select $\varnothing$, otherwise $\varnothing$ is masked); if only one branch is chosen, we perform traversal of a certain relation, which the module will further infer; if multiple branches are chosen, we take conjunction over the chosen branches. Since each element of $\texttt{pset}_t$ is a set, we design $D(\cdot)$ with an order-invariant DeepSets architecture [24] to extract the representation. For each $\mathbf{B} \in \texttt{pset}_t$, we obtain its representation with $D(\mathbf{B})$. Then we score each set with a final scoring network, which also takes the question embedding $\mathbf{q}$ as input: $\texttt{score}_\mathbf{B} \propto S(D(\mathbf{B}), \mathbf{q})$. See Appendix A for design choice of $D(\cdot)$ and $S(\cdot, \cdot)$.

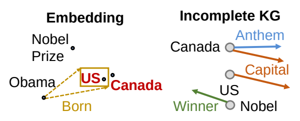

Figure 2: We prune the search space for relation prediction.

Relation Prediction. When we only select a single branch $\mathbf{B} \in [\{\mathbf{b_1^t}\}, \ldots, \{\mathbf{b_n^t}\}]$, we further predict a relation for this branch to traverse (in the embedding space). We use an additional network for relation inference: $R(D(\mathbf{B}), \mathbf{q})$, which outputs a distribution over all the relations $\mathcal{R}$ on the KG. Besides, we design heuristics to reduce the number of valid relations to predict at each step. For example, if the selected branch is "[Obama, [Born]]", then we know the relation we further predict

[1]Nearly all natural language questions have less than 4 branches/topic entities, hence it is tractable.

should be an attribute for *countries*. In this case, we prune the relation sets $\mathcal{R}$ by only predicting from the attributes of entities close to the embedding of the selected branch. As shown in Figure 2, when predicting the next relation for "[Obama, [Born]]", we find "US" and "Canada" are close to the branch embedding (the brown box), then the relation to traverse will be a union of the attributes of "US" and "Canada": {"Anthem", "Capital"}, while we ignore "Winner".

3.3 Iterative Query Synthesis and Question Answering

Inference. Given a question, we start from its topic entities, iteratively synthesize the query tree and execute the query tree in the embedding space via the two modules. At each step, we choose the action with the maximum probability. After the synthesis is terminated, we take the embedding of the final query tree[2] and rank all the entities by the distance between the entity and the query tree in the vector space. See Appendix C for details of distance function and Appendix D for complexity analysis. The score of a query tree for a given question is the average Hits@1 of the answers.

Training. During training, we first pretrain the knowledge embedding module by sampling (query tree, answer) pairs from the given KG, so that we can have a nice initialization for the knowledge embedding module. See Appendix B for details. After pretraining, we train the whole framework using (training question, answer) pairs. We keep a replay buffer for each training question and search query trees online for each question. The replay buffer stores the top-3 ranking query trees and the traces for a given question. We directly optimize a standard supervised loss for the two modules. Although the replay buffer may contain spurious queries/programs, the optimization follows the insights that neural networks will fit the correct label more easily [25].

4 Experiments and Discussions

Dataset. We evaluate LEGO on MetaQA [22], which is a large-scale multi-hop KGQA benchmark dataset with more than 400k questions. The questions in MetaQA span from 1-hop to 3-hop reasoning steps, and can be answered on a given KG. In order to evaluate the robustness against an incomplete KG, following prior work [5], we use the same incomplete KG with only 50% edges.

Baseline. We compare with two state-of-the-art methods: (1) Pullnet [5], which iteratively retrieves a subgraph from the KG starting from the topic entities and obtains answers by ranking entities on the subgraph by graph neural networks; (2) EmbedKGQA [6], which learns a score function between the question embeddings (from Bert [23]) and entity embeddings (from ComplEx [18], a KG embedding method). For sanity check, we also execute the ground truth query tree by (3) traversing on the incomplete KG or (4) using the knowledge embedding (KE) module. More details can be found in the Appendix E.

Hits@1	1-hop	2-hop	3-hop	All
Traverse w/ ground truth query tree	63.3	45.8	45.3	51.5
KE w/ ground truth query tree	70.8	62.1	66.4	66.6
Pullnet [5]	65.1	52.1	59.7	59.2
EmbedKGQA* [6]	**70.6**	54.3	53.5	60.2
LEGO	69.3	**57.8**	**63.8**	**63.8**

Table 1: Hits@1 results of MetaQA on 50% KG. *We rerun EmbedKGQA to guarantee that baselines and ours use the exact same incomplete KG.

Results. We evaluate the accuracy using the Hits@1 metrics. As shown in Table 1, our method achieves comparable results with EmbedKGQA on single-hop questions while outperforms both Pullnet and EmbedKGQA by at least 8% relative Hits@1, demonstrating the effectiveness of LEGO in modeling the reasoning step of multi-hop questions. When given the ground truth query tree, we could directly use our KE module to embed it, which achieves the best result.

Overall, our experiment demonstrates that our framework outperforms previous methods in answering multi-hop questions. And we plan to evaluate on more complex KGQA datasets including ComplexWebQuestions [26], and further consider comparative or superlative constraints in query tree synthesis for questions such as "Who are the latest Canadian Turing Award winners".

[2]We guarantee that only when the query tree $\mathbf{g}$ has a single branch can it select termination.

References

[1] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, “Freebase: a collaboratively created graph database for structuring human knowledge,” in *ACM SIGMOD international conference on Management of data (SIGMOD)*, pp. 1247–1250, ACM, 2008.

[2] F. M. Suchanek, G. Kasneci, and G. Weikum, “Yago: a core of semantic knowledge,” in *Proceedings of the International World Wide Web Conference (WWW)*, pp. 697–706, ACM, 2007.

[3] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, and T. M. Mitchell, “Toward an architecture for never-ending language learning,” in *AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

[4] H. Liu and P. Singh, “Conceptnet—a practical commonsense reasoning tool-kit,” *BT technology journal*, vol. 22, no. 4, pp. 211–226, 2004.

[5] H. Sun, T. Bedrax-Weiss, and W. W. Cohen, “Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text,” in *Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[6] A. Saxena, A. Tripathi, and P. Talukdar, “Improving multi-hop question answering over knowledge graphs using knowledge base embeddings,” in *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[7] A. Bordes, S. Chopra, and J. Weston, “Question answering with subgraph embeddings,” in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[8] K. Xu, S. Reddy, Y. Feng, S. Huang, and D. Zhao, “Question answering on freebase via relation extraction and textual evidence,” in *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016.

[9] M. Yu, W. Yin, K. S. Hasan, C. d. Santos, B. Xiang, and B. Zhou, “Improved neural relation detection for knowledge base question answering,” in *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

[10] C. Liang, J. Berant, Q. Le, K. D. Forbus, and N. Lao, “Neural symbolic machines: Learning semantic parsers on freebase with weak supervision,” in *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

[11] Y. Lan and J. Jiang, “Query graph generation for answering multi-hop complex questions from knowledge bases,” in *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2020.

[12] Z.-Y. Chen, C.-H. Chang, Y.-P. Chen, J. Nayak, and L.-W. Ku, “Uhop: An unrestricted-hop relation extraction framework for knowledge-based question answering,” in *Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019.

[13] S. W.-t. Yih, M.-W. Chang, X. He, and J. Gao, “Semantic parsing via staged query graph generation: Question answering with knowledge base,” in *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015.

[14] J. Bao, N. Duan, Z. Yan, M. Zhou, and T. Zhao, “Constraint-based question answering with knowledge graph,” in *International Conference on Computational Linguistics (COLING)*, 2016.

[15] K. Luo, F. Lin, X. Luo, and K. Zhu, “Knowledge base question answering via encoding of complex query graphs,” in *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[16] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, “Translating embeddings for modeling multi-relational data,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2013.

[17] Z. Sun, Z.-H. Deng, J.-Y. Nie, and J. Tang, “Rotate: Knowledge graph embedding by relational rotation in complex space,” in *International Conference on Learning Representations (ICLR)*, 2019.

[18] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, “Complex embeddings for simple link prediction,” in *International Conference on Machine Learning (ICML)*, 2016.

[19] W. Hamilton, P. Bajaj, M. Zitnik, D. Jurafsky, and J. Leskovec, “Embedding logical queries on knowledge graphs,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[20] H. Ren, W. Hu, and J. Leskovec, “Query2box: Reasoning over knowledge graphs in vector space using box embeddings,” in *International Conference on Learning Representations (ICLR)*, 2020.

[21] H. Ren and J. Leskovec, “Beta embeddings for multi-hop logical reasoning in knowledge graphs,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[22] Y. Zhang, H. Dai, Z. Kozareva, A. J. Smola, and L. Song, “Variational reasoning for question answering with knowledge graph,” in *AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[23] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” in *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

[24] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola, “Deep sets,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[25] D. Arpit, S. Jastrzębski, N. Ballas, D. Krueger, E. Bengio, M. S. Kanwal, T. Maharaj, A. Fischer, A. Courville, Y. Bengio, *et al.*, “A closer look at memorization in deep networks,” in *International Conference on Machine Learning (ICML)*, 2017.

[26] A. Talmor and J. Berant, “The web as a knowledge-base for answering complex questions,” in *Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

Appendix

A Design Choice of the Query Synthesis Module

Here we discuss the details of the architecture design of the $D(\cdot)$ and $S(\cdot, \cdot)$ networks in the query synthesis module. Since D takes a set of branches $\mathbf{B} \in [\varnothing, \{\mathbf{b_1}\}, \ldots, \{\mathbf{b_1}, \mathbf{b_2}\}, \ldots, \{\mathbf{b_1}, \ldots, \mathbf{b_n}\}]$ as input, we adopt an order-invariant DeepSets architecture [24], where we first use a 2-layer MLP to obtain the initial representation for each branch in the set and then use max-pooling, before we use another 2-layer MLP to obtain the final representation for the set of branches. For $\varnothing$, we manually set $D(\varnothing) = \mathbf{0}$. For S, it aims to score a set of branches conditioned on the input question, so we directly concatenate the set representation obtained by D with the Bert embedding $\mathbf{q}$ of the question. After we score all branches in the powerset using D and S, we normalize it with softmax.

B Pretraining Details

Given a KG, we follow the practices of Query2box [20] and synthesize query trees of different structures. As shown in Figure 3, given a query structure, we need to instantiate it for a query tree, where essentially we need to ground the blue nodes (topic entities) and all the edges in the query structure. For instantiation, we adopt a top-down strategy, where we first sample a random node on the KG and treat this node as the green node and iteratively ground the edges by sampling the neighboring edges of the green node. The process is iteratively executed until we have instantiated all the blue nodes. Then we traverse the KG using the query tree for answers, and add this new (query tree, answer) pair to our pretraining dataset.

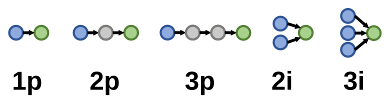

Figure 3: The query structures on which we instantiate grounded queries and pre-train the knowledge embedding module.

C Distance Function

Given the final query tree with a single branch $\mathbf{g} = [\mathbf{b}]$, we define the distance between $\mathbf{b}$ and an entity embedding $\mathbf{v}$ on KG using the box distance as in Query2box [20]. Here $\mathbf{b}$ is a box with center and offset, and $\mathbf{v}$ is a single point in the embedding space.

$$\begin{aligned}
\mathrm{dist}_{\mathrm{box}}(\mathbf{v}; \mathbf{b}) &= \mathrm{dist}_{\mathrm{outside}}(\mathbf{v}; \mathbf{b}) + \alpha \cdot \mathrm{dist}_{\mathrm{inside}}(\mathbf{v}; \mathbf{b}), \\
\mathrm{dist}_{\mathrm{outside}}(\mathbf{v}; \mathbf{b}) &= \|\mathrm{Max}(\mathbf{v} - \mathbf{b}_{\mathrm{max}}, \mathbf{0}) + \mathrm{Max}(\mathbf{b}_{\mathrm{min}} - \mathbf{v}, \mathbf{0})\|_1, \\
\mathrm{dist}_{\mathrm{inside}}(\mathbf{v}; \mathbf{b}) &= \|\mathrm{Cen}(\mathbf{b}) - \mathrm{Min}(\mathbf{b}_{\mathrm{max}}, \mathrm{Max}(\mathbf{b}_{\mathrm{min}}, \mathbf{v}))\|_1.
\end{aligned}$$

where $\mathbf{b}_{\mathrm{max}} = \mathrm{Cen}(\mathbf{b}) + \mathrm{Off}(\mathbf{b}) \in \mathbb{R}^d$, $\mathbf{b}_{\mathrm{min}} = \mathrm{Cen}(\mathbf{b}) - \mathrm{Off}(\mathbf{b}) \in \mathbb{R}^d$ and $0 < \alpha < 1$ is a fixed scalar and we used 0.02 in our experiments.

	Train	Dev	Test
1-hop	96,106	9,992	9,947
2-hop	118,980	14,872	14,872
3-hop	114,196	14,274	14,274

Table 2: Statistics of MetaQA

D Complexity Analysis

Given a KG $\mathcal{G}$, with $|\mathcal{V}|$ number of entities and the maximum degree $\Delta(\mathcal{G})$, and a k-hop question, we list below the worst case asymptotic complexity of traversing $\mathcal{G}$ following the structured query as well as embedding the structured query. For traversal, the complexity is $\min(\mathcal{O}(\Delta(\mathcal{G})^k), \mathcal{O}(k|\mathcal{V}|^2))$ since they need to track and model all the intermediate entities; while the complexity of embedding-based methods is $\mathcal{O}(k + |\mathcal{V}|)$, linear with respect to the number of hops and the number of entities on $\mathcal{G}$.

E Experimental Details

We experimented on MetaQA [22], the statistics of the dataset can be found in Table 2.

For all the baselines and our method, we use the same pretrained case-insensitive 768 dimensional Bert embedding (without finetuning) [23] to obtain the question representation for fair comparison.