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

Hongyu Ren [1] **Hanjun Dai** [2] **Bo Dai** [2] **Xinyun Chen** [3] **Michihiro Yasunaga** [1] **Haitian Sun** [4] **Dale Schuurmans** [2] **Jure Leskovec** [1] **Denny Zhou** [2]

Abstract

Answering complex natural language questions on knowledge graphs (KGQA) is a challenging task. It requires reasoning with 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 **L**atent **E**xecution-**G**uided reas**O**ning framework to handle this challenge in KGQA. LEGO works in an iterative way, which alternates between (1) a Query Synthesizer, which synthesizes a reasoning action and grows the query tree step-by-step, and (2) a Latent Space Executor that executes the reasoning action in the latent embedding space to combat against the missing information in KG. To learn the synthesizer without step-wise supervision, we design a generic latent execution guided bottom-up search procedure to find good execution traces efficiently in the vast query space. Experimental results on several KGQA benchmarks demonstrate the effectiveness of our framework compared with previous state of the art.

1. Introduction

Answering complex natural language questions with multi-hop reasoning steps on incomplete knowledge graphs (KGs) is a fundamental, yet challenging task (KGQA) (Sun et al., 2019a; Saxena et al., 2020; Bordes et al., 2014; Xu et al., 2016; Yu et al., 2017; Liang et al., 2017). KGQA takes a question in natural language as an input, with the goal to identify a set of KG entities that form the answer. There are

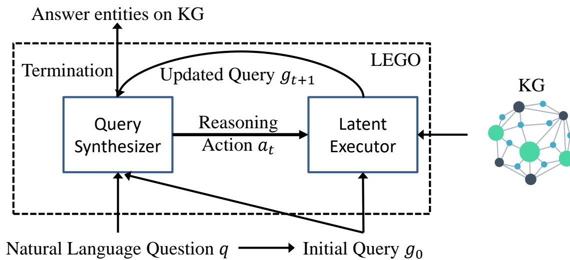

Figure 1. LEGO answers a question by iteratively performing execution-guided query synthesis and latent query execution.

two important subproblems that need to be addressed: (1) bridging the gap between natural language question and the entities in the KG as structured query needs to be generated and (2) a robust reasoning algorithm that efficiently locates the answer entities with missing links in KGs.

Previous neural KGQA models that are based on semantic parsing (Lan & Jiang, 2020; Chen et al., 2019b; Yih et al., 2015; Bao et al., 2016; Luo et al., 2018) first synthesize a tree-structured query by parsing the questions, and then execute the query on KG, *i.e.*, traverse the KG for answers. Compared with non-parsing methods (Sun et al., 2019a; Saxena et al., 2020), these models achieve better empirical results and interpretability since the underlying structured query captures the reasoning process. However, the performance of these parsing-based methods is hindered by three major challenges. The first challenge is the scale and incomplete nature of KGs. Real-world KGs (Bollacker et al., 2008; Suchanek et al., 2007; Liu & Singh, 2004) often have millions of entities, and multi-hop traversal on a KG leads to an exponential growth in computation time and space. Since KGs are often noisy and incomplete, executing even the ground truth query may still not return the fully correct answer set. Another challenge is that the query synthesis process and the execution process are separate and disjoint, leading to a combinatorial search space, especially for multi-hop reasoning. It is currently addressed by KG context-free beam search (Lan & Jiang, 2020; Chen et al., 2019b). Finally, these parsing-based models mostly require ground truth queries for supervision in training. However, such supervision is hard to obtain in practice.

[1]Stanford University [2]Google Brain [3]UC Berkeley [4]Carnegie Mellon University. Correspondence to: Hongyu Ren <hyren@cs.stanford.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

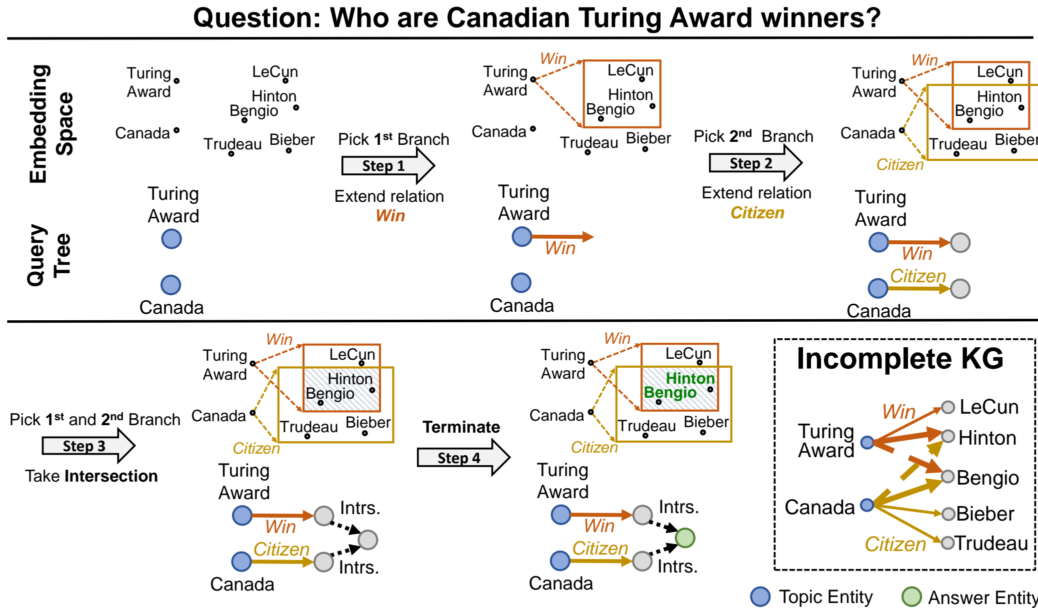

Figure 2. Given an input natural language question, LEGO starts from topic entities and iteratively perform query synthesis and query execution in the latent space. LEGO is robust to incomplete KGs, whereas direct traversal will not return all the answers given the incomplete KG (dashed lines denote missing links).

A recent line of work (Hamilton et al., 2018; Ren et al., 2020; Ren & Leskovec, 2020; Sun et al., 2020) 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 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. For example, Query2box (Q2B) (Ren et al., 2020) embeds queries as hyper-rectangles (box) and the answers as point vectors enclosed in the box. However, such execution relies on the presence of the structured logical queries, and how to generalize this line of work to take a *natural language question* as the input remains unexplored.

In this work, we present LEGO, a Latent Execution-Guided reasOning framework, for KGQA (Figure 1). LEGO consists of a *Latent Space Executor* and a *Query Synthesizer*, which interact iteratively to identify the answer entities in the KG. Given a natural language question, the latent space executor starts with the named entities and their KG embeddings (initial query embedding). Afterwards, at each step, the query synthesizer infers the next reasoning action based on the question embedding (obtained by pretrained language models (Devlin et al., 2019)) as well as the current partial query embedding; then the executor performs this new reasoning action in the latent KG embedding space, and updates the query embedding as well as the query tree accordingly. The proposed LEGO naturally addresses the abovementioned challenges of KGQA: it executes queries in the latent space so that the execution is robust against missing edges; meanwhile, the synthesis becomes a step-by-step procedure guided by the execution context of the current query tree, decomposing the structured output space of queries.

Before we introduce the mechanism to extract training supervision for the executor and the synthesizer, we first illustrate the alternating procedure between these two modules for reasoning in Figure 2. For example, to answer the question "Who are Canadian Turing Award winners?", we start with the entities {"Canada", "Turing Award"} mentioned in the question, and initialize the query embedding with the embedding of the topic entities based on Q2B. The query synthesizer will take as input the question embedding as well as the query embedding, and infer the reasoning action, *e.g.*, pick the branch "Turing Award" and traverse by the relation "Win". Then the executor will update the embedding of the first branch by performing relation projection in the embedding space. The process is iteratively executed until the query synthesizer outputs "Terminate", marking the end of the reasoning process. Finally, the answers are those enclosed in the final box embedding of the query.

The main challenge of applying our approach is that the correct query tree is unknown for a given natural language question. In order to train LEGO, we propose *execution-guided action space pruning* for efficient online searching to obtain the valid execution traces given the questions. We score a candidate query by evaluating whether it is close to the ground truth answers in the embedding space. For each question, we keep a buffer of good candidate queries producing similar answers to the ground truth, although some

spurious queries (noisy candidates) exist. Then we optimize the model to fit the questions and their corresponding candidate queries in the buffer. Inspired by prior findings that deep neural networks can fit the correct labels more easily than the noisy ones (Arpit et al., 2017; Zhang et al., 2017; Liu et al., 2020), we propose stochastic hard EM (Min et al., 2019; Chen et al., 2020b; Liang et al., 2018), *i.e.*, for each question, we only optimize the model with the candidate query with the *lowest* loss in a minibatch. This simple and effective trick demonstrates robustness against the spurious queries during training.

We evaluate LEGO on three large-scale KGQA datasets. LEGO outperforms previous state-of-the-art KGQA methods in answering questions on massive and incomplete KGs. Extensive ablation studies further demonstrate the effectiveness and scalability of multiple components in our method. The implementation of LEGO can be found in `http://github.com/snap-stanford/lego`.

2. Related Work

KGQA. Traditional KGQA methods (Berant et al., 2013; Bast & Haussmann, 2015) use hand-engineered templates to parse the question and synthesize the query (Poon & Domingos, 2009). However, these methods require strong domain knowledge to manually design the set of rules and features to reduce the search space. To overcome the requirement of the rules, recent KGQA methods leverage neural semantic parsing with a deep network (Berant et al., 2013; Lan & Jiang, 2020; Liang et al., 2017; Yih et al., 2015; 2016; Luo et al., 2018; Chen et al., 2019b; Hu et al., 2018; Qiu et al., 2020b). However, these neural semantic parsing methods require a ground truth query for supervision, which is often hard to achieve in the real world. Besides, these methods directly execute the synthesized query on the KG, which is affected by the missing links. Another line of KGQA methods (Xu et al., 2016; Dong et al., 2015; Miller et al., 2016) aim to retrieve information from the KG using RL agent (Xiong et al., 2017; Qiu et al., 2020a; Lin et al., 2018; Das et al., 2018) or graph nets (Sun et al., 2018a; 2019a; Xiong et al., 2019), however, these methods rely on heuristics, *e.g.*, shortest paths between topic entities and answers, most of which are spurious on an incomplete KG. These methods further leverage text corpora to increase performance (Sun et al., 2019a; Das et al., 2017). Another recent line of work uses KG embeddings (Saxena et al., 2020; Sun et al., 2020) to combat the incompleteness of KG. However, they require the prior knowledge of the query structure of the question (Sun et al., 2020) or do not consider the latent reasoning structure of questions (Saxena et al., 2020). Besides, neither can deal with complex questions with multiple topic entities and logical operations, *e.g.*, intersection. Our method designs a latent execution-guided reasoning framework that infers the latent structure of a question as well as reasons in the latent space for multi-hop logical inference.

Structured Representation of Natural Language. While most existing work on semantic parsing requires the ground truth programs for training (Zelle & Mooney, 1996; Zettlemoyer & Collins, 2012; Jia & Liang, 2016; Yu et al., 2018b;a; Keysers et al., 2020), some recent approaches have been proposed to improve the model performance with weak supervision, i.e., the training supervision only contains the ground truth execution results on tabular databases (Liang et al., 2017; Neelakantan et al., 2017; Krishnamurthy et al., 2017; Guu et al., 2017; Liang et al., 2018). However, they assume that the databases are complete, and thus the result is always correct when executing the ground truth program. On the other hand, we need to address the challenge of execution on incomplete KGs, in addition to the weak supervision problem. Besides structured databases, there are other works on learning to synthesize programs for unstructured text understanding, including reading comprehension (Chen et al., 2020b; Amini et al., 2019) and sequence-to-sequence translation (Chen et al., 2020a; Nye et al., 2020b). These works demonstrate better generalization performance than approaches that directly predict the final answers, sometimes without the usage of ground truth programs for training. Again, these works require the access to an oracle executor, and thus the model only needs to synthesize the programs.

Execution-Guided Program Synthesis. Our execution-guided reasoning framework is related to the line of research on execution-guided program synthesis (Odena et al., 2020; Chen et al., 2019a; Wang et al., 2018; Nye et al., 2020a). While most work on neural program synthesis only uses the execution results to select from top candidate programs generated from the beam search, recent works have shown that utilizing the execution results of partial programs improve the synthesizer performance, including text-to-SQL tasks (Wang et al., 2018), synthesis of sequential program statements (Sun et al., 2018b; Zohar & Wolf, 2018; Ellis et al., 2019), and synthesis of programs including nested statements or branches (Odena et al., 2020; Chen et al., 2019a). Different from our work, all these works obtain the execution results from an oracle program executor, rather than learn to execute the programs. When the partial program semantics is too tedious to formally define, or the partial program is even not executable, computing the partial program execution becomes challenging. Therefore, some recent works propose to learn neural networks to represent a program executor, including reading comprehension (Gupta et al., 2019), visual question answering (Andreas et al., 2016), 3D rendering (Tian et al., 2019), and short programs with loops and higher-order functions (Nye et al., 2020a). In this work, our model learns to predict the answer from executing queries on incomplete KGs.

3. Problem Setting

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 relation r between pairs of entities. Given a natural language question q, we aim to extract its answer entities by reasoning on $\mathcal{G}$. Following the standard setting in KGQA (Saxena et al., 2020), we assume that the topic entities of the question, *e.g.*, "Canada" and "Turing Award" in Figure 2, are given and linked to nodes on the KG. For each question q, there exists an underlying query tree corresponding to it, where the nodes represent a set of concrete KG entities and each edge belongs to relation traversal or a logical operation, *e.g.*, conjunction. Among all the nodes of the query tree, each leaf node represents the set of one topic entity and the single root represents the answer set. We emphasize that we only have access to a training dataset of [natural language question, answer entity] pairs *without the knowledge of ground truth queries* in our setting.

4. LEGO Framework

Our LEGO framework consists of a latent space executor (Sec. 4.1) and a query synthesizer (Sec. 4.2), which perform embedding-based execution and context-aware synthesis respectively. Given an input question q and its topic entities $[e_1, \dots, e_n]$, we use a pretrained language model (Devlin et al., 2019; Reimers & Gurevych, 2019) to obtain the representation $\mathbf{q} \in \mathbb{R}^d$, and our framework adopts a bottom-up strategy that synthesizes the query tree from the given topic entities (leaves in the tree). At each step, the query synthesizer infers an action guided by the current partial execution result, and the latent executor further executes this action to grow and update the query tree *in the embedding space*. The context-aware synthesis terminates until the root node is constructed, as shown in Figure 2. The overall training and inference of LEGO can be found in Algs. 1 and 2.

4.1. Latent Space Executor

The latent space executor executes a reasoning action given the current (partial) query tree in a low-dimensional latent space. Given the latent execution results of its partial query tree $\mathbf{g_t} = [\mathbf{b_1^t}, \dots, \mathbf{b_n^t}]$ at step t, where $\mathbf{b_i^t}$ represents the embedding of branch b_i^t of the query tree, the executor runs a reasoning action, which is the output of the query synthesizer (detailed in Sec. 4.2), to expand the query tree.

We use the Query2box (Q2B) model (Ren et al., 2020) for latent space representation, which embeds a query into a hyper-rectangle (box) in the Euclidean space. A box $\mathbf{b} = (\text{Cen}(\mathbf{b}), \text{Off}(\mathbf{b})) \in \mathbb{R}^{2d}$ is defined as the region:

$$\mathbf{b} \equiv \{\mathbf{v} \in \mathbb{R}^d : \text{Cen}(\mathbf{b}) - \text{Off}(\mathbf{b}) \preceq \mathbf{v} \preceq \text{Cen}(\mathbf{b}) + \text{Off}(\mathbf{b})\},$$

where $\text{Cen}(\mathbf{b}) \in \mathbb{R}^d$ is the center of the box, and $\text{Off}(\mathbf{b}) \in \mathbb{R}^d_{\geq 0}$ is the positive offset of the box. Q2B represents each entity $e \in \mathcal{V}$ as a point (box with the zero offset): $\mathbf{e} = (\text{Cen}(\mathbf{e}), \mathbf{0})$, each relation r as an embedding in $\mathbb{R}^{2d}$, and provides two logical operators

$$\mathcal{P} : \mathbb{R}^{2d} \times \mathbb{R}^{2d} \to \mathbb{R}^{2d} \quad \text{and} \quad \mathcal{I} : \mathbb{R}^{2d} \times \cdots \times \mathbb{R}^{2d} \to \mathbb{R}^{2d}$$

to perform relation projection and box intersection in the embedding space respectively. See Figure 2 and Ren et al. (2020) for more details. Note that our latent space executor is agnostic to any specific KG embeddings, and we take Q2B as one example here. Different design choices can be found in Appendix A.

Overall, the latent space executor (LSE) takes a reasoning action a_t and the query embedding $\mathbf{g_t}$ at timestep t as the input, and deterministically outputs the resulting query embedding $\mathbf{g_{t+1}} = \text{LSE}(\mathbf{g_t}, a_t)$, *i.e.*, the executor models the conditional distribution $p(\mathbf{g_{t+1}} | \mathbf{g_t}, a_t)$, which is a Dirac-delta distribution. Valid reasoning actions $a_t \in \mathcal{A}$ include:

(1) $a_t = (\{b_i^t\}, r)$: extension of one branch b_i^t with a relation edge r, this represents one relation projection from the set of entities in b_i^t using r, *e.g.*, step 1 and 2 in Figure 2. The executor updates the i-th component of the query embedding $\mathbf{g_t}$ accordingly: $\mathbf{g_{t+1}}[i] = \mathcal{P}(\mathbf{b_i^t}, \mathbf{r}) = \mathbf{b_i^t} + \mathbf{r}$;

(2) $a_t = (B, -1)$: conjunction of multiple branches $B \subseteq \{b_i^t\}_{i=1}^n, |B| > 1$, this action takes the intersection of the set of entities in each $b \in B$, *e.g.*, step 3 in Figure 2. We use the intersection operator $\mathcal{I}$, remove all embeddings $\mathbf{b_i^t}$ with $b_i^t \in B$ from $\mathbf{g_t}$, and append $\mathbf{b_{\text{int}}^t} = \mathcal{I}(\mathbf{B})$ to the end of $\mathbf{g_t}$;

$$\text{Cen}(\mathbf{b_{\text{int}}^t}) = \sum_i \mathbf{a}_i \odot \text{Cen}(\mathbf{b_i^t}),\ \mathbf{a}_i = \frac{\exp(f_\omega(\mathbf{b_i^t}))}{\sum_j \exp(f_\omega(\mathbf{b_j^t}))},$$
$$\text{Off}(\mathbf{b_{\text{int}}^t}) = \text{Min}(\{\text{Off}(\mathbf{b_i^t})\}_{b_i^t \in B}) \odot \sigma(D_\omega(\mathbf{B})); \qquad (1)$$

(3) $a_t = (\varnothing, -1)$: termination, *e.g.*, step 4 in Figure 2.

Pretraining Latent Space Executor. The objective of the executor is that we can embed the query tree (or any its subquery) such that the query (or subquery) embedding is close to the embedding of the answers (or any intermediate entities). We pretrain the parameters of the latent space executor, including all entity embeddings $\mathbf{e},\ \forall e \in \mathcal{V}$, relation embeddings $\mathbf{r},\ \forall r \in \mathcal{R}$, neural networks f_ω and D_ω used in the operators. The pretraining procedure includes sampling query trees online from the given KG and optimize a contrastive loss with sampled positive and negative answers. Note that this process does not require knowledge of any natural language questions. The pretraining details can be found in Appendix B.

4.2. Query Synthesizer

Given the embedding of the question q, the query synthesizer will act as a controller that infers a series of reasoning actions for the latent space executor to perform reasoning step-by-step. Here we introduce how we model the

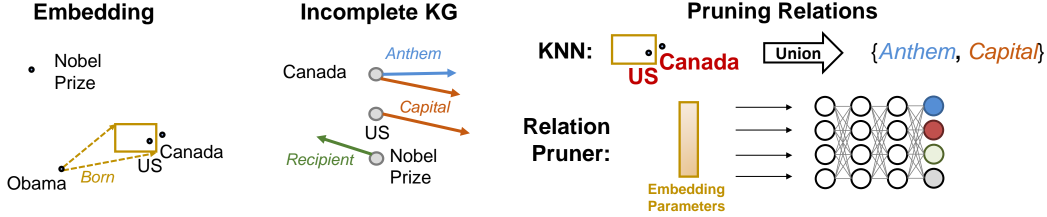

Figure 3. LEGO pretrains a relation pruner that directly outputs a distribution over all relations to prune the search space when predicting the next relation for a branch. In this example, the relation "Receipient" is pruned.

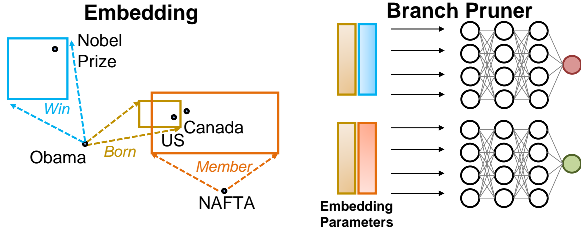

Figure 4. LEGO trains a branch pruner to limit the search space of branch selection. The pruner takes as input a set of branches $\{b_i\}$ and outputs the probability of taking intersection of $\{b_i\}$. The pruner predicts high/low probability, When the branches are similar (brown + orange boxes) or different (brown + cyan boxes).

step-wise query synthesis under the guidance from the execution context, i.e., $p(a_t|\mathbf{g_t}, \mathbf{q})$. Specifically, in t-th step, the latent space executor maintains the partial query tree $\mathbf{g_t} = [\mathbf{b_1^t}, \ldots, \mathbf{b_n^t}]$ as the context. Taking the current context into account, the query synthesizer is parameterized as

$$p_\theta(a_t = (B, r)|\mathbf{g_t}, \mathbf{q}) = p_\theta(r|B, \mathbf{g_t}, \mathbf{q}) p_\theta(B|\mathbf{g_t}, \mathbf{q}), \quad (2)$$

which first selects one or multiple branches and then chooses one of the candidate relations. These actions will be sent to the latent space executor to extend the query tree.

We model $p_\theta(B|\mathbf{g_t}, \mathbf{q})$ with a categorical distribution, *i.e.*, directly choosing from the powerset of the branches $\{\varnothing, \{b_1^t\}, \ldots, \{b_1^t, b_2^t\}, \ldots, \{b_1^t, \ldots, b_n^t\}\}$. Since each choice corresponds to a subset of branches of g_t, we design $D_\theta(\cdot)$ with an order-invariant DeepSets architecture (Zaheer et al., 2017) to obtain the representation. For each $B \subseteq \{b_i^t\}_{i=1}^n$, we obtain its representation with $D_\theta(\mathbf{B})$. Then we parameterize $p_\theta(B|\mathbf{g_t}, \mathbf{q})$ with a scoring network, which also takes the question embedding $\mathbf{q}$ as input:

$$p_\theta(B|\mathbf{g_t}, \mathbf{q}) \propto S_\theta(D_\theta(\mathbf{B}), \mathbf{q}). \quad (3)$$

If only one branch is chosen, *i.e.*, $B \in \{\{b_1^t\}, \ldots, \{b_n^t\}\}$, the query synthesizer further models $p_\theta(r|B, \mathbf{g_t}, \mathbf{q})$ using an additional network for relation inference:

$$p_\theta(r|B, \mathbf{g_t}, \mathbf{q}) \propto R_\theta(D_\theta(\mathbf{B}), \mathbf{q}), \quad (4)$$

which is a distribution over all the relations $\mathcal{R}$ on the KG. If multiple branches are chosen, we take conjunction over the chosen branches, and $p_\theta(r = -1|B, \mathbf{g_t}, \mathbf{q}) = 1$. We use $\varnothing$ to represent the termination (note only when the query tree has exactly one branch will the model select $\varnothing$, otherwise $\varnothing$ is not a valid action and will be masked). See Appendix C for design choice of $D_\theta(\cdot)$, $S_\theta(\cdot, \cdot)$ and $R_\theta(\cdot, \cdot)$.

4.3. Iterative Query Synthesis and Question Answering

Given a question q and the initial query tree g_0, here we introduce how LEGO iteratively synthesizes and embeds the query tree by execution in the embedding space. With the latent space executor and the query synthesizer, we can model all query trees by

$$p_\theta(\mathbf{g_T}|\mathbf{q}) = \int \prod_t p_\theta(a_t|\mathbf{g_t}, \mathbf{q})\, da_0, \ldots, a_{T-1},$$

where T is the maximum allowed size of a query tree. Since the search space is $\mathcal{O}(\#\text{action}^T)$, where $|\#\text{action}| \approx |\mathcal{R}|$ and $|\mathcal{R}|$ can be over 2000 for a KG like Freebase (Bollacker et al., 2008), finding the query tree with maximum likelihood is intractable. We use beam search with beam size k to approximate the best solution. We start from g_0, iteratively synthesize and embed the query tree in the embedding space. At each step t, we choose all the actions with top-k probability based on the query synthesizer and use latent space executor to achieve k updated query trees. After the synthesis is terminated, we take the embedding of the query tree with the maximum probability and rank all the entities on KG by the distance between the entity and the query tree in the vector space. The model selects the entity closest to the query embedding as the final answer to the question. See Appendix D for details of distance function and Appendix E for complexity analysis.

4.4. Module Training

During training of the query synthesizer, we are given a dataset only containing [natural language training question, answer] pairs. To obtain fine-grained query trees for supervising query synthesizer, we propose an *execution-guided* search mechanism that will be explained in detail in Section 5. Suppose at the moment we have a buffer $\mathcal{D}_q$ of potential query trees for each individual natural language training question q. We can rank each candidate query tree in each $\mathcal{D}_q$ by the average distance between the candidate query embedding and the answer embed-

ding. The replay buffer keeps the top-k candidate query trees and the traces for the question q: $\mathcal{D}_q : \{\tau_1, \ldots, \tau_k\}$, where $\tau_j = (\mathbf{g_0}, a_0, \ldots, a_{t_j-1}, \mathbf{g_{t_j}})$ and $\log p_\theta(\tau_j|\mathbf{q}) = \sum_t \log p_\theta(a_t|\mathbf{g_t}, \mathbf{q})$.

Stochastic Hard EM. One can directly optimize a standard supervised loss for the query synthesizer. Since the targeted execution traces are not provided and thus obtained by searching, there may contain spurious queries/programs, which leads to correct answers from incorrect execution paths. These spurious queries do not correspond to the question and confuse the model during training. Inspired by the insights that neural networks will fit the correct label more easily (Arpit et al., 2017; Zhang et al., 2017; Liu et al., 2020), we extend hard EM (Min et al., 2019; Liang et al., 2018) to a stochastic version for more efficient optimization. The idea of Hard EM is that given a question, among *all* candidate queries, Hard EM only optimizes the model to fit the query candidate with the lowest loss. We make it a stochastic version in order to reduce the computation overhead. Specifically, during each iteration, we sample a minibatch of questions of size n_q and also stochastically sample a minibatch of candidate query trees from the replay buffer of size n_c for each sampled question. Then we calculate the loss for each [question, candidate query] pairs and perform a min-pooling over each question, as shown in Eq. 5 below,

$$\ell = \sum_i^{n_q} \min_{j=1,\ldots,n_c} -\log p_\theta(\tau_j^i|\mathbf{q_i}). \tag{5}$$

This simple trick has shown effective in the presence of the noisy labels without additional computation overhead.

5. Execution-Guided Search and Pruning

Here we introduce the execution-guided search to efficiently obtain the candidate query trees for LEGO training. As analyzed in Section 4.3, the search space of a single question is $\mathcal{O}(\#\text{action}^{\#\text{hop}})$, which can be very large ($\mathcal{O}(10^{18})$ as shown in Table 1) when the background KG has a large number of relation types. We introduce learnable $p^B_{\mathbf{g},\mathbf{q}}$ and $p^r_{B,\mathbf{g},\mathbf{q}}$, which will use the partial execution embeddings from the latent space executor to prune the search space of both the branch selection and the relation prediction for collecting valid execution traces.

5.1. Pruning Branch(es) Selection

Although the model can freely choose from the powerset of the branches, yet some branches inherently cannot be selected simultaneously to take intersection regardless of the input question q. As an example shown in Figure 4, if the query tree is $g = [b_1, b_2]$, where $b_1 =$ [Obama, [Born]] and $b_2 =$ [Obama, [Win]], then the model should not take the intersection of the two branches because the b_1 represents a set of countries while the b_2 represents a set of awards. Thus, we further introduce a branch pruner f_ϕ to evaluate whether a given set of branches can be intersected during candidate query searching. Given the pretrained latent space executor, the pruner takes multiple branch embeddings as input and outputs a single scalar as the score $f_\phi(\mathbf{B}) \in (0, 1)$, when $B \subseteq \{b_i\}_{i=1}^n, |B| > 1$; when $|B| = 1$, we assume $f_\phi(\mathbf{B}) = 1$. In order to prune the search space, we introduce a threshold S, only B with $f_\phi(\mathbf{B}) > S$ will it be selected.

$$p^B_{\mathbf{g},\mathbf{q}} \propto p_\theta(B|\mathbf{g}, \mathbf{q}) \cdot 1[f_\phi(\mathbf{B}) > S] \tag{6}$$

Note the pruner f_ϕ is *independent of the questions*, it only reflects whether two given branches should possibly be intersected or not.

Pretraining Branch Pruner f_ϕ. We pretrain the branch pruner f_ϕ by sampling positive and negative branch sets from the KG. A positive branch set represents a set of branches with common answers on the KG, which means that it makes sense for the branches in the set to take intersection. On the contrary, branches in a negative branch set do not share answers, which means they should not be intersected with high probability. For sampling positive set, we randomly pick one node on the KG as the common answer, and reverse-construct the branches. For sampling negative set, we directly randomly sample a set of branches on the KG for simplicity. Note the branches sampled in pretraining do not correspond to any natural language question.

Algorithm 1 Latent Execution-Guided Reasoning (Training)

Input: KG $\mathcal{G} = (\mathcal{V}, \mathcal{R})$, training dataset $\{(q, a)\}_{\text{train}}$, an empty buffer $\mathcal{D}$ for each question.

Training:

(1) Pretrain latent space executor, including entity and relation embedding $\{\mathbf{e}\}_{e\in\mathcal{V}}$, $\{\mathbf{r}\}_{r\in\mathcal{V}}$, f_ω and D_ω (Eq. 1).

(2) Pretrain branch f_ϕ and relation pruner p_ϕ (Eqs. 6, 7)

(3) Search candidate queries for training questions q with latent execution-guided pruning. Record top ranking query tree g and traces τ in the buffer $\mathcal{D}$.

(4) Train synthesizer using the candidate queries with stochastic hard EM, including D_θ, R_θ and S_θ (Eqs. 3, 4).

5.2. Pruning Relation Prediction

We also design a learnable heuristic to reduce the number of valid relations in each step of each target execution trace based on the embedding of the selected branch.

One empirical observation is the valid next-step relations for a query branch usually take the union of the relations associated with the "similar" entities to the current branch on the KG. For example, as shown in Figure 3, 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 be predicted should be a union of the attributes of "US" and "Canada": {"Anthem", "Capital"} in most cases, while we should ignore "Receip-

Algorithm 2 Latent Execution-Guided Reasoning (Inference)

Input: KG, question q with embedding $\mathbf{q}$ and its topic entities $[e_1, \ldots, e_n]$, initial query tree $\mathbf{g_0} = [\mathbf{e_1}, \ldots, \mathbf{e_n}]$, executor, synthesizer, (assume beam size equals 1).
Inference:
repeat
(1) Synthesizer infers the reasoning action $a_t = \text{argmax}_a\, p_\theta(a|\mathbf{g_t}, \mathbf{q})$ (Eq. 2).
(2) Executor executes this reasoning action in the latent space and updates the query tree $g_{t+1} = \text{LSE}(\mathbf{g_t}, \mathbf{a_t})$.
until Synthesizer outputs "Termination".
(3) Calculate the distance between the final query tree $\mathbf{g}$ and the KG entities $\mathbf{e}$.

ient", which is disjoint and irrelevant no matter what the natural language question is. So one simple way to prune the search space of relations is prioritizing those associated with the similar entities, measured by the distance between the current branch and the entities in the embedding space.

However, one major limitation of this method is that it requires a nearest neighbor search at each step for relation prediction, and the computation complexity is linear with respect to the number of entities $|\mathcal{V}|$ on KG in the worst case. To address the limitation, we further train a neural network to prune the relations given the selected branch. The relation pruner takes as input the embedding of the selected branch b and outputs a distribution over all the relations $p_\phi(r|\mathbf{b})$. When predicting the relation for next step, we only look at the top-k relations, *i.e.*,

$$p^r_{\{b\},\mathbf{g},\mathbf{q}} \propto p_\theta(r|\{b\}, \mathbf{g}, \mathbf{q}) \cdot 1[r \in \arg\text{top}\, k\, p_\phi(r|\mathbf{b})] \quad (7)$$

This relation pruner leverages an important inductive bias for generalization: similar branch embedding may represent a similar set of entities. We emphasize that the relation pruner also does not rely on natural language questions.

Pretraining Relation Pruner p_ϕ. We pretrain the relation pruner by sampling [synthetic query, relation] pairs from the KG. Specifically, we first sample a random node on KG as answer and then use this answer to instantiate a multi-hop query in a top-down fashion. Then we take the union of the relations associated with this answer as the positive classes and the others as the negative classes, and we optimize a multi-label classification loss. Note again that this pretraining process is general and the sampled synthetic queries do not correspond to natural language questions. See Appendix F for more details on pretraining pruners.

6. Experiments

We evaluate LEGO on three large-scale multi-hop KGQA benchmark datasets. We show that (1) our LEGO framework outperforms state-of-the-art KGQA methods on these datasets; (2) execution-guided pruning using pretrained branch and relation pruner significantly reduces the search space with high recall of the hidden ground truth candidate query trees; (3) optimization using the stochastic hard EM demonstrates robustness against noisy candidate query trees.

Table 1. Statistics of the KG for the three datasets.

	Entities	Relations	Observed Edges	Missing Edges
MetaQA	43,234	18	134,574	133,680
WQSP (30%)	158,674	1,632	451,634	1,053,800
WQSP (50%)	158,674	1,632	752,717	752,717
CWQ	409,829	1,836	2,151,671	2,151,671

Table 2. Statistics of the three datasets.

	Train	Dev	Test
MetaQA-1hop	96,106	9,992	9,947
MetaQA-2hop	118,980	14,872	14,872
MetaQA-3hop	114,196	14,274	14,274
WQSP	2,848	250	1,639
CWQ	27,623	3,518	3,531

6.1. Experimental Setup

Datasets. We evaluate LEGO on three large-scale multi-hop KGQA benchmark datasets: MetaQA (Zhang et al., 2018), WebQuestionsSP (WQSP) (Yih et al., 2015) and ComplexWebQuestions (CWQ) (Talmor & Berant, 2018). The questions in MetaQA span from 1-hop to 3-hop path-like reasoning steps, and can be answered on the given KG based on WikiMovies (Miller et al., 2016). In order to evaluate the robustness against an incomplete KG, following prior work (Sun et al., 2019a), we use the incomplete KG with only 50% edges. Both WQSP and CWQ contain natural language questions that can be answered using the Freebase KG. The questions may contain multiple topic entities and constraints so that the reasoning process requires intersection operations. We prepare a KG subgraph of Freebase with 150k and 400k entities for WQSP and CWQ respectively, containing relevant entities and their 3-hop neighbors with mentioned relation types in the datasets. We further randomly drop 50% of the edges as the incomplete version. Note that we make sure that if we traverse the full (100%) KG using the ground truth query tree, we can answer all questions. The statistics of the KG and the datasets can be found in Table 1 and 2.

Baselines. We compare with three state-of-the-art methods: **(1) Pullnet** (Sun et al., 2019a), which iteratively retrieves a subgraph from the KG starting from the topic entities and obtains answers by ranking entities on the subgraph by graph nets (Sun et al., 2018a); **(2) EmbedKGQA** (Saxena et al., 2020), which learns a score function between the question embeddings and the entity embeddings from a learned KG embedding method; **(3) EMQL** (Sun et al., 2020), which learns a dense and a sparse representation of the query and uses these joint representation to obtain the answers. Besides, we also compare LEGO with **KG traversal** using the ground truth query, whose accuracy would be the upper bound of the semantic-parsing based KGQA meth-

Table 3. Hits@1 results of MetaQA on 50% KG. Our method achieves at least 3.6% better than the SOTA baselines.

Hits@1	1-hop	2-hop	3-hop	All
KG Traversal (w/ GT query)	63.3	45.8	45.3	51.5
Latent Execution (w/ GT query)	70.8	62.1	66.4	66.6
Pullnet (Sun et al., 2019a)	65.1	52.1	59.7	59.2
EmbedKGQA (Saxena et al., 2020)	**70.6**	54.3	53.5	60.2
EMQL (Sun et al., 2020)	63.8	47.6	48.1	53.2
LEGO (ours)	69.3	**57.8**	**63.8**	**63.8**

Table 4. Hits@1 results of WQSP and CWQ. Ours achieve 2.6% and 1.1% better results respectively.

Hits@1	CWQ	WQSP
KG Traversal (w/ GT query)	25.2	56.9
Latent Execution (w/ GT query)	46.4	70.0
Pullnet (Sun et al., 2019a)	26.8	47.4
EmbedKGQA (Saxena et al., 2020)	-	42.5
LEGO (ours)	**29.4**	**48.5**

ods. However, since the KG is incomplete, these methods can still only obtain a subset of answers. We tune all the methods on *the same KG* for all the datasets. More details of training and baselines can be found in Appendix G.

6.2. Main Results

Following the standard setup in KGQA (Saxena et al., 2020; Sun et al., 2019a; 2020), we evaluate the accuracy using the Hits@1 metrics. As shown in Table 3, our method achieves slightly worse results than EmbedKGQA on *single-hop* questions while outperforms both Pullnet and EmbedKGQA by at least 3.5% Hits@1 on *multi-hop questions*. On both CWQ and WQSP, we also achieve SOTA results, demonstrating the effectiveness of LEGO in modeling the reasoning steps of multi-hop questions. EMQL is not suitable for these two datasets since it requires prior knowledge of the query template/structure. Note that ours without using any ground truth query is better than traversing the KG using the ground truth query on both MetaQA (12.3%) and CWQ (4.2%). When using our executor to execute the ground truth query, we could achieve the best performance, serving as the upper bound of LEGO.

6.3. Search Space Pruning

For both the WQSP and the CWQ datasets, since the KG has a huge number of relations (> 1000), enumerating all the candidate queries becomes intractable. As introduced in Sec. 5, we pretrain a branch and relation pruner on the KG to perform execution guided search instead.

Branch Pruning. For all the ground truth query trees of the questions in the dataset, we generate a dataset of partial queries to evaluate the branch pruner. Specifically, we take those query trees with multiple topic entities, which means that there exists at least one intersection step in the query tree. We add the partial query before the intersection operation to the dataset for evaluation, these branches should be positive examples. Then we perturb these branches to create negative examples. For example, given a query tree of a test question: [[NobelPrize, [winnerOf, bornIn]], [WorldCup, [heldIn]]], these two branches were together added to the dataset as positive samples. We further perturb the first branch to [NobelPrize, [winnerOf]], now the two branches should not be intersected, and we add these perturbed examples to the dataset.

Table 5. Results of branch pruning on WQSP and CWQ. The pruner can perfectly classify whether a set of branches should be intersected. Selecting $S = 0.8$ (Eq. 6) can cover more than 91% intersection steps when searching for candidates.

	$S = 0.1$	$S = 0.4$	$S = 0.8$	AUC	#Positive	#Negative
WQSP (30%)	0.97	0.96	0.95	0.99	975	229
WQSP (50%)	0.99	0.98	0.96	0.99	975	229
CWQ	0.98	0.95	0.91	0.96	17596	12320

Table 6. Results of relation pruning on WQSP and CWQ.

	$k = 5$	$k = 50$	MRR	MR	$\|\mathcal{R}\|$
WQSP (30%)	0.68	0.98	0.4	8.14	1632
WQSP (50%)	0.72	0.98	0.42	7.44	1632
CWQ	0.61	0.93	0.4	14.3	1836

As shown in Table 5, our pretrained branch pruner achieves almost perfect AUC and accuracy in classifying whether a given set of branches should be intersected or not. Selecting the threshold $S = 0.8$ (in Eq. 6) can already covers more than 91% of the intersection steps for both WQSP and CWQ.

Relation Pruning. Similar to the branch pruning, we also create a dataset to evaluate the performance of the pretrained relation pruner based on the WQSP and CWQ test questions. For all query trees of the test questions, we generate all the relation projection steps of the full query tree. For example, given the query tree, [[NobelPrize, [winnerOf, bornIn]], [WorldCup, [heldIn]]], it has three relation projection steps: (1) [NobelPrize, []] $\rightarrow$ “winnerOf”; (2) [NobelPrize, [winnerOf]] $\rightarrow$ “bornIn”; (3) [WorldCup, []] $\rightarrow$ “heldIn”. We evaluate the performance of the relation pruner by calculating the mean reciprocal rank of these targeted relation. Note that when training the relation pruner, it only samples queries from the KG and has no knowledge of what the query trees of the test questions will be.

As shown in Table 6, our pretrained relation pruner has 0.42 and 0.4 mean reciprocal rank on WQSP and CWQ respectively. We can safely reduce the number of relations to search from all 1836 relations to only the top 50 relations, reducing more than 99.92% the search space for a 2-hop question on CWQ.

See Appendix H for example questions with the ground truth query as well as the top-ranking candidate queries.

6.4. Ablation

Severely Incomplete KG. We test LEGO on WQSP with 70% missing edges in KG. We still achieve the best result,

Table 7. Hits@1 results of WQSP on 30% incomplete KG. Ours achieves 3.4% better performance.

Hits@1	WQSP (30% KG)
KG Traversal (w/ GT query)	37.3
Latent Execution (w/ GT query)	53.2
Pullnet (Sun et al., 2019a)	34.6
EmbedKGQA (Saxena et al., 2020)	31.4
LEGO (ours)	**38.0**

outperforming Pullnet and EmbedKGQA by 3.4% and 6.6% respectively, shown in Table 7. Note that the margin is even larger than that on 50% KG, demonstrating the robustness of LEGO on more severely incomplete KGs.

Table 8. Hits@1 results of LEGO without stochastic hard EM on CWQ and WQSP.

Hits@1	CWQ	WQSP
LEGO w/o S-Hard EM	27.8	47.6
LEGO	**29.4**	**48.5**

LEGO W/O Stochastic Hard EM. We further analyze the stochastic hard EM trick. The performance on CWQ and WQSP decreases by up to 1.6% without using stochastic hard EM, which demonstrates the robustness against spurious queries (noisy labels).

Table 9. Hits@1 results and the percentage of questions with ground truth query tree in the resulting candidate set after searching on WQSP given the same computation budget.

	GT query searched %	Hits@1
LEGO w/o Pruning	3.3	13.2
LEGO	**65.5**	**48.5**

LEGO W/O Pruning. Given the same computation budget, we further search the candidate queries for training questions without the search space pruning. At each step, instead of predicting top 50 probable relations according to Eq. 7, we randomly select 50. Only 3.3% training questions discovered the ground truth query (Table 9). The synthesizer trained by this candidate set becomes much worse.

7. Conclusion

We proposed LEGO for multi-hop KGQA. LEGO consists of a latent space executor and a query synthesizer, iteratively synthesizing and executing the query in the embedding space. With execution-guided search space pruner, LEGO achieves the state-of-the-art performance.

Acknowledgements

We thank the anonymous reviewers for providing feedback on our manuscript. We also gratefully acknowledge the support of DARPA under Nos. HR00112190039 (TAMI), N660011924033 (MCS); ARO under Nos. W911NF-16-1-0342 (MURI), W911NF-16-1-0171 (DURIP); NSF under Nos. OAC-1835598 (CINES), OAC-1934578 (HDR), CCF-1918940 (Expeditions), IIS-2030477 (RAPID), NIH under No. R56LM013365; Stanford Data Science Initiative, Wu Tsai Neurosciences Institute, Chan Zuckerberg Biohub, Amazon, JPMorgan Chase, Docomo, Hitachi, Intel, JD.com, KDDI, NVIDIA, Dell, Toshiba, Visa, and UnitedHealth Group. Hongyu Ren is supported by the Masason Foundation Fellowship and the Apple PhD Fellowship. Jure Leskovec is a Chan Zuckerberg Biohub investigator.

References

Amini, A., Gabriel, S., Lin, S., Koncel-Kedziorski, R., Choi, Y., and Hajishirzi, H. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2019.

Andreas, J., Rohrbach, M., Darrell, T., and Klein, D. Learning to compose neural networks for question answering. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, 2016.

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

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

Bast, H. and Haussmann, E. More accurate question answering on freebase. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015.

Berant, J., Chou, A., Frostig, R., and Liang, P. Semantic parsing on freebase from question-answer pairs. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2013.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. Freebase: a collaboratively created graph database for structuring human knowledge. In *ACM SIGMOD international conference on Management of data (SIGMOD)*. ACM, 2008.

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

Chen, X., Liu, C., and Song, D. Execution-guided neural program synthesis. In *International Conference on Learning Representations (ICLR)*, 2019a.

Chen, X., Liang, C., Yu, A. W., Song, D., and Zhou, D. Compositional generalization via neural-symbolic stack machines. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020a.

Chen, X., Liang, C., Yu, A. W., Zhou, D., Song, D., and Le, Q. V. Neural symbolic reader: Scalable integration of distributed and symbolic representations for reading comprehension. In *International Conference on Learning Representations (ICLR)*, 2020b.

Chen, Z.-Y., Chang, C.-H., Chen, Y.-P., Nayak, J., and Ku, L.-W. 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)*, 2019b.

Das, R., Zaheer, M., Reddy, S., and McCallum, A. Question answering on knowledge bases and text using universal schema and memory networks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

Das, R., Dhuliawala, S., Zaheer, M., Vilnis, L., Durugkar, I., Krishnamurthy, A., Smola, A., and McCallum, A. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. 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.

Dong, L., Wei, F., Zhou, M., and Xu, K. Question answering over freebase with multi-column convolutional neural networks. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2015.

Ellis, K., Nye, M. I., Pu, Y., Sosa, F., Tenenbaum, J. B., and Solar-Lezama, A. Write, execute, assess: Program synthesis with a repl. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Gupta, N., Lin, K., Roth, D., Singh, S., and Gardner, M. Neural module networks for reasoning over text. In *International Conference on Learning Representations (ICLR)*, 2019.

Guu, K., Pasupat, P., Liu, E., and Liang, P. From language to programs: Bridging reinforcement learning and maximum marginal likelihood. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2017.

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

Hu, S., Zou, L., and Zhang, X. A state-transition framework to answer complex questions over knowledge base. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Jia, R. and Liang, P. Data recombination for neural semantic parsing. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016.

Keysers, D., Schärli, N., Scales, N., Buisman, H., Furrer, D., Kashubin, S., Momchev, N., Sinopalnikov, D., Stafiniak, L., Tihon, T., et al. Measuring compositional generalization: A comprehensive method on realistic data. In *International Conference on Learning Representations (ICLR)*, 2020.

Krishnamurthy, J., Dasigi, P., and Gardner, M. Neural semantic parsing with type constraints for semi-structured tables. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

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

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

Liang, C., Norouzi, M., Berant, J., Le, Q., and Lao, N. Memory augmented policy optimization for program synthesis and semantic parsing. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Lin, X. V., Socher, R., and Xiong, C. Multi-hop knowledge graph reasoning with reward shaping. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

Liu, H. and Singh, P. Conceptnet—a practical commonsense reasoning tool-kit. *BT technology journal*, 2004.

Liu, S., Niles-Weed, J., Razavian, N., and Fernandez-Granda, C. Early-learning regularization prevents memorization of noisy labels. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

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

Miller, A., Fisch, A., Dodge, J., Karimi, A.-H., Bordes, A., and Weston, J. Key-value memory networks for directly reading documents. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

Min, S., Chen, D., Hajishirzi, H., and Zettlemoyer, L. A discrete hard em approach for weakly supervised question answering. *arXiv preprint arXiv:1909.04849*, 2019.

Neelakantan, A., Le, Q. V., Abadi, M., McCallum, A., and Amodei, D. Learning a natural language interface with neural programmer. In *International Conference on Learning Representations (ICLR)*, 2017.

Nye, M., Pu, Y., Bowers, M., Andreas, J., Tenenbaum, J. B., and Solar-Lezama, A. Representing partial programs with blended abstract semantics. *arXiv preprint arXiv:2012.12964*, 2020a.

Nye, M. I., Solar-Lezama, A., Tenenbaum, J. B., and Lake, B. M. Learning compositional rules via neural program synthesis. *arXiv preprint arXiv:2003.05562*, 2020b.

Odena, A., Shi, K., Bieber, D., Singh, R., and Sutton, C. Bustle: Bottom-up program-synthesis through learning-guided exploration. *arXiv preprint arXiv:2007.14381*, 2020.

Poon, H. and Domingos, P. Unsupervised semantic parsing. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2009.

Qiu, Y., Wang, Y., Jin, X., and Zhang, K. Stepwise reasoning for multi-relation question answering over knowledge graph with weak supervision. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020a.

Qiu, Y., Zhang, K., Wang, Y., Jin, X., Bai, L., Guan, S., and Cheng, X. Hierarchical query graph generation for complex question answering over knowledge graph. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020b.

Reimers, N. and Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

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

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

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

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

Sun, H., Dhingra, B., Zaheer, M., Mazaitis, K., Salakhutdinov, R., and Cohen, W. W. Open domain question answering using early fusion of knowledge bases and text. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018a.

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

Sun, H., Arnold, A. O., Bedrax-Weiss, T., Pereira, F., and Cohen, W. W. Faithful embeddings for knowledge base queries. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Sun, S.-H., Noh, H., Somasundaram, S., and Lim, J. Neural program synthesis from diverse demonstration videos. In *International Conference on Machine Learning (ICML)*, 2018b.

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

Talmor, A. and Berant, J. 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.

Tian, Y., Luo, A., Sun, X., Ellis, K., Freeman, W. T., Tenenbaum, J. B., and Wu, J. Learning to infer and execute 3d shape programs. In *International Conference on Learning Representations (ICLR)*, 2019.

Wang, C., Tatwawadi, K., Brockschmidt, M., Huang, P.-S., Mao, Y., Polozov, O., and Singh, R. Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100*, 2018.

Xiong, W., Hoang, T., and Wang, W. Y. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

Xiong, W., Yu, M., Chang, S., Guo, X., and Wang, W. Y. Improving question answering over incomplete kbs with

knowledge-aware reader. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019.

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

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

Yih, W.-t., Richardson, M., Meek, C., Chang, M.-W., and Suh, J. The value of semantic parse labeling for knowledge base question answering. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2016.

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

Yu, T., Yasunaga, M., Yang, K., Zhang, R., Wang, D., Li, Z., and Radev, D. Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018a.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2018b.

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

Zelle, J. M. and Mooney, R. J. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, 1996.

Zettlemoyer, L. S. and Collins, M. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Uncertainty in Artificial Intelligence (UAI)*, 2012.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations (ICLR)*, 2017.

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

Zohar, A. and Wolf, L. Automatic program synthesis of long programs with a learned garbage collector. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

Appendix

A. Design Choice of the Latent Space Executor

Besides Query2box (Ren et al., 2020), which is introduced in Sec. 4.1, we also design a new latent space executor based on RotatE (Sun et al., 2019b). RotatE is also a translation-based knowledge graph embedding method. It models a triple (h, r, t) as a relation traversal in the Complex space. The goal is $\mathbf{t} = \mathbf{h} \circ \mathbf{r}$, where $\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{C}^d$, $|\mathbf{r_i}| = 1$. Since the rotation operation is naturally compositional, we further extend RotatE to handle multi-hop KG reasoning.

Given the latent execution results of a partial 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. And we use a RotatE embedding to represent b_i^t: $\mathbf{b_i^t} \in \mathbb{C}^d$. We also provide two logical operators:

$$\mathcal{P} : \mathbb{C}^d \times \mathbb{C}^d \to \mathbb{C}^d \quad \text{and} \quad \mathcal{I} : \mathbb{C}^d \times \cdots \times \mathbb{C}^d \to \mathbb{C}^d$$

to perform relation projection and intersection in the embedding space respectively. The RotatE-based latent executor models the conditional distribution $p(\mathbf{g_{t+1}}|\mathbf{g_t}, a_t)$ as follows. For all valid reasoning actions $a_t \in \mathcal{A}$:

(1) $a_t = (\{b_i^t\}, r)$: extension of one branch b_i^t with a relation edge r, this represents one relation projection from the set of entities in b_i^t using r, *e.g.*, step 1 and 2 in Figure 2. The executor updates the i-th component of the query embedding $\mathbf{g_t}$ accordingly: $\mathbf{g_{t+1}}[i] = \mathcal{P}(\mathbf{b_i^t}, \mathbf{r}) = \mathbf{b_i^t} \circ \mathbf{r}$;

(2) $a_t = (B, -1)$: conjunction of multiple branches $B \subseteq \{b_i^t\}_{i=1}^n, |B| > 1$, this action takes the intersection of the set of entities in each $b \in B$, *e.g.*, step 3 in Figure 2. We use the intersection operator $\mathcal{I}$, remove all embeddings $\mathbf{b_i^t}$ with $b_i^t \in B$ from $\mathbf{g_t}$, and append $\mathbf{b}_{\text{int}}^{\mathbf{t}} = \mathcal{I}(\mathbf{B}) = D_\omega(\mathbf{B})$ to the end of $\mathbf{g_t}$, where D_ω is a neural network with DeepSet architecture (Zaheer et al., 2017);

(3) $a_t = (\varnothing, -1)$: termination, *e.g.*, step 4 in Figure 2.

B. Pretraining Details of Latent Space Executor

Given a KG, we follow the practices of Query2box (Ren et al., 2020) and synthesize query trees of different structures. As shown in Figure 5, 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, answer] pair to our pretraining dataset.

Given a batch of [query g, answer v] pairs, we may first embed the query tree using the latent space executor, then we may optimize a loss to minimize the distance between the query embedding $\mathbf{g}$ and the answer embedding $\mathbf{v}$ while maximize the distance between the query embedding $\mathbf{g}$ and k negative samples $\{\mathbf{v_j'}\}$:

$$L = -\log \sigma \left(\gamma - \text{dist}(\mathbf{v}; \mathbf{g})\right) - \sum_{j=1}^{k} \frac{1}{k} \log \sigma \left(\text{dist}(\mathbf{v_j'}; \mathbf{g}) - \gamma\right). \quad (8)$$

The distance function varies for different executors, which will be detailed in Appendix D.

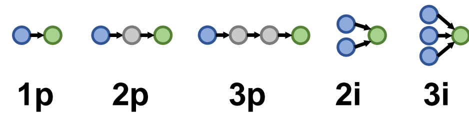

Figure 5. The query structures on which we instantiate grounded queries and pretrain the knowledge embedding module.

C. Design Choice of the Query Synthesis Module

Here we discuss the details of the architecture design of the $D_\theta(\cdot)$, $S_\theta(\cdot, \cdot)$, $R_\theta(\cdot, \cdot)$ networks in the query synthesis module. Since D_θ 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 (Zaheer et al., 2017), 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 $D_\theta(\varnothing) = \mathbf{0}$. For S_θ, it aims to score a set of branches conditioned on the input question, so we directly concatenate the set representation obtained by D_θ with the Bert embedding $\mathbf{q}$ of the question. After

we score all branches in the powerset using D_θ and S_θ, we normalize it with Softmax. For R_θ, it has the same input with S_θ, hence we adopt the same architecture, and only differ in the design of the last layer, where instead of selecting branches, R_θ outputs a distribution over all the relations.

D. Distance Function

Given the final query tree with a single branch $\mathbf{g}$, we define the distance between $\mathbf{g}$ and an entity embedding $\mathbf{v}$ on KG.

If the latent space executor is based on Query2box, then we use the box distance as in Query2box (Ren et al., 2020). Here $\mathbf{g}$ is a box with center and offset, and $\mathbf{v}$ is a single point in the embedding space.

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

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

If the latent space executor is based on RotatE, then we define distance as L1 distance between the query embedding and the entity embedding: $\mathrm{dist}_{\mathrm{rotate}}(\mathbf{v};\mathbf{g}) = \|\mathbf{v}-\mathbf{g}\|_1$.

E. 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 $\mathcal{G}$.

F. Pretraining Details of Pruners

F.1. Branch Pruner

In order to pretrain the branch pruner f_ϕ, we need to sample positive branch sets and negative branch sets, where positive branch sets represent a set of branches that have shared answers, while negative branch sets represent a set of branches that do not have shared answers. Specifically, we look at several query templates/structures, including 2i and 3i as shown in Fig. 5. We instantiate 2i and 3i queries on KG, and the instantiated queries will be viewed as positives since they all have shared answers. We then randomly sample branches from KG, and view these randomly sampled branches as negatives. Note again that this pretraining process does not involve natural language questions.

F.2. Relation Pruner

For pretraining the relation pruner p_ϕ, we need to sample [query, relation] pairs from KG. The trick is to first sample a pair of [query, answer] and then take the union of all the relations associated with the answer in order to obtain [query, relation] pairs. In detail, we instantiate all 5 query templates/structures {1p, 2p, 3p, 2i, 3i} for pretraining relation pruners. We have also tried to only use queries of structure {1p, 2p, 3p}, the performance are comparable.

G. Experimental Details

For all the baselines and our method, we use the same pretrained case-insensitive 768 dimensional Bert embedding (without finetuning) (Devlin et al., 2019) to obtain the question representation for fair comparison.

H. Example Candidate Queries

We also list some candidate queries our model finds for question from the WebQuestionSP dataset (Yih et al., 2015). As shown in Figure 6, our method mostly finds the correct candidate queries for the questions, (the concrete percentage

can be found in Table 9). Although the ground truth query may not always achieve the highest score (the closest to the answers) measured by mean reciprocal rank (MRR). Some other non-ground truth queries also make sense. For example, the candidates we find for question "what is nina dobrev nationality" contain a relation path ["place_of_birth", "location.contained_by"], which may still provide meaningful supervision signal for the synthesizer.

Question	**Ground Truth**	**Candidates (query, mrr)**
who does joakim noah play for	[topic, ['pro_athlete.teams', 'sports_team_roster.team']]	1. [topic, ['sports.sports_team_roster.player', 'sports.sports_team.roster']] 0.5 2. [topic, ['sports.sports_team_roster.player', 'sports.sports_team_roster.team']] 0.5 3. **[topic, ['sports.pro_athlete.teams', 'sports.sports_team_roster.team']]** 0.5
what is nina dobrev nationality	[topic, ['people.person.nationality']]	1. [topic, ['people.person.languages', 'language.language_family.geographic_distribution']] 0.509 2. [topic, ['people.person.place_of_birth', 'location.contained_by']] 0.5 3. **[topic, ['people.person.nationality']]** 0.455
what movies does taylor lautner play in	[topic, ['film.actor.film', 'film.performance.film']]	1. [topic, ['film.performance.actor', 'film.film.starring']] 1.0 2. **[topic, ['film.actor.film', 'film.performance.film']]** 1.0 3. [topic, ['film.actor.film', 'film.film.starring']] 1.0
what type of guitar does kirk hammett play	[[topic1, ['music.group_member.instruments_played']], [topic2, ['music.instrument.family']]]	1. **[[topic1, ['music.group_member.instruments_played']], [topic2, ['music.instrument.family']]]** 1.0 2. [[topic1, ['music.group_member.instruments_played']], [topic2, ['music.composition.composer']]] 1.0 3. [[topic1, ['music.group_member.instruments_played']], [topic2, ['music.group_membership.role']]] 1.0

Figure 6. Example questions from WebQuestion datasets with the ground truth query and the candidate queries our model finds.

I. Results of Different Latent Space Executors

Here we show the H@1max and H@10 results of LEGO with different latent space executors (RotatE and Q2B). Note the H@1 max measures whether the top ranking entiyt is the answer and H@10 measures percentage of the (filtered) rank of *all answers* is among top 10 (typically used in KG completion). Our method LEGO with both executors achieve better performance than baselines, which suggests that LEGO is robust to the specific embedding models. PullNet has lower H@10 than H@1 max because it ranks answers on a subset of KG entities it retrieves and the recall of the answers is low.

Table 10. Different embeddings (H@1max / H@10).

	CWQ	WQSP (50%)	WQSP (30%)
Pullnet	26.8 / 33.9	47.4 / 39.1	34.6 / 23.2
EmbedKGQA	-	42.5 / 60.6	31.4 / 41.4
LEGO (RotatE)	**29.4 / 49.4**	**48.5 / 67.3**	38.0 / 48.2
LEGO (Q2B)	28.9 / 48.5	48.3 / 66.3	**39.2 / 50.8**

Table 11. Relation Pruner of LEGO on CWQ using RotatE and Box as latent space executor.

	$k = 5$	$k = 50$	MRR	MR	$\|\mathcal{R}\|$
RotatE	0.61	0.93	0.4	14.3	1836
Q2B	0.62	0.93	0.41	14.0	1836

Table 12. Branch Pruner of LEGO on CWQ using RotatE and Box as latent space executor.

	$S = 0.1$	$S = 0.4$	$S = 0.8$	AUC	#Positive	#Negative
RotatE	0.98	0.95	0.91	0.96	17596	12320
Q2B	0.99	0.97	0.95	0.98	17596	12320