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 $\mathbf{g_t} = [\mathbf{b^t_1}, \dots, \mathbf{b^t_n}]$ at step t, where $\mathbf{b^t_i}$ represents the embedding of branch b^t_i of the query tree. And we use a RotatE embedding to represent b^t_i: $\mathbf{b^t_i} \in \mathbb{C}^d$. 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) $a_t = (\{b^t_i\}, r)$: extension of one branch b^t_i with a relation edge r, this represents one relation projection from the set of entities in b^t_i 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: $\mathbf{g_{t+1}}[i] = \mathcal{P}(\mathbf{b^t_i}, \mathbf{r}) = \mathbf{b^t_i} \circ \mathbf{r}$;

(2) $a_t = (B, -1)$: conjunction of multiple branches $B \subseteq \{b^t_i\}^n_{i=1}$, $|B| > 1$, 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 $\mathbf{b^t_i}$ with $b^t_i \in B$ from $\mathbf{g_t}$, and append $\mathbf{b^t_{\text{int}}} = \mathcal{I}(\mathbf{B}) = D_\omega(\mathbf{B})$ 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.

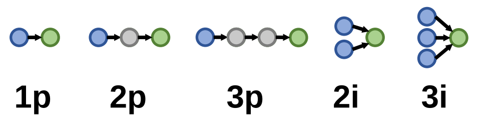


Figure 5. The query structures on which we instantiate grounded queries and pretrain the knowledge embedding module.

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 $\mathbf{B} \in [\varnothing, \{\mathbf{b_1}\}, \dots, \{\mathbf{b_1}, \mathbf{b_2}\}, \dots, \{\mathbf{b_1}, \dots, \mathbf{b_n}\}]$ 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 $\mathbf{g}_{\mathrm{max}} = \mathrm{Cen}(\mathbf{g}) + \mathrm{Off}(\mathbf{g}) \in \mathbb{R}^d$, $\mathbf{g}_{\mathrm{min}} = \mathrm{Cen}(\mathbf{g}) - \mathrm{Off}(\mathbf{g}) \in \mathbb{R}^d$ 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

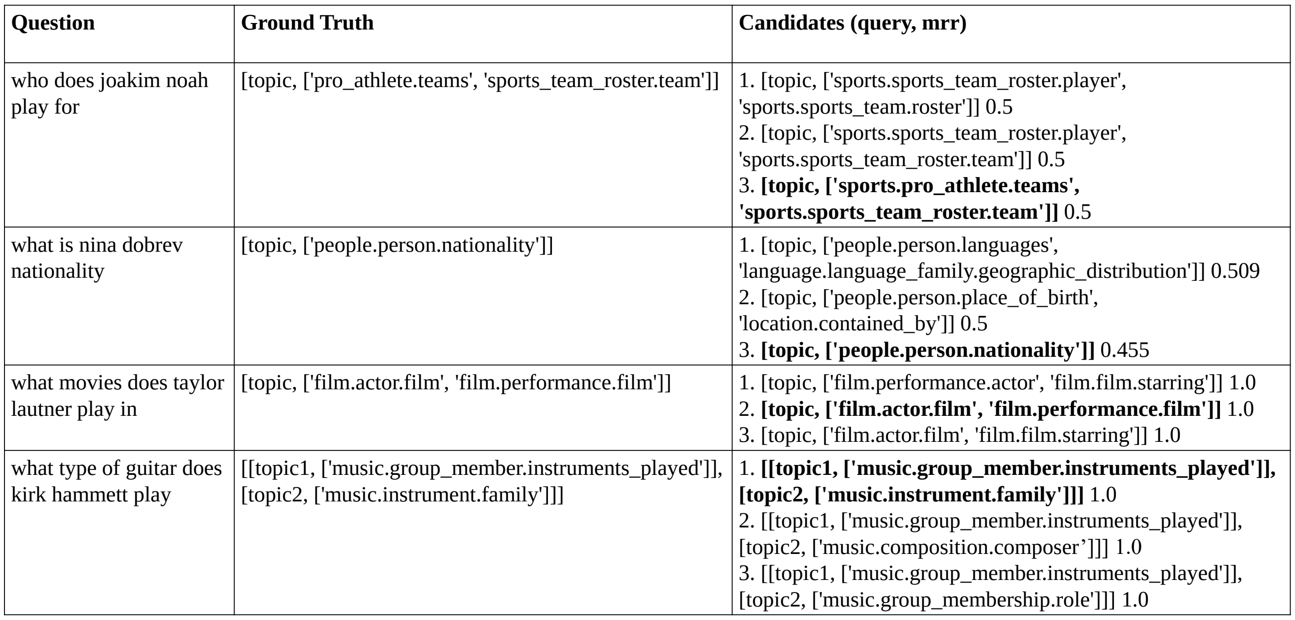
For pretraining the relation pruner p_ϕ, we need to sample [query, relation] pairs from KG. The trick is to first sample a pair of [query, answer] and then take the union of all the relations associated with the answer in order to obtain [query, relation] pairs. In detail, we instantiate all 5 query 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