ReTraCk: A Flexible and Efficient Framework for Knowledge Base Question Answering

Shuang Chen[†*], **Qian Liu**[♡*], **Zhiwei Yu**[§*], **Chin-Yew Lin**[§], **Jian-Guang Lou**[§], **Feng Jiang**[†]
[†]Harbin Institute of Technology, Harbin, China
[♡]Beihang University, Beijing, China
[§]Microsoft Research, Beijing, China
[†]hitercs@gmail.com;[♡]qian.liu@buaa.edu.cn;[§]{zhiwyu,cyl,jlou}@microsoft.com

Abstract

We present **Re**triever-**Tra**nsducer-**C**hec**k**er (ReTraCk), a neural semantic parsing framework for large scale knowledge base question answering (KBQA). ReTraCk is designed as a modular framework to maintain high flexibility. It includes a retriever to retrieve relevant KB items efficiently, a transducer to generate logical form with syntax correctness guarantees and a checker to improve the transduction procedure. ReTraCk is ranked at top1 overall performance on the GrailQA leaderboard[1] and obtains highly competitive performance on the typical WebQuestionsSP benchmark. Our system can interact with users timely, demonstrating the efficiency of the proposed framework.[2]

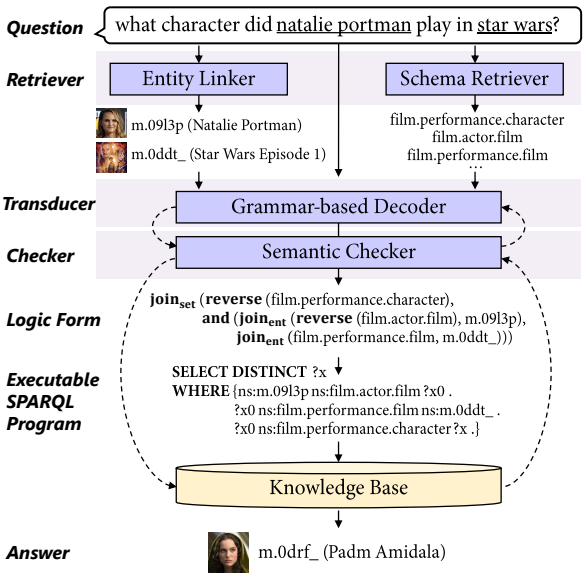

Figure 1: The **Re**triever-**Tra**nsducer-**C**hec**k**er (ReTraCk) framework.

1 Introduction

Knowledge base question answering (KBQA) is an important task in natural language processing that aims to satisfy users' information needs based on factual information stored in knowledge bases. Over the years, it has attracted a great deal of research attention from academia and industry. Early KBQA systems are generally rule-based. They rely on predefined rules or templates to parse questions into logical forms (Cabrio et al., 2012; Abujabal et al., 2017), suffering from coverage and scalability problems. Recently, researchers usually focus more on neural semantic parsing approaches. These data-driven parsing methods (Yih et al., 2015; Jia and Liang, 2016; Dong and Lapata, 2016; Liang et al., 2017; Gu et al., 2021) significantly improve the state-of-the-art (SOTA) performance on KBQA tasks.

Although various neural semantic parsing methods have been proposed for KBQA, there are few works investigating how to leverage the advantages of SOTA models to build a comprehensive system, and how to fit the system with practical application purpose (e.g., balancing effectiveness and efficiency). To investigate, we identify two key issues hindering the development of KBQA systems.

On the one hand, there is a lack of a generic and extensible framework for KBQA. For example, the popular SEMPRE[3] toolkit (Berant et al., 2013) provides infrastructures to develop statistical semantic parsers for KBQA with rich features, but its performance and scalability are inferior to recent neural semantic parsing methods. The TRANX toolkit[4] (Yin and Neubig, 2018) employs a transition-based neural semantic parser to model the logical form generation procedure as a

[*]The first three authors contributed equally. This work was conducted during Shuang and Qian's internship at Microsoft Research Asia.
[1]https://dki-lab.github.io/GrailQA/
[2]https://aka.ms/ReTraCk
[3]https://github.com/percyliang/sempre
[4]https://github.com/pcyin/tranX

Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations, pages 325–336, August 1st - August 6th, 2021.

sequence of tree-constructing actions under grammar specification. However, TRANX does not include the essential retriever components used in grounding, and thus does not support KBQA by now.

On the other hand, recent neural semantic parsing methods mostly emphasize performance on benchmark datasets while neglecting the efficiency (speed) dimension. This limits the understanding of how designed approaches fit into real applications. For example, the popular *query graph generation* methods generate and rank a set of query graphs (Yih et al., 2015; Maheshwari et al., 2019; Lan and Jiang, 2020). Since all query graph candidates keep in line with the knowledge base (KB) structure, these methods take full advantage of the KB. However, they suffer from poor efficiency due to the large number of candidates and heavily querying on KB. To verify that, we performed a preliminary study on available SOTA models[5,6,7,8,9,10]. According to our study, these models either have difficulties in supporting interactive online services, or limit the candidate space for specific datasets, which makes them difficult to apply in practice.

To this end, we present ReTraCk, a practical framework for large scale KBQA. We hope ReTraCk can help standardize the KBQA model design process and lower the barrier of entry for new practitioners. ReTraCk is designed with the following principles in mind:

- **Flexibility** ReTraCk employs a modular architecture, which decouples the dependencies among components as much as possible to enable quick integration of novel components. For example, our system supports two different kinds of schema retrievers, namely *dense schema retriever* and *neighbor schema retriever*[11].

- **Efficiency** ReTraCk falls into the transduction family, which is fast during the generation process. Besides, we retrieve entities and relevant schema items (relations and types) in parallel by leveraging the recent advance of entity linking (Orr et al., 2021) and dense retrieval (Wu et al., 2020; Karpukhin et al., 2020). Our system can interact with users timely, demonstrating the efficiency of the proposed ReTraCk framework.

- **Effectiveness** ReTraCk is designed to enhance the controllability of transduction-based methods in both syntax level and semantic level. It first employs a grammar based decoder (Yin and Neubig, 2018) to guarantee the syntax correctness. Then it leverages a checker to alleviate the semantic inconsistency issues. Inspired by previous work, four checking mechanisms are proposed and implemented in the checker: instance-level checking (Liang et al., 2017), ontology-level checking (Chen et al., 2018), real execution (Wang et al., 2018) and the novel virtual execution. The experimental results verify the significant effectiveness of our proposed checker. Notably, the checker is also flexible enough to be easily extended with new mechanisms. Finally, ReTraCk achieves state-of-the-art performance on GrailQA and achieves highly competitive performance on WebQuestionsSP.

2 ReTraCk Framework

Given an input question $\mathbf{q}$, ReTraCk parses the question into a logical form which can be deterministically converted into a SPARQL query to retrieve answers from the knowledge base $\mathcal{K}$. Generally $\mathcal{K}$ consists of two parts: an ontology $\mathcal{O} \subseteq \mathcal{T} \times \mathcal{R} \times \mathcal{T}$, which defines the schema structure, and the fact triples $\mathcal{F} \subseteq \mathcal{E} \times \mathcal{R} \times (\mathcal{E} \cup \mathcal{T} \cup \mathcal{L})$. Here, $\mathcal{T}$ is the set of types, $\mathcal{R}$ is the set of relations, $\mathcal{E}$ is the set of entities, and $\mathcal{L}$ is the set of literals.

As shown in Fig. 1, ReTraCk consists of three components: retriever, transducer and checker. The *retriever* consists of an entity linker, which links explicit entity mentions to corresponding entities, and a schema retriever, which retrieves relevant *schema items* (types and relations) mentioned either explicitly or implicitly in the question. Given the retrieved *KB items* (entities, types, and relations), the *transducer* employs a grammar-based decoder to generate the logical form with syntax correctness guarantees. Meanwhile, the transducer interacts with the *checker* to discourage generating programs that are semantically inconsistent with KB.

To make ReTraCk more accessible and interpretable for end users, we build a user interface. As shown in Fig. 2, users can type a question in the text box. The interface then displays retrieved

[5] http://github.com/nju-websoft/SPARQA
[6] http://github.com/lanyunshi/Multi-hopComplexKBQA
[7] http://github.com/OceanskySun/GraftNet
[8] https://github.com/scottyih/STAGG
[9] https://github.com/guoday/Dialog-to-Action
[10] https://github.com/dongpobeyond/Seq2Act
[11] This module is implemented in our codebase. The detailed analysis is in the Appendix.

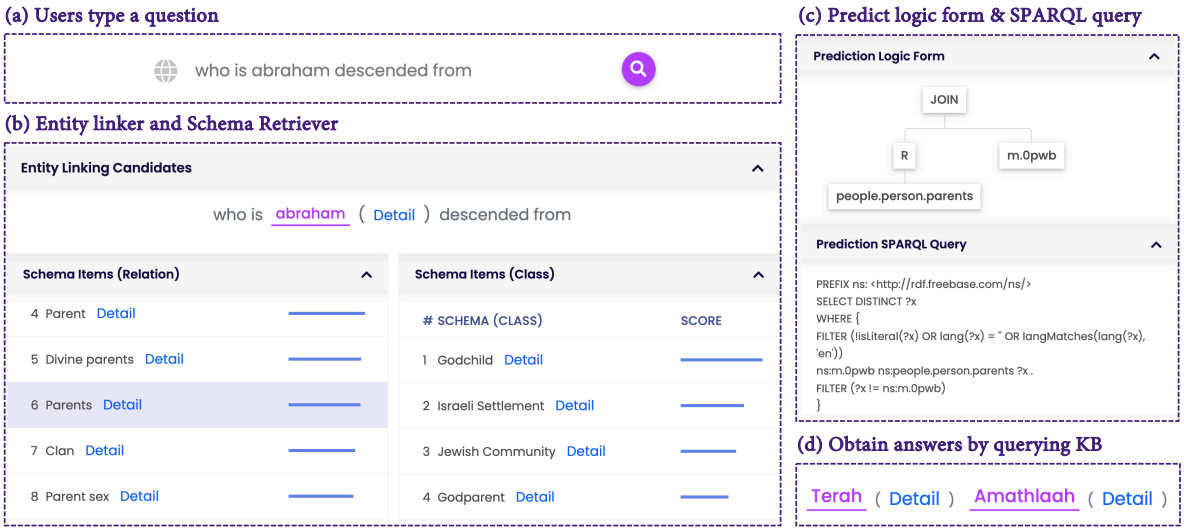

Figure 2: The main user interface of ReTraCk.

KB items, a graph visualization of predicted logical forms, generated SPARQL query and predicted answer (s). The schema items selected by our transducer are shaded. Besides, users can refer to more information of any KB item by clicking on the subsequent "Detail". Next, we will introduce each component in detail.

2.1 Retriever

Entity Linker The entity linker used in this work follows the entity linking pipeline described in Gu et al. (2021). It firstly detects entity mentions using a BERT-based NER system, then generates candidate entities along with their prior score based on an alias map mined from the KB and FACC1 (Gabrilovich et al., 2013). As for entity disambiguation, we implement a prior baseline which selects the most popular entity based on the prior score. Besides, we also implement an alternative model by leveraging BOOTLEG (Orr et al., 2021) enriched with the prior features[12]. Due to space limitations, the model details and its comparison with the entity linker used in Gu et al. (2021) are put in the Appendix.

Schema Retriever As schema items are not always mentioned explicitly in the question and their vocabularies are much fewer than entities[13], we leverage the dense retriever framework (Mazaré et al., 2018; Humeau et al., 2020; Wu et al., 2020) to obtain the related types and relations. To be specific, we train a bi-encoder architecture (Wu et al., 2020) such that related schema items are close to the question embedding. This architecture allows for fast real-time inference, as it is able to cache the encoded candidates.

We use two independent BERT-base encoders (Devlin et al., 2019) to represent the input question $\mathbf{e_q}$ and candidate schema items $\mathbf{e_s}$ by extracting the upper most layer representation corresponding to the [CLS] token. The matching score for each pair (q_g, s_i) is calculated by the dot-product:

$$s(q_g, s_i) = \mathbf{e}_{q_g} \cdot \mathbf{e}_{s_i}. \quad (1)$$

Given a question q, we retrieve the top k schema items with the highest scores during inference time.

2.2 Transducer

Following previous work (Guo et al., 2018, 2019) - especially the s-expression design principle (Gu et al., 2021), we design a set of grammar rules for the logical form. As shown in Table 1, there are two kinds of grammars in our definition: knowledge-agnostic grammar and knowledge-specific grammar. To incorporate these predefined grammar rules, we introduce a question encoder and a grammar-based decoder (Liu et al., 2020).

Question Encoder To capture contextual information in a question, we apply a Bidirectional Long Short-Term Memory Neural Network (BiLSTM) (Hochreiter and Schmidhuber, 1997; Schuster and Paliwal, 1997) as our question encoder. For each

[12] In our demo system, we choose the prior baseline method since it is more memory efficient than the BOOTLEG (Orr et al., 2021) method.

[13] In the latest version of Freebase, there are more than 120 million entities, 16k types and 20k relations.

Grammar Rule	Description
root→ set | num	start of the grammar rule sequence
set→ and(set$_1$, set$_2$)	set$_1$ ∩ set$_2$
set→ join$_{\text{ent}}$(rel, ent)	$\{e_1 \mid (e_1, \text{ent}) \in \text{rel}\}$
set→ join$_{\text{set}}$(rel, set)	$\{e_1 \mid (e_1, e_2) \in \text{rel}$ and $e_2 \in \text{set}\}$
set→ argmax(set, rel)	$\{e_1 \mid (e_1, e_2) \in \text{rel}$ and e_2 is the largest$\}$
set→ argmin(set, rel)	$\{e_1 \mid (e_1, e_2) \in \text{rel}$ and e_2 is the smallest$\}$
set→ gt(rel, num)	$\{e_1 \mid (e_1, e_2) \in \text{rel}$ and $e_2 > \text{num}\}$
set→ ge(rel, num)	$\{e_1 \mid (e_1, e_2) \in \text{rel}$ and $e_2 \geq \text{num}\}$
set→ lt(rel, num)	$\{e_1 \mid (e_1, e_2) \in \text{rel}$ and $e_2 < \text{num}\}$
set→ le(rel, num)	$\{e_1 \mid (e_1, e_2) \in \text{rel}$ and $e_2 \leq \text{num}\}$
rel→ join$_{\text{rel}}$(rel$_1$, rel$_2$)	$\{(e_1, e_2) \mid (e_1, e) \in \text{rel}_1$ and $(e, e_2) \in \text{rel}_2\}$
rel→ reverse(rel)	$\{(e_1, e_2) \mid (e_2, e_1) \in \text{rel}\}$
num→ count(set)	number of entities in set
rel→ relation	instantiate a relation in $\mathcal{I}$
set→ type	instantiate a type in $\mathcal{I}$
ent→ entity | literal	instantiate an entity or literal in $\mathcal{I}$
num→ literal	instantiate a grammar rule for any literal in $\mathcal{I}$

Table 1: Knowledge-agnostic (**Top**) and knowledge-specific (**Bottom**) grammar rule definitions used in our grammar-based decoder. Knowledge-specific grammar rules change with the retrieved KB items $\mathcal{I}$. Here, `set` denotes a set of entities, `rel` denotes the set of (head, tail) entity tuples.

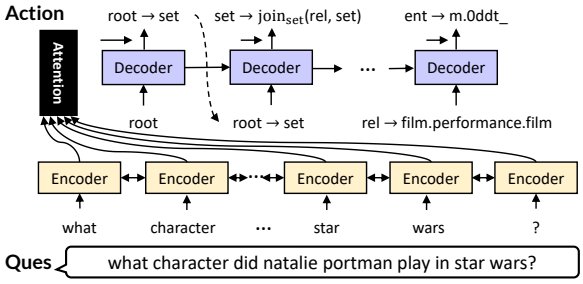

Figure 3: Decoding procedure of the example in Fig. 1.

token q_i in $\mathbf{q}$, we obtain its contextual representation as $\mathbf{h}_i^E = [\overrightarrow{\mathbf{h}_i^E}; \overleftarrow{\mathbf{h}_i^E}]$, where the forward hidden state $\overrightarrow{\mathbf{h}_i^E}$ is computed by passing the word embedding of q_i into a forward LSTM. The backward hidden state is computed similarly.

Grammar-based Decoder Once the question representation is prepared, the grammar-based decoder starts to produce the target logical form step by step with attention on the question. Our decoder regards each logical form as a structure and outputs its corresponding grammar rule/action [14] sequence $\mathbf{a} = (a_1, \cdots, a_K)$.

At each decoding step, a nonterminal (e.g., set) is expanded using one of its valid grammar rules. For example, at time step k, the LSTM decoder $\mathbf{LSTM}^{\overrightarrow{D}}$ accepts the embedding of the previous output $\phi^a(a_{k-1})$ as input and updates its hidden state as:

$$\mathbf{h}_k^{\overrightarrow{D}} = \mathbf{LSTM}^{\overrightarrow{D}}\left([\phi^a(a_{k-1}); \mathbf{c}_{k-1}], \mathbf{h}_{k-1}^{\overrightarrow{D}}\right), \quad (2)$$

[14] We use *grammar rule* and *action* interchangeably.

where $\mathbf{c}_{k-1}$ is the context vector obtained by attending on each encoder hidden state $\mathbf{h}_i^E$. As for ϕ^a, it behaves differently for knowledge-agnostic grammar rules and knowledge-specific grammar rules. For knowledge-agnostic grammar rules, ϕ^a returns a trainable global embedding. For knowledge-specific grammar rules, ϕ^a returns its related KB item representation, obtained by averaging over all word representations.

When predicting a_k, the probability of selecting the action γ follows:

$$P(a_k = \gamma) \propto \exp\left(\phi^a(\gamma) \tanh([\mathbf{h}_k^{\overrightarrow{D}}; \mathbf{c}_k]\mathbf{W}^o)\right), \quad (3)$$

where $\mathbf{W}^o$ is a learned matrix.

BERT Encoding Motivated by the success of pretrained language models on cross-domain text-to-SQL tasks (Hwang et al., 2019), we augment our model with BERT (Devlin et al., 2019). First, we concatenate the questions with all retrieved KB items as input for BERT to strengthen the connection between them. Then, we replace the word embeddings mentioned above with deep contextual representations from the last layer of BERT of each question token and each KB item, respectively. In a case where the total number of words in the retrieved KB items exceeds the maximum length constraint of BERT, we split these KB items into different blocks and encode them with the question separately (Gu et al., 2021).

2.3 Checker

Inspired by previous work (Liang et al., 2017; Chen et al., 2018; Wang et al., 2018), we design a pluggable module named *checker* to improve the decoding process by leveraging semantics of KB.

Instance-level Checking relies on the KB linkage information at the instance level (i.e., entities and their connected relations), which means that instance-level checking only deals with cases where the current action is a child node of action `set`→ join$_{\text{ent}}$(`rel`, `ent`) in the abstract syntax tree (AST). As illustrated in Fig. 4, when expanding the nonterminal `ent`, any retrieved KB entity can return a valid grammar rule such as `ent`→`m.04bmk` or `ent`→`m.04vd3`. However, only `m.04vd3` can pass the instance-level checking, since other candidates do not share direct links with the decoded relation `tv.tv_episode_segment.subjects`.

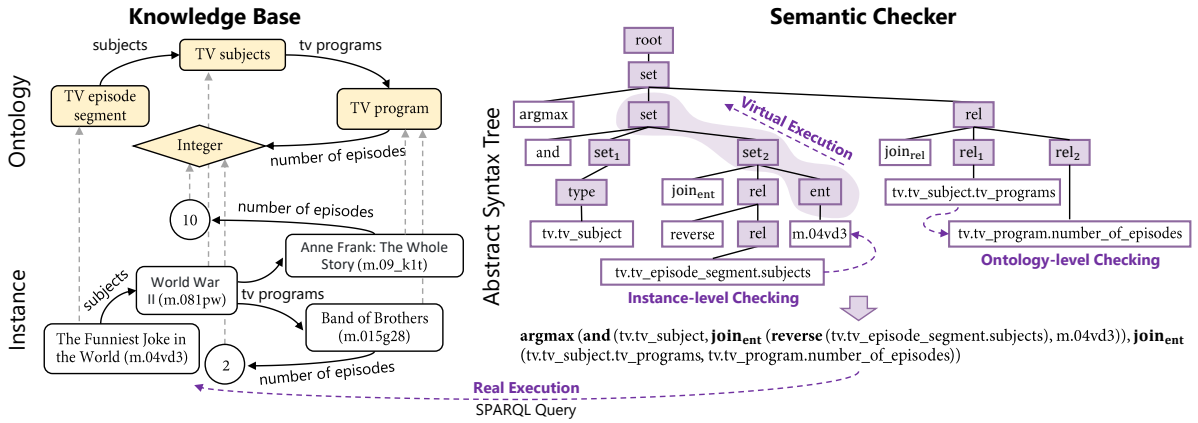

Figure 4: The illustration of four checking mechanisms in the check given the question "What was the subject of the TV show with the most number of episodes and featured on killer joke?"

Ontology-level Checking performs checking with the help of KB linkage information at the ontology level (i.e., types and bridging relations). Taking the right subtree presented in Fig. 4 as an example, when expanding the second `rel`, we employ ontology-level checking to determine its valid semantic scope. According to the semantics of the grammar rule $\texttt{set} \rightarrow \text{join}_{\text{rel}}(\texttt{rel}_1, \texttt{rel}_2)$, the type set of the head entity in $\texttt{rel}_2$ must overlap with the type set of the tail entity in $\texttt{rel}_1$, by which the candidate `rel`→`tv.tv_program.number_of_episodes` is selected. Although ontology-level checking applies to more situations than instance-level, it is weaker in terms of checking effectiveness and needs constraints of high coverage.

Real Execution When decoding reaches the end, an action sequence can be converted into a logical form, and finally into a SPARQL query. As depicted in Fig. 4, the real execution simply takes the final SPARQL query and tries to execute it over KB. If the query cannot be executed successfully, or the result is empty, it means that the corresponding action sequence cannot meet the executable requirement. In practice, we utilize the real execution to check all complete action sequence candidates searched by the beam search procedure, until an action sequence passes checking.

Virtual Execution The real execution cannot intervene in the middle of program generation, which leads to candidates of low quality in the final beam (e.g., no candidate can be executed). Meanwhile, since real execution relies on SPARQL, it is relatively slow as SPARQL queries are executed over tremendous (e.g., millions) entities with multi-hop relations. Instead, we propose virtual execution to alleviate these issues. As illustrated in Fig. 4, when a sub-program (i.e., shaded in purple) is fully produced, virtual execution is triggered to run bottom-up and check if the virtual answer set is empty. If so, the action sequence is removed from the beam. At each node, this virtual execution performs according to the program function semantics at the ontology-level. Taking $\texttt{rel} \rightarrow \text{reverse}(\texttt{rel})$ as an illustration, the virtual answer is obtained by reversing each tuple (head entity type, end entity type) in `rel`. Such virtual execution is very fast since the ontology only contains thousands of relations and types. Meanwhile, it can prune programs earlier; before the real execution.[15]

3 Experiments

3.1 Datasets and Metrics

GrailQA (Gu et al., 2021) is a challenging crowd-sourced KBQA dataset containing 64,331 questions involving up to 4 relations. This dataset is created to evaluate three levels of generalization scenarios in KBQA: *i.i.d.*, *compositional*, and *zero-shot*, which account for 25%, 25%, and 50% of the test set, respectively. We refer readers to Gu et al. (2021) for more details.

WebQuestionsSP (WebQSP) (Yih et al., 2016) is a popular KBQA dataset with 4,937 questions, requiring up to 2-hop relation path inference. Originally it splits into 3,298 questions as train set and 1,639 questions as test set. We randomly sample 200 questions from the train set as a dev set.

On GrailQA, we use official evaluation metrics: *exact match accuracy* (**EM**) and **F1**. Consistent

[15] More details are described in the Appendix.

	Overall		I.I.D.		Compositional		Zero-shot	
	EM	**F1**	**EM**	**F1**	**EM**	**F1**	**EM**	**F1**
Query-Graph Generation methods								
QGG (Lan and Jiang, 2020)	–	36.7	–	40.5	–	33.0	–	36.6
GloVe + RANKING (Gu et al., 2021)	39.5	45.1	62.2	67.3	40.0	47.8	28.9	33.8
BERT + RANKING (Gu et al., 2021)	50.6	58.0	59.9	67.0	45.5	53.9	**48.6**	**55.7**
Transduction-based methods								
GloVe + TRANSDUCTION (Gu et al., 2021)	17.6	18.4	50.5	51.6	16.4	18.5	3.0	3.1
BERT + TRANSDUCTION (Gu et al., 2021)	33.3	36.8	51.8	53.9	31.0	36.0	25.7	29.3
Ours	**58.1**	**65.3**	**84.4**	**87.5**	**61.5**	**70.9**	44.6	52.5
– Checker	41.6	44.2	73.2	74.5	43.4	48.3	26.2	28.4

Table 2: EM and F1 results on the hidden test set of GrailQA.

Method	**F1**	**Hits@1**
IR-based methods		
EmbedKGQA*♡ (Saxena et al., 2020)	–	72.5
EmbedKGQA* (Saxena et al., 2020)	–	66.6
PullNet* (Sun et al., 2019)	–	68.1
GRAFT-Net* (Sun et al., 2018)	62.8	67.8
Query-Graph Generation methods		
GrailQA RANKING*♡ (Gu et al., 2021)	67.0	–
STAGG♡ (Yih et al., 2015)	69.0	–
Topic Units♡ (Lan et al., 2019)	67.9	–
TextRay♡ (Bhutani et al., 2019)	60.3	–
QGG♡ (Lan and Jiang, 2020)	74.0	–
UHop (Chen et al., 2019)	68.5	–
Transduction-based methods		
NSM♡ (Liang et al., 2017)	69.0	–
Ours*	74.7	74.6
– Checker	62.0	61.7
Ours	71.0	71.6
– Checker	56.9	57.4

Table 3: F1 and Hits@1 results on WebQSP. * denotes using oracle entity linking annotations. ♡ denotes using fixed number of hops assumption.

with previous work, we use **F1** and **Hits@1** as evaluation metrics on WebQSP.

3.2 Implementation Details

We implemented our model based on PyTorch (Paszke et al., 2019) and AllenNLP (Gardner et al., 2018). With respect to BERT, we utilize the uncased BERT-base model from the Transformers library (Wolf et al., 2020). In training, we employed the Adam optimizer (Kingma and Ba, 2015). The learning rate is set to 1e-3, except for BERT, which is set to 2e-5. Our model training time on a single Tesla V100 is approximately 20h[16].

As for dense retriever, on GrailQA dataset, we retrieve top-100 type items and top-150 relation items. On WebQSP dataset, we retrieve top-200 type items and top-500 relation items.

[16]Due to space limitation, we put the detailed hyperparameters setting in the Appendix.

3.3 Baseline Models

We compare our model with previous state-of-the-art models on GrailQA (Lan and Jiang, 2020; Gu et al., 2021) and WebQSP (Liang et al., 2017; Sun et al., 2019; Saxena et al., 2020; Lan and Jiang, 2020). Notably, both TRANSDUCTION and RANKING models proposed by Gu et al. (2021) on GrailQA can be based on either GloVe (Pennington et al., 2014) or BERT (Devlin et al., 2019). We compare with them under all settings.

3.4 Results

We test ReTraCk with two configurations, with or without Checker. As shown in Table 2, ReTraCk significantly outperforms the previous SOTA model BERT + RANKING (F1 +7.3, EM +7.5) and achieves an improvement (F1 +28.5, EM + 24.8) over the previous best transduction-based model BERT + TRANSDUCTION on GrailQA.

Table 3 shows model performance on WebQSP. Given predicted entities, our model outperforms previous models (except for QGG (Lan and Jiang, 2020)) and even outperforms these models with oracle entities: GRAFT-Net, PullNet, and EmbedKGQA. Given oracle entities, the performance of our model further boosts to 74.7 F1, which shows the potential gains with a better entity linker.

While most SOTA models constrain their answer space by assuming a fixed number of hops, we conduct experiments on both datasets without such assumptions, which simulates real world scenarios. QGG works well on WebQSP by accessing the KB via SPARQL when generating the query graph at each step. However, as noted in Gu et al. (2021), extending QGG to consider 3-hops relations on GrailQA will take a few months to train, which is time consuming. It works poorly on GrailQA

Example	Right
Query: which journal did don slater serve as editor on the editor in chief? **Predict**: (AND book.journal (JOIN book.periodical.editorial_staff (AND (JOIN book.editorial_tenure.editor m.05ws_t6) (JOIN book.editorial_tenure.title m.02wk2cy)))) **Golden**: (AND book.journal (JOIN book.periodical.editorial_staff (AND (JOIN book.editorial_tenure.editor m.05ws_t6) (JOIN book.editorial_tenure.title m.02wk2cy))))	✓
Query: which exhibition has the same exhibition curator with venice biennale of architecture taiwan pavillion 2006? **Predict**: (AND exhibitions.exhibition (JOIN exhibitions.exhibition.curators (JOIN (R exhibitions.exhibition.curators) m.064dsyn))) **Golden**: (AND exhibitions.exhibition (JOIN (R exhibitions.exhibition_curator.exhibitions_curated) (JOIN exhibitions.exhibition_curator.exhibitions_curated m.064dsyn)))	✓
Query: how is surface density measured in international system of units? **Predict**: (AND measurement_unit.unit_of_density (JOIN measurement_unit.unit_of_density.measurement_system m.0c13h)) **Golden**: (AND measurement_unit.unit_of_surface_density (JOIN measurement_unit.unit_of_surface_density.measurement_system m.0c13h))	×

Table 4: Case Study. Three examples from the development set of GrailQA dataset. Brown words denote semantically equivalent schema items. Red words denote inconsistent schema items.

under 2-hop assumption.

By removing the checker module, the performance drops 21.1 and 14.1 F1 points on GrailQA and WebQSP respectively, which demonstrates the significant effectiveness of the checker. Except for QGG mentioned above, GrailQA RANKING model takes an average 115.5 seconds[17] to process one query, which is not applicable for online systems. In contrast, ReTraCk takes only 1.62 seconds per query on average at its current implementation which demonstrates its efficiency.

3.5 Case Study

To demonstrate ReTraCk's capability, we show three typical examples from the development set of GrailQA dataset in Table 4. In the first case, ReTraCk accurately links two mentions (*don slater* and *editor in chief*) in the query to corresponding entities (`m.05ws_t6` and `m.02wk2cy`) in Freebase. It also retrieves all necessary schema items (three relations and one type) via schema retriever. The transducer equipped with checker accurately understands the meaning of query and compose the complex logical form with five operators. The predicted logical form is exactly the same as the golden logical form. As for the second case, ReTraCk parses the query to a logical form which is semantically equivalent to the golden logical form, which demonstrates the existence of *program alias*. As for the third case, ReTraCk ignores the semantics conveyed by the word *surface* in the query, and selects wrong schema item `unit_of_density` instead of `unit_of_surface_density`. This example shows that our model sometimes only captures part of the semantics in the query and misses some span information.

[17]Data are derived from https://github.com/dki-lab/GrailQA

4 Conclusion

We present ReTraCk, a semantic parsing framework for KBQA. ReTraCk is flexible and efficient, achieving strong results on two distinct KBQA datasets. We hope that ReTraCk will be beneficial for future research efforts towards developing better KBQA systems.

Acknowledgements

We would like to thank Audrey Lin and Börje F. Karlsson for their constructive comments and useful suggestions, and all the anonymous reviewers for their helpful feedback. We also thank Yu Gu for evaluating our submissions on the test set of the GrailQA benchmark and sharing preprocessed data on GrailQA and WebQuestionsSP.

References

Abdalghani Abujabal, Rishiraj Saha Roy, Mohamed Yahya, and Gerhard Weikum. 2017. QUINT: Interpretable question answering over knowledge bases. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 61–66, Copenhagen, Denmark. Association for Computational Linguistics.

Hannah Bast and Elmar Haussmann. 2015. More accurate question answering on freebase. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1431–1440.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on Freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, Seattle, Washington, USA. Association for Computational Linguistics.

Nikita Bhutani, Xinyi Zheng, and H. V. Jagadish. 2019. Learning to answer complex questions over knowledge bases with query composition. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3-7, 2019*, pages 739–748. ACM.

Elena Cabrio, Julien Cojan, Alessio Palmero Aprosio, Bernardo Magnini, Alberto Lavelli, and Fabien Gandon. 2012. Qakis: an open domain qa system based on relational patterns. In *International Semantic Web Conference, ISWC 2012*.

Bo Chen, Le Sun, and Xianpei Han. 2018. Sequence-to-action: End-to-end semantic graph generation for semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 766–777. Association for Computational Linguistics.

Zi-Yuan Chen, Chih-Hung Chang, Yi-Pei Chen, Jijnasa Nayak, and Lun-Wei Ku. 2019. Uhop: An unrestricted-hop relation extraction framework for knowledge-based question answering. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 345–356. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

Evgeniy Gabrilovich, Michael Ringgaard, and Amarnag Subramanya. 2013. Facc1: Freebase annotation of clueweb corpora, version 1 (release date 2013-06-26, format version 1, correction level 0). *Note: http://lemurproject. org/clueweb09/FACC1/Cited by*, 5:140.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew E. Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. Allennlp: A deep semantic natural language processing platform. *CoRR*, abs/1803.07640.

Yu Gu, Sue Kase, Michelle Vanni, Brian Sadler, Percy Liang, Xifeng Yan, and Yu Su. 2021. Beyond iid: three levels of generalization for question answering on knowledge bases. *The Web Conference*.

Daya Guo, Duyu Tang, Nan Duan, Ming Zhou, and Jian Yin. 2018. Dialog-to-action: Conversational question answering over a large-scale knowledge base. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2946–2955.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. 2020. Poly-encoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. *CoRR*, abs/1902.01069.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6769–6781. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Yunshi Lan and Jing Jiang. 2020. Query graph generation for answering multi-hop complex questions from knowledge bases. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 969–974, Online. Association for Computational Linguistics.

Yunshi Lan, Shuohang Wang, and Jing Jiang. 2019. Knowledge base question answering with topic units. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 5046–5052. ijcai.org.

Chen Liang, Jonathan Berant, Quoc V. Le, Kenneth D. Forbus, and Ni Lao. 2017. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 23–33. Association for Computational Linguistics.

Qian Liu, Bei Chen, Jiaqi Guo, Jian-Guang Lou, Bin Zhou, and Dongmei Zhang. 2020. How far are we from effective context modeling? an exploratory study on semantic parsing in context. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 3580–3586. ijcai.org.

Gaurav Maheshwari, Priyansh Trivedi, Denis Lukovnikov, Nilesh Chakraborty, Asja Fischer, and Jens Lehmann. 2019. Learning to rank query graphs for complex question answering over knowledge graphs. In *The Semantic Web - ISWC 2019 - 18th International Semantic Web Conference, Auckland, New Zealand, October 26-30, 2019, Proceedings, Part I*, volume 11778 of *Lecture Notes in Computer Science*, pages 487–504. Springer.

Pierre-Emmanuel Mazaré, Samuel Humeau, Martin Raison, and Antoine Bordes. 2018. Training millions of personalized dialogue agents. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2775–2779, Brussels, Belgium. Association for Computational Linguistics.

Laurel Orr, Megan Leszczynski, Simran Arora, Sen Wu, Neel Guha, Xiao Ling, and Christopher Re. 2021. Bootleg: Chasing the tail with self-supervised named entity disambiguation. *The Conference on Innovative Data Systems Research (CIDR)*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4498–4507, Online. Association for Computational Linguistics.

Mike Schuster and Kuldip K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.*, 45(11):2673–2681.

Haitian Sun, Tania Bedrax-Weiss, and William W. Cohen. 2019. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 2380–2390. Association for Computational Linguistics.

Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. 2018. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4231–4242, Brussels, Belgium. Association for Computational Linguistics.

Chenglong Wang, Po-Sen Huang, Alex Polozov, Marc Brockschmidt, and Rishabh Singh. 2018. Execution-guided neural program decoding. *CoRR*, abs/1807.03100.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Ledell Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. 2020. Scalable zero-shot entity linking with dense entity retrieval. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6397–6407. Association for Computational Linguistics.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1321–1331. The Association for Computer Linguistics.

Wen-tau Yih, Matthew Richardson, Christopher Meek, Ming-Wei Chang, and Jina Suh. 2016. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 201–206.

Pengcheng Yin and Graham Neubig. 2018. TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018: System Demonstrations, Brussels, Belgium, October 31 - November 4, 2018*, pages 7–12. Association for Computational Linguistics.

A Entity Linker

The entity linker used in this paper follows the typical pipeline that consists of three sub-modules: *mention detection*, *candidate generation* and *entity disambiguation*. Following the previous work Gu et al. (2021), we use a BERT-based NER system[18] to detect the entity mentions and literals (e.g., numerical values and datetime) in the question. Then we generate candidate entities along with their prior probability using an alias map mined from the KB and FACC1 (Gabrilovich et al., 2013), a large entity linking corpus.

For entity disambiguation, we adopt the state-of-the-art neural entity disambiguation model BOOTLEG (Orr et al., 2021)[19] which shows decent generalization performance over long-tail entities. In BOOTLEG, each entity is represented with three levels information: its unique entity embedding, attached types' embedding and relations' embedding, and leverage BERT (Devlin et al., 2019) to encode the context. Besides, we also combine the prior score from the candidate generation step and the context compatibility score from BOOTLEG with two fully connected layers of 100 hidden units and ReLU non-linearities. Note that existing KBQA datasets do not provide the mention boundary annotations. We generated the distantly supervised training data for both named entity recognition and entity disambiguation by aligning the natural language question with entities' observed aliases mined from the candidate generation step.

We evaluate the performance of our entity linker on GrailQA dev set and WebQSP test set. We compare its performance with the following baselines: 1) **Aqqu** (Bast and Haussmann, 2015) which is a rule based entity linker using linguistic and entity popularity features. 2) **GrailQA** (Gu et al., 2021) which is a prior baseline. 3) **Prior** which is a prior baseline implemented by us. 4) **BOOTLEG** (Orr et al., 2021) which is trained using distantly aligned question answering data. 5) **BOOTLEG + Prior** which is the full disambiguation model used in this paper.

As you can see from Table 5, our Prior performs slightly better than the GrailQA (Gu et al., 2021)'s Prior by 0.8 F1 points on GrailQA. What's interesting is that the BOOTLEG trained with GrailQA data is even inferior than Prior baseline by 4.8 F1 points. However, BOOTLEG + Prior improves over BOOTLEG and Prior by 4.4 F1 points and 9.2 F1 points respectively. The above experiment results show that the prior feature is very important and orthogonal to the BOOTLEG model in the question entity linking. As shown in Table 6, similar conclusions can be derived from the experiment results on WebQSP dataset. Compared with experiments on GrailQA, the performance of BOOTLEG is lower with only 58.5 F1 score and the improvement of BOOTLEG + Prior over Prior is reduced by 1.7 F1 points. This is mainly because the size of training data of WebQSP (3,098 instances) is much smaller than GrailQA (44,337 instances) which limits the learning of BOOTLEG model.

B Dense Schema Retriever

In principle, the encoders can be implemented by any neural networks (Karpukhin et al., 2020). We use two independent BERT-base encoders (Devlin et al., 2019).

Training The goal of training the encoders is to create a vector space such that relevant schema

[18] https://github.com/kamalkraj/BERT-NER
[19] http://ai.stanford.edu/blog/bootleg/

	Overall	I.I.D.	Com	Zero
Aqqu (Bast and Haussmann, 2015)	14.5	–	–	–
GrailQA (Test set) (Gu et al., 2021)	75.2	–	–	–
GrailQA (Dev set) (Gu et al., 2021)	72.2	–	–	–
Prior	73.0	78.6	74.9	69.7
BOOTLEG	68.2	78.6	70.6	62.5
BOOTLEG + Prior	77.4	86.6	81.3	71.9

Table 5: F1 scores of various Entity linking models on GrailQA dev set.

	Precision	Recall	F1
Prior	81.2	81.7	81.4
BOOTLEG	58.3	58.6	58.5
BOOTLEG + Prior	82.8	83.3	83.1

Table 6: Entity linking performance (set level metric P/R/F1) on WebQSP test set.

items get higher scores with the given question. For each pair of question and schema item (q_i, s_i) in a batch of size B, the loss is computed as:

$$L(q_i,s_i) = -s(q_i, s_i) + \log \sum_{j=1}^{B} \exp(s(q_i, s_j)). \quad (4)$$

In-batch negatives have shown to be effective for learning a bi-encoder architecture (Karpukhin et al., 2020). To use in-batch negatives, we separate relevant schema items of the same question into different mini-batches. In this way, there are B training instances in each batch and $B-1$ negative candidates for each question.

Dense Schema Retriever v.s. Neighbor Schema Retriever To prune the decoding vocabulary space, Gu et al. (2021) retrieves schema items that are reachable by anchor entities within 2-hops in KB, which is named after *neighbor schema retriever*. In this section, we compare the performance of dense schema retriever proposed in this work with the neighbor schema retriever. Fig. 5 shows the recall of the schema items with respect to top-k retrieved candidates on GrailQA dev set. Neighbor schema retriever obtains 69.2% type recall with an average of 112.1 candidate items while dense schema retriever achieves 73.3% recall with only 2 candidates and 98.5% recall with 100 candidates. Similar trends can be found in the relation recall curve in Fig. 5. Dense schema retriever not only improves the recall of schema items, but also reduces the candidate size, which benefits the downstream transducer model.

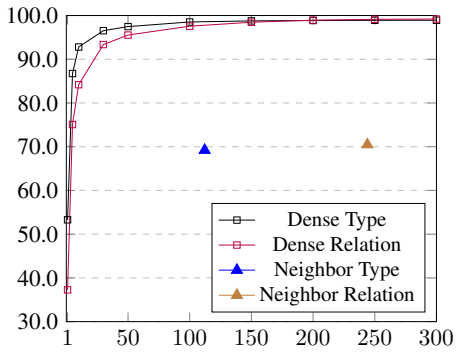

Figure 5: Top-k recall of schema retriever on GrailQA dev set.

C Checking Procedure

The usage of 4 functions (`instance_checking`, `type_checking`, `virtual_execution` and `real_execution`) are explained in the paper. Here we present an algorithm to introduce the checking procedure better, as show in Algorithm 1.

D Detailed Hyper-parameter Setting

Entity Linker For the BERT-based NER model, we use the uncased BERT-base model from the Transformers library trained with AdamW optimizer (learning rate: 5e-5) for 5 epochs. For the entity disambiguation model, we use the default parameters from BOOTLEG. On GrailQA dataset, we use the uncased BERT-base model trained with SparseDenseAdam optimizer implemented by BOOTLEG (learning rate: 1e-4) for 5 epochs. We add two fully connected layers of 100 hidden units and ReLU non-linearities to combine BOOTLEG and the prior score feature. The entity embedding size is set to 256, type and relation embedding size is set to 128. The entity embedding mask percentage is set to 0.8. On the smaller dataset WebQSP, except training with a larger number of epochs (50), and the embedding size is set to 64 to avoid overfitting, everything is the same as the model on GrailQA. Through our experiments, we select the best model based on the F1 score on

Algorithm 1 Checking Process

Input: valid action candidates $\mathcal{C}$, decoded logical form beam $\mathcal{L}$, knowledge base $\mathcal{K}$
Output: logical form beam for the next step $\hat{\mathcal{L}}$
Algorithm:
$\hat{\mathcal{L}} = \emptyset$
Procedure `static_checking`$(\mathcal{C}, \mathcal{L}, \mathcal{K})$
  for each action sequence $\mathbf{s}$ in $\mathcal{L}$ **do**
    for each valid action candidate c in $\mathcal{C}$ **do**
      if not `instance_checking`$(\mathbf{s}, c)$ **then**
        continue
      if not `ontology_checking`$(\mathbf{s}, c)$ **then**
        continue
      ▷ novel checking techniques can be added here
      $\hat{\mathbf{s}} \leftarrow \langle \mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_{|\mathbf{s}|}, c \rangle$
      $\hat{\mathcal{L}} \leftarrow \hat{\mathcal{L}} \cup \{\hat{\mathbf{s}}\}$
$\hat{\mathcal{L}} =$ `kbest_beam`$(\hat{\mathcal{L}}, k)$ ▷ keep top k scoring candidates in $\hat{\mathcal{L}}$
Procedure `dynamic_checking`$(\hat{\mathcal{L}})$
  for each action sequence $\hat{\mathbf{s}}$ in $\hat{\mathcal{L}}$ **do**
    $\tau = \hat{\mathbf{s}}_{|\hat{\mathbf{s}}|}$
    While τ corresponds to a full sub-program **do**
      $r =$ `virtual_execution`(τ)
      if not r **then**
        $\hat{\mathcal{L}} \leftarrow \hat{\mathcal{L}}$ remove $\hat{\mathbf{s}}$
        break
      $\tau \leftarrow$ parent node of τ in AST tree
    if $\hat{\mathbf{s}}$ arrives at the end **then**
      $r =$ `real_execution`$(\hat{\mathbf{s}})$
      if r **then**
        $\hat{\mathcal{L}} \leftarrow \{\hat{\mathbf{s}}\}$ ▷ only keep the first executable $\hat{\mathbf{s}}$
        break
return $\hat{\mathcal{L}}$

dev set of each dataset. We pass top-3 and top-5 candidate entities per entity mention to the downstream transducer model on GrailQA and WebQSP dataset respectively.

Dense Schema Retriever We use the uncased BERT-base model from the Transformers library trained with AdamW optimizer (learning rate: 1e-5) for 10 epochs. We select the best model based on the recall of schema items on the dev set of each dataset. On GrailQA dataset, we retrieve top-100 type items and top-150 relation items. On WebQSP dataset, we retrieve top-200 type items and top-500 relation items.

Parser We implement our model based on PyTorch and AllenNLP. With respect to BERT, we use the uncased BERT-base model from Transformers library. In training, we employ the Adam optimizer. The learning rate of our model is set to 1e-3, except for BERT, which is set to 2e-5. The training time of our model on single Tesla V100 is approximately 20 hours. We select the best model based on the exact match ratio between the predicted logical form and golden logical form.