A Tale of Two Linkings: Dynamically Gating between Schema Linking and Structural Linking for Text-to-SQL Parsing

Sanxing Chen[◇] **Aidan San**[◇] **Xiaodong Liu**[♣] **Yangfeng Ji**[◇]
[◇]University of Virginia
[♣]Microsoft Research
{sc3hn, aws9xm, yangfeng}@virginia.edu
xiaodl@microsoft.com

Abstract

In Text-to-SQL semantic parsing, selecting the correct entities (tables and columns) for the generated SQL query is both crucial and challenging; the parser is required to connect the natural language (NL) question and the SQL query to the structured knowledge in the database. We formulate two linking processes to address this challenge: *schema linking* which links explicit NL mentions to the database and *structural linking* which links the entities in the output SQL with their structural relationships in the database schema. Intuitively, the effectiveness of these two linking processes changes based on the entity being generated, thus we propose to dynamically choose between them using a gating mechanism. Integrating the proposed method with two graph neural network-based semantic parsers together with BERT representations demonstrates substantial gains in parsing accuracy on the challenging Spider dataset. Analyses show that our proposed method helps to enhance the structure of the model output when generating complicated SQL queries and offers more explainable predictions.

1 Introduction

Semantic parsing, which aims at mapping natural language (NL) utterances to computer understandable logic forms or programming languages, has been an active research topic in the field of natural language processing (NLP) for decades (Zettlemoyer and Collins, 2005; Liang et al., 2011). Although a variety of logic forms have been studied by researchers, Text-to-SQL has particularly attracted a large amount of attention due to the desire of natural language interfaces to database (NLIDB) (Warren and Pereira, 1982; Zelle and Mooney, 1996; Dong, 2019) for both scientific and industrial reasons. Recently, there is a growing interest in neural based Text-to-SQL semantic parsing, thanks to the development of new evaluation paradigms and datasets (Iyer et al., 2017; Zhong et al., 2017; Yu et al., 2018; Yu et al., 2019).

Text-to-SQL parsing requires strict *structured prediction* due to its application scenario where the output SQL will be sent to an executor program directly. To enhance the capacity of an auto-regressive model to capture structural information, current state-of-the-art semantic parsers usually adopt a grammar-based decoder (Xiao et al., 2016; Yin and Neubig, 2017; Krishnamurthy et al., 2017). Rather than directly generating the tokens in a traditional sequence-to-sequence manner, grammar-based decoders produce a sequence of production rules to construct an abstract syntax tree (AST) of the corresponding SQL. As the grammar constraints narrows down the search space to only grammatically valid ASTs, those parsers can usually generate well-formed SQL skeletons (Guo et al., 2019; Bogin et al., 2019a).

However, it is still difficult for current state-of-the-art models to fill in the skeletons with semantically correct entities, especially when they are required to generalize to unseen DB schemas (Yu et al., 2018; Suhr et al., 2020). To predict the correct entity, the model should have a database (DB) schema grounded understanding of the NL question, which means that the model should be able to jointly learn the semantics in the NL question and the structured knowledge in a given database. We formulate two types of entity generation problems, which can be addressed by the following two linking processes respectively.

Proceedings of the 28th International Conference on Computational Linguistics, pages 2900–2912
Barcelona, Spain (Online), December 8-13, 2020

	Q: *For each continent, list its id, name, and how many countries it has?*
1	`SELECT t1.contid, t1.continent, COUNT(*) FROM continents AS t1 JOIN countries AS t2 ON t1.contid = t2.continent GROUP BY t1.contid;`
	Q1: *What is the average, minimum, and maximum age of all singers from France?*
2	Q2: *What is the average, minimum, and maximum age for all French singers?*
	`SELECT AVG(age), MIN(age), MAX(age) FROM singer WHERE country = 'France';`
	Q: *What is the first name and gender of the all the students who have more than one pet?*
3	`SELECT t1.fname, t1.sex FROM student AS t1 JOIN has_pet AS t2 ON t1.stuid = t2.stuid GROUP BY t1.stuid HAVING COUNT(*) > 1`

Table 1: Several examples that are taken from the Spider dataset. Entity mentions are underlined in NL questions. Q1 and Q2 are paraphrases of each other which should lead to the same SQL result. Wavy underline indicates the mention can only be resolved by linking to a cell value or common sense reasoning.

Schema linking. Schema linking (Guo et al., 2019; Wang et al., 2020) is an instance of entity linking (Shen et al., 2014) in the context of linking to relational DB schema. Text-to-SQL semantic parsers should learn to recognize an entity mention in the NL question and link it to the corresponding unique entity in the DB schema. This task can be challenging due to the diversity and ambiguity NL mentions. However, in practice, the solution is often relatively easy when a particular entity is well realized with similar wording in both the NL question and DB schema. As shown in Table 1, in Spider (Yu et al., 2018), the underlined mentions can almost exactly match the corresponding schema entities. Therefore, current state-of-the-art parsers normally address this problem with simple string matching or embedding matching modules.

Structural linking. While some entities are generated because they are mentioned in the NL question, others can be generated because of their role as special functional components in SQL, *e.g.*, `contid` and `stuid` in the `ON` clauses of the first and third examples in Table 1. These entities usually cannot find their corresponding mentions in the NL question but can be induced by the structural constraints of SQL. Such phenomenons are generally referred to as the *structural mismatch* between NL and formal languages (Kwiatkowski et al., 2013; Berant and Liang, 2014). This process frequently occurs when generating complex SQL queries. We propose to treat this entity generation process as finding a structural link between current candidates and past generated entities. Although previous work has considered some simple structural constraints (Guo et al., 2019), to the best of our knowledge, we are the first to formally describe this process.

While schema linking and structural linking can complement each other (*e.g.*, the entity used by the `GROUP BY` clause needs to be a column in a previously selected table and also has its mention in the NL question), they actually address different types of problems. In most cases, *e.g.*, the examples in Table 1 described before, a decoder may need to discriminate one from another for better generation performance.

In this work, we propose to use a *dynamic gating* mechanism to serve as a switch between the two linking processes. Schema linking can be implemented as the decoder attending to the encoded representations of the NL question to find an entity mention, then locating the corresponding entity from the schema. On the other hand, in structural linking, our decoder performs self-attention over those past decoder states where an entity was generated and further makes a decision between copying one of the previously generated entities or taking one of its linked entities (*e.g.*, the foreign key of a previously joined table) based on the schema structure. From the model's perspective, it can also be viewed as a memory pointer network that enhances the structured prediction ability of an auto-regressive model, and the dynamic gating determines when to emphasize this enhancement. Our proposed method can be easily applied to most semantic parsers as long as their decoders explicitly or implicitly have two modules that deal with schema linking and structural linking.

We integrate the dynamic gating technique to two state-of-the-art Text-to-SQL parsing models (Bogin et al., 2019a; Bogin et al., 2019b) and further augment them with pretrained BERT (Devlin et al., 2019) word representations. We evaluate our model on the Spider dataset which is challenging because of its

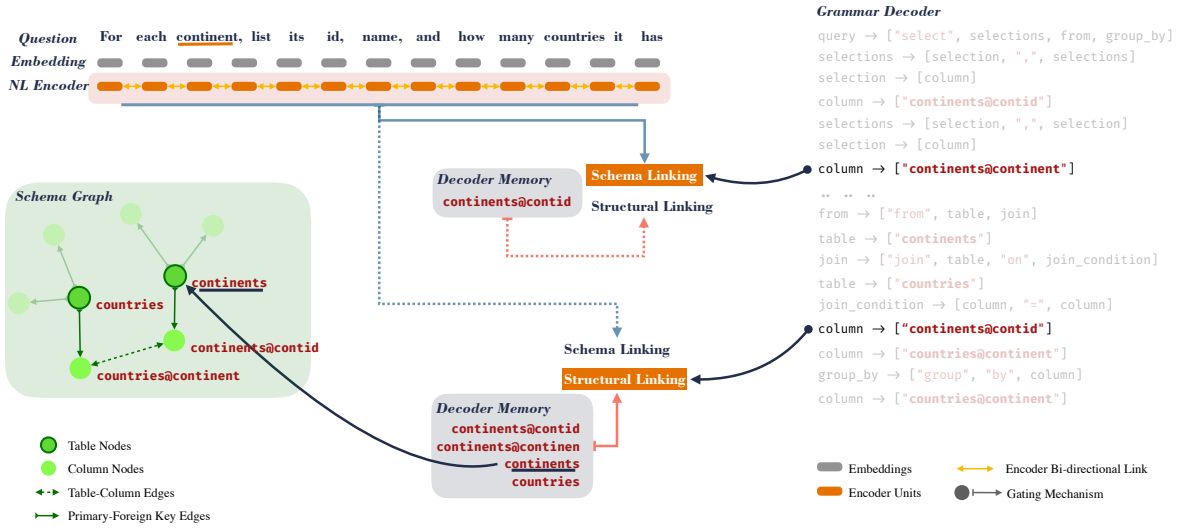

Figure 1: An illustration of our proposed method when running the first example shown in Table 1. This figure shows two independent entity generation procedures, the top one favors schema linking while the bottom one favors structural linking. Some details are omitted for the sake of simplicity. Grammars are simplified to fit in the limited space, readers are encouraged to refer to (Krishnamurthy et al., 2017) for details.

cross-domain setting where a model needs to generalize to not only complex SQL but also unseen DBs. Experimental results show that our proposed method consistently yields an improvement of more than 3% on exact set matching accuracy and sees the most benefits when generating complex SQL. Further analysis confirms that the models are dynamically switching between the two linking processes.

2 Approach

In this section, we first formulate the Text-to-SQL semantic parsing task in §2.1. We will then describe the details of our proposed method in §2.2.

2.1 Text-to-SQL Semantic Parsing

The task of Text-to-SQL semantic parsing is to predict a SQL query $\mathcal{S}$ based on input $(\mathcal{Q}, \mathcal{G})$ where $\mathcal{Q} = \{q_1, \ldots, q_{|\mathcal{Q}|}\}$ is the NL question and $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is the DB schema being queried. In the schema, $\mathcal{V} = \{(e_1, t_1), \ldots, (e_{|\mathcal{V}|}, t_{|\mathcal{V}|})\}$ is a set which usually contains two types of entities (*i.e.*, tables and columns[1]) and their textual descriptions (*i.e.*, table names and column names), while $\mathcal{E} = \{(e_1^{(s)}, e_1^{(t)}, l_1), \ldots, (e_{|\mathcal{E}|}^{(s)}, e_{|\mathcal{E}|}^{(t)}, l_{|\mathcal{E}|})\}$ contains the relations l between source entity $e^{(s)}$ and target entity $e^{(t)}$, *e.g.*, table-column relationships, foreign-primary key relationships,[2] etc. The output $\mathcal{S} = \{a_1, \ldots, a_{|\mathcal{S}|}\}$ is a sequence of decoder actions which further compose an AST of SQL.

Typical state-of-the-art Text-to-SQL parsers, consist of three components: a NL encoder, a schema encoder and a grammar decoder (Guo et al., 2019; Bogin et al., 2019a).

The **NL encoder** takes the NL question tokens $\mathcal{Q}$ as input, maps them to word embeddings $\boldsymbol{E}_{\mathcal{Q}}$, then feeds them to a Bi-LSTM (Hochreiter and Schmidhuber, 1997). The hidden states of the Bi-LSTM serve as the contextual word representation of each token.

The **schema encoder** takes $\mathcal{G}$ as input and builds a relation-aware entity representation for every entity in the schema. The initial representation of an entity is a combination of its words embeddings and type information. Then self-attention (Zhang et al., 2019; Shaw et al., 2019) or graph-based models (Bogin et al., 2019a; Wang et al., 2020) are utilized to exploit the relational information between each pair of

[1]Note that columns may have more fine-grained types like binary, numeric, string and date/time, primary/foreign etc.

[2]In SQL, a foreign key in one table is used to refer to a primary key in another table to link these two tables together for joint queries.

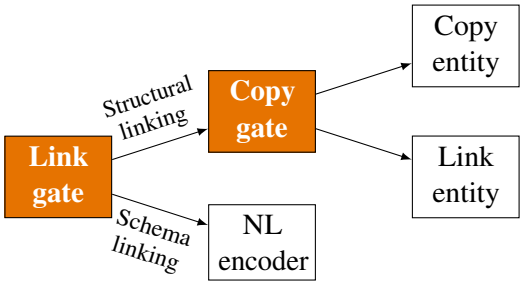

Figure 2: An illustration of our gating mechanism.

entities from the DB schema, thus produce the final representation of all entities $\boldsymbol{H}_{\mathcal{V}} \in \mathbb{R}^{|\mathcal{V}| \times \dim}$. We will detail this in §3.

Finally, a **grammar decoder** (Xiao et al., 2016; Yin and Neubig, 2017; Krishnamurthy et al., 2017) generates an AST of output SQL in a depth-first order. The decoder is typically an auto-regressive model (*e.g.*, LSTM) which estimates the probability of generating an action sequence.

There are *two cases* of using actions. Depending a specific case, an action is either (i) producing a new production rule to unfold the leftmost non-terminal node in the AST, or (ii) generating an entity (*e.g.*, a table or a column) from the DB schema if it is required by last output production rule. In the former case, at step t, the decoder normally uses its hidden states $\boldsymbol{h}_t$ to retrieve a context vector $\boldsymbol{c}_t$ from the NL encoder. Then an action embedding $\boldsymbol{a}_t$ is produced based on the concatenation of $\boldsymbol{h}_t$ and $\boldsymbol{c}_t$. This action embedding will directly predict a production rule from the target vocabulary which is a subset of a fixed number of production rules. For the latter case, the decoder needs to estimate a probability distribution over a schema-specific vocabulary under grammatical constraints which come from the structure of both the output SQL and the DB schema, as well as semantic constraints implied in the NL question.

2.2 Dynamic Gating

In this paper, we focus on the decision made when the decoder is looking for an entity to fill in a slot (*i.e.*, case (ii) in the last paragraph). Our decoder predicts the entity based on a mixed probability model consisting of two processes:

- **Schema linking.** The decoder attends to the output of the NL encoder (which can be seen as selecting a most relevant NL mention), then finds the corresponding entity based on string-matching or embedding-matching results.
- **Structural linking.** The decoder self-attends to the output states from those previous decoding steps which have generated entities, then finds another entity which is structurally linked to the attended entity.

The choice between them is controlled by a gating mechanism called the *link gate*.

Formally, the marginal probability of generating an entity e is defined as follows:

$$\Pr(a_t = e) = \Pr(e|\textsc{Schm}) \Pr(\textsc{Schm}) + \Pr(e|\textsc{Strct}) \Pr(\textsc{Strct}) \tag{1}$$

where $\Pr(\textsc{Schm})$ and $\Pr(\textsc{Strct})$ are the probability of choosing schema linking and structural linking respectively. They are further computed as:

$$\rho_{\text{link}} = \text{Sigmoid}(\text{FF}(\boldsymbol{a}_t)) \tag{2}$$

$$\Pr(\textsc{Schm}) = \rho_{\text{link}} \tag{3}$$

$$\Pr(\textsc{Strct}) = 1 - \rho_{\text{link}} \tag{4}$$

Equation 2 stands for our proposed link gate which is computed by the action embedding $\boldsymbol{a}_t$. The reason for purely basing the gate value on $\boldsymbol{a}_t$ is that intuitively the choice between the two processes is about the

role of the current entity we want to generate. The role of an entity is determined by the SQL clause that contains it. Since $\boldsymbol{a}_t$ is directly used to predict a production rule in case (i), it should be able to capture this information. The link gate allows the decoder to dynamically choose between information from our two linking processes, and prevents them from interfering each other.

In practice, we model the probability of the schema linking process generating an entity mentioned in the NL question, namely $\Pr(e|\textsc{Schm})$, as a multiplication of the attention weights $\boldsymbol{\lambda} \in \mathbb{R}^{|\mathcal{Q}|}$ over the NL encoder outputs and a schema linking matrix $M \in \mathbb{R}^{|\mathcal{Q}|\times|\mathcal{V}|}$. The probability of the structural linking process, namely $\Pr(e|\textsc{Strct})$, is similarly computed by multiplying decoder self-attention weights and a structural linking matrix $T \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$.

The structural linking matrix T captures the relationship between every pair of entities given the relational DB schema. Common structural links include relations between a table and its columns, a table and its primary/foreign key, a primary key and one of its linked foreign keys in other tables, etc. There are also multi-steps links which are the combinations of the one-step links listed above. Note that there may not be a unique link between every pair of entities and some entities may not have a link between them at all. Meanwhile, an entity can link to itself which can be considered to be a special zero-step structural link. It is the only structural link modeled in most current Text-to-SQL semantic parsers.

We compute the structural linking score between an entity e_i and e_j by an additive attention mechanism (Bahdanau et al., 2015), as follows:

$$T_{i,j} = \mathbf{v}_\alpha^\top \tanh\left(\mathbf{W}_\alpha \left[\boldsymbol{e}_i; \boldsymbol{e}_j\right]\right) \tag{5}$$

where $\boldsymbol{e}_i$ and $\boldsymbol{e}_j$ are the corresponding entity representations retrieved from $\boldsymbol{H}_\mathcal{V}$, while $\mathbf{v}_\alpha$ and $\mathbf{W}_\alpha$ are both trainable parameters. Compared to the dot-product attention used in Bogin et al. (2019a) and Bogin et al. (2019b), the additive attention we used here is expected to capture more of the structural relationship between entity pairs rather than only the similarity of entity representations.

T is expected to capture all types of relationships between entities, but it can be overwhelmed by the large workload. So, we single out the zero-step relationship (*i.e.*, copying) and address it by another structural linking matrix $T_{\text{copy}} = \boldsymbol{I}$, which is trivially an identity matrix in this case. To choose between copying and other types of links, a *copy gate* (ρ_{copy}) is obtained in the same manner as to how we compute the link gate in Equation 2.

We use decoder self-attention to find the past generated entity which could have structural constraints on the entity that we currently want to generate.

$$\boldsymbol{\beta} = \text{Softmax}(\text{Attention}(\boldsymbol{a}_t, \boldsymbol{H}_m)) \tag{6}$$

$$\boldsymbol{H}_m = \{\boldsymbol{a}_i | i < t, a_i \in e\} \tag{7}$$

$\boldsymbol{H}_m$ is a memory matrix consisting of the action embeddings from every past decoding step which has generated an entity. In this way, we compute two sets of attention weights $\boldsymbol{\beta}_{\text{copy}}$ and $\boldsymbol{\beta}_{\text{link}}$ for copying and linking separately using the additive attention (Bahdanau et al., 2015) again. The motivation for separate attention weights is that these two linking patterns might need to attend to different generated entities.

Overall the probability of generating an entity via structural linking is modeled as:

$$\Pr(a_t = e_i|\textsc{Strct}) = \rho_{\text{copy}}(\boldsymbol{\beta}_{\text{copy}} T_{\text{copy}})_i + (1 - \rho_{\text{copy}})(\boldsymbol{\beta}_{\text{link}} T)_i \tag{8}$$

Finally, this probability is mixed with the probability of schema linking controlled by the link gate.

3 Model Implementation

In this section, we describe how we integrate our proposed method into a grammar decoder and leverage the entity representation from a GNN module.

We use the type constrained grammar decoder from Krishnamurthy et al. (2017). To predict a_t at time step t, the decoder will first obtain the context vector $\boldsymbol{c}_t$ from the NL encoder by performing dot-product

attention (Luong et al., 2015). Then the action embedding is generated by a feed-forward network taking the concatenation of decoder hidden state and context vector as input.

$$\boldsymbol{a}_t = \mathrm{FF}([\boldsymbol{h}_t; \boldsymbol{c}_t]) \tag{9}$$

$\boldsymbol{a}_t$ is used to predict the production rule or estimate the gate values in the entity generation process.

We adopt the idea from (Bogin et al., 2019a; Bogin et al., 2019b) to learn a schema relation-aware entity representation $\boldsymbol{H}_{\mathcal{V}}$ by a GNN module.[3] The initial embedding of each entity $h_e^{(0)}$ is defined as a non-linear transformation of the combination of its type embedding and the average over the word embeddings of its neighbors in the schema graph. In later time steps, the hidden state is updated by a gated recurrent unit (Cho et al., 2014; Li et al., 2016) as $h_e^{(l)} = \mathrm{GRU}\left(h_e^{(l-1)}, x_e^{(l)}\right)$, where the input $x_e^{(l)}$ is defined as a weighted summation over the hidden states of its neighbor entities:

$$x_e^{(l)} = \sum_{t \in \{\leftarrow, \rightarrow, \leftrightarrow\}} \sum_{(s,e,l) \in \mathcal{E}, l=t} \mathbf{W}_t h_s^{(l-1)} + b_t$$

They consider three edge types, *i.e.*, bidirectional edges between a table and its contained columns $\leftrightarrow$, unidirectional edges between a foreign key and a connected primary key $\leftarrow$ and its reverse version $\rightarrow$. Given a fixed GNN recurrence step L, we have the final hidden states of all the entities in the graph as the entity representation $\boldsymbol{H}_{\mathcal{V}} = \{h_e^{(L)} | (e,t) \in \mathcal{V}\}$. We also adopt their schema linking module to create a schema linking matrix M based on word embedding similarity and some simple manually design features (*e.g.*, editing distance and lemma). In their GLOBALGNN (Bogin et al., 2019b), an additional GNN and an auxiliary training loss are added to filter out irrelevant nodes in the graph, thus producing a better entity representation.

To augment our model with pretrained BERT embeddings, we follow Hwang et al. (2019) and Zhang et al. (2019) to feed the concatenation of NL question and the textual descriptions of DB entities to BERT and use the top layer hidden states of BERT as the input embeddings.

4 Experiments

We evaluate the effectiveness of our proposed method by integrating it into two state-of-the-art semantic parsers on the Spider dataset and further ablate out some components to understand their contributions.

4.1 Experiment Setup

We implement our model using PyTorch (Paszke et al., 2019) and AllenNLP (Gardner et al., 2018). For the GNN and GLOBALGNN models we revise and build upon the code released in (Bogin et al., 2019a; Bogin et al., 2019b). We re-ran the experiment and report the results on our re-implementation and found our results slightly improves upon their reported results. In BERT experiments, we use the base uncased BERT model with 768 hidden size provided by HuggingFace's Transformers library (Wolf et al., 2019). We follow the database split setting of Spider, where any databases that appear at testing time are ensured to be unseen at training time. Our code and models are available at `https://github.com/sanxing-chen/linking-tale`.

4.2 Experimental Results

The experimental results in Table 3 show that our proposed gating mechanism leads a substantial improvement on all the GNN, GLOBALGNN, and BERT baselines. Spider questions are divided into different levels of difficulty (hardness). Most of the improvements come from gains in complicated (*i.e.*, Medium, Hard and Extra Hard) SQL generation. Specially, we observe up to 13.8% gains in the Hard set when applying our method on the GLOBALGNN baseline. One major contribution comes from the partial matching F1 score of `IUEN` (*i.e.*, SQL clauses `INTERSECT`, `UNION`, `EXCEPT`, `NESTED` which only appear in Hard and Extra Hard levels) increasing from 25.4% to 39.7%. We also notice that the

[3] We choose these models because of the ability of GNNs to model various types of structural links, and they are among a few state-of-the-art models that are publicly available at the time of writing.

Hardness	# Example
Easy	250
Medium	440
Hard	174
Extra	170
All	1034

Table 2: Number of examples in the development set of Spider with different hardness levels associated with the SQL need to be generated.

Model	Acc.	Easy	Medium	Hard	Extra
GNN	47.7%	68.8%	51.8%	31.2%	22.9%
+ Ours	50.7%	66.4%	54.8%	42.8%	25.3%
GLOBALGNN	49.3%	69.2%	53.0%	32.8%	27.6%
+ Ours	52.8%	70.4%	55.7%	46.6%	25.9%
+ BERT	53.5%	76.0%	57.3%	36.2%	28.3%
+ BERT + Ours	57.6%	73.6%	61.6%	48.9%	32.9%

Table 3: Exact Set Matching Accuracy on SQL queries with different hardness levels in the development set of Spider. Greatest improvements in the Hard level; small fluctuation in Easy level due to gate bias.

SQL output well-formedness is improved. For instance, before applying our method the decoder would occasionally select the same columns twice to perform the `ON` clause.[4] After applying our dynamic gating, this issue is virtually eliminated (error rate from 2% to 0.2%).[5]

As shown in Figure 3, the values of both gates are polarized to 0 or 1, thus making the gating mechanism act as a binary gate. These statistics coincide with our hypothesis that *most entity generation decisions in Text-to-SQL can be solely made by evidence from either schema linking or structural linking*. In addition, among all the cases where structural linking is chosen and the copy gate takes control, fewer than 20% of cases favor copying. This suggests that there are lots of circumstances where different kinds of structural linking are adopted.

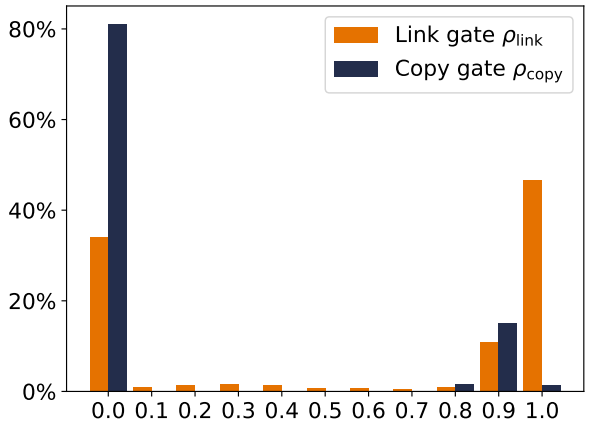

Figure 3: Value distributions of the link gate and copy gate measured in dev set of Spider using the GLOBALGNN model. Values of copy gate are considered when the corresponding link gate value is small ($\rho_{\text{link}} < 0.1$).

4.3 Alternative Approaches and Ablation

We also conduct several experiments to examine several design choices in our proposed method.

Sharing Action Embedding. In the design of our gating mechanism, one critical decision is to use the action embedding $\boldsymbol{a}_t$ to perform decoder self-attention and produce the gating values. This is based on our intuition that the action embedding captures the structural information of the output SQL at the current position. To verify this decision, we conduct an ablation experiment by using a *dedicated embedding* to produce the gating value. This dedicated embedding is produced in exactly same way as we generated $\boldsymbol{a}_t$ in Equation 9, but uses a different set of parameters for the feed-forward network. As we can see from Figure 4 ("dedicated embed"), sharing the parameters with the action embedding is important.

Keeping entities. Guo et al. (2019) also uses a memory-augmented pointer network to perform a copy mechanism which assists column selection. In contrast with our memory matrix consisting of action embeddings (Equation 7), their memory matrix consists of the entity embeddings of columns that have been selected previously. They further remove the columns from the candidates in the schema linking process once they are generated to prevent the decoder from repeatedly generating the same columns. To

[4] This is different from the case of intentionally self join.

[5] Although this issue can be resolved by engineering more grammar rules, we leave it as an indicator of the improvement of the well-formedness of the output SQL.

NL	*Show the stadium name and the number of concerts in each stadium.*
SQL	SELECT stadium.name (ρ_l = N/A, ρ_c = N/A) , COUNT(∗) FROM concert (ρ_l = 0.15, ρ_c = 0.00) JOIN stadium (ρ_l = 0.00, ρ_c = 0.00) ON concert.stadium_id (ρ_l = 0.09, ρ_c = 0.00) = stadium.stadium_id (ρ_l = 0.00, ρ_c = 0.00) GROUP BY concert.stadium_id (ρ_l = 0.00, ρ_c = 0.96) ;
NL	*Which city has most number of departing flights?*
SQL	SELECT airports.city (ρ_l = N/A, ρ_c = N/A) FROM airports (ρ_l = 0.82, ρ_c = 0.01) JOIN flights (ρ_l = 0.00, ρ_c = 0.00) ON flights.airportcode (ρ_l = 0.00, ρ_c = 0.00) = flights.sourceairport (ρ_l = 0.00, ρ_c = 0.00) GROUP BY airports.city (ρ_l = 0.50, ρ_c = 1.00) ORDER BY COUNT (∗) DESC LIMIT 1

Table 5: Sample predictions of our model. ρ_{link} and ρ_{copy} are abbreviated as ρ_l and ρ_c respectively. Gate values are not applicable (denoted by N/A) for the first entity since it has no previously generated entity. Some of the results reflect gate bias, see text for details.

determine if our dynamic gating and structural linking module can add enough structural constraints to the decoding process to resolve this kind of problem, we conduct an experiment where we also remove the entities in the schema linking process after they are generated (*i.e.*, "removing entity" in Figure 4) to see if it further improves the model. Our results shows that this change can actually hurts the model. Specifically, we observe a drop in accuracy of the `WHERE` and `IUEN` clauses, which suggests that in our context, the information about a specific entity in the schema linking process is still useful even after the entity has been generated once.

Copy Gate. In addition, removing the copy gate and copy mechanism also harms the performance of our model (*i.e.*, "without copy" in Figure 4). This result confirms that it is beneficial to handle different types of structural links separately. We hypothesize that different types of structural links conflict with each other, so they are hard to fit in one structural linking matrix. Overall, these two results further supports our claim that the copy gate can determine when to copy or link to an entity by itself.

Model	Dev Acc. (%)	
	GLOBALGNN	GLOBALBERT
Base	49.3	53.5
Ours	**52.8**	**57.6**
dedicated embed	50.0	55.4
removing entity	49.4	57.1
without copy	50.9	55.8

Table 4: Alternative approaches and ablation results.

4.4 Error Analysis and Discussion

Gate bias. Error analysis reveals that our gating mechanism is biased, *e.g.*, for the first few entities being selected in a SQL query the link gate is trained to favor schema linking in most cases. But, such bias could sometimes be wrong. In such cases where structural linking is needed but absent, the model may select duplicate columns or the wrong table during decoding. Similarly, the copy gate might be biased toward copying an entity from memory in `GROUP BY` clauses. Out of all the SQL clause components, only the `GROUP BY` clause's partial matching F1 score drops (about 3%) due to this copy gate bias. It is true that the entity needed in the `GROUP BY` clause is usually selected, but the information from schema linking can still be beneficial,[6] *e.g.*, in the second example of Table 5, the model wants to copy the wrong entity but the link gate rectifies it with schema linking information. Our gating mechanism only relies on the current action embedding (which can be seen as short-term structural information) to determine the gate values. We believe that introducing more global structural constraints is a promising direction to find a more flexible and accurate gating mechanism.

Short attention spans. In our experiments, we notice that the action history the model usually attends to is very short, *i.e.*, the model only utilizes the the output memory of the most recent three entities in 99% of cases. This coincides with similar findings in language modeling (Daniluk et al., 2017) where

[6]The semantics of some pronouns (*e.g.*, "each" and "which" in the examples of Table 5) in NL question match with the `GROUP BY` clause, but this could be a dataset bias.

the augmented-memory was expected to facilitate the modeling of long-range dependencies but failed to do so. Although long-term context is important for language modeling, it is not as important in our Text-to-SQL scenario since most dependencies in programming languages like SQL lie within a short span. We are interested in exploring semantic parsing tasks which requires long-term structural constraints using our method in the future.

Structural linking patterns. We have shown the effectiveness of our method in dealing with different types of structural links separately using different components of the model in the previous section. So far, the only special type of structural links we can explicitly model is the copy mechanism, and we treat all other types of links uniformly using additive attention in Equation 5. This might limit the model's ability to take advantage of the complicated relationships between entities. Currently, the entity representation provided by the GNN model is still difficult to explain, because the node representations contain a mix of information from different message-passing steps. One could imagine training GNNs with different message-passing steps each modeling a different level of structural linking, could lead to a more clear and expressive linking pattern.

5 Related Work

Semantic parsing. Semantic parsing research focus on mapping NL to formal languages like lambda calculus (Zettlemoyer and Collins, 2005; Kwiatkowksi et al., 2010; Liang et al., 2011; Dong and Lapata, 2016), Prolog-style queries (Zelle and Mooney, 1996; Tang and Mooney, 2000), and more recently to SQL (Warren and Pereira, 1982; Popescu et al., 2003; Giordani and Moschitti, 2009; Zhong et al., 2017; Iyer et al., 2017). It can also tackle the problem of parsing NL descriptions to complicated general-purpose programming language such as Python (Ling et al., 2016; Rabinovich et al., 2017; Yin and Neubig, 2017). Our proposed method is tested for Text-to-SQL parsing and can be adapted to other semantic parsing applications.

Structural mismatch. Programming languages like SQL express the same intent in a completely different way from NL by design (Kate, 2008). The phenomenon called structural mismatch widely exists between NL and various programming language and is a major challenge in semantic parsing (Dong, 2019). To alleviate the structural mismatch problem, early approaches rely on linguistic formalisms like parsing results from flexible CCGs (Zettlemoyer and Collins, 2005; Zettlemoyer and Collins, 2007; Kwiatkowski et al., 2011; Kwiatkowski et al., 2013). Chen et al. (2016) proposed to use sentence rewriting to revise the NL question to a new question which has the same structure with the targeted logical form. Recently, Guo et al. (2019) proposed to first translate the NL question to an intermediate representation (IR) designed to bridge NL and SQL, then use a deterministic algorithm to convert the IR to SQL. In addition to taking a considerable amount of engineering effort, their designed IR is still unable to cover some SQL grammars like the self-join in the `ON` clause, and is more challenging to apply to other programming languages. We deal with this problem by explicitly modeling the prediction structure with external predefined structure (*i.e.*, DB schema) by structural linking.

Memory pointer network. Memory networks were first introduced in the context of the question answering task, where they served as a differentiable long-term knowledge base to enhance an auto-regressive model's poor memory (Weston et al., 2015; Sukhbaatar et al., 2015). Copy mechanisms use attention as a pointer to select and copy items from source text, thus addressing the problem of a variable output vocabulary size (Vinyals et al., 2015; See et al., 2017). Recent research has applied memory-augmented pointer networks to various NLP tasks, including task-oriented dialogue (Wu et al., 2019) and also semantic parsing (Liang et al., 2017; Guo et al., 2019). Our dynamic gating mechanism can also be seen a memory controller, except our memory is read-only and acts as both the query and key in a pointer network. Different from most current techniques, our pointer network does not only perform copying but can also point to a start point of structural linking.

6 Conclusion

In this paper, we formulated the entity generation process in Text-to-SQL semantic parsing as two kinds of linking problems, namely schema linking and structural linking. We further proposed a dynamic gating

mechanism to explicitly model the decision between these two linking processes. Experimental results show the effectiveness of our proposed method and confirm our intuitions. In the future, we would like to apply our proposed method to other semantic parsing tasks, such as general purpose code generation, where structural constraints may be more important.

Acknowledgments

We thank Yu Bai and members of the UVA NLP group for valuable discussion and feedback. We also thank all anonymous reviewers for their helpful comments and suggestions.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*.

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1415–1425, Baltimore, Maryland, June. Association for Computational Linguistics.

Ben Bogin, Jonathan Berant, and Matt Gardner. 2019a. Representing schema structure with graph neural networks for text-to-SQL parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4560–4565, Florence, Italy, July. Association for Computational Linguistics.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019b. Global reasoning over database structures for text-to-SQL parsing. 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)*, pages 3659–3664, Hong Kong, China, November. Association for Computational Linguistics.

Bo Chen, Le Sun, Xianpei Han, and Bo An. 2016. Sentence rewriting for semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 766–777, Berlin, Germany, August. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October. Association for Computational Linguistics.

Michał Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel. 2017. Frustratingly short attention spans in neural language modeling. In *International Conference on Learning Representations*.

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, June. 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, August. Association for Computational Linguistics.

Li Dong. 2019. *Learning Natural Language Interfaces with Neural Models*. phdthesis, the University of Edinburgh.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2018. AllenNLP: A deep semantic natural language processing platform. In *Proceedings of Workshop for NLP Open Source Software (NLP-OSS)*, pages 1–6, Melbourne, Australia, July. Association for Computational Linguistics.

Alessandra Giordani and Alessandro Moschitti. 2009. Semantic mapping between natural language questions and sql queries via syntactic pairing. In *International Conference on Application of Natural Language to Information Systems*, pages 207–221. Springer.

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, July. Association for Computational Linguistics.

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

Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. In *Proceedings of the Second KR2ML workshop at NeurIPS 2019*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, Vancouver, Canada, July. Association for Computational Linguistics.

Rohit Kate. 2008. Transforming meaning representation grammars to improve semantic parsing. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 33–40, Manchester, England, August. Coling 2008 Organizing Committee.

Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526, Copenhagen, Denmark, September. Association for Computational Linguistics.

Tom Kwiatkowksi, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic CCG grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1223–1233, Cambridge, MA, October. Association for Computational Linguistics.

Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2011. Lexical generalization in CCG grammar induction for semantic parsing. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1512–1523, Edinburgh, Scotland, UK., July. Association for Computational Linguistics.

Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. 2013. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1545–1556, Seattle, Washington, USA, October. Association for Computational Linguistics.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2016. Gated graph sequence neural networks. In *International Conference on Learning Representations*.

Percy Liang, Michael Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 590–599, Portland, Oregon, USA, June. Association for Computational Linguistics.

Chen Liang, Jonathan Berant, Quoc 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 (Volume 1: Long Papers)*, pages 23–33, Vancouver, Canada, July. Association for Computational Linguistics.

Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Fumin Wang, and Andrew Senior. 2016. Latent predictor networks for code generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 599–609, Berlin, Germany, August. Association for Computational Linguistics.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September. Association for Computational Linguistics.

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 *Advances in Neural Information Processing Systems 32*, pages 8024–8035.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157. ACM.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149, Vancouver, Canada, July. Association for Computational Linguistics.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada, July. Association for Computational Linguistics.

Peter Shaw, Philip Massey, Angelica Chen, Francesco Piccinno, and Yasemin Altun. 2019. Generating logical forms from graph representations of text and entities. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 95–106, Florence, Italy, July. Association for Computational Linguistics.

Wei Shen, Jianyong Wang, and Jiawei Han. 2014. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460.

Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring unexplored generalization challenges for cross-database semantic parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8372–8388, Online, July. Association for Computational Linguistics.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. 2015. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448.

Lappoon R. Tang and Raymond J. Mooney. 2000. Automated construction of database interfaces: Intergrating statistical and relational learning for semantic parsing. In *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 133–141, Hong Kong, China, October. Association for Computational Linguistics.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online, July. Association for Computational Linguistics.

David H.D. Warren and Fernando C.N. Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. *American Journal of Computational Linguistics*, 8(3-4):110–122.

Jason Weston, Sumit Chopra, and Antoine Bordes. 2015. Memory networks. In *International Conference on Learning Representations*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Chien-Sheng Wu, Richard Socher, and Caiming Xiong. 2019. Global-to-local memory pointer networks for task-oriented dialogue. In *International Conference on Learning Representations*.

Chunyang Xiao, Marc Dymetman, and Claire Gardent. 2016. Sequence-based structured prediction for semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1341–1350, Berlin, Germany, August. Association for Computational Linguistics.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada, July. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium, October-November. Association for Computational Linguistics.

Tao Yu, Rui Zhang, Michihiro Yasunaga, Yi Chern Tan, Xi Victoria Lin, Suyi Li, Heyang Er, Irene Li, Bo Pang, Tao Chen, Emily Ji, Shreya Dixit, David Proctor, Sungrok Shim, Jonathan Kraft, Vincent Zhang, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. SParC: Cross-domain semantic parsing in context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4511–4523, Florence, Italy, July. Association for Computational Linguistics.

John M. Zelle and Raymond J. Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, pages 1050–1055. AAAI Press.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, pages 658–666, Arlington, Virginia, United States. AUAI Press.

Luke Zettlemoyer and Michael Collins. 2007. Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 678–687, Prague, Czech Republic, June. Association for Computational Linguistics.

Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir Radev. 2019. Editing-based SQL query generation for cross-domain context-dependent questions. 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)*, pages 5338–5349, Hong Kong, China, November. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.