RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers

Bailin Wang[*][†]
University of Edinburgh
bailin.wang@ed.ac.uk

Richard Shin[*][‡]
UC Berkeley
ricshin@cs.berkeley.edu

Xiaodong Liu   Oleksandr Polozov   Matthew Richardson
Microsoft Research, Redmond
{xiaodl,polozov,mattri}@microsoft.com

Abstract

When translating natural language questions into SQL queries to answer questions from a database, contemporary semantic parsing models struggle to generalize to unseen database schemas. The generalization challenge lies in (a) encoding the database relations in an accessible way for the semantic parser, and (b) modeling alignment between database columns and their mentions in a given query. We present a unified framework, based on the relation-aware self-attention mechanism, to address schema encoding, schema linking, and feature representation within a text-to-SQL encoder. On the challenging Spider dataset this framework boosts the exact match accuracy to 57.2%, surpassing its best counterparts by 8.7% absolute improvement. Further augmented with BERT, it achieves the new state-of-the-art performance of 65.6% on the Spider leaderboard. In addition, we observe qualitative improvements in the model's understanding of schema linking and alignment. Our implementation will be open-sourced at https://github.com/Microsoft/rat-sql.

1 Introduction

The ability to effectively query databases with natural language (NL) unlocks the power of large datasets to the vast majority of users who are not proficient in query languages. As such, a large body of research has focused on the task of translating NL questions into SQL queries that existing database software can execute.

The development of large annotated datasets of questions and the corresponding SQL queries has catalyzed progress in the field. In contrast to prior semantic parsing datasets (Finegan-Dollak et al., 2018), new tasks such as WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018b) pose the real-life challenge of *generalization to unseen database schemas*. Every query is conditioned on a multi-table database schema, and the databases do not overlap between the train and test sets.

Schema generalization is challenging for three interconnected reasons. First, any text-to-SQL parsing model must encode the schema into representations suitable for decoding a SQL query that might involve the given columns or tables. Second, these representations should encode all the information about the schema such as its column types, foreign key relations, and primary keys used for database joins. Finally, the model must recognize NL used to refer to columns and tables, which might differ from the referential language seen in training. The latter challenge is known as *schema linking* – aligning entity references in the question to the intended schema columns or tables.

While the question of *schema encoding* has been studied in recent literature (Bogin et al., 2019a), *schema linking* has been relatively less explored. Consider the example in Figure 1. It illustrates the challenge of ambiguity in linking: while *"model"* in the question refers to car_names.model rather than model_list.model, *"cars"* actually refers to both cars_data and car_names (but not car_makers) for the purpose of table joining. To resolve the column/table references properly, the semantic parser must take into account both the known schema relations (*e.g.* foreign keys) and the question context.

Prior work (Bogin et al., 2019a) addressed the schema representation problem by encoding the directed graph of foreign key relations in the schema with a graph neural network (GNN). While effective, this approach has two important shortcomings. First, it does not contextualize schema encoding with the question, thus making reasoning about
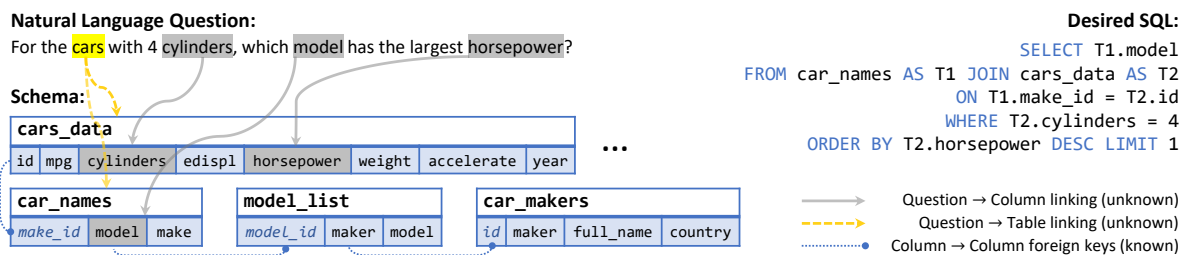

Figure 1: A challenging text-to-SQL task from the Spider dataset.

schema linking difficult after both the column representations and question word representations are built. Second, it limits information propagation during schema encoding to the predefined graph of foreign key relations. The advent of self-attentional mechanisms in NLP (Vaswani et al., 2017) shows that global reasoning is crucial to effective representations of relational structures. However, we would like any global reasoning to still take into account the aforementioned schema relations.

In this work, we present a unified framework, called RAT-SQL,[1] for encoding relational structure in the database schema and a given question. It uses *relation-aware self-attention* to combine global reasoning over the schema entities and question words with structured reasoning over predefined schema relations. We then apply RAT-SQL to the problems of schema encoding and schema linking. As a result, we obtain 57.2% exact match accuracy on the Spider test set. At the time of writing, this result is the state of the art among models unaugmented with pretrained BERT embeddings – and further reaches to the overall state of the art (65.6%) when RAT-SQL is augmented with BERT. In addition, we experimentally demonstrate that RAT-SQL enables the model to build more accurate internal representations of the question's true alignment with schema columns and tables.

2   Related Work

Semantic parsing of NL to SQL recently surged in popularity thanks to the creation of two new multi-table datasets with the challenge of schema generalization – WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018b). Schema encoding is not as challenging in WikiSQL as in Spider because it lacks multi-table relations. Schema linking is relevant for both tasks but also more challenging in Spider due to the richer NL expressiveness and less restricted SQL grammar observed in it. The state of the art semantic parser on WikiSQL (He et al.,

2019) achieves a test set accuracy of 91.8%, significantly higher than the state of the art on Spider.

The recent state-of-the-art models evaluated on Spider use various attentional architectures for question/schema encoding and AST-based structural architectures for query decoding. IRNet (Guo et al., 2019) encodes the question and schema separately with LSTM and self-attention respectively, augmenting them with custom type vectors for schema linking. They further use the AST-based decoder of Yin and Neubig (2017) to decode a query in an intermediate representation (IR) that exhibits higher-level abstractions than SQL. Bogin et al. (2019a) encode the schema with a GNN and a similar grammar-based decoder. Both works emphasize schema encoding and schema linking, but design separate featurization techniques to augment *word vectors* (as opposed to *relations between words and columns*) to resolve it. In contrast, the RAT-SQL framework provides a unified way to encode arbitrary relational information among inputs.

Concurrently with this work, Bogin et al. (2019b) published Global-GNN, a different approach to schema linking for Spider, which applies global reasoning between question words and schema columns/tables. Global reasoning is implemented by *gating* the GNN that encodes the schema using the question token representations. This differs from RAT-SQL in two important ways: (a) question word representations influence the schema representations but not vice versa, and (b) like in other GNN-based encoders, message propagation is limited to the schema-induced edges such as foreign key relations. In contrast, our relation-aware transformer mechanism allows encoding arbitrary relations between question words and schema elements explicitly, and these representations are computed jointly over all inputs using self-attention.

We use the same formulation of relation-aware self-attention as Shaw et al. (2018). However, they only apply it to sequences of words in the context of machine translation, and as such, their relation

---

[1] **R**elation-**A**ware **T**ransformer.

types only encode the relative distance between two words. We extend their work and show that relation-aware self-attention can effectively encode more complex relationships within an unordered set of elements (in our case, columns and tables within a database schema as well as relations between the schema and the question). To the best of our knowledge, this is the first application of relation-aware self-attention to joint representation learning with both predefined and softly induced relations in the input structure. Hellendoorn et al. (2020) develop a similar model concurrently with this work, where they use relation-aware self-attention to encode data flow structure in source code embeddings.

Sun et al. (2018) use a heterogeneous graph of KB facts and relevant documents for open-domain question answering. The nodes of their graph are analogous to the database schema nodes in RAT-SQL, but RAT-SQL also incorporates the question in the same formalism to enable joint representation learning between the question and the schema.

3 Relation-Aware Self-Attention

First, we introduce *relation-aware self-attention*, a model for embedding semi-structured input sequences in a way that jointly encodes pre-existing relational structure in the input as well as induced "soft" relations between sequence elements in the same embedding. Our solutions to schema embedding and linking naturally arise as features implemented in this framework.

Consider a set of inputs $X = \{x_i\}_{i=1}^n$ where $x_i \in \mathbb{R}^{d_x}$. In general, we consider it an unordered set, although x_i may be imbued with positional embeddings to add an explicit ordering relation. A *self-attention* encoder, or *Transformer*, introduced by Vaswani et al. (2017), is a stack of *self-attention layers* where each layer (consisting of H *heads*) transforms each x_i into $y_i \in \mathbb{R}^{d_x}$ as follows:

$$e_{ij}^{(h)} = \frac{x_i W_Q^{(h)} (x_j W_K^{(h)})^\top}{\sqrt{d_z/H}}; \quad \alpha_{ij}^{(h)} = \operatorname{softmax}_j \{e_{ij}^{(h)}\}$$

$$z_i^{(h)} = \sum_{j=1}^n \alpha_{ij}^{(h)} (x_j W_V^{(h)}); \quad z_i = \operatorname{Concat}(z_i^{(1)}, \cdots, z_i^{(H)})$$

$$\tilde{y}_i = \operatorname{LayerNorm}(x_i + z_i)$$

$$y_i = \operatorname{LayerNorm}(\tilde{y}_i + \operatorname{FC}(\operatorname{ReLU}(\operatorname{FC}(\tilde{y}_i)))) \tag{1}$$

where FC is a fully-connected layer, LayerNorm is *layer normalization* (Ba et al., 2016), $1 \le h \le H$, and $W_Q^{(h)}, W_K^{(h)}, W_V^{(h)} \in \mathbb{R}^{d_x \times (d_x/H)}$.

One interpretation of the embeddings computed by a Transformer is that each head of each layer

computes a *learned relation* between all the input elements x_i, and the strength of this relation is encoded in the attention weights $\alpha_{ij}^{(h)}$. However, in many applications (including text-to-SQL parsing) we are aware of some preexisting relational features between the inputs, and would like to bias our encoder model toward them. This is straightforward for non-relational features (represented directly in each x_i). We could limit the attention computation only to the "hard" edges where the preexisting relations are known to hold. This would make the model similar to a *graph attention network* (Veličković et al., 2018), and would also impede the Transformer's ability to learn *new* relations. Instead, RAT provides a way to communicate known relations to the encoder by adding their representations to the attention mechanism.

Shaw et al. (2018) describe a way to represent *relative position information* in a self-attention layer by changing Equation (1) as follows:

$$e_{ij}^{(h)} = \frac{x_i W_Q^{(h)} (x_j W_K^{(h)} + \boldsymbol{r_{ij}^K})^\top}{\sqrt{d_z/H}}$$

$$z_i^{(h)} = \sum_{j=1}^n \alpha_{ij}^{(h)} (x_j W_V^{(h)} + \boldsymbol{r_{ij}^V}). \tag{2}$$

Here the $\boldsymbol{r_{ij}}$ terms encode the known relationship between the two elements x_i and x_j in the input. While Shaw et al. used it exclusively for relative position representation, we show how to use the same framework to effectively bias the Transformer toward arbitrary relational information.

Consider R relational features, each a binary relation $\mathcal{R}^{(s)} \subseteq X \times X$ ($1 \le s \le R$). The RAT framework represents all the pre-existing features for each edge (i,j) as $\boldsymbol{r_{ij}^K} = \boldsymbol{r_{ij}^V} = \operatorname{Concat}(\boldsymbol{\rho_{ij}^{(1)}}, \ldots, \boldsymbol{\rho_{ij}^{(R)}})$ where each $\boldsymbol{\rho_{ij}^{(s)}}$ is either a *learned embedding* for the relation $\mathcal{R}^{(s)}$ if the relation holds for the corresponding edge (*i.e.* if $(i,j) \in \mathcal{R}^{(s)}$), or a zero vector of appropriate size. In the following section, we will describe the set of relations our RAT-SQL model uses to encode a given database schema.

4 RAT-SQL

We now describe the RAT-SQL framework and its application to the problems of schema encoding and linking. First, we formally define the text-to-SQL semantic parsing problem and its components. In the rest of the section, we present our implementation of schema linking in the RAT framework.

Type of x	Type of y	Edge label	Description
Column	Column	SAME-TABLE	x and y belong to the same table.
		FOREIGN-KEY-COL-F	x is a foreign key for y.
		FOREIGN-KEY-COL-R	y is a foreign key for x.
Column	Table	PRIMARY-KEY-F	x is the primary key of y.
		BELONGS-TO-F	x is a column of y (but not the primary key).
Table	Column	PRIMARY-KEY-R	y is the primary key of x.
		BELONGS-TO-R	y is a column of x (but not the primary key).
Table	Table	FOREIGN-KEY-TAB-F	Table x has a foreign key column in y.
		FOREIGN-KEY-TAB-R	Same as above, but x and y are reversed.
		FOREIGN-KEY-TAB-B	x and y have foreign keys in both directions.

Table 1: Description of edge types present in the directed graph $\mathcal{G}$ created to represent the schema. An edge exists from source node $x \in \mathcal{S}$ to target node $y \in \mathcal{S}$ if the pair fulfills one of the descriptions listed in the table, with the corresponding label. Otherwise, no edge exists from x to y.
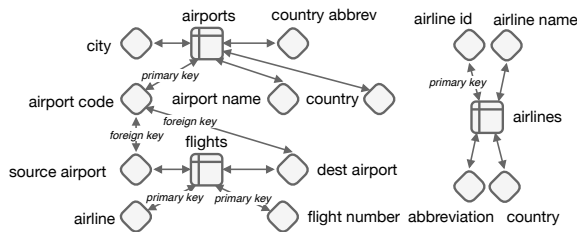

Figure 2: An illustration of an example schema as a graph $\mathcal{G}$. We do not depict all the edges and label types of Table 1 to reduce clutter.

Figure 3: One RAT layer in the schema encoder.

4.1 Problem Definition

Given a natural language question Q and a schema $\mathcal{S} = \langle \mathcal{C}, \mathcal{T} \rangle$ for a relational database, our goal is to generate the corresponding SQL P. Here the question $Q = q_1 \ldots q_{|Q|}$ is a sequence of words, and the schema consists of columns $\mathcal{C} = \{c_1, \ldots, c_{|\mathcal{C}|}\}$ and tables $\mathcal{T} = \{t_1, \ldots, t_{|\mathcal{T}|}\}$. Each column name c_i contains words $c_{i,1}, \ldots, c_{i,|c_i|}$ and each table name t_i contains words $t_{i,1}, \ldots, t_{i,|t_i|}$. The desired program P is represented as an *abstract syntax tree* T in the context-free grammar of SQL.

Some columns in the schema are *primary keys*, used for uniquely indexing the corresponding table, and some are *foreign keys*, used to reference a primary key column in a different table. In addition, each column has a *type* $\tau \in \{\texttt{number}, \texttt{text}\}$.

Formally, we represent the database schema as a directed graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$. Its nodes $\mathcal{V} = \mathcal{C} \cup \mathcal{T}$ are the columns and tables of the schema, each labeled with the words in its name (for columns, we prepend their type τ to the label). Its edges $\mathcal{E}$ are defined by the pre-existing database relations, described in Table 1. Figure 2 illustrates an example graph (with a subset of actual edges and labels).
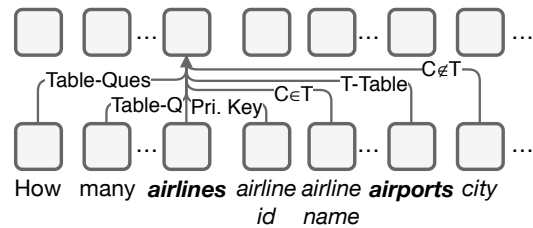
While $\mathcal{G}$ holds all the known information about the schema, it is insufficient for appropriately encoding a previously unseen schema *in the context of the question Q*. We would like our representations of the schema $\mathcal{S}$ and the question Q to be *joint*, in particular for modeling the alignment between them. Thus, we also define the *question-contextualized schema graph* $\mathcal{G}_Q = \langle \mathcal{V}_Q, \mathcal{E}_Q \rangle$ where $\mathcal{V}_Q = \mathcal{V} \cup Q = \mathcal{C} \cup \mathcal{T} \cup Q$ includes nodes for the question words (each labeled with a corresponding word), and $\mathcal{E}_Q = \mathcal{E} \cup \mathcal{E}_{Q \leftrightarrow \mathcal{S}}$ are the schema edges $\mathcal{E}$ extended with additional special relations between the question words and schema members, detailed in the rest of this section.

For modeling text-to-SQL generation, we adopt the *encoder-decoder framework*. Given the input as a graph $\mathcal{G}_Q$, the *encoder* f_{enc} embeds it into joint representations $\boldsymbol{c}_i$, $\boldsymbol{t}_i$, $\boldsymbol{q}_i$ for each column $c_i \in \mathcal{C}$, table $t_i \in \mathcal{T}$, and question word $q \in Q$ respectively. The decoder f_{dec} then uses them to compute a distribution $\Pr(P \mid \mathcal{G}_Q)$ over the SQL programs.

4.2 Relation-Aware Input Encoding

Following the state-of-the-art NLP literature, our encoder first obtains the *initial* representations $\boldsymbol{c}_i^{\text{init}}$, $\boldsymbol{t}_i^{\text{init}}$ for every node of $\mathcal{G}$ by **(a)** retrieving a pre-

trained Glove embedding (Pennington et al., 2014) for each word, and **(b)** processing the embeddings in each multi-word label with a bidirectional LSTM (BiLSTM) (Hochreiter and Schmidhuber, 1997). It also runs a separate BiLSTM over the question Q to obtain initial word representations q_i^{init}.

The initial representations c_i^{init}, t_i^{init}, and q_i^{init} are independent of each other and devoid of any relational information known to hold in $\mathcal{E}_Q$. To produce joint representations for the entire input graph $\mathcal{G}_Q$, we use the relation-aware self-attention mechanism (Section 3). Its input X is the set of all the node representations in $\mathcal{G}_Q$:

$$X = (c_1^{\text{init}}, \cdots, c_{|\mathcal{C}|}^{\text{init}}, t_1^{\text{init}}, \cdots, t_{|\mathcal{T}|}^{\text{init}}, q_1^{\text{init}}, \cdots, q_{|Q|}^{\text{init}}).$$

The encoder f_{enc} applies a stack of N relation-aware self-attention layers to X, with separate weight matrices in each layer. The final representations c_i, t_i, q_i produced by the Nth layer constitute the output of the whole encoder.

Alternatively, we also consider pre-trained BERT (Devlin et al., 2019) embeddings to obtain the initial representations. Following (Huang et al., 2019; Zhang et al., 2019), we feed X to the BERT and use the last hidden states as the initial representations before proceeding with the RAT layers.[2]

Importantly, as detailed in Section 3, every RAT layer uses self-attention between *all elements of the input graph* $\mathcal{G}_Q$ to compute new contextual representations of question words and schema members. However, this self-attention is *biased* toward some pre-defined relations using the edge vectors r_{ij}^K, r_{ij}^V in each layer. We define the set of used relation types in a way that directly addresses the challenges of schema embedding and linking. Occurrences of these relations between the question and the schema constitute the edges $\mathcal{E}_{Q \leftrightarrow S}$. Most of these relation types address schema linking (Section 4.3); we also add some auxiliary edges to aid schema encoding (see Appendix A).

4.3 Schema Linking

Schema linking relations in $\mathcal{E}_{Q \leftrightarrow S}$ aid the model with aligning column/table references in the question to the corresponding schema columns/tables. This alignment is implicitly defined by two kinds of information in the input: *matching names* and *matching values*, which we detail in order below.

Name-Based Linking Name-based linking refers to *exact* or *partial* occurrences of the column/table names in the question, such as the occurrences of *"cylinders"* and *"cars"* in the question in Figure 1. Textual matches are the most explicit evidence of question-schema alignment and as such, one might expect them to be directly beneficial to the encoder. However, in all our experiments the representations produced by vanilla self-attention were insensitive to textual matches even though their initial representations were identical. Brunner et al. (2020) suggest that representations produced by Transformers mix the information from different positions and cease to be directly interpretable after 2+ layers, which might explain our observations. Thus, to remedy this phenomenon, we explicitly encode name-based linking using RAT relations.

Specifically, for all n-grams of length 1 to 5 in the question, we determine (1) whether it exactly matches the name of a column/table (*exact match*); or (2) whether the n-gram is a subsequence of the name of a column/table (*partial match*).[3] Then, for every (i, j) where $x_i \in Q$, $x_j \in \mathcal{S}$ (or vice versa), we set $r_{ij} \in \mathcal{E}_{Q \leftrightarrow S}$ to QUESTION-COLUMN-M, QUESTION-TABLE-M, COLUMN-QUESTION-M or TABLE-QUESTION-M depending on the type of x_i and x_j. Here M is one of EXACTMATCH, PARTIALMATCH, or NOMATCH.

Value-Based Linking Question-schema alignment also occurs when the question mentions any *values* that occur in the database and consequently participate in the desired SQL, such as *"4"* in Figure 1. While this example makes the alignment explicit by mentioning the column name *"cylinders"*, many real-world questions do not. Thus, linking a value to the corresponding column requires background knowledge.

The database itself is the most comprehensive and readily available source of knowledge about possible values, but also the most challenging to process in an end-to-end model because of the privacy and speed impact. However, the RAT framework allows us to outsource this processing to the database engine to *augment* $\mathcal{G}_Q$ with potential value-based linking without exposing the model itself to the data. Specifically, we add a new COLUMN-VALUE relation between any word q_i and column name c_j s.t. q_i occurs as a value

---

[2]In this case, the initial representations c_i^{init}, t_i^{init}, q_i^{init} are not strictly independent although still yet uninfluenced by $\mathcal{E}$.

[3]This procedure matches that of Guo et al. (2019), but we use the matching information differently in RAT.

(or a full word within a value) of c_j. This simple approach drastically improves the performance of RAT-SQL (see Section 5). It also directly addresses the aforementioned DB challenges: **(a)** the model is never exposed to database content that does not occur in the question, **(b)** word matches are retrieved quickly via DB indices & textual search.

Memory-Schema Alignment Matrix Our intuition suggests that the columns and tables which occur in the SQL P will generally have a corresponding reference in the natural language question. To capture this intuition in the model, we apply relation-aware attention as a pointer mechanism between every memory element in y and all the columns/tables to compute explicit *alignment matrices* $L^{\mathrm{col}} \in \mathbb{R}^{|y| \times |\mathcal{C}|}$ and $L^{\mathrm{tab}} \in \mathbb{R}^{|y| \times |\mathcal{T}|}$:

$$\tilde{L}^{\mathrm{col}}_{i,j} = \frac{y_i W^{\mathrm{col}}_Q (\boldsymbol{c}^{\mathrm{final}}_j W^{\mathrm{col}}_K + \boldsymbol{r}^K_{ij})^\top}{\sqrt{d_x}} \quad (3)$$

$$\tilde{L}^{\mathrm{tab}}_{i,j} = \frac{y_i W^{\mathrm{tab}}_Q (\boldsymbol{t}^{\mathrm{final}}_j W^{\mathrm{tab}}_K + \boldsymbol{r}^K_{ij})^\top}{\sqrt{d_x}}$$

$$L^{\mathrm{col}}_{i,j} = \mathrm{softmax}_j \{ \tilde{L}^{\mathrm{col}}_{i,j} \} \qquad L^{\mathrm{tab}}_{i,j} = \mathrm{softmax}_j \{ \tilde{L}^{\mathrm{tab}}_{i,j} \}$$

Intuitively, the alignment matrices in Eq. (3) should resemble the real discrete alignments, therefore should respect certain constraints like sparsity. When the encoder is sufficiently parameterized, sparsity tends to arise with learning, but we can also encourage it with an explicit objective. Appendix B presents this objective and discusses our experiments with sparse alignment in RAT-SQL.

4.4 Decoder

The decoder f_{dec} of RAT-SQL follows the tree-structured architecture of Yin and Neubig (2017). It generates the SQL P as an abstract syntax tree in depth-first traversal order, by using an LSTM to output a sequence of *decoder actions* that either (i) expand the last generated node into a grammar rule, called APPLYRULE; or when completing a leaf node, (ii) choose a column/table from the schema, called SELECTCOLUMN and SELECTTABLE.

Formally, $\Pr(P \mid \mathcal{Y}) = \prod_t \Pr(a_t \mid a_{<t}, \mathcal{Y})$ where $\mathcal{Y} = f_{\mathrm{enc}}(\mathcal{G}_Q)$ is the final encoding of the question and schema, and $a_{<t}$ are all the previous actions. In a tree-structured decoder, the LSTM state is updated as $\boldsymbol{m}_t, \boldsymbol{h}_t = f_{\mathrm{LSTM}} ([\boldsymbol{a}_{t-1} \parallel \boldsymbol{z}_t \parallel \boldsymbol{h}_{p_t} \parallel \boldsymbol{a}_{p_t} \parallel \boldsymbol{n}_{f_t}], \boldsymbol{m}_{t-1}, \boldsymbol{h}_{t-1})$ where $\boldsymbol{m}_t$ is the LSTM cell state, $\boldsymbol{h}_t$ is the LSTM output at step t, $\boldsymbol{a}_{t-1}$ is the embedding of the previous action, p_t is the step corresponding to
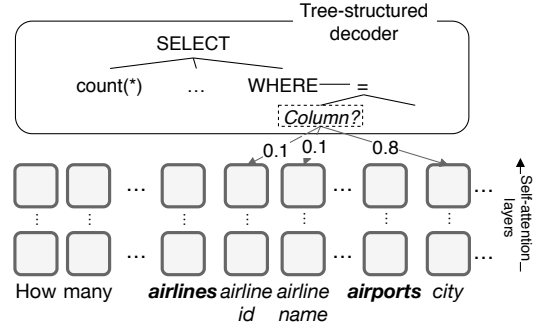

Figure 4: Choosing a column in a tree decoder.

expanding the parent AST node of the current node, and $\boldsymbol{n}_{f_t}$ is the embedding of the current node type. Finally, $\boldsymbol{z}_t$ is the context representation, computed using multi-head attention (with 8 heads) on $\boldsymbol{h}_{t-1}$ over $\mathcal{Y}$.

For APPLYRULE$[R]$, we compute $\Pr(a_t = \mathrm{APPLYRULE}[R] \mid a_{<t}, y) = \mathrm{softmax}_R (g(\boldsymbol{h}_t))$ where $g(\cdot)$ is a 2-layer MLP with a tanh non-linearity. For SELECTCOLUMN, we compute

$$\tilde{\lambda}_i = \frac{\boldsymbol{h}_t W^{\mathrm{sc}}_Q (y_i W^{\mathrm{sc}}_K)^T}{\sqrt{d_x}} \qquad \lambda_i = \mathrm{softmax}_i \{ \tilde{\lambda}_i \}$$

$$\Pr(a_t = \mathrm{SELECTCOLUMN}[i] \mid a_{<t}, y) = \sum_{j=1}^{|y|} \lambda_j L^{\mathrm{col}}_{j,i}$$

and similarly for SELECTTABLE. We refer the reader to Yin and Neubig (2017) for details.

5 Experiments

We implemented RAT-SQL in PyTorch (Paszke et al., 2017). During preprocessing, the input of questions, column names and table names are tokenized and lemmatized with the StandfordNLP toolkit (Manning et al., 2014). Within the encoder, we use GloVe (Pennington et al., 2014) word embeddings, held fixed in training except for the 50 most common words in the training set. For RAT-SQL BERT, we use the WordPiece tokenization. All word embeddings have dimension 300. The bidirectional LSTMs have hidden size 128 per direction, and use the recurrent dropout method of Gal and Ghahramani (2016) with rate 0.2. We stack 8 relation-aware self-attention layers on top of the bidirectional LSTMs. Within them, we set $d_x = d_z = 256$, $H = 8$, and use dropout with rate 0.1. The position-wise feed-forward network has inner layer dimension 1024. Inside the decoder, we use rule embeddings of size 128, node type embeddings of size 64, and a hidden size of 512 inside the LSTM with dropout of 0.21.

Model	Dev	Test
IRNet (Guo et al., 2019)	53.2	46.7
Global-GNN (Bogin et al., 2019b)	52.7	47.4
IRNet V2 (Guo et al., 2019)	55.4	48.5
RAT-SQL (ours)	**62.7**	**57.2**
With BERT:		
EditSQL + BERT (Zhang et al., 2019)	57.6	53.4
GNN + Bertrand-DR (Kelkar et al., 2020)	57.9	54.6
IRNet V2 + BERT (Guo et al., 2019)	63.9	55.0
RYANSQL V2 + BERT (Choi et al., 2020)	**70.6**	60.6
RAT-SQL + BERT (ours)	69.7	**65.6**

Table 2: Accuracy on the Spider development and test sets, compared to the other approaches at the top of the dataset leaderboard as of May 1st, 2020. The test set results were scored using the Spider evaluation server.

We used the Adam optimizer (Kingma and Ba, 2015) with the default hyperparameters. During the first $warmup\_steps = max\_steps/20$ steps of training, the learning rate linearly increases from 0 to 7.4×10^{-4}. Afterwards, it is annealed to 0 with $7.4 \times 10^{-4}(1 - \frac{step-warmup\_steps}{max\_steps-warmup\_steps})^{-0.5}$. We use a batch size of 20 and train for up to 40,000 steps. For RAT-SQL + BERT, we use a separate learning rate of 3×10^{-6} to fine-tune BERT, a batch size of 24 and train for up to 90,000 steps.

Hyperparameter Search We tuned the batch size (20, 50, 80), number of RAT layers (4, 6, 8), dropout (uniformly sampled from [0.1, 0.3]), hidden size of decoder RNN (256, 512), max learning rate (log-uniformly sampled from [5×10^{-4}, 2×10^{-3}]). We randomly sampled 100 configurations and optimized on the dev set. RAT-SQL + BERT reuses most hyperparameters of RAT-SQL, only tuning the BERT learning rate (1×10^{-4}, 3×10^{-4}, 5×10^{-4}), number of RAT layers (6, 8, 10), number of training steps (4×10^4, 6×10^4, 9×10^4).

5.1 Datasets and Metrics

We use the Spider dataset (Yu et al., 2018b) for most of our experiments, and also conduct preliminary experiments on WikiSQL (Zhong et al., 2017) to confirm generalization to other datasets. As described by Yu et al., Spider contains 8,659 examples (questions and SQL queries, with the accompanying schemas), including 1,659 examples lifted from the Restaurants (Popescu et al., 2003; Tang and Mooney, 2000), GeoQuery (Zelle and Mooney, 1996), Scholar (Iyer et al., 2017), Academic (Li and Jagadish, 2014), Yelp and IMDB (Yaghmazadeh et al., 2017) datasets.

As Yu et al. (2018b) make the test set accessible only through an evaluation server, we perform

Split	Easy	Medium	Hard	Extra Hard	All
RAT-SQL					
Dev	80.4	63.9	55.7	40.6	62.7
Test	74.8	60.7	53.6	31.5	57.2
RAT-SQL + BERT					
Dev	86.4	73.6	62.1	42.9	69.7
Test	83.0	71.3	58.3	38.4	65.6

Table 3: Accuracy on the Spider development and test sets, by difficulty as defined by Yu et al. (2018b).

Model	Accuracy (%)
RAT-SQL + value-based linking	**60.54 ± 0.80**
RAT-SQL	55.13 ± 0.84
w/o schema linking relations	40.37 ± 2.32
w/o schema graph relations	35.59 ± 0.85

Table 4: Accuracy (and ±95% confidence interval) of RAT-SQL ablations on the dev set.

most evaluations (other than the final accuracy measurement) using the development set. It contains 1,034 examples, with databases and schemas distinct from those in the training set. We report results using the same metrics as Yu et al. (2018a): exact match accuracy on all examples, as well as divided by difficulty levels. As in previous work on Spider, these metrics do not measure the model's performance on generating values in the SQL.

5.2 Spider Results

In Table 2 we show accuracy on the (hidden) Spider test set for RAT-SQL and compare to all other approaches at or near state-of-the-art (according to the official leaderboard). RAT-SQL outperforms all other methods that are not augmented with BERT embeddings by a large margin of 8.7%. Surprisingly, it even beats other BERT-augmented models. When RAT-SQL is further augmented with BERT, it achieves the new state-of-the-art performance. Compared with other BERT-argumented models, our RAT-SQL + BERT has smaller generalization gap between development and test set.

We also provide a breakdown of the accuracy by difficulty in Table 3. As expected, performance drops with increasing difficulty. The overall generalization gap between development and test of RAT-SQL was strongly affected by the significant drop in accuracy (9%) on the extra hard questions. When RAT-SQL is augmented with BERT, the generalization gaps of most difficulties are reduced.

Ablation Study Table 4 shows an ablation study over different RAT-based relations. The ablations
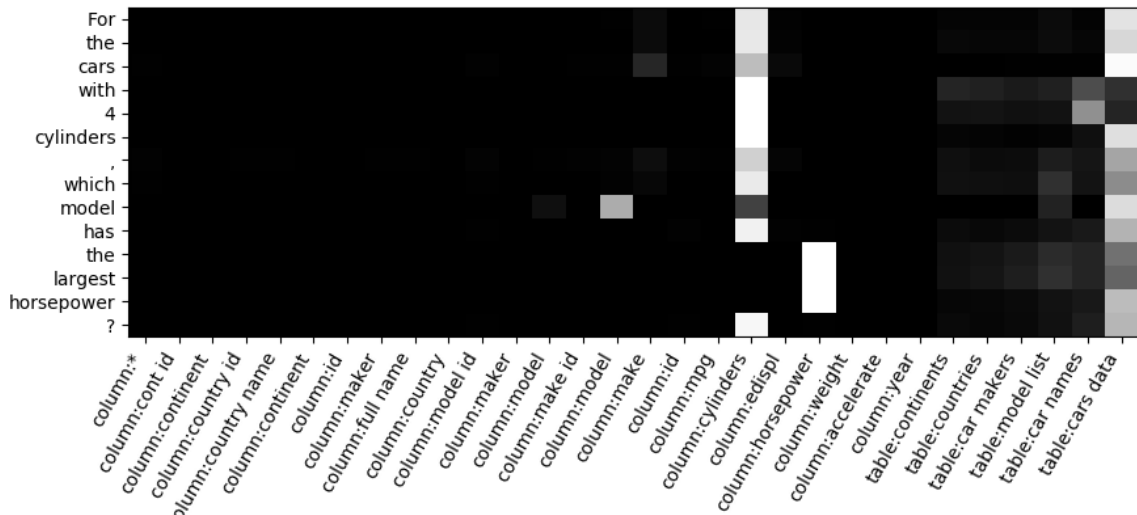
Figure 5: Alignment between the question *"For the cars with 4 cylinders, which model has the largest horsepower"* and the database `car_1` schema (columns and tables) depicted in Figure 1.

are run on RAT-SQL without value-based linking to avoid interference with information from the database. Schema linking and graph relations make statistically significant improvements (p<0.001). The full model accuracy here slightly differs from Table 2 because the latter shows the best model from a hyper-parameter sweep (used for test evaluation) and the former gives the mean over five runs where we only change the random seeds.

5.3 WikiSQL Results

We also conducted preliminary experiments on WikiSQL (Zhong et al., 2017) to test generalization of RAT-SQL to new datasets. Although WikiSQL lacks multi-table schemas (and thus, its challenge of schema encoding is not as prominent), it still presents the challenges of schema linking and generalization to new schemas. For simplicity of experiments, we did not implement either BERT augmentation or execution-guided decoding (EG) (Wang et al., 2018), both of which are common in state-of-the-art WikiSQL models. We thus only compare to the models that also lack these two enhancements.

While not reaching state of the art, RAT-SQL still achieves competitive performance on WikiSQL as shown in Table 5. Most of the gap between its accuracy and state of the art is due to the simplified implementation of *value decoding*, which is required for WikiSQL evaluation but not in Spider. Our value decoding for these experiments is a simple token-based pointer mechanism, which often fails to retrieve multi-token value constants accurately. A robust value decoding mechanism in

RAT-SQL is an important extension that we plan to address outside the scope of this work.

5.4 Discussions

Alignment Recall from Section 4 that we explicitly model the alignment matrix between question words and table columns, used during decoding for column and table selection. The existence of the alignment matrix provides a mechanism for the model to align words to columns. An accurate alignment representation has other benefits such as identifying question words to copy to emit a constant value in SQL.

In Figure 5 we show the alignment generated by our model on the example from Figure 1.[4] For the three words that reference columns (*"cylinders"*, *"model"*, *"horsepower"*), the alignment matrix correctly identifies their corresponding columns. The alignments of other words are strongly affected by these three keywords, resulting in a sparse span-to-column like alignment, *e.g. "largest horsepower"* to `horsepower`. The tables `cars_data` and `cars_names` are implicitly mentioned by the word *"cars"*. The alignment matrix successfully infers to use these two tables instead of `car_makers` using the evidence that they contain the three mentioned columns.

The Need for Schema Linking One natural question is how often does the decoder fail to select the correct column, even with the schema encoding and linking improvements we have made. To

---

[4]The full alignment also maps from column and table names, but those end up simply aligning to themselves or the table they belong to, so we omit them for brevity.

Model	Dev		Test	
	LF Acc%	Ex. Acc%	LF Acc%	Ex. Acc%
IncSQL (Shi et al., 2018)	49.9	84.0	49.9	83.7
MQAN (McCann et al., 2018)	76.1	82.0	75.4	81.4
RAT-SQL (ours)	73.6	79.5	73.3	78.8
Coarse2Fine (Dong and Lapata, 2018)	72.5	79.0	71.7	78.5
PT-MAML (Huang et al., 2018)	63.1	68.3	62.8	68.0

Table 5: RAT-SQL accuracy on WikiSQL, trained without BERT augmentation or execution-guided decoding (EG). Compared to other approaches without EG. "LF Acc" = Logical Form Accuracy; "Ex. Acc" = Execution Accuracy.

Model	Acc.
RAT-SQL	62.7
RAT-SQL + Oracle columns	69.8
RAT-SQL + Oracle sketch	73.0
RAT-SQL + Oracle sketch + Oracle columns	99.4

Table 6: Accuracy (exact match %) on the development set given an oracle providing correct columns and tables ("Oracle columns") and/or the AST sketch structure ("Oracle sketch").

answer this, we conducted an oracle experiment (see Table 6). For "oracle sketch", at every grammar nonterminal the decoder is forced to choose the correct production so the final SQL sketch exactly matches that of the ground truth. The rest of the decoding proceeds conditioned on that choice. Likewise, "oracle columns" forces the decoder to emit the correct column/table at terminal nodes.

With both oracles, we see an accuracy of 99.4% which just verifies that our grammar is sufficient to answer nearly every question in the data set. With just "oracle sketch", the accuracy is only 73.0%, which means 72.4% of the questions that RAT-SQL gets wrong and could get right have incorrect column or table selection. Similarly, with just "oracle columns", the accuracy is 69.8%, which means that 81.0% of the questions that RAT-SQL gets wrong have incorrect structure. In other words, most questions have both column and structure wrong, so both problems require important future work.

Error Analysis An analysis of mispredicted SQL queries in the Spider dev set showed three main causes of evaluation errors. **(I)** 18% of the mispredicted queries are in fact *equivalent* implementations of the NL intent with a different SQL syntax (*e.g.* ORDER BY C LIMIT 1 vs. SELECT MIN(C)). Measuring execution accuracy rather than exact match would detect them as valid. **(II)** 39% of errors involve a wrong, missing, or extraneous column in the SELECT clause. This is a limitation of our schema linking mechanism, which, while substantially improving column

resolution, still struggles with some ambiguous references. Some of them are unavoidable as Spider questions do not always specify which columns should be returned by the desired SQL. Finally, **(III)** 29% of errors are missing a WHERE clause, which is a common error class in text-to-SQL models as reported by prior works. One common example is domain-specific phrasing such as *"older than 21"*, which requires background knowledge to map it to age > 21 rather than age < 21. Such errors disappear after in-domain fine-tuning.

6 Conclusion

Despite active research in text-to-SQL parsing, many contemporary models struggle to learn good representations for a given database schema as well as to properly link column/table references in the question. These problems are related: to encode & use columns/tables from the schema, the model must reason about their role in the context of the question. In this work, we present a unified framework for addressing the schema encoding and linking challenges. Thanks to relation-aware self-attention, it jointly learns schema and question representations based on their alignment with each other and schema relations.

Empirically, the RAT framework allows us to gain significant state of the art improvement on text-to-SQL parsing. Qualitatively, it provides a way to combine predefined *hard* schema relations and inferred *soft* self-attended relations in the same encoder architecture. This representation learning will be beneficial in tasks beyond text-to-SQL, as long as the input has some predefined structure.

Acknowledgments

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. *arXiv:1607.06450*.

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.

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 3657–3662.

Gino Brunner, Yang Liu, Damian Pascual, Oliver Richter, Massimiliano Ciaramita, and Roger Wattenhofer. 2020. On identifiability in Transformers. In *International Conference on Learning Representations*.

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. 2020. RYANSQL: Recursively applying sketch-based slot fillings for complex text-to-SQL in cross-domain databases. *arXiv preprint arXiv:2004.03125*.

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.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742, Melbourne, Australia. Association for Computational Linguistics.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving Text-to-SQL Evaluation Methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360.

Yarin Gal and Zoubin Ghahramani. 2016. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In *Advances in Neural Information Processing Systems 29*, pages 1019–1027.

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.

Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. 2019. X-SQL: reinforce schema representation with context. *arXiv preprint arXiv:1908.08113*.

Vincent J. Hellendoorn, Charles Sutton, Rishabh Singh, Petros Maniatis, and David Bieber. 2020. Global relational models of source code. In *International Conference on Learning Representations*.

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

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. 2019. Music Transformer. In *International Conference on Learning Representations*.

Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen-tau Yih, and Xiaodong He. 2018. Natural language to structured query generation via meta-learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 732–738.

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.

Amol Kelkar, Rohan Relan, Vaishali Bhardwaj, Saurabh Vaichal, and Peter Relan. 2020. Bertrand-DR: Improving text-to-SQL using a discriminative re-ranker. *arXiv preprint arXiv:2002.00557*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.

Fei Li and H. V. Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. 2018. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

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.

Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-Attention with Relative Position Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468.

Tianze Shi, Kedar Tatwawadi, Kaushik Chakrabarti, Yi Mao, Oleksandr Polozov, and Weizhu Chen. 2018. IncSQL: Training Incremental Text-to-SQL Parsers with Non-Deterministic Oracles. *arXiv:1809.05054 [cs]*.

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.

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.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. *International Conference on Learning Representations*.

Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018. Robust Text-to-SQL Generation with Execution-Guided Decoding. *arXiv:1807.03100 [cs]*.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Sqlizer: Query synthesis from natural language. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM*, pages 63:1–63:26.

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.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018a. SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-Domain Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663.

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. 2018b. 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.

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*, pages 1050–1055.

Rui Zhang, Tao Yu, He Yang 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*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *arXiv:1709.00103 [cs]*.

A Auxiliary Relations for Schema Encoding

In addition to the schema graph edges $\mathcal{E}$ (Section 4.2) and schema linking edges (Section 4.3), the edges in $\mathcal{E}_Q$ also include some auxiliary relation types to aid the relation-aware self-attention. Specifically, for each $x_i, x_j \in \mathcal{V}_Q$:

- If $i = j$, then COLUMN-IDENTITY or TABLE-IDENTITY.

- $x_i \in Q$, $x_j \in Q$: QUESTION-DIST-d, where

$$d = \mathsf{clip}(j - i, D),$$
$$\mathsf{clip}(a, D) = \max(-D, \min(D, a)).$$

  We use $D = 2$.

- Otherwise, one of COLUMN-COLUMN, COLUMN-TABLE, TABLE-COLUMN, or TABLE-TABLE.

B Alignment Loss

The memory-schema alignment matrix is expected to resemble the real discrete alignments, therefore should respect certain constraints like sparsity. For example, the question word *"model"* in Figure 1 should be aligned with `car_names.model` rather than `model_list.model` or `model_list.model_id`. To further bias the soft alignment towards the real discrete structures, we add an auxiliary loss to encourage sparsity of the alignment matrix. Specifically, for a column/table that is mentioned in the SQL query, we treat the model's current belief of the best alignment as the ground truth. Then we use a cross-entropy loss, referred as *alignment loss*, to strengthen the model's belief:

$$align\_loss = -\frac{1}{|Rel(\mathcal{C})|} \sum_{j \in Rel(\mathcal{C})} \log \max_i L_{i,j}^{\mathrm{col}}$$
$$-\frac{1}{|Rel(\mathcal{T})|} \sum_{j \in Rel(\mathcal{T})} \log \max_i L_{i,j}^{\mathrm{tab}}$$

where $Rel(\mathcal{C})$ and $Rel(\mathcal{T})$ denote the set of *relevant* columns and tables that appear in the SQL.

In earlier experiments, we found that the alignment loss did improve the model (statistically significantly, from 53.0% to 55.4%). However, it does not make a statistically significant difference in our final model in terms of overall exact match. We hypothesize that hyperparameter tuning that caused us

Model	Exact Match	Correctness
RAT-SQL	0.59	0.81
RAT-SQL + BERT	0.67	0.86

Table 7: Consistency of the two RAT-SQL models.

to increase encoding depth eliminated the need for explicit supervision of alignment. With few layers in the Transformer, the alignment matrix provided additional degrees of freedom, which became unnecessary once the Transformer was sufficiently deep to build a rich joint representation of the question and the schema.

C Consistency of RAT-SQL

In Spider dataset, most SQL queries correspond to more than one question, making it possible to evaluate the consistency of RAT-SQL given paraphrases. We use two metrics to evaluate the consistency: 1) *Exact Match* – whether RAT-SQL produces the exact same predictions given paraphrases, 2) *Correctness* – whether RAT-SQL achieves the same correctness given paraphrases. The analysis is conducted on the development set.

The results are shown in Table 7. We found that when augmented with BERT, RAT-SQL becomes more consistent in terms of both metrics, indicating the pre-trained representations of BERT are beneficial for handling paraphrases.