Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing

Xi Victoria Lin **Richard Socher** **Caiming Xiong**
Salesforce Research
`{xilin,rsocher,cxiong}@salesforce.com`

Abstract

We present BRIDGE, a powerful sequential architecture for modeling dependencies between natural language questions and relational databases in cross-DB semantic parsing. BRIDGE represents the question and DB schema in a tagged sequence where a subset of the fields are augmented with cell values mentioned in the question. The hybrid sequence is encoded by BERT with minimal subsequent layers and the text-DB linking is realized via fine-tuned deep attention of BERT. Combined with a pointer-generator decoder with schema-consistency driven search space pruning, BRIDGE attained state-of-the-art performance on the well-studied Spider benchmark (65.5% dev, 59.2% test), despite being much simpler than most recently proposed models for this task. Our analysis shows that BRIDGE effectively captures the desired cross-modal dependencies and has the potential to generalize to more text-DB oriented tasks. Our implementation will be open-sourced at `http://anonymous.url`.

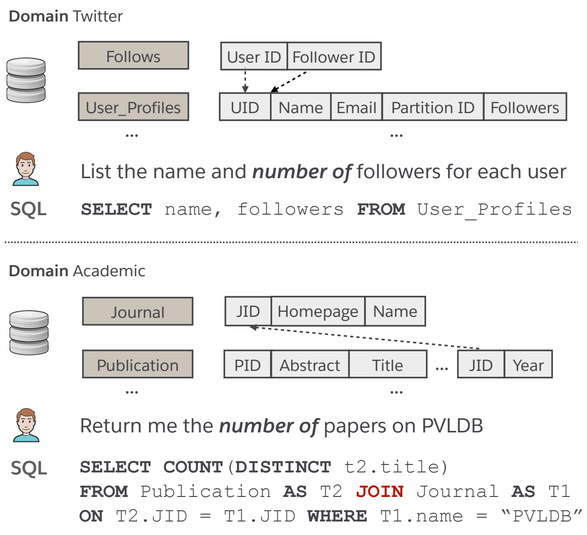

Domain Twitter
Follows
User ID | Follower ID
User_Profiles
UID | Name | Email | Partition ID | Followers
List the name and *number of* followers for each user
SQL
SELECT name, followers FROM User_Profiles
Domain Academic
Journal
JID | Homepage | Name
Publication
PID | Abstract | Title | ... | JID | Year
Return me the *number of* papers on PVLDB
SQL
SELECT COUNT(DISTINCT t2.title)
FROM Publication AS T2 JOIN Journal AS T1
ON T2.JID = T1.JID WHERE T1.name = "PVLDB"

Figure 1: Two questions from the Spider dataset with similar intent resulted in completely different SQL logical forms on two DBs. In cross-DB text-to-SQL semantic parsing, the interpretation of a natural language question is strictly grounded in the underlying relational DB schema.

1 Introduction

Text-to-SQL semantic parsing addresses the problem of mapping natural language utterances to executable relational DB queries. Early work in this area focus on training and testing the semantic parser on a single DB (Hemphill et al., 1990; Dahl et al., 1994; Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Dong and Lapata, 2016). However, DBs are widely used in many domains and developing a semantic parser for each individual DB is unlikely to scale in practice.

More recently, large-scale datasets consisting of hundreds of DBs and the corresponding question-SQL pairs have been released (Yu et al., 2018; Zhong et al., 2017; Yu et al., 2019b,a) to encourage the development of semantic parsers that can work well across different DBs (Guo et al., 2019; Bogin et al., 2019b; Zhang et al., 2019; Wang et al., 2019; Suhr et al., 2020; Choi et al., 2020). The setup is challenging as it requires the model to interpret a question conditioned on a relational DB unseen during training and accurately express the question intent via SQL logic. Consider the two examples shown in Figure 1, both questions have the intent to count, but the corresponding SQL queries are drastically different due to differences in the target DB schema. As a result, cross-DB text-to-SQL semantic parsers cannot trivially memorize seen SQL patterns, but instead has to accurately model the natural language question, the target DB structure, and the contextualization of both.

State-of-the-art cross-DB text-to-SQL semantic parsers adopt the following design principles to address the aforementioned challenges. First, the question and schema representation should be contextualized with each other (Hwang et al., 2019; Guo et al., 2019; Wang et al., 2019; Yin et al., 2020). Second, large-scale pre-trained language models (LMs) such as BERT (Devlin et al., 2019) and Roberta (Liu et al., 2019c) can significantly boost parsing accuracy by providing better representations of text and capturing long-term dependencies. Third, under data privacy constraints, leveraging available DB content can resolve ambiguities in the DB schema (Bogin et al., 2019b; Wang et al., 2019; Yin et al., 2020). Consider the second example in Figure 1, knowing "PLVDB" is a value of the field `Journal.Name` helps the model to generate the `WHERE` condition.

We present BRIDGE, a powerful sequential text-DB encoding framework assembling the three design principles mentioned above. BRIDGE represents the relational DB schema as a tagged sequence concatenated to the question. Different from previous work which proposed special-purpose layers for modeling the DB schema (Bogin et al., 2019a,b; Zhang et al., 2019; Choi et al., 2020) and cross text-DB linking (Guo et al., 2019; Wang et al., 2019), BRIDGE encodes the tagged hybrid sequence with BERT and lightweight subsequent layers – two single-layer bi-directional LSTMs (Hochreiter and Schmidhuber, 1997). Each schema component (table or field) is simply represented using the hidden state of its special token in the hybrid sequence. To better align the schema components with the question, BRIDGE augments the hybrid sequence with *anchor texts*, which are automatically extracted DB cell values mentioned in the question. Anchor texts are appended to their corresponding fields in the hybrid sequence (Figure 2). The text-DB alignment is then implicitly achieved via fine-tuned BERT attention between overlapped lexical tokens.

Combined with a pointer-generator decoder (See et al., 2017) and schema-consistency driven search space pruning, BRIDGE performs competitively on the well studied Spider benchmark (Structure Acc: 65.6% dev, 59.2% test, top-4 rank; Execution Acc: 59.9% test, top-1 rank), outperforming most of recently proposed models with more sophisticated neural architectures. Our analysis shows that when applied to Spider, the BERT-encoded hybrid representation can effectively capture useful cross-modal dependencies and the anchor text augmentation resulted in significant performance improvement.

2 Model

In this section, we present the BRIDGE encoding framework combined with a sequential pointer-generator to perform end-to-end cross-DB text-to-SQL semantic parsing.

2.1 Problem Definition

We formally defined the cross-DB text-to-SQL task as the following. Given a natural language question Q and the schema $\mathcal{S} = \langle \mathcal{T}, \mathcal{C} \rangle$ for a relational database, the parser needs to generate the corresponding SQL query Y. The schema consists of tables $\mathcal{T} = \{t_1, \ldots, t_N\}$ and fields $\mathcal{C} = \{c_{11}, \ldots, c_{1|T_1|}, \ldots, c_{n1}, \ldots, c_{N|T_N|}\}$. Each table t_i and each field c_{ij} has a textual name. Some fields are *primary keys*, used for uniquely indexing eachEar data record, and some are *foreign keys*, used to reference a primary key in a different table. In addition, each field has a *data type*, $\tau \in$ {`number`, `text`, `time`, `boolean`, *etc.*}.

Most existing solutions for this task do not consider DB content (Zhong et al., 2017; Yu et al., 2018). Recent approaches show accessing DB content significantly improves system performance (Liang et al., 2018; Wang et al., 2019; Yin et al., 2020). We consider the setting adopted by Wang et al. (2019) where the model has access to the value set of each field instead of full DB content. For example, the field `Property_Type_Code` in Figure 2 can take one of the following values: {"Apartment", "Field", "House", "Shop", "Other"}. We call such value sets *picklists*. This setting protects individual data record and sensitive fields such as user IDs or credit numbers can be hidden.

2.2 Question-Schema Serialization and Encoding

As shown in Figure 2, we represent each table with its table name followed by its fields. Each table name is preceded by the special token `[T]` and each field name is preceded by `[C]`. The representations of multiple tables are concatenated to form a serialization of the schema, which is surrounded by two `[SEP]` tokens and concatenated to the question. Finally, following the input format of BERTt, the question is preceded by `[CLS]` to form the hybrid question-schema serialization

$$\begin{aligned} X = &\texttt{[CLS]}, Q, \texttt{[SEP]}, \texttt{[T]}, t_1, \texttt{[C]}, c_{11} \ldots, c_{1|T_1|}, \\ &\texttt{[T]}, t_2, \texttt{[C]}, c_{21}, \ldots, \texttt{[C]}, c_{N|T_N|}, \texttt{[SEP]}. \end{aligned}$$

X is encoded with BERT, followed by a bi-directional LSTM to form the base encoding $\boldsymbol{h}_\text{X} \in \mathbb{R}^{|X|\times n}$. The question segment of $\boldsymbol{h}_\text{X}$ is passed through another bi-LSTM to obtain the question encoding $\boldsymbol{h}_\text{Q} \in \mathbb{R}^{|Q|\times n}$. Each table/field is represented using the slice of $\boldsymbol{h}_\text{X}$ corresponding to its special token `[T]`/`[C]`.

Meta-data Features We train dense look-up features to represent meta-data of the schema. This includes whether a field is a primary key ($\boldsymbol{f}_\text{pri} \in \mathbb{R}^{2\times n}$), whether the field appears in a foreign key pair ($\boldsymbol{f}_\text{for} \in \mathbb{R}^{2\times n}$) and the data type of the field ($\boldsymbol{f}_\text{type} \in \mathbb{R}^{|\tau|\times n}$). These meta-data features are fused with the base encoding of the schema component via a projection layer g to obtain the following encoding output:

$$\boldsymbol{h}_S^{t_i} = g([\boldsymbol{h}_\text{X}^p; \boldsymbol{0}; \boldsymbol{0}; \boldsymbol{0}]), \tag{1}$$

$$\begin{aligned}\boldsymbol{h}_S^{c_{ij}} &= g([\boldsymbol{h}_\text{X}^q; \boldsymbol{f}_\text{pri}^u; \boldsymbol{f}_\text{for}^v; \boldsymbol{f}_\text{type}^w]) \\ &= \text{ReLU}(\boldsymbol{W}_g[\boldsymbol{h}_\text{X}^m; \boldsymbol{f}_\text{pri}^u; \boldsymbol{f}_\text{for}^v; \boldsymbol{f}_\text{type}^w] + \boldsymbol{b}_g)\end{aligned} \tag{2}$$

$$\boldsymbol{h}_S = [\boldsymbol{h}^{t_1}, \ldots, \boldsymbol{h}^{t_{|T|}}, \boldsymbol{h}^{c_{11}}, \ldots, \boldsymbol{h}^{c_{N|T_N|}}] \in \mathbb{R}^{|S|\times n}, \tag{3}$$

where p is the index of `[T]` associated with table t_i in X and q is the index of `[C]` associated with field c_{ij} in X. u, v and w are feature indices indicating the properties of c_{ij}. $[\boldsymbol{h}_\text{X}^m; \boldsymbol{f}_\text{pri}^u; \boldsymbol{f}_\text{for}^v; \boldsymbol{f}_\text{type}^w] \in \mathbb{R}^{4n}$ is the concatenation of the four vectors. The meta-data features are specific to fields and the table representations are fused with place-holder $\boldsymbol{0}$ vectors.

2.3 Bridiging

Modeling only the table/field names and their relations is not always enough to capture the semantics of the schema and its dependencies with the question. Consider the example in Figure 2, `Property_-Type_Code` is a general expression not explicitly mentioned in the question and without access to the set of possible field values, it is difficult to associate "houses" and "apartments" with it. To resolve this problem, we make use of *anchor text* to link value mentions in the question with the corresponding DB fields. We perform fuzzy string match between Q and the picklist of each field in the DB. The matched field values (anchor texts) are inserted into the question-schema representation X, succeeding the corresponding field names and separated by the special token `[V]`. If multiple values were matched for one field, we concatenate all of them in matching order (Figure 2). If a question mention is matched with values in multiple fields. We add all matches and let the model learn to resolve ambiguity[1].

The anchor texts provide additional lexical clues for BERT to identify the corresponding mention in Q. And we name this mechanism "bridging".

2.4 Decoder

We use an LSTM-based pointer-generator (See et al., 2017) with multi-head attention (Vaswani et al., 2017) as the decoder. The decoder starts from the final state of the question encoder. At each step, the decoder performs one of the following actions: generating a token from the vocabulary $\mathcal{V}$, copying a token from the question Q or copying a schema component from $\mathcal{S}$.

Mathematically, at each step t, given the decoder state $\boldsymbol{s}_t$ and the encoder representation $[\boldsymbol{h}_Q; \boldsymbol{h}_S] \in \mathbb{R}^{(|Q|+|\mathcal{S}|)\times n}$, we compute the multi-head attention as defined in Vaswani et al. (2017):

$$e_{tj}^{(h)} = \frac{\boldsymbol{s}_t W_U^{(h)} (\boldsymbol{h}_j W_V^{(h)})^\top}{\sqrt{n/H}}; \quad \alpha_{tj}^{(h)} = \operatorname*{softmax}_j \left\{ e_{tj}^{(h)} \right\} \tag{4}$$

$$z_t^{(h)} = \sum_{j=1}^{|Q|+|\mathcal{S}|} \alpha_{tj}^{(h)} (\boldsymbol{h}_j W_V^{(h)}); \quad z_t = [z_t^{(1)}; \cdots; z_t^{(H)}], \tag{5}$$

where $h \in [1, \ldots, H]$ is the head number and H is the total number of heads.

The scalar probability of generating from $\mathcal{V}$ and the output distribution are

$$p_\text{gen}^t = \text{sigmoid}(\boldsymbol{s}_t \boldsymbol{W}_\text{gen}^s + \boldsymbol{z}_t \boldsymbol{W}_\text{gen}^z + \boldsymbol{b}_\text{gen}) \tag{6}$$

$$p_\text{out}^t = p_\text{gen}^t P_\mathcal{V}(y_t) + (1 - p_\text{gen}^t) \sum_{j:\tilde{X}_j = y_t} \alpha_{tj}^{(H)}, \tag{7}$$

where $P_\mathcal{V}(y_t)$ is the softmax LSTM output distribution and $\tilde{X}$ is the length-($|Q| + |\mathcal{S}|$) sequence that consists of only the question words and special tokens `[T]` and `[C]` from X. We use the attention weights of the last head to compute the pointing distribution[2].

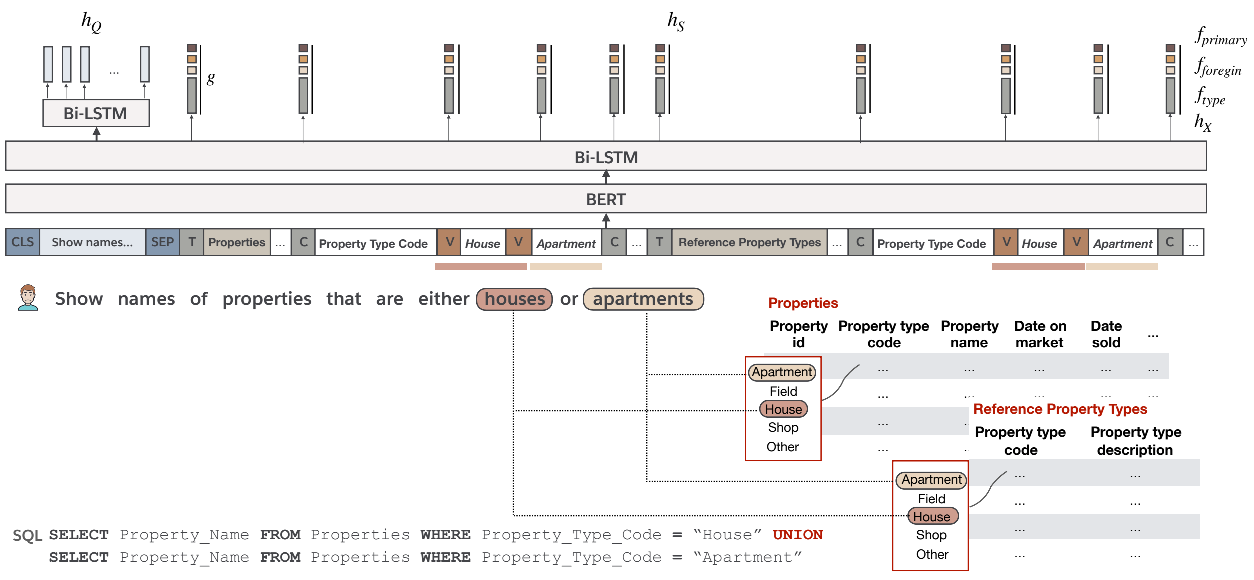
We extend the input state to the LSTM decoder using *selective read* proposed by Gu et al. (2016). The technical details of this extension can be found in §A.2.

2.5 Schema-Consistency Guided Decoding

We propose a simple pruning strategy for sequence decoders, based on the fact that the DB fields ap-

[1]This approach may over-match anchor texts from fields other than those in the correct SQL query. Yet keeping the additional matches in X may provide useful information rather than noise. We plan to verify this in future work.

[2]In practice we find this approach better than using just one head or using the average of multiple head weights.



```
SQL SELECT Property_Name FROM Properties WHERE Property_Type_Code = "House" UNION
    SELECT Property_Name FROM Properties WHERE Property_Type_Code = "Apartment"
```

Figure 2: The BRIDGE encoder. The two phrases "houses" and "apartments" in the input question both matched to two DB fields. The matched values are appended to the corresponding field names in the hybrid sequence.

peared in each SQL clause must only come from the tables in the `FROM` clause.

Generating SQL Clauses in Execution Order To this end we rearrange the clauses of each SQL query in the training set into the standard DB execution order (Rob and Coronel, 1995) shown in table 1. For example, the SQL `SELECT COUNT(*) FROM Properties` is converted to `FROM Properties SELECT COUNT(*)`[3]. We can show that all SQL queries with clauses in execution order satisfy the following lemma

Lemma 1 *Let Y_{exec} be a SQL query with clauses arranged in execution order, then any table field in Y_{exec} must appear after the table.*

As a result, we adopt a binary attention mask ξ

$$\tilde{\alpha}_t^{(H)} = \alpha_t^{(H)} \cdot \xi \tag{8}$$

which initially has entries corresponding to all fields set to 0. Once a table t_i is decoded, we set all entries in ξ corresponding to $\{c_{i1}, \ldots, c_{i|T_i|}\}$ to 1. This allows the decoder to only search in the space specified by the condition in Lemma 1 with little overhead in decoding speed.

3 Related Work

Text-to-SQL Semantic Parsing Recently the field has witnessed a re-surge of interest for text-to-SQL semantic parsing, by virtue of the newly released large-scale datasets (Zhong et al., 2017; Yu et al., 2018; Zhang et al., 2019) and matured neural network modeling tools (Vaswani et al., 2017; Shaw et al., 2018; Devlin et al., 2019). While existing models have surpassed human performance on benchmarks consisting of single-table and simple SQL queries (Hwang et al., 2019; Lyu et al., 2020; He et al., 2019a), ample space of improvement still remains for the Spider benchmark which consists of relational DBs and complex SQL queries[4].

Recent work have proposed increasingly complicated architectures for this problem (Guo et al., 2019; Wang et al., 2019; Choi et al., 2020), with extensions in both the encoder and the decoder. Bogin et al. (2019a,b) proposed to encode relational DB schema as a graph and also use the graph structure to guide decoding. Guo et al. (2019) proposes schema-linking and SemQL, an intermediate SQL representation customized for questions in the Spider dataset which was synthesized via a tree-based decoder. Wang et al. (2019) proposes RAT-SQL, a unified graph encoding mechanism which effectively covers relations in the schema graph and its linking with the question. The overall architec-

Written:	SELECT	FROM	WHERE	GROUPBY	HAVING	ORDERBY	LIMIT
Exec:	FROM	WHERE	GROUPBY	HAVING	SELECT	ORDERBY	LIMIT

Table 1: The written order vs. execution order of all SQL clauses appeared in Spider.

[3]More complex examples can be found in Table A1.

[4]https://yale-lily.github.io/spider

ture of RAT-SQL is deep, consisting of 8 relational self-attention layers on top of BERT-large.

In comparison, BRIDGE uses BERT combined with minimal subsequent layers. It uses a simple sequence decoder with search space-pruning heuristics and applies little abstraction to the SQL surface form. Its encoding architecture took inspiration from the table-aware BERT encoder proposed by Hwang et al. (2019), which is very effective for WikiSQL but has not been successful adapted to Spider. Yavuz et al. (2018) also uses question-value matches to achieve high-precision condition predictions on WikiSQL. While previous work such as (Guo et al., 2019; Wang et al., 2019; Yin et al., 2020) use feature embeddings or relational attention layers to explicitly model schema linking, BRIDGE models the linking implicitly with BERT and lexical anchors.

Textual-Tabular Data Joint Representation and Pre-training BRIDGE is a general framework for jointly representing question, relational DB schema and DB values, and has the potential to be applied to a wide range of problems that requires joint textual-tabular data understanding. Recently, Yin et al. (2020) proposes TaBERT, an LM for jointly representing textual and tabular data pre-trained over millions of web tables. Similarly, Herzig et al. (2020) proposes TaPas, a pre-trained text-table LM that supports arithmetic operations for weakly supervised table QA. Both TaBERT and TaPaSand supports arit focus on representing text with a single table. TaBERT was applied to Spider by encoding each table individually and modeling cross-table correlation through hierarchical attention. In comparison, BRIDGE serialized the relational DB schema and uses BERT to model cross-table dependencies. TaBERT adopts the "content snapshot" mechanism which retrieves rows from a table most similar to the input question and jointly encodes them with the table header. Compared to BRIDGE which uses the anchor texts, table rows are not always available if DB content access is restricted. Furthermore, anchor texts provide more focused signals that link the text and the DB schema.

4 Experiment Setup

4.1 Dataset

We evaluate BRIDGE using Spider (Yu et al., 2018), a large-scale, human annotated, cross-database text-to-SQL benchmark. Table 2 shows the statistics of its train/dev/test splits. The test set is hidden. We run hyperparameter search and analysis on the dev set and report the test set performance only using our best approach.

	# Q	# SQL	#DB
Train	8,695	4,730	140
Dev	1,034	564	20
Test	2,147	–	40

Table 2: Spider Dataset Statistics

We also report SOTA results on WikiSQL (Zhong et al., 2017) in § A.7.

4.2 Evaluation Metrics

We report the official evaluation metrics proposed by the Spider team.

Exact Set Match (E-SM) This metrics evaluates the structural correctness of the predicted SQL by checking the orderless set match of each SQL clause in the predicted query w.r.t. the ground truth. It ignores errors in the predicted values.

Execution Accuracy (EA) This metrics checks if the predicted SQL is executable on the target DB and if the execution results of match those of the ground truth. It is a performance upper bound as two SQL queries with different semantics can execute to the same results on a DB.

4.3 Implementation Details

Anchor Text Selection Given a Spider DB, we compute the pickist of each field using the official DB files. We convert the question and field values into lower cased character sequences and compute the longest sub-sequence match with heuristically determined matching boundaries[5]. We restrict the matching character sequence to a phrase in the question and exclude all number matches[6]. We define a matching score function and only consider matches that score above a tuned threshold (described in detail in §A.3). We include up to k matches per field, and break ties by taking the longer match.

[5]This allows us to capture plural/singular nouns and other trivial string variations.

[6]A number mention in the question often does not correspond to a DB cell (e.g. "shoes lower than $50") or cannot effectively discriminate between different fields.

Data Repair The original Spider dataset contains errors in both the example files and database files. We manually corrected some errors in the train and dev examples. For comparison with others in §5.1, we report metrics using the official dev/test sets. For our own ablation study and analysis, we report metrics using the corrected dev files. We also use a high-precision heuristics to identify missing foreign key pairs in the databases and combine them with the released ones during training and inference: if two fields of different tables have identical name and one of them is a primary key, we count them as a foreign key pair[7].

Training We train our model using cross-entropy loss. We use Adam-SGD (Kingma and Ba, 2015) with default parameters and a mini-batch size of 32. We use the uncased BERT-base model from the Huggingface's transformer library (Wolf et al., 2019). We set all LSTMs to 1-layer and set the hidden state dimension $n = 512$. We train a maximum of 50,000 steps and set the learning rate to $5e-4$ in the first 5,000 iterations and linearly shrink it to 0. We fine-tune BERT with a fine-tuning rate linearly increasing from $3e-5$ to $8e-5$ in the first 5,000 iterations and linearly decaying to 0. We randomly permute the table order in a DB schema and drop one table which does not appear in the ground truth with probability 0.3 in every training step. The training time of our model on a Tesla V100-SXM2-16GB GPU is approximately 33h (including intermediate results verification time).

Decoding The decoder uses a generation vocabulary consisting of 70 SQL keywords and reserved tokens, plus the 10 digits to generate numbers not explicitly mentioned in the question (e.g. "first", "second", "youngest" etc.). We use a beam size of 256 for leaderboard evaluation. All other experiments uses a beam size of 16. We use schema-consistency guided decoding during inference only. It cannot guarantee schema consistency[8] and we run a static SQL correctness check on the beam search output to eliminate predictions that are either syntactically incorrect or violates schema consistency[9] If no predictions in the beam satisfy the two criteria, we output a default SQL query which count the number of entries in the first table.

Model	**Dev**	**Test**
Global-GNN (Bogin et al., 2019b) ♠	52.7	47.4
EditSQL + BERT (Zhang et al., 2019)	57.6	53.4
GNN + Bertrand-DR (Kelkar et al., 2020)	57.9	54.6
IRNet + BERT (Guo et al., 2019)	61.9	54.7
RAT-SQL v2 ♠ (Wang et al., 2019)	62.7	57.2
RYANSQL + BERT$_L$ (Choi et al., 2020)	*66.6*	58.2
RYANSQL v2 + BERT$_L$ ◇	*70.6*	*60.6*
RAT-SQL v3 + BERT$_L$ ♠ (Wang et al., 2019)	*69.7*	*65.6*
BRIDGE (k = 1) (ours) ♠ ♡	65.3	–
BRIDGE (k = 2) (ours) ♠ ♡	**65.5**	**59.2**

Table 3: Exact set match on the Spider dev and test sets, compared to the other top-performing approaches on the leaderboard as of June 1st, 2020. The test set results were issued by the Spider team. BERT$_L$ denotes BERT$_{\text{LARGE}}$. ◇ denotes approaches without publication reference. ♠ denotes approaches using DB content. ♡ denote approaches that output executable SQL queries.

5 Results

5.1 End-to-end Performance Evaluation

Table 3 shows the E-SM accuracy of BRIDGE compared to other approaches ranking at the top of the Spider leaderboard. BRIDGE performs very competitively, significantly outperforming most of recently proposed architectures with more complicated, task-specific layers (Global-GNN, EditSQL+BERT, IRNet+BERT, RAT-SQL v2, RYANSQL+BERT$_L$). We find changing k from 1 to 2 yield marginal performance improvement since only 77 SQL queries in the dev set contains more than one textual values (Figure A1). In addition, BRIDGE generates executable SQL queries by copying values from the input question while most existing models do not. As of June 1st, 2020, BRIDGE ranks top-1 on the Spider leaderboard by execution accuracy.

The two approaches significantly better than BRIDGE by E-SM are RYANSQL v2+BERT$_L$ and RAT-SQL v3+BERT$_L$. We further look at the performance comparison with RAT-SQL v3+BERT$_L$ across different difficulty level in Table 4. Both

[7] We exclude common field names such as "name", "id" and "code" in this procedure.

[8] Consider the example SQL query shown in Table A2 which satisfies the condition of Lemma 1, the table `VOTING_RECORD` only appears in the first sub-query, and the field `VOTING_RECORD.PRESIDENT_Vote` in the second sub-query is out of scope.

[9] Prior work such as (Wang et al., 2018) performs the more aggressive execution-guided decoding. However, it is difficult to apply this approach to complex SQL queries (Zhong et al., 2017). We build a static SQL analyzer on top of the Mozilla SQL Parser (`https://github.com/mozilla/moz-sql-parser`). Our static checking approach handles complex SQL queries and avoids DB execution overhead.

Model	Easy	Medium	Hard	Ex-Hard	All
count	250	440	174	170	1034
Dev					
BRIDGE (k = 2) ♠	88.4	68	51.7	39.4	65.5
RAT-SQL v3+B$_L$ ♠	86.4	**73.6**	**62.1**	**42.9**	**69.7**
Test					
BRIDGE (k = 2) ♠	80	62	51	35.6	59.2
IRNet+B	77.2	58.7	48.1	25.3	54.7
RAT-SQL v3+B$_L$ ♠	**83.0**	**71.3**	**58.3**	**38.4**	**65.6**

Table 4: E-SM broken by hardness level compared to other top-performing approaches on Spider leaderboard.

Model	Exact Set Match (%)	
	Mean	Max
BRIDGE (k = 2)	65.8 ± 0.8	66.9
- SC-guided decoding	65.4 ± 0.7	66.3 **(-0.6)**
- static SQL check	64.8 ± 0.9	65.9 **(-1.0)**
- execution order	64.2 ± 0.1	64.3 **(-2.6)**
- table shuffle & drop	63.9 ± 0.3	64.3 **(-2.6)**
- anchor text	63.3 ± 0.6	63.9 **(-3.0)**
- BERT	17.7 ± 0.7	18.3 **(-48.6)**

Table 5: BRIDGE ablations on the dev set. We report the exact set match accuracy of each model variations averaged over 3 runs.

model achieves > 80% E-SM accuracy in the easy category, but BRIDGE shows more significant overfitting. BRIDGE also underperforms RAT-SQL v3+BERT$_L$ in the other three categories, with considerable gaps in medium and hard.

As descirbed in §3, RAT-SQL v3 uses very different encoder and decoder architectures compared to BRIDGE and it is difficult to conduct a direct comparison without a model ablation[10]. We hypothesize that the most critical difference that leads to the performance gap is in their encoding schemes. RAT-SQL v3 explicitly models the question-schema-value matching via a graph and the matching condition (full-word match, partial match, etc.) are used to label the graph edge. BRIDGE represents the same information in a tagged sequence and uses fine-tuned BERT to implicitly obtain such mapping. While the anchor text selection algorithm (§4.3) has taken into account string variations, BERT may not be able to capture the linking when string variations exist – it has not seen tabular input during pre-training. The tokenization scheme adopted by BERT and other pre-trained LMs (e.g. GPT-2) cannot effectively capture partial string matches in a novel input (e.g. "cats" and "cat" are two different words in the vocabularies of BERT and GPT-2). We think recent works on text-table joint pretraining have the potential to overcome this problem (Yin et al., 2020; Herzig et al., 2020). In addition, RAT-SQL v3 uses BERT$_{\text{LARGE}}$ which has a significantly larger number of parameters than BRIDGE. We plan to also train BRIDGE with BERT-large in the future for an equal comparison.

5.2 Ablation Study

We perform a thorough ablation study to show the contribution of each BRIDGE sub-component (Table 5). Overall, all sub-components significantly contributed to the model performance. The decoding search space pruning strategies we introduced (including generation in execution order, schema-consistency guided decoding and static SQL correctness check) are effective, with absolute E-SM improvements ranging from 0.6% to 2.6%. However, encoding techniques for bridging textual and tabular input contribute more. Especially, adding anchor texts results in an absolute E-SM improvement of 3%. A further comparison between BRIDGE with and without anchor texts (Table A3) shows that anchor text augmentation improves the model performance at all hardness levels, especially in the hard and extra-hard categories. Shuffling and randomly dropping non-ground-truth tables during training also significantly helps our approach, as it increases the diversity of DB schema seen by the model and reduces overfitting to a particular table arrangement.

Moreover, BERT is critical to the performance of BRIDGE, magnifying performance of the base model by more than three folds. This is considerably larger than the improvement prior approaches have obtained from adding BERT. Consider the performances of RAT-SQL v2 and RAT-SQL v2+BERT$_L$ in Table 3, the improvement with BERT$_L$ is 7%. This shows that simply adding BERT to existing approaches results in significant redundancy in the model architecture. We perform a qualitative attention analysis in §A.8 to show that after fine-tuning, the BERT layers effectively capture the linking between question mentions and the anchor texts, as well as the relational DB structures.

[10]RAT-SQL v3 entered the leaderboard within a month of EMNLP deadline and hasn't released its source code.

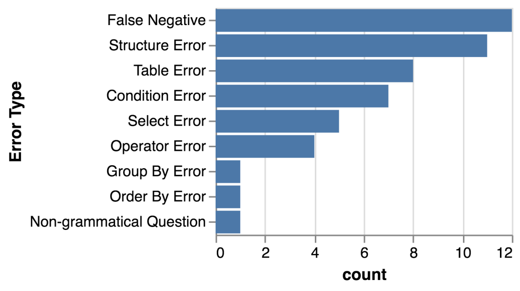


Figure 3: BRIDGE error type counts.

5.3 Error Analysis

We randomly sampled 50 dev set examples for which the best BRIDGE model failed to produce a prediction that matches the ground truth and manually categorized the errors. Each example is assigned to only the category it fits most.

Error Types Figure 3 shows the number of examples in each category. 24% of the examined predictions are false negatives. Among them, 7 are semantically equivalent to the ground truths; 4 contain `GROUP BY` keys different but equivalent to those of the ground truth (e.g. `GROUY BY car_models.name` vs. `GROUP BY car_models.id`); 1 has the wrong ground truth annotation. Among the true negatives, 11 have SQL structures completely deviated from the ground truth. 22 have errors that can be pinpointed to specific clauses: `FROM` (8), `WHERE` (7), `SELECT` (5), `GROUP BY` (1), `ORDER BY` (1). 4 have errors in the operators: 3 in the aggregation operator and 1 in the comparison operator. 1 example has non-grammatical natural language question.

Error Causes A prominent cause of errors for BRIDGE is irregular design and naming in the DB schema. Table A4 shows 3 examples where BRIDGE made a wrong prediction from the medium hardness level in the dev set. In the second example, the DB contains a field named “hand” which stores information that indicates whether a tennis player is right-handed or left-handed. While “hand” is already a rarely seen field name (comparing to “name”, “address” etc.), the problem is worsened by the fact that the field values are acronyms which bypassed the anchor text match. Similarly, in the third example, BRIDGE fails to detect that “highschooler”, normally written as “high schooler” is a synonym of student. Occasionally, however, BRIDGE still makes mistakes w.r.t. schema components explicitly mentioned in the question, as shown by the first example. Addressing such error cases could further improve its performance.

6 Discussion

Anchor Selection BRIDGE adopts simple string matching for anchor text selection. Our early experiments show that improving anchor text selection accuracy significantly improves the end-to-end accuracy. Extending anchor text matching to cases beyond simple string match (e.g. “LA”→“Los Angeles”) is also a future direction. Moreover, this step can be trained, either independently or jointly with the text-to-SQL objective. Currently BRIDGE ignores number mentions, which is not always ideal. We can introduce features indicating information such as a specific number mention in the question falls within the range of a specific column. Furthermore, we can enable copying of the anchor texts into the output SQL to improve execution accuracy (currently BRIDGE only copy values from the question).

Input Size As BRIDGE serializes all inputs into a sequence with special tags, a fair concern is that the input would be too long for large relational DBs. We believe this can be addressed with recent architecture advancements in transformers (Beltagy et al., 2020), which have scaled up the attention mechanism to model very long sequences.

Relation Encoding BRIDGE fuses DB schema meta data features to each individual table field representations. This mechanism is not as strong as directly modeling the original graph structure. It works well in Spider, where the foreign key pairs often have exactly the same names. We consider regularizing specific attention heads to capture DB connections (Strubell et al., 2018) a promising way to model the graph structure of relational DBs within the BRIDGE framework without introducing (a lot of) additional parameters.

7 Conclusion

We present BRIDGE, a powerful sequential architecture for modeling dependencies between natural language question and relational DBs in cross-DB semantic parsing. BRIDGE serializes the question and DB schema into a tagged sequence and maximally utilizes pre-trained LMs such as BERT to capture the linking between text mentions and the DB schema components. It uses anchor texts to further improve the alignment between the two cross-modal inputs. Combined with a simple sequential pointer-generator decoder with schema-consistency driven search space pruning, BRIDGE attained state-of-the-art performance on Spider. In the future, we plan to study the application of BRIDGE to other text-DB tasks such as fact checking and weakly supervised question answering.

References

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *CoRR*, abs/2004.05150.

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 Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4560–4565. 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 2019, Hong Kong, China, November 3-7, 2019*, pages 3657–3662. Association for Computational Linguistics.

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. *CoRR*, abs/2004.03125.

Deborah A. Dahl, Madeleine Bates, Michael Brown, William M. Fisher, Kate Hunicke-Smith, David S. Pallett, Christine Pao, Alexander I. Rudnicky, and Elizabeth Shriberg. 1994. Expanding the scope of the ATIS task: The ATIS-3 corpus. In *Human Language Technology, Proceedings of a Workshop held at Plainsboro, New Jerey, USA, March 8-11, 1994*.

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, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1*, pages 4171–4186.

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, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O. K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

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 Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4524–4535.

Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. 2019a. X-SQL: reinforce schema representation with context. *CoRR*, abs/1908.08113.

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

Charles T. Hemphill, John J. Godfrey, and George R. Doddington. 1990. The ATIS spoken language systems pilot corpus. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, USA, June 24-27, 1990*.

Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Seattle, Washington, United States. To appear.

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

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

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 *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Anna Korhonen, David R. Traum, and Lluís Màrquez, editors. 2019. *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*. Association for Computational Linguistics.

Chen Liang, Mohammad Norouzi, Jonathan Berant, Quoc V. Le, and Ni Lao. 2018. Memory augmented policy optimization for program synthesis and semantic parsing. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 10015–10027.

Tianyu Liu, Fuli Luo, Pengcheng Yang, Wei Wu, Baobao Chang, and Zhifang Sui. 2019a. Towards comprehensive description generation from factual attribute-value tables. In (Korhonen et al., 2019), pages 5985–5996.

Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019b. Multi-task deep neural networks for natural language understanding. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4487–4496, Florence, Italy. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019c. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Qin Lyu, Kaushik Chakrabarti, Shobhit Hathi, Souvik Kundu, Jianwen Zhang, and Zheng Chen. 2020. Hybrid ranking network for text-to-sql. Technical Report MSR-TR-2020-7, Microsoft Dynamics 365 AI.

Peter Rob and Carlos Coronel. 1995. *Database systems - design, implementation, and management (2. ed.)*. Boyd and Fraser.

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, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 1073–1083.

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, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 464–468. Association for Computational Linguistics.

Emma Strubell, Patrick Verga, Daniel Andor, David Weiss, and Andrew McCallum. 2018. Linguistically-informed self-attention for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 5027–5038. Association for Computational Linguistics.

Alane Suhr, Ming-Wei Chang, Peter Shaw, and Kenton Lee. 2020. Exploring unexplored generalization challenges for cross-database semantic parsing. In *The 58th annual meeting of the Association for Computational Linguistics (ACL)*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008.

Jesse Vig. 2019. A multiscale visualization of attention in the transformer model. *arXiv preprint arXiv:1906.05714*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Margot Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *ArXiv*, abs/1911.04942.

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

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

Semih Yavuz, Izzeddin Gur, Yu Su, and Xifeng Yan. 2018. What it takes to achieve 100 percent condition accuracy on wikisql. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 1702–1711. Association for Computational Linguistics.

Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. 2020. Tabert: Pretraining for joint understanding of textual and tabular data. *CoRR*, abs/2005.08314.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir R. Radev. 2018. 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, Brussels, Belgium, October 31 - November 4, 2018*, pages 1653–1663.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Richard Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter S. Lasecki, and Dragomir R. Radev. 2019a. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. *CoRR*, abs/1909.05378.

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 R. Radev. 2019b. Sparc: Cross-domain semantic parsing in context. In (Korhonen et al., 2019), pages 4511–4523.

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 and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 2.*, pages 1050–1055.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005*, pages 658–666. AUAI Press.

Rui Zhang, Tao Yu, Heyang Er, Sungrok Shim, Eric Xue, Xi Victoria Lin, Tianze Shi, Caiming Xiong, Richard Socher, and Dragomir R. Radev. 2019. Editing-based SQL query generation for cross-domain context-dependent questions. *CoRR*, abs/1909.00786.

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

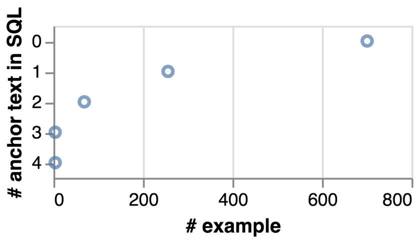


Figure A1: Distribution of # textual values in the ground truth SQL queries on Spider dev set.

A Appendix

A.1 Examples of SQL queries with clauses arranged in execution order

We show more examples of complex SQL queries with their clauses arranged in written order vs. execution order in Table A1.

A.2 Selective read decoder extension

The selective read operation was introduced by Gu et al. (2016). It extends the input state to the decoder LSTM with the corresponding encoder hidden states of the tokens being copied. This way the decoder was provided information on which part of the input has been copied.

Specically, we modified the input state of our decoder LSTM to the following:

$$\boldsymbol{y}_t = [\boldsymbol{e}_{t-1}; \zeta_{t-1}] \in \mathbb{R}^{2n}, \tag{9}$$

where $\boldsymbol{e}_{t-1} \in \mathbb{R}^n$ is either the embedding of a generated vocabulary token or a learned vector indicating if a table, field or question token is copied in step $t-1$. $\zeta_{t-1} \in \mathbb{R}^n$ is the *selective read* vector, which is a weighted sum of the encoder hidden states corresponding to the tokens copied in step $t-1$:

$$\zeta(y_{t-1}) = \sum_{j=1}^{|Q|+|S|} \rho_{t-1,j}\boldsymbol{h}_j; \quad \rho_{t-1,j} = \begin{cases} \frac{1}{K}\alpha^{(H)}_{t-1,j}, & \tilde{X}_j = y_{t-1} \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Here $K = \sum_{j:\tilde{X}_j = y_{t-1}} \alpha^{(H)}_{t-1,j}$ is a normalization term considering there may be multiple positions equals to y_{t-1} in $\tilde{X}$.

A.3 Anchor text selection

We convert the question and field values into lower cased character sequences and compute the longest sub-sequence match with heuristically determined matching boundaries. For example, the sentence "how many students keep cats as pets?" matches with the cell value "cat" (s_c) and the matched substring is "cat" (s_m). We further search the question starting from the start and end character indices i, j of s_m in the question to make sure that word boundaries can be detected within $i-2$ to $j+2$, otherwise the match is invalidated. This excludes matches which are sub-strings of the question words, e.g. "cat" vs. "category". Denoting matched whole-word phrase in the question as s_q, we define the question match score and cell match score as

$$\beta_q = |s_m|/|s_q| \tag{11}$$

$$\beta_c = |s_c|/|s_q| \tag{12}$$

We define a coarse accuracy measurement to tune the question match score threshold θ_q and the cell match threshold θ_c. Namely, given the list of matched anchor texts $\mathcal{P}$ obtained using the aforementioned procedure and the list of textual values $\mathcal{G}$ extracted from the ground truth SQL query, when compute the percentage of anchor texts appeared in $\mathcal{G}$ and the percentage of values in $\mathcal{G}$ that appeared in $\mathcal{P}$ as approximated precision (p') and recall (r'). Note that this metrics does not evaluate if the matched anchor texts are associated with the correct field.

For $k = 2$, we set $\theta_q = 0.5$ and $\theta_c = 0.8$. On the training set, the resulting $p' = 73.7, r' = 74.9$. 25.7% examples have at least one anchor text match with 1.89 average number of matches per example among them. On the dev set, the resulting $p' = 90.0, r' = 92.2$. 30.9% examples have at least one match with 1.73 average number of matches per example among them. The training set metrics are lower as some training databases do not have DB content files.

A.4 Anchor text ablation by hardness level

Table A3 shows the E-SM comparison between models with and without anchor text augmentation at different hardness level. Anchor text augmentation improves performance at all hardness levels, with the improvement especially significant in the hard and extra-hard categories.

A.5 Sample error cases

Table A4 shows examples of mistakes by BRIDGE on the Spider dev set, all selected from the medium hardness level.

A.6 Performance by database

Figure A2 shows the E-SM accuracy of BRIDGE for different DBs of the Spider dev set. We notice drastic performance differences across DBs. While

Written:	`SELECT rid FROM routes WHERE dst_apid IN (SELECT apid FROM airports WHERE country = 'United States') AND src_apid IN (SELECT apid FROM airports WHERE country = 'United States')`
Exec:	`FROM routes WHERE dst_apid IN (FROM airports WHERE country = 'United States' SELECT apid) AND src_apid IN (FROM airports WHERE country = 'United States' SELECT apid) SELECT rid`
Written:	`SELECT t3.name FROM publication_keyword AS t4 JOIN keyword AS t1 ON t4.kid = t1.kid JOIN publication AS t2 ON t2.pid = t4.pid JOIN journal AS t3 ON t2.jid = t3.jid WHERE t1.keyword = "Relational Database" GROUP BY t3.name HAVING COUNT(DISTINCT t2.title) = 60`
Exec:	`FROM publication_keyword AS t4 JOIN keyword AS t1 ON t4.kid = t1.kid JOIN publication AS t2 ON t2.pid = t4.pid JOIN journal AS t3 ON t2.jid = t3.jid WHERE t1.keyword = "Relational Database" GROUP BY t3.name HAVING COUNT(DISTINCT t2.title) = 60 SELECT t3.name`
Written:	`SELECT COUNT(DISTINCT state) FROM college WHERE enr < (SELECT AVG(enr) FROM college)`
Exec:	`FROM college WHERE enr < (FROM college SELECT AVG(enr)) SELECT COUNT(DISTINCT state)`
Written:	`SELECT DISTINCT T1.LName FROM STUDENT AS T1 JOIN VOTING_RECORD AS T2 ON T1.StuID = PRESIDENT_Vote EXCEPT SELECT DISTINCT LName FROM STUDENT WHERE Advisor = "2192"`
Exec:	`FROM STUDENT AS T1 JOIN VOTING_RECORD AS T2 ON T1.StuID = PRESIDENT_Vote SELECT DISTINCT T1.LName EXCEPT FROM STUDENT WHERE Advisor = 2192 SELECT DISTINCT LName`

Table A1: Examples of complex SQL queries with clauses in the normal order and the DB execution order.

```
FROM STUDENT JOIN VOTING_RECORD ON STUDENT.StuID = VOTING_RECORD.PRESIDENT_Vote SELECT
DISTINCT STUDENT.LName EXCEPTFROM STUDENT WHERE STUDENT.Advisor = 2192 SELECT DISTINCT
VOTING_RECORD.PRESIDENT_Vote
```

Table A2: An example sequence satisfies the condition of Lemma 1 but violates schema consistency. Here the field `VOTING_RECORD.PRESIDENT_Vote` in the second sub-query is out of scope.

Model	**Easy**	**Medium**	**Hard**	**Ex-Hard**	**All**
count	250	440	174	170	1034
BRIDGE ($k = 2$)	**88.7**	**68.4**	**54**	**44**	**66.9**
-value augmentation	85.5	66.6	49.4	39.8	63.9

Table A3: Comparison between BRIDGE and BRIDGE without value augmentation on our manually corrected dev set.

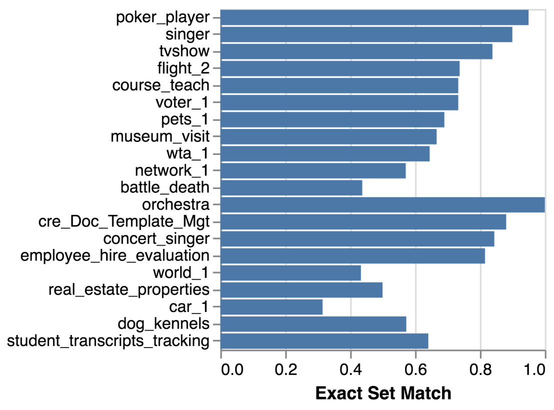


Figure A2: BRIDGE E-SM accuracy broken down by DB in Spider dev set. From top to bottom, the DBs are sorted by their schema sizes from small to large.

BRIDGE achieves near perfect score on some, the performance is only 30%-40% accurate on the others. The performance does not always negatively correlates with the schema size. We hypothesize that the model scores better on DB schema similar to those seen during training and plan to verify this in future work.

A.7 WikiSQL Experiments

We test BRIDGE on WikiSQL and report the comparison to other top-performing entries on the leaderboard in Table A5. BRIDGE achieves SOTA performance on WikiSQL, surpassing the widely cited SQLova model (Hwang et al., 2019) by a significant margin. Among the baselines shown in Table A5, SQLova is the one that's strictly comparable to BRIDGE as both use BERT-large-uncased. Hydra-Net uses RoBERTa-Large (Liu et al., 2019a) and X-SQL uses MT-DNN (Liu et al., 2019b). The introduction of anchor text from the table enables BRIDGE to be the best-performing model without execution-guided decoding (Wang et al., 2018). However, it seems to also reduce the degree the

	What are the names and release years for all the songs of the youngest singer? **concert_singer**
✗	`SELECT Song_Name, Age FROM singer ORDER BY Age LIMIT 1`
✓	`SELECT song_name, song_release_year FROM singer ORDER BY age LIMIT 1`
	What are the full names of all left handed players, in order of birth date? **WTA_1**
✗	`SELECT first_name, last_name FROM players ORDER BY birth_date`
✓	`SELECT first_name, last_name FROM players WHERE hand = 'L' ORDER BY birth_date`
	What are the names of students who have 2 or more likes? **network_1**
✗	`SELECT Likes.student_id FROM Likes JOIN Friend ON Likes.student_id = Friend.student_id GROUP BY Likes.student_id HAVING COUNT(*) >= 2`
✓	`SELECT T2.name FROM Likes AS T1 JOIN Highschooler AS T2 ON T1.student_id = T2.id GROUP BY T1.student_id HAVING count(*) >= 2`

Table A4: Errors cases of BRIDGE on the Spider dev set. The samples were randomly selected from the medium hardness level. ✗denotes the wrong predictions made by BRIDGE and ✓denotes the ground truths.

Model	**Dev**		**Test**	
	EM	**EX**	**EM**	**EX**
SQLova (Hwang et al., 2019)	81.6	87.2	80.7	86.2
X-SQL (He et al., 2019b)	83.8	89.5	83.3	88.7
HydraNet (Lyu et al., 2020)	83.6	89.1	83.8	89.2
BRIDGE +B$_L$ (k = 2) ♠	**85.1**	**91.1**	**84.8**	**90.4**
SQLova+EG (Hwang et al., 2019)	84.2	90.2	83.6	89.6
BRIDGE +B$_L$ (k = 2)+EG ♠	86.1	**92.5**	85.8	91.7
X-SQL+EG (He et al., 2019b)	86.2	92.3	86.0	91.8
HydraNet+EG (Lyu et al., 2020)	**86.6**	92.4	**86.5**	**92.2**

Table A5: Comparison between BRIDGE and other top-performing models on the WikiSQL leaderboard as of August 20, 2020. ♠ denotes approaches using DB content. +EG denotes approaches using execution-guided decoding.

model can benefit from it (after adding execution-guided decoding, the improvement by BRIDGE is significantly less than the other models).

A.8 Visualizing fine-turned BERT attention of BRIDGE

We visualize attention in the fine-tuned BERT layers of BRIDGE to qualitatively evaluate if the model functions as an effective text-DB encoder as we expect. We use the BERTViz library[11] developed by Vig (2019).

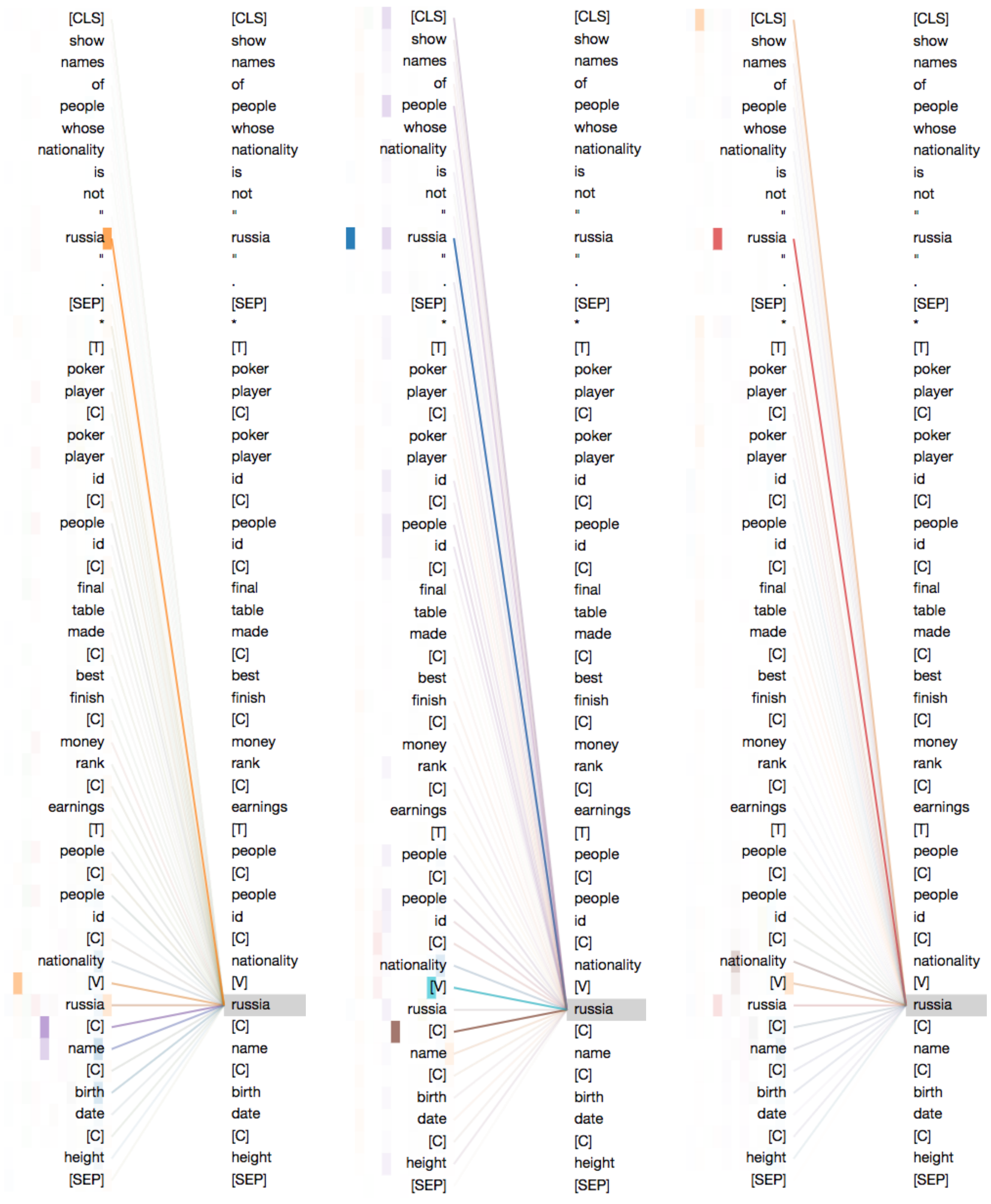
We perform the analysis on the smallest DB in the Spider dev set to ensure the attention graphs are readable. This DB consists of two tables, `Poker_Player` and `People` that store information of poker players and their match results. While the BERT attention is a complicated computation graph consisting of 12 layers and 12 heads, we were able to identify prominent patterns in a subset of the layers.

First, we examine if anchor texts indeed have the effect of bridging information across the textual and tabular segments. The example question we use is "show names of people whose nationality is not Russia" and "Russia" in the field `People.Nationality` is identified as the anchor text. As show in Figure A3 and Figure A4, we find strong connection between the anchor text and their corresponding question mention in layer 2, 4, 5, 10 and 11.

We further notice that the layers effectively captures the relational DB structure. As shown in Figure A5 and Figure A6, we found attention patterns in layer 5 that connect tables with their primary keys and foreign key pairs.

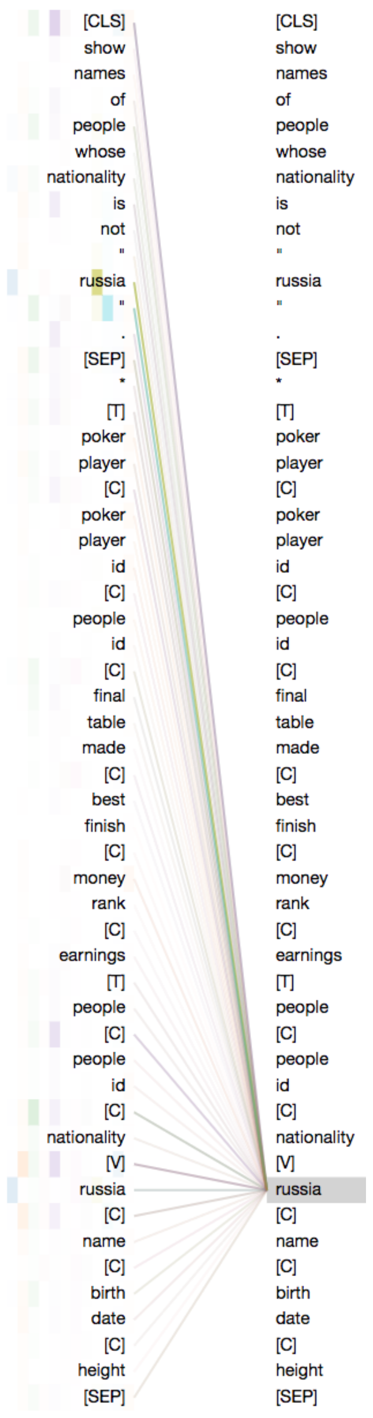
We notice that all interpretable attention connections are between lexical items in the input sequence, not including the special tokens (`[T]`, `[C]`, `[V]`). This is somewhat counter-intuitive as the subsequent layers of BRIDGE use the special tokens to represent each schema component. We hence examined attention over the special tokens (Figure A7) and found that they function as bindings of tokens in the table names and field names. The pattern is especially visible in layer 1. As shown in Figure A7, each token in the table name "poker player" has high attention to the corresponding `[T]`. Similarly, each token in the field name "poker player ID" has high attention to the corresponding `[C]`. We hypothesize that this way the special tokens function similarly as the cell pooling layers proposed in TaBERT (Yin et al., 2020).

[11] https://github.com/jessevig/bertviz

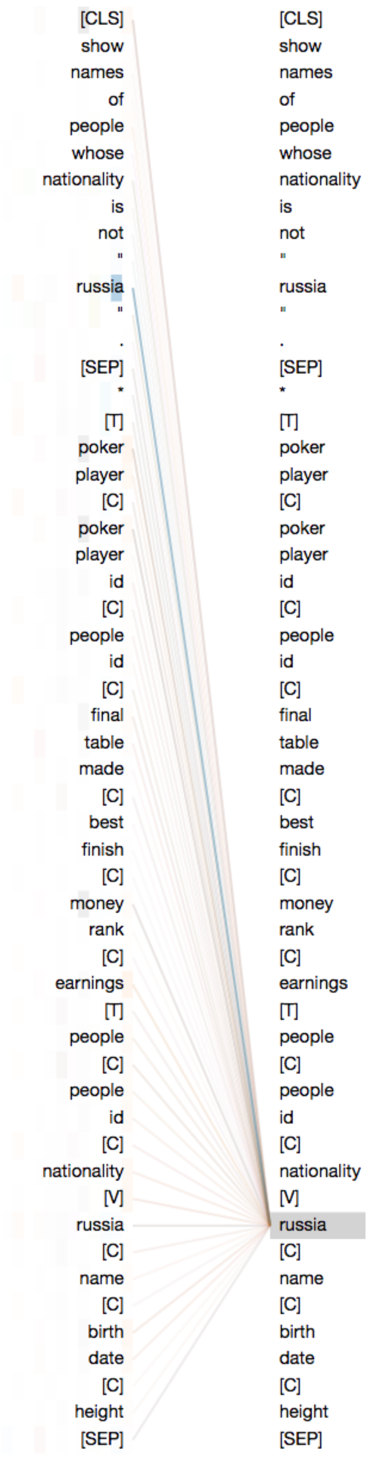


(a) Layer = 2 (b) Layer = 4 (c) Layer = 5

Figure A3: Visualization of attention to anchor text “Russia” from other words. In the shown layers, weights from the textual mention “Russia” is significantly higher than the other tokens.

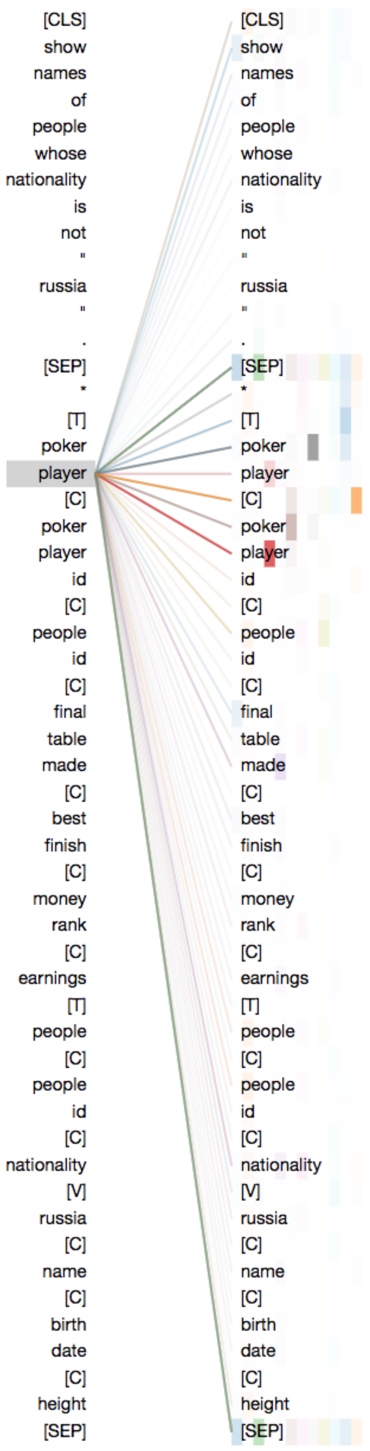


(a) Layer = 10

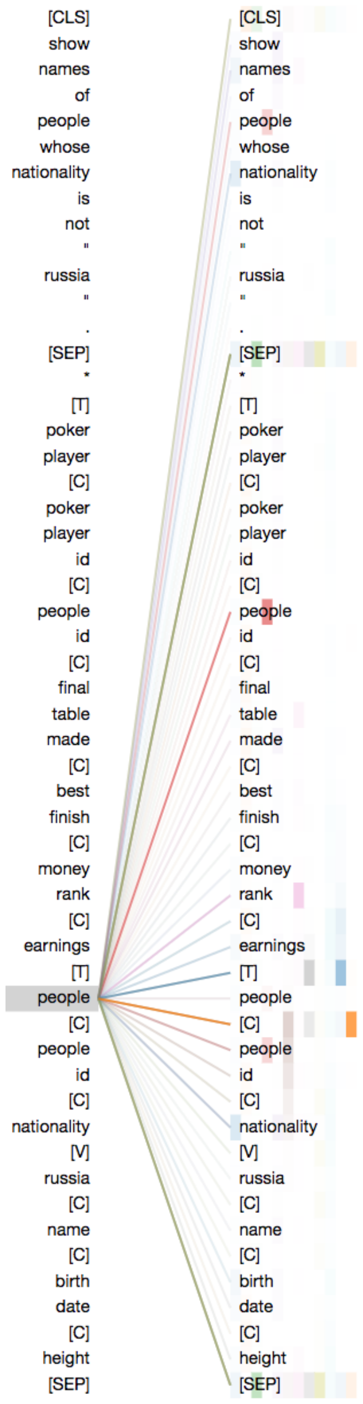


(b) Layer = 11

Figure A4: Visualization of attention to anchor text "Russia" from other words. Continue from Figure A3.

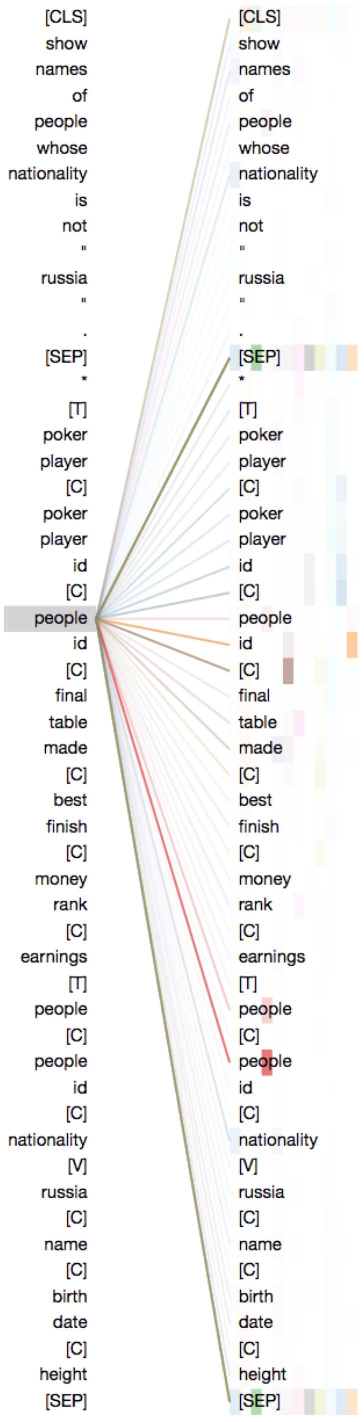


(a) Table `Poker_Player`

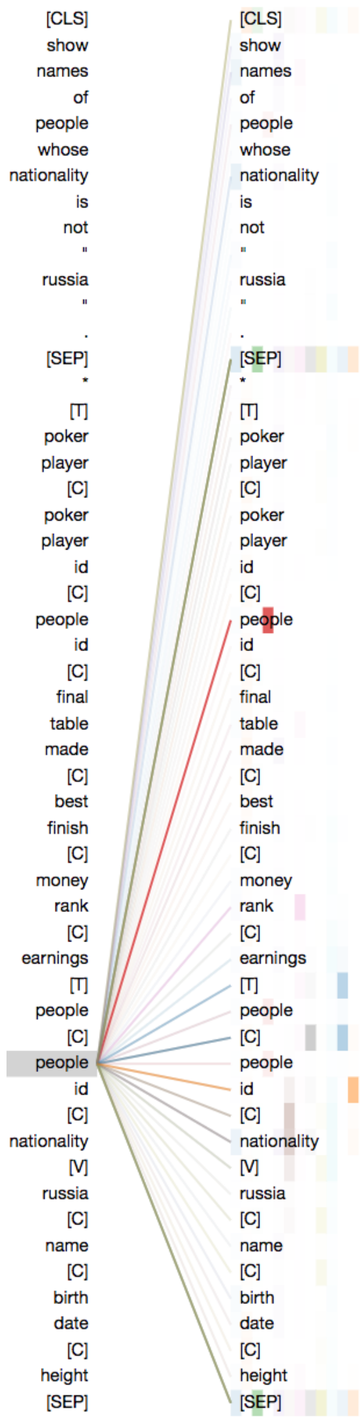


(b) Table `People`

Figure A5: Visualization of attention in layer 5 from tables to their primary keys. In Figure A5b, the table name `People` has high attention weights to `Poker_Player.People_ID`, a foreign key referring to its primary key `People.People_ID`.

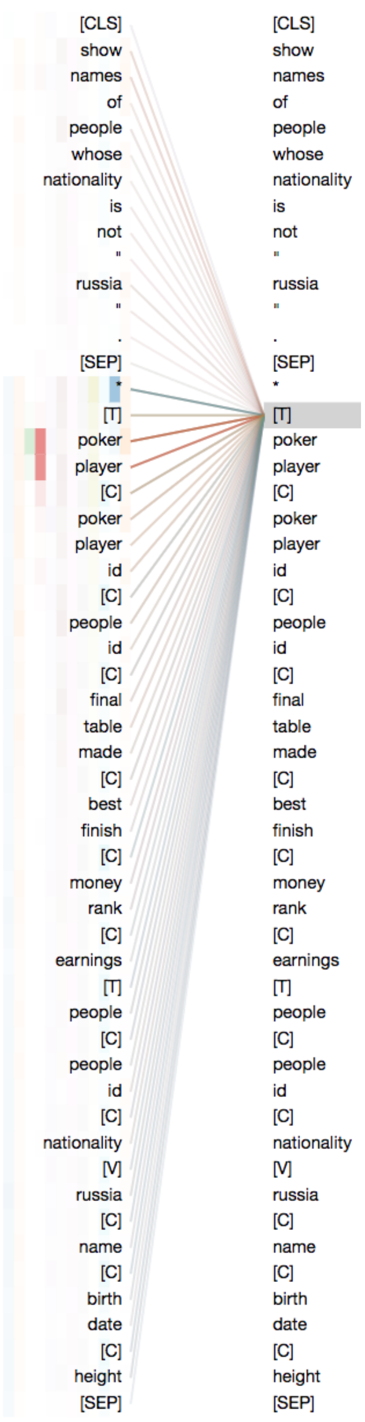


(a) `Poker_Player.People_ID` $\longrightarrow$ `People.People_ID`

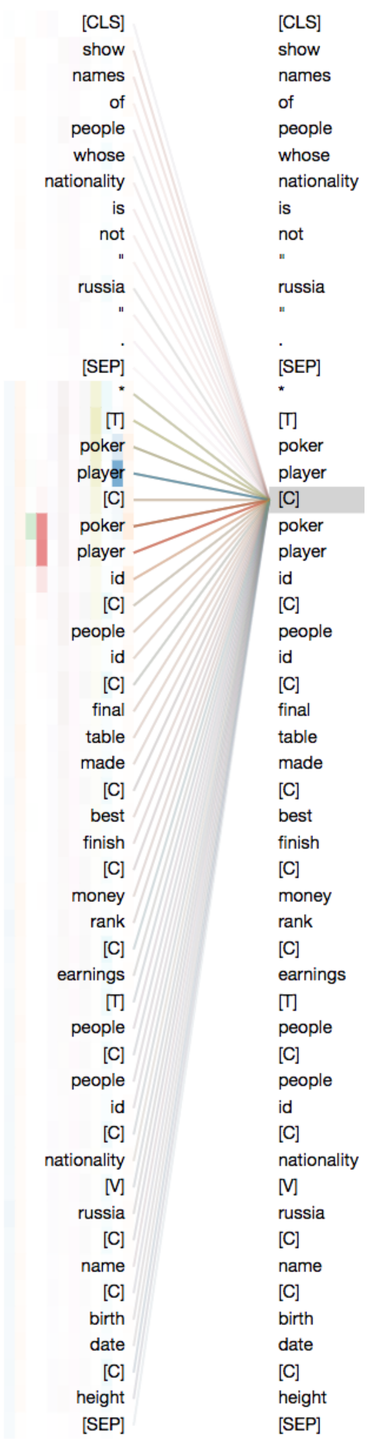


(b) `People.People_ID` $\longrightarrow$ `Poker_Player.People_ID`

Figure A6: Visualization of attention in layer 5 between a pair of foreign keys.



(a)



(b)

Figure A7: Visualization of attention over special tokens `[T]` and `[C]` in layer 1.