DuoRAT: Towards Simpler Text-to-SQL Models

Torsten Scholak*, Raymond Li*, Dzmitry Bahdanau, Harm de Vries, Chris Pal
Element AI, a ServiceNow company

Abstract

Recent neural text-to-SQL models can effectively translate natural language questions to corresponding SQL queries on unseen databases. Working mostly on the Spider dataset, researchers have proposed increasingly sophisticated solutions to the problem. Contrary to this trend, in this paper we focus on simplifications. We begin by building DuoRAT, a re-implementation of the state-of-the-art RAT-SQL model that unlike RAT-SQL is using only relation-aware or vanilla transformers as the building blocks. We perform several ablation experiments using DuoRAT as the baseline model. Our experiments confirm the usefulness of some techniques and point out the redundancy of others, including structural SQL features and features that link the question with the schema[1].

1 Introduction

Language user interfaces to databases allow non-specialists to retrieve and process information that might otherwise not be easily available to them. Much of the recent research in this area has focused on neural models that can generalize to new relational databases without any human intervention. Given a relational database schema (and often also content), such models translate the user's question directly into an SQL query (Zhong et al., 2017; Yu et al., 2018a; Bogin et al., 2019). Such cross-database *text-to-SQL* research was spurred by the introduction of large datasets such as WikiSQL (Zhong et al., 2017) and Spider (Yu et al., 2018b) that feature utterance-query pairs for hundreds or even thousands of databases.

State-of-the-art text-to-SQL models employ many sophisticated techniques. These include, but are not limited to, grammar-constrained models and recurrent neural networks with parent feeding (Yin and Neubig, 2018), intermediate meaning representations (Guo et al., 2019; Suhr et al., 2020), relation-aware attention (Wang et al., 2020), schema linking and table joining heuristics (Guo et al., 2019), slot filling (Choi et al., 2020) and re-ranking (Kelkar et al., 2020) models. The high complexity of these models raises the barrier of entry and can slow down text-to-SQL research.

In this work, we attempt to distill the essence of high-performing text-to-SQL systems. We start with a transformer-only reimplementation of the state-of-the-art RAT-SQL model (Wang et al., 2020). Importantly, our resulting DuoRAT model trains three times faster than RAT-SQL. We then systematically study how DuoRAT can be simplified without losing performance. Our ablation study confirms the usefulness of many but not all techniques employed in RAT-SQL. For example, we show that the benefits of explicit matching of question spans with the column or table names (*name-based schema linking*, NBSL) become marginal when a pretrained transformer (Devlin et al., 2018) is used to jointly encode the question and the schema. By contrast, we confirm the benefit of using a grammar to constrain the inference to only produce well-formed queries. These and other findings of our work bring much-needed insight of what enables higher performance in modern text-to-SQL models.

2 Methods

Our base model, DuoRAT, is a reimplementation of RAT-SQL (Wang et al., 2020). It is an encoder-decoder model with attention and a pointer-network copy mechanism, Fig. 1. Contrary to RAT-SQL, both the encoder and the decoder are relation-aware transformers (Shaw et al., 2018). The input is modelled as a labelled directed graph, where the nodes are the input tokens and the edges are the so-called *relations*, see below.

*Equal contribution, order was determined by a quantum random number draw.

[1]Code available at `https://github.com/ElementAI/duorat`.

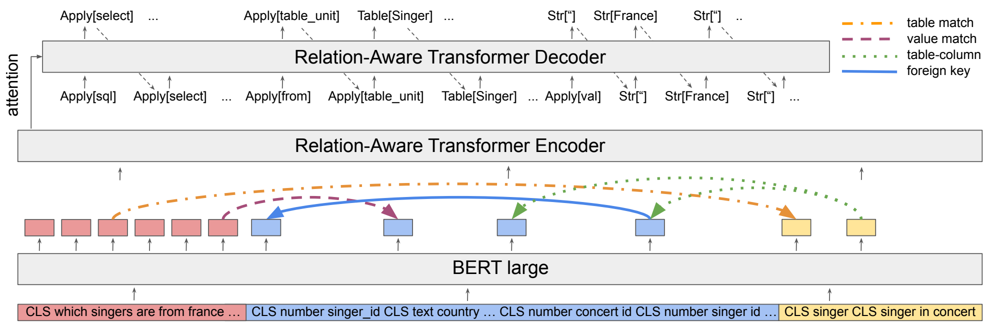

Figure 1: The DuoRAT architecture. The encoder consists of a stack of BERT layers and several relation-aware self-attention layers, while the decoder is a relation-aware transformer with encoder cross-attention. The inputs to BERT are, from left to right, the question (red), each column type and name (blue), and each table name (yellow). Column and table representations are pooled (blue and yellow boxes), the question's representation is not (red boxes).

Relation-Aware Attention Compared to vanilla self-attention, relation-aware self-attention takes two additional tensor inputs, key relations $r_{ij}^{(K)}$ and value relations $r_{ij}^{(V)}$, that amplify or diminish contributions in the scaled dot-product attention for each of the H attention heads:

$$e_{ij}^{(h)} = \frac{x_i W_Q^{(h)} \left(x_j W_K^{(h)} + r_{ij}^{(K)} \right)^\intercal}{\sqrt{d_z/H}} \tag{1}$$

$$\alpha_{ij}^{(h)} = \frac{\exp e_{ij}^{(h)}}{\sum_{k=1}^{n} \exp e_{ik}^{(h)}} \tag{2}$$

$$z_i^{(h)} = \sum_{j=1}^{n} \alpha_{ij}^{(h)} \left(x_j W_V^{(h)} + r_{ij}^{(V)} \right), \tag{3}$$

where $x_i \in \mathbb{R}^{d_x}$ is the i-th element of the input sequence, $\alpha_{ij}^{(h)}$ is the attention weight coefficient for the h-th head, and $z_i^{(h)} \in \mathbb{R}^{d_z/H}$ is the i-th output element. The indices i and j run from 1 to n, where n is the length of the sequence. $W_Q^{(h)}$, $W_K^{(h)}$, and $W_V^{(h)} \in \mathbb{R}^{d_x \times d_z/H}$ are trainable weight matrices. $r_{ij}^{(K)} \in \mathbb{R}^{d_z/H}$ and $r_{ij}^{(V)} \in \mathbb{R}^{d_z/H}$ represent a directed labelled edge pointing from the i-th input x_i to the j-th input x_j. Following Shaw et al. (2018), we set $r_{ij}^{(K)} = r_{ij}^{(V)} = r_{ij}$. The relations r_{ij} are shared across all layers. Let R be the total number of relational edge labels. If the relation $s \in \{1, \ldots, R\}$ exists between the i-th and j-th input, then we assign the s-th learned embedding $r_{ij}^{(s)}$ to r_{ij}. Otherwise, we use padding.

Encoder The DuoRAT encoder is divided into two stages, a pretrained relation-unaware transformer stage followed by a relation-aware transformer stage that is trained from scratch. The first stage is initialized with BERT weights (Devlin et al., 2018) and is fed embeddings of the question tokens, the table name tokens, the column name tokens, and one token for each column data type. We add [CLS] tokens between segments, cf. Fig. 1 for the input layout. The second stage has two inputs: an input sequence and the input relations corresponding to graph nodes and labelled edges, respectively. The input sequence is comprised of the BERT outputs for all question token positions and the [CLS] token position outputs for each table and each column. We use relational edge labels similar to those introduced by RAT-SQL. The labels are divided into three groups; (i) schema-linking relations, (ii) table-column relations, and (iii) foreign-key relations. (i) Schema-linking relations provide explicit alignment between the question and the schema. We distinguish between name-based schema linking (NBSL) and content-based schema linking (CBSL), where the former uses the names of tables and columns only and the latter uses the database content. An example for NBSL is when the question references a table by name, like "singer" in Fig. 1. CBSL identifies when the question references a value in a database column, e.g. the word "France" in Fig. 1. We use a common schema-linking heuristic where question n-grams are compared at the character level with names of tables and columns for NBSL and with the contents of column cells for CBSL. (ii) The table-column relations describe which columns belong to which tables, and which columns occur in the same table. Finally, (iii) the foreign-key relations indicate the foreign key constraints between columns. See Appendix A for a complete list of the encoder relations.

Decoder We have extended the TRANX framework for grammar-constrained sequence prediction

(Yin and Neubig, 2018) to relation-aware transformers. Like in the original framework, the decoder is restricted to generate only those sequences of grammar actions that encode a valid SQL abstract syntax tree (AST), see Appendix B. We consider two output grammars, one for complete SQL and another for SQL with underspecified `FROM` clause ($\text{SQL}^{\texttt{UF}}$) (Suhr et al., 2020). In a $\text{SQL}^{\texttt{UF}}$ query, the `FROM` clause is replaced by the `UF` clause that contains only the tables that were not mentioned in other clauses of the original SQL query. After decoding a $\text{SQL}^{\texttt{UF}}$ query, we recover the `FROM` clause by adding tables from other clauses and joining them using the foreign-key relations.

RAT-SQL and the TRANX framework use a custom parent-feeding LSTM decoder where the LSTM is fed also its own state from a previous step at which the constructor of the current action's parent AST node was generated. By contrast, in DuoRAT's relation-aware transformer decoder, we experiment with relations that are derived from the structure of the SQL program code, see Appendix A for a list. The relations can bias the transformer decoder towards attending AST parent or sibling nodes, allowing for the model to get a sense of the AST's structure. However, it is unclear whether or not this is necessary in a model with self-attention.

The decoder is coupled to the encoder via a relation-aware memory attention mechanism. Here we use relations to indicate which tokens from the input were copied to the output, that is, either question tokens, tables, or columns, depending on the type of the literal that was produced.

3 Experiments

For most of our experiments we use the Spider dataset (Yu et al., 2018b) and evaluate the predicted SQL with the *exact-match* (EM) accuracy from the official Spider evaluation script. The Spider training set contains 8,659 questions for 146 databases. The Spider development set contains 1,034 questions for 20 databases. We exclude the `baseball1` questions from the training data because the schema of this database is too large. To compare DuoRAT to the models in the literature we evaluate it on the original development set as released on January 8, 2019. In all other experiments we use the corrected development set that was released on June 7, 2020.

We also test our Spider-trained models on several earlier single-database text-to-SQL datasets, see Section 3.4 for more details on that and Appendix C for details on the training procedure.

System	EM (dev.)
RYANSQL (Choi et al., 2020)	70.6
RAT-SQL (Wang et al., 2020)	69.7
IRNet (Guo et al., 2019)	65.5
DuoRAT (original dev. set)	68.7 ± 0.7
DuoRAT (corrected dev. set)	69.9 ± 0.8

Table 1: Exact-match (EM) performance on the Spider development set. For DuoRAT we report results on the original and the corrected development set. For the other models only the original development set performance is available.

3.1 Comparing DuoRAT to Other Models

Table 1 compares DuoRAT's performance on the Spider development set to that of other state-of-the-art models[2]. DuoRAT performs similarly to its close relative RAT-SQL and outperforms other recently proposed models. Importantly, DuoRAT training takes roughly two days compared to six days for RAT-SQL. We associate the difference in speed with replacing RAT-SQL's LSTM-with-parent-feeding decoder with a transformer.

3.2 Encoder Ablations

Schema Linking Prior work (Wang et al., 2020; Guo et al., 2019) attributes a high value to schema linking, that is, to the engineering of features that ground the user utterance in the database domain. However, this insight rests entirely on experiments without BERT. We find that, for our BERT-based DuoRAT model, name-based schema linking (NBSL) can be disabled with a negligible loss in performance (see Table 2) while content-based schema linking (CBSL) can not.

The result suggests that a BERT encoder fine-tunes to perform computation that makes heuristic NBSL redundant. To gain further understanding of whether or how BERT does this, we conduct an experiment in which the inputs to BERT are divided into two logical segments: the question and the schema. We shape the attention mask such that the question segment attends to the schema or the schema attends to the question, or both, or neither. The results are shown in Table 2. We observe that, for the best performance, BERT should

[2]Results taken from the Spider leaderboard at `https://yale-lily.github.io/spider` on October 19, 2020.

Model Variant			Exact Match (dev.)	
CBSL	Q→S	S→Q	w NBSL	w/o NBSL
✓	✓	✓	69.9 ± 0.8	68.6 ± 1.0
	✓	✓	67.5 ± 0.6	66.7 ± 1.0
✓	✓		64.3 ± 0.4	63.2 ± 0.7
✓		✓	64.4 ± 0.9	64.5 ± 0.8
✓			63.9 ± 0.8	59.1 ± 1.1

Table 2: Results with and without name-based (NBSL) or content-based schema-linking (CBSL) for various attention masks. Q→S: the question can attend to the schema. S→Q: the schema can attend to the question.

Model Variant	EM (dev.)
DuoRAT	69.9 ± 0.8
w/o AST relations	69.7 ± 0.9
w/o copied-from relations	69.2 ± 1.0
w/o any decoder relations	69.4 ± 1.1
w/o constraining during training	69.0 ± 1.1*
w/o any constraining	63.2 ± 0.9*

Table 3: Decoder ablations. Middle: decoder relations. Bottom: grammar constraining at training or inference. *In each case one out of five jobs diverged during training and was removed from the ensemble.

Model Variant	EM (dev.)
`[column][table]` + relations	69.6 ± 0.8
`[column][table]`	60.0 ± 0.7
`[table[column]]`	65.8 ± 1.3

Table 4: Encoding of schema structure. Ordering of schema elements can be: (i) `[column][table]`: first all the column types and names, then all the table names. (ii) `[table[column]]`: each table is followed by the columns of that table.

be jointly embedding the question and the schema. We can neither embed the question separately from the schema nor the schema separately from the question without substantial performance losses. Interestingly, once we cut all the attention connections between the question and the schema, explicit NBSL becomes essential. This confirms our hypothesis that joint BERT-based encoding of the question and the schema is the cause of the low importance of NBSL in our model.

Schema Structure Representation Various ways of encoding which columns belong to which table have been explored: Suhr et al. (2020) orders the schema elements such that each table name is followed by the names of the columns of that table, RAT-SQL (Wang et al., 2020) represents schema structure as table-column relations in the RAT encoder. Our experiments show that encoding the schema structure via encoder relations gives the best performance (first row of Table 4), and encoding it in the order of its elements (third row) is better than not encoding it at all (second row). Additional results can be found in Appendix D.

3.3 Decoder Ablations

Decoder Relations And Grammar-Based Constraining Table 3 shows results of ablation studies in which (i) different kinds of decoder relations were removed, in particular those that provide program structure information, and (ii) grammar-based constraining was deactivated during training and/or inference. The experiments provide the following insights: (i) A vanilla transformer decoder can be used without loss of performance. The relations that provide information about the AST and about which literals were copied from the input are not useful. We speculate that AST information can be more useful in deeply nested SQL expressions of which Spider contains only few. (ii) By contrast, grammar constraining at inference leads to significant performance improvements. Notably, training tends to be less stable when not using grammar constraints. The usefulness of grammar constraining can be explained by the fact that it reduces the output space and makes the decoder more data-efficient.

Output Format In this section, we examine the performance on the Spider dataset when outputting complete SQL and when outputting simplified SQL with underspecified `FROM` clause, SQL$^{\texttt{UF}}$. The results are reported in Table 6. Our first insight is that DuoRAT performance with and without SQL$^{\texttt{UF}}$ is almost the same. Our second insight is that when the encoder does not have access to information about the foreign keys, SQL$^{\texttt{UF}}$ brings a significant improvement. The best result is still achieved with a model that uses the foreign-key input relations.

3.4 Testing on Single-Database Datasets

A known issue of the Spider dataset is that the question wordings are unnaturally close to the respective queries (Suhr et al., 2020). To complement our studies on Spider, we perform additional experiments on single-database text-to-SQL datasets that are devoid of this issue. Suhr et al. (2020) propose a demanding cross-domain generalization evaluation whereby models are trained on Spider and

Dataset	DuoRAT + SQL$^{\mathrm{UF}}$	w/o CBSL	w/o SQL$^{\mathrm{UF}}$
GeoQuery	54.6 ± 3.9	49.2 ± 4.2	48.4 ± 4.9
Academic	31.4 ± 4.9	18.0 ± 3.9	10.3 ± 1.2
IMDB	38.5 ± 8.1	39.2 ± 5.6	20.0 ± 5.0
Yelp	33.5 ± 3.8	25.7 ± 4.7	27.8 ± 4.3

Table 5: Ablation results with the underspecified FROM (SQL$^{\mathrm{UF}}$) approach on single-database text-to-SQL datasets. CBSL stands for content-based schema linking. Pink shading indicates statistically significant gap between the ablated model and the complete one.

Model Variant	Exact Match (dev.)
DuoRAT	69.9 ± 0.8
w/o foreign-key inputs	67.6 ± 0.7
+ SQL$^{\mathrm{UF}}$, w/o foreign-key inputs	69.0 ± 0.8
+ SQL$^{\mathrm{UF}}$	70.5 ± 1.2

Table 6: Results using SQL$^{\mathrm{UF}}$ and/or foreign-key inputs.

tested on single-database datasets by comparing the execution results of the predicted and the gold queries. We follow this methodology and filter the datasets to only use those question-query pairs for which execution accuracy evaluation is appropriate (see Suhr et al. (2020) for filtering details). To focus on evaluating the query structure, we replace predicted string literals with the most similar ones from the gold query (details of this procedure can be found in Appendix E).

Of the 8 datasets that Suhr et al. (2020) consider we exclude ATIS, Advising, and Scholar for being too different from Spider and Restaurants for having just 27 examples after filtering. What remains are the SQL version of the GeoQuery dataset (Zelle and Mooney, 1996) as well as the small Academic, IMDB, and Yelp datasets (Finegan-Dollak et al., 2018). After filtering, these datasets are left with 532, 180, 107 and 54 examples, respectively. Note that these single-database datasets are partially contained in Spider. To avoid testing on training data, we train new models only on the about 7,000 examples produced by the Spider data collection effort.

In this round of analysis we focus on the impact of CBSL and the underspecified FROM clause (SQL$^{\mathrm{UF}}$) technique (Suhr et al., 2020). We expect both methods to be especially useful for out-of-distribution generalization, despite the limited importance that Spider evaluation attributes to them. The results in Table 5 show that, in line with our intuition, CBSL and SQL$^{\mathrm{UF}}$ bring performance gains on 2 and 3 out of 4 datasets, respectively.

Dataset	DuoRAT + SQL$^{\mathrm{UF}}$	Suhr et al. (2020)
GeoQuery	54.59 ± 3.91	41.6
Academic	23.28 ± 2.93	8.2
IMDB	34.95 ± 5.66	24.6
Yelp	30.19 ± 3.27	19.8

Table 7: DuoRAT performance on single-database text-to-SQL datasets. The model is trained to predict SQL$^{\mathrm{UF}}$, i.e. SQL with underspecified FROM clause.

To enable comparison with results by Suhr et al. (2020), we also report the results without literal replacement in Table 7. DuoRAT performs consistently better than the model by Suhr et al. (2020).

4 Conclusion

Our investigations have revealed several possible simplifications of relation-aware text-to-SQL transformer models. In particular, we have shown that a transformer decoder with vanilla self- and memory-attention is sufficient, and that heuristic schema linking based on table and/or column names brings only a marginal benefit. To the contrary, we confirm the importance of grammar-constrained decoding, relational schema representations, content-based schema linking. Looking forward, we believe that content-based schema-linking will remain important, while the impact of name-based schema linking will further decrease as the language models get bigger and absorb more data. This prediction is based on the fact that the mapping from an entity to the entity type that name-based schema linking effectively performs can be highly domain- and schema-specific. Last but not least, we have shown that predicting a more compact SQL version with an underspecified FROM clause improves the model's out-of-distribution performance, despite bearing little influence on the model's performance on Spider.

In future work, we will combine the successful simplifications from this paper to build the simplest yet high-performing text-to-SQL model. One promising direction for further simplification is to use a pretrained encoder-decoder pair as proposed in Raffel et al. (2020) and Lewis et al. (2019).

References

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing. *arXiv:1905.06241 [cs]*. ArXiv: 1905.06241.

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:2004.03125 [cs]*. ArXiv: 2004.03125.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. 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, NAACL 2019*.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving Text-to-SQL Evaluation Methodology. *arXiv:1806.09029 [cs]*. ArXiv: 1806.09029.

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

Amol Kelkar, Rohan Relan, Vaishali Bhardwaj, Saurabh Vaichal, and Peter Relan. 2020. Bertrand-DR: Improving Text-to-SQL using a Discriminative Re-ranker. *arXiv:2002.00557 [cs, stat]*. ArXiv: 2002.00557.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proceedings of the 2015 International Conference on Learning Representations, ICLR 2015*. ArXiv: 1412.6980.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. BART: Denoising Sequence-to-Sequence Pretraining for Natural Language Generation, Translation, and Comprehension. *arXiv e-prints*, page arXiv:1910.13461.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21(140):1–67.

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, New Orleans, Louisiana. 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 *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8372–8388, Online. Association for Computational Linguistics.

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

Daniel C. Wang, Andrew W. Appel, Jeff L. Korn, and Christopher S. Serra. 1997. The zephyr abstract syntax description language. In *Proceedings of the Conference on Domain-Specific Languages on Conference on Domain-Specific Languages (DSL), 1997*, DSL'97, page 17, USA. USENIX Association.

Pengcheng Yin and Graham Neubig. 2018. TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation. *arXiv:1810.02720 [cs]*. ArXiv: 1810.02720.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. TypeSQL: Knowledge-based Type-Aware Neural Text-to-SQL Generation. *arXiv:1804.09769 [cs]*. ArXiv: 1804.09769.

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. *arXiv:1809.08887 [cs]*. ArXiv: 1809.08887.

John M. Zelle and Raymond J. Mooney. 1996. Learning to Parse Database Queries Using Inductive Logic Programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, AAAI'96, pages 1050–1055. AAAI Press. Event-place: Portland, Oregon.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic Evaluation for Text-to-SQL with Distilled Test Suites. *arXiv:2010.02840 [cs]*. ArXiv: 2010.02840.

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

A Table of Relations

Table 8 lists all relations in the DuoRAT encoder while Table 9 lists all relations used in the decoder.

B Grammar-Constrained Sequence Prediction

The SQL grammar is defined in the Zephyr Abstract Syntax Description Language (ASDL) (Wang et al., 1997). Given the grammar, the ground-truth SQL query is parsed to an abstract syntax tree (AST). This tree is then serialized into a sequence of production actions, choosing the depth-first, left-to-right order as reference. The decoder is trained to either generate actions from a vocabulary of actions or to copy literals from the question or the schema. During inference, the predicted action sequence is deserialized back to an AST, from which the final SQL query string is assembled. Actions that are invalid at any decoding step are masked such that the decoder is constrained to only produce grammatically valid SQL.

Each input element to the DuoRAT decoder consists of the concatenation of an action embedding, a field embedding, and a field type embedding which are derived from the ASDL grammar.

C Training Procedure

The BERT stage of the encoder is initialized with BERT-large weights, whereas the RAT stage of the encoder is initialized randomly. BERT's input sequence length is limited to 512 tokens. Inputs that exceed this limit are truncated. We use 8 RAT layers with each $H^{(\text{enc})} = 8$ heads, an embedding dimension of $d_x^{(\text{enc})} = d_z^{(\text{enc})} = 1024$, a feed-forward network with 1024 dimensions, and a dropout probability of 0.1.

In the decoder, we use 2 randomly initialized RAT layers with $H^{(\text{dec})} = 8$ heads each, embedding dimensions for actions, fields, and field types of 64 each for a total of $d_x^{(\text{dec})} = d_z^{(\text{dec})} = 192$, a feed-forward network with 256 dimensions, and a dropout probability of 0.1. The memory pointer has a projection size of 50, and the minimum required number of times a literal action token must occur in the input of the training set to be added to the vocabulary is 5.

We train using Adam (Kingma and Ba, 2015) with default parameters for 100,000 steps in PyTorch's automatic mixed precision mode (AMP). We use a step-wise linear learning-rate schedule with a warm-up period from 0 up to 0.0001 in 2,000 steps followed by a 98,000 steps long cool-down period from 0.0001 down to 0. All model weights are trainable. The learning rate for BERT is always 8 times lower than for the rest of the model. Each training batch has 9 items, we use gradient accumulation to bring the batch size up to effectively 27. We report results obtained with the beam size of 1. Training and evaluating DuoRAT on a single V100 with 32 GB HBM2 takes 2 days. For each training run, we log the performance of the model checkpoint with the best validation accuracy during training measured in intervals of 5,000 steps. In the tables, we report the peak performance averaged over 5 training runs.

D Additional Encoder Ablations

This section reports some additional results regarding the encoding of the schema structure from Section 3.2. When encoding the schema structure through the order of its elements, Suhr et al. (2020) proposed to shuffle the order of the tables and columns to regularize training. We test this regularization and report the results in Table 10. This brings a slight improvement (fourth row of Table 10), but still gives results below a model that encodes this structure through encoder relations. Additional experiments not reported in Table 10 showed that using different ordering and shuffling did not bring an advantage when using the table-column relations.

We have also experimented with removing the foreign key relations from the model's input. The results reported in Table 6 confirm that conditioning the model on these relations is indeed necessary for the best performance.

E Details on Literal Substitution For Experiments on Single-Database Datasets

When following the protocol of Suhr et al. (2020), we observed that the model often made errors when copying literals, especially long ones. Often the literal copying mistake was the only one in an otherwise correctly predicted query, see Table 11 for an example. This issue limited our ability to rigorously assess

Relative Positioning Relations	
Q→Q Distance d	Question token positions are separated by the distance d
C→C Distance d	Column positions are separated by the distance d
T→T Distance d	Table positions are separated by the distance d
Table And Column Relations	
C→C Table Match	Columns are in the same table
C→T Any Table	Column is wildcard (*) in table
C→T Table Match	Column is part of table
T→C Any Table	Table contains wildcard (*) column
T→C Table Match	Table contains column
Primary-Key And Foreign-Key Relations	
C→C Foreign-Key Forward	Column is foreign key to other column
C→C Foreign-Key Backward	Other column is foreign key to this column
C→T Foreign-Key	Column is foreign key of table
C→T Primary-Key	Column is primary key of table
T→C Foreign-Key	Table has column as foreign key
T→C Primary-Key	Table has column as primary key
T→T Foreign-Key Forward	Table has foreign key to other table
T→T Foreign-Key Backward	Other table has foreign key to this table
T→T Foreign-Key Bidirectional	Tables have foreign keys in both directions
Name-Based Schema Linking Relations	
Q→C Name-based Match c	Question token matches column name with confidence c
Q→T Name-based Match c	Question token matches table name with confidence c
C→Q Name-based Match c	Column name matches question token with confidence c
T→Q Name-based Match c	Table name matches question token with confidence c
Content-Based Schema Linking Relations	
Q→C Content-based match c	Question token matches column cell content with confidence c
C→Q Content-based match c	Column cell content matches question token with confidence c
Padding Relations	
Q→Q Default	No Q→Q relation exists
Q→C Default	No Q→C relation exists
Q→T Default	No Q→T relation exists
C→Q Default	No C→Q relation exists
C→C Default	No C→C relation exists
C→T Default	No C→T relation exists
T→Q Default	No T→Q relation exists
T→C Default	No T→C relation exists
T→T Default	No T→T relation exists
Default	At least one token position is not populated

Table 8: List of encoder relations and their purposes. A Q→Q relation points from one question token positions to another, a Q→C relation points from a question token position to a column, and so on. The distances d count from $-D$ to D in increments of 1, where the horizon D is configurable. The confidences c are binary and either `high` or `low`.

Self-Attention	
Parent-Child	Node is parent of a node in the decoded AST
Child-Parent	Node is child of a node in the decoded AST
Identity	Self-loop, connects a node in the decoded AST to itself
Sibling-Distance d	Relative distance d of nodes that have the same parent
Default	Padding, at least one token position is not populated
Memory Attention	
Copied-From	Input tokens that were copied
Default	Padding, at least one token position is not populated

Table 9: List of decoder relations and their purposes.

Model Variant	EM (dev.)
`[column][table]` + relations	69.6 ± 0.8
`[column][table]`	60.0 ± 0.7
`[table[column]]`	65.8 ± 1.3
`[table[column]]` + shuffling	66.5 ± 1.3

Table 10: Complimentary results on the encoding of schema structure. Ordering of schema elements can be: (i) `[column][table]`: first all the column types and names, then all the table names. (ii) `[table[column]]`: each table is followed by the columns of that table.

Question:	return me the abstract of "Making database systems usable" .
Predicted:	`SELECT publication.abstract FROM publication` `WHERE publication.title = "Making Systems Usable"`
Ground truth:	`SELECT publication.abstract FROM publication` `WHERE publication.title = "Making Database Systems Usable"`

Table 11: An example from the Academic dataset on which DuoRAT makes a literal copying error.

performance difference between different DuoRAT versions. To be able to evaluate improvements in predicting query structure, we modified their experimental protocol by replacing predicted literals with most similar ground-truth ones (we used the Levenshtein distance to assess similarity). Note that ignoring literals during evaluation is a standard practice in text-to-SQL literature (see e.g. exact match metric for Spider (Yu et al., 2018b), logical form accuracy for WikiSQL (Zhong et al., 2017) or the recently proposed test-suite accuracy by Zhong et al. (2020)). In future work, the literal copying issue can be addressed by generating copies of Spider queries with longer literals and adding them to the training set.