Texila International Journal of Academic Research
ISSN: 2520-3088
DOI: 10.21522/TIJAR.2014.10.02.Art001

Light RAT-SQL: A RAT-SQL with More Abstraction and Less Embedding of Pre-existing Relations

Nathan Manzambi Ndongala
Ph.D, Department of Computer Science, Texila American University, Guyana

Abstract

RAT-SQL is among the popular framework used in the Text-To-SQL challenges for jointly encoding the database relations and questions in a way to improve the semantic parser. In this work, we propose a light version of the RAT-SQL where we dramatically reduced the number of the preexisting relations from 55 to 7 (Light RAT-SQL-7) while preserving the same parsing accuracy. To ensure the effectiveness of our approach, we trained a Light RAT-SQL-2, (with 2 embeddings) to show that there is a statistically significant difference between RAT-SQL and Light RAT-SQL-2 while Light RAT-SQL-7 can compete with RAT-SQL.

Keywords: *Deep learning, Natural Language Processing, Neural Semantic Parsing, Relation Aware Transformer, RAT-SQL, Text-To-SQL, Transformer.*

Introduction

The RAT-SQL [1] has been used in Text-to-SQL [2-4] as an encoder transformer. The RAT-SQL framework jointly encodes the question and the schema database to improve the generalization even in unseen databases by the model during the training process. RAT-SQL is based on the relation-aware self-attention mechanism, and address schema encoding and schema linking within a text-to-SQL encoder.

The core of RAT-SQL is the abstract pre-existing relation between input tokens. The RAT-SQL model implementation has been trained with more than 50 embedding relation types.

The management of relations in the Relation Attention Transformer is challenging: Having more relations can lead the model to capture noise and having fewer relations, the model can miss another important relation trend in data.

Previous methods, [2] empirically noticed when injecting syntactic dependency in the graph of RAT-SQL, that having many relations can lead to overfitting. Another insight about pre-existing relations is when a pre-trained language model as BERT [5, 6] is used to enhance RAT-SQL the name-based schema linking (NBSL) become marginal [3] but neither method explicitly assess the acceptable threshold of the number of relations to take into account in a Transformer with pre-existing relations.

In this work, we attempt to respond to the following questions:

1. How can we reduce the number of "pre-existing relations" while preserving high-quality parsing?
2. To what extent does exact match accuracy depend on pre-existing abstract relations in Relation-Aware Transformer / RAT-SQL?

First, we constrain the number of relations to be equal to the number of heads of the model. Each head tends to specialize [7], and we hypothesize then each relation will be learned by one head of the model.

Secondly, we present a new structure of a relation graph inspired by RAT-SQL [1], SS^2SQL [2], and database theory [8].

Received: 07-02-2023 ***Accepted: 20-02-2023*** ***Published on: 28.04.2023***
Corresponding Author: nathanmanzambi@yahoo.fr

Our findings are as follows, without any enhancement of the pre-trained Language model:

1. Pre-existing relations in RAT-SQL are important and can lift the model accuracy and improve alignment between the Question and the Schema.
2. Constrain the number of relation types to be equal to the number of heads allows having more abstraction in relation types.
3. It is better to have more abstraction in relation types but too abstract relation types can lead to an underfit model.

Related Work

Text-To-SQL

Most Text-to-SQL [9-11] models are built around an encoder-decoder pattern. The challenge is to design a system that can predict the SQL query from a question and the details of the schema. The relational database schema is made up of tables linked together by columns. This semantic parsing prediction task is difficult due to the tree structure of the SQL programming language.

Early work on semantic parsing with an encoder-decoder model focused on designing a decoder capable of constraining the output space to the outputs that matter [12]. The goal is to avoid invalid parsers because most Seq2seq models performed poorly in semantic parsing tasks. Two families have emerged to build such decoders, namely token-based decoding [13-15], and grammar-based decoding [16-19]. Token-based decoding is the Seq2Seq model where the output space is made up of tokens, but they are constrained to be relevant at every time step. And grammar-based decoding, the output space is the production rules, and the grammar defines the constraints.

Apart from these two families, [20, 21] use Transformers [22, 23] with tree positional encodings to enable tree-based decoders and get a good result. [24] use an intermediate representation SemQL query which bridges Natural Language and SQL.

The works on the Encoder side [1, 2, 24-29] focus on jointly encoding the Question and the Database Schema. With the emergence of the language model [5, 6, 30-40], the trend is to improve encoding with a language model. PICARD [41] effectively constrained decoding with large pre-trained language models, T5 [42].

Another way to approach Text-To-SQL is to design a pre-trained language model from tabular data [4, 43-46].

RAT-SQL

Text-to-SQL encoder should be able to encode the database relations in a way for optimizing the semantic parser and model alignment between the database schema and the given query. For unseen database schemas, the way the inputs (schema: tables & columns, and question) are jointly encoded is crucial for generalization purposes. The RAT-SQL framework [1] designed from the relation-aware self-attention mechanism [47], used as pointer networks [48], fixes this generalization issue on the Text-to-SQL challenge.

The relation-aware self-attention [47] uses the relative position distance in input tokens as edges of the graph while RAT-SQL uses an embedding of abstract type of existing relations between tokens. But both biases the self-attention equation in the same way to inject the edges of the graph in their model.

Here below equations of relation-aware self-attention:

$$e_{i,j}^{h} = \left(\frac{\left(W^{q} X \left(W^{k} X + R \right)^{T} \right)}{\sqrt{d_k}} \right) (1)$$

$$\alpha_{i,j}^{h} = softmax\left(e_{i,j}^{h} \right) (2)$$

$$z_{i}^{h} = \sum_{v} \alpha_{i,j}^{h} \left(W^{v} X + R \right) (3)$$

where matrices W^{q}, W^{k}, *and* W^{v} are trainable parameters in self-attention.

Relations Structure

The self-attention as presented in Transformer [23] is a directed graph, it is not

symmetric. During the learning process, the self-attention blocks learn the optimal attention weight that represents the relation existing between tokens. This is more efficient than recurrent models [49-51] since each part of the inputs has a direct relation with other parts while the recurrent models encode left to right and right to left dependency and struggle with long sequences.

Relations between tokens can be diverse depending on semantic meaning. In the relational graph attention transformers, the pre-existing relations have to enhance self-attention. That is why the structure of pre-existing relations becomes crucial to avoid the bad effect on self-attention. [28] tackled the schema representation challenge by encoding the directed graph of foreign key relations in the schema with a graph neural network (GNN). In RAT-SQL instead, the authors use the relation-aware self-attention as a model, and define the 3 main structures: Schema Structure, Schema Linking and Question Structure.

S^2SQL [2] injects the syntactic dependency information of questions into RAT-SQL, but it does not use relative distance in question tokens, instead, it uses the first-order distance in dependency syntactic between tokens. And adds orthogonality constraints on relations to avoid coupling between edges.

Method: Light RAT-SQL

We present now Light RAT-SQL, our model to reduce the pre-exissting embedding in transformer.

Text-to-SQL Problem Definition

Given a natural language question $\mathbf{Q=q1\ldots q_{|Q|}}$, a schema $\mathbf{S = <C, T>}$ with columns $\mathbf{C=\{c1,\ldots, c_{|C|}\}}$ and tables $\mathbf{T\{ t1,\ldots, t_{|T|}\}}$.

The goal of text-to-SQL is to create the SQL query $\boldsymbol{y}$ for the question sentence. The pattern used in such a challenge is an encoder and decoder architecture with attention mechanisms.

The encoder encodes input as graph $\mathbf{G=<V, R>}$ where $\mathbf{V=Q\ U\ T\ U\ C}$ are nodes of types { **Q, T, C** } The initial embedding matrix X $\mathbf{R^{Vxd}}$, [$\mathbf{V = Q+T+C,}$ and **d** is the dimension model] is flattened and the edge $\boldsymbol{R}$ is the know relation between two input tokens.

In this paper, our proposed model improves the encoder side, especially, the existing RAT-SQL model [1] and self-attention with a relative position representation framework [47].

Please see [16] work for a thorough description of the decoder side.

The Number of Abstract Type Relation

A great number of pre-existing relational features between the inputs can lead to overfitting the model. [2] experienced the same when injecting the syntax dependency in RAT-SQL. To fix this problem, in their model S^2SQL, they used the first-order distance in dependency syntactic between tokens and then reduce the number of relation types [for the relation Question graph] to 3: Forward, Backward, and None.

[7] shows that each head in multi-head attention tends to specialize. They found 3 categories: Positional heads, Syntactic heads, and heads that point to rare words.

1. Positional heads that attend largely to their neighbor.
2. Syntactic heads are tokens that have a specific syntactic relation.
3. Headings that point to uncommon words in the sentence.

Pruning the others is the finest technique to demonstrate the relevance of their head category. They retained 17 out of 48 heads with nearly the same BLEU score by largely keeping the heads that are classified in the differentiated categories, as illustrated. And this corresponds to around 2/3 of the encoder's heads. Even though they only identified 3 types of essential heads by looking at their attention matrices and pruning the others, we hypothesize that in the self-attention mechanism with pre-existing relation embedding, each head will be specialized in each abstract type during the learning process. We already know one of these embeddings is the

abstract type *"None"* that express that there is no link between two tokens.

In other words, the number of heads (#head) in the model gives us the degree of freedom (DOF) to define the number of pre-existing relations (or abstract type relations) that we can use in the model. We define it as follows:

$$\boldsymbol{DOF = \#head - 1}\ (\mathbf{4}).$$

How to Constrain each Head to Learn at Most one Embedding Relation?

Algorithm 1 Spreading relations through heads:

1. **Input**: relation, head, embedding.
2. *# relation shape: (query, key, features).*
3. *# head : number of head in the model.*
4. *#embedding: lookup table that stores embeddings of a fixed dictionary and size.*
5. *#The first indice in our fixed dictionary is 0 and it is linked to "None" relation.*
6. **Output**: r_out # relation output.
7. *#r_out shape: (head, query, key, features.)*
8. Init of r_out #*zeros like (head, query, key, features).*
9. r_out[0]= relation.
10. **For** i=1 to head-1.
11. mask ← (relation==embedding(i)).
12. r_out[i] ← mask relation.
13. **return** r_out.

Where ⊙ is Hadamard product and # a comment line.

Algorithm 1 helps propagate the embedding of pre-existing relations through the heads. This makes each "head" specialized, thus pointing to tokens with a specific relationship type.

Graph Structure

Since we will have 8 heads as in RAT-SQL, so we will describe DOF=7 abstract relation types to handle the text-to-SQL challenge.

Question Structure $\boldsymbol{R^{question}}$: Relations between two question tokens that show their grammatical dependency or their distance (or relative positions).

As [2] we model the syntactic relation between two tokens (vi, vj) by abstract types.

$$\boldsymbol{Rijquestion} = \begin{Bmatrix} Forward, if\ D(vi,vj) = 1 \\ Backward, if\ D(vj,vi) = 1 \\ None, otherwise \end{Bmatrix} (5).$$

More details about first-order distance in dependency syntactic can be found here [2].

Schema Structure $\boldsymbol{R^{schema}}$: Relations within a database schema. Most relational databases in production are normalized in Boyce–Codd's normal form [8]. Because of this normalization, we know that for any X, Y, and Z, sets of attributes in a relation from a database:

1. All attributes in each relation of the database are atomic (First Normal Form or 1NF).
2. If X → Y i.e., Y has a functional dependency on X means that X is a primary key and X and Y are in the same relation (or table) and Y depends only on X (2NF).
3. The transitivity functional dependency (X → Z and Z → Y then X → Y) is only allowed between 2 relations (or tables) and not in the same relation (or table) (3NF).
4. If all dependencies of the relationships depend on X and X is a key or a super key then R is in Boyce–Codd normal form (BCNF).

We can define all structures of the database using functional dependency. The graph of functional dependency is the best tool to describe and understand all database structures.

In our model, if there is a functional dependency or backward functional dependency between 2 columns, so edge will be defined by the abstract type **"FD"**. The abstract type describing the relationship between the column and the table will be **"TC"**. And relations between 2 tables will be described by **"TT".**

Linking Structure $\boldsymbol{R^{linking}}$: For the linking between the database schema and the question, we follow RAT-SQL [1]. The difference between Light RAT-SQL and RAT-SQL in schema linking structure comes in the level of abstraction. In RAT-SQL the schema linking is more detailed: exact name matching, partial matching, and value base linking. The implementation of RAT-SQL has more than 10 relations for Schema Linking.

1. CEM=Column Exact Matching (Backward and Forward).
2. TEM=Table Exact Matching (Backward and Forward).
3. CPM= Column Partial Matching (Backward and Forward).
4. TPM= Table Partial Matching (Backward and Forward).
5. Value-Based Number (Backward and Forward).
6. Value-Based Time (Backward and Forward)
7. Value-Based Cell Match (Backward and Forward).

We propose schema linking more abstract than RAT-SQL and consists of 2 abstract types:

1. NBL (Name Base Linking): Linking between a Question and a Table or Column (forward and backward with the same nomenclature)
2. VBL (Value Base Linking): Linking between a Question and a Column(forward and backward with the same nomenclature) references any values found in the Question and SQL query.

Here below the "None" and all 7 relation types:

Table 1. The List of All Relations Structures used in Light RAT-SQL

Abstract Type	**Description**	**X**	**Y**
None	**X** and **Y** have no relation	Q/T/C	Q/T/C
TT	Table X and Table **Y** are linked by a foreign key **[Forward and Backward]**	T	T
TC	Column **Y** belongs to table **X** or Column **X** belongs to table **Y**	T/C	C/T
FD	There is a functional dependency between Column **X** and Column **Y or between Y and X.**	C	C
Forward	**Y** is the target word of **X** under syntax dependency	Q	Q
Backward	**Y** is the source word of **X** under syntax dependency.	Q	Q
NBL	Name-based linking refers to exact or partial occurrences of either the table name or Column name **Y** in the question **X\ [Forward and Backward]**	Q/T or C	T or C/Q
VBL	Value-based-Linking: question **X** references any values found in the database and so participates in the SQL query **[Forward and Backward]**	Q/C	C/Q

*Legend: **X**= Source token | **Y**=Target token | **T** = TABLE | **C**=Column | **Q**=Question*

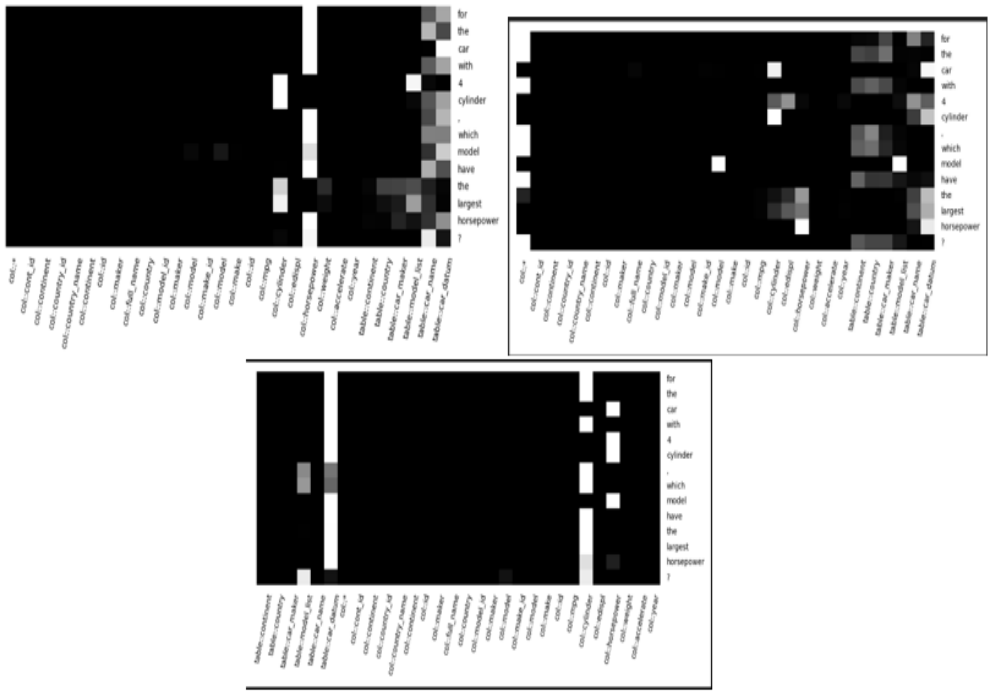
Experiments

Dataset

We use the Spider dataset [52]. Spider is a text-to-SQL benchmark that is vast, sophisticated, and cross-domain. We followed the formal evaluation process to report exact match accuracy.

Implementation

We use the same architecture as in the RAT-SQL with the Spider dataset, which means that the input of questions, column names, and Table names is tokenized and lemmatized using the StanfordNLP toolkit [53]. We use GloVe [54] word embeddings within the encoder, which are fixed in training except for the 50 most common terms in the training set. The bidirectional LSTMs [49] have a hidden size of 128 per direction and employ a recurrent dropout [55] approach with a rate of 0:2. On top of the bidirectional LSTMs, we build 8 relation-aware self-attention layers. To have the same configuration as RAT-SQL, we set **dx = dz = 256, H = 8**, and employ.



Left: The RAT-SQL - Right: Light RAT-SQL and bottom: Light RAT-SQL-2

Figure 1. Alignment between the Question "For the Cars with 4 Cylinders, which Model has the Largest Horsepower" and the Database Car_1 Schema (Columns and Tables)

Dropout with a rate of 0:1. The inner layer dimension of the position-wise feed-forward network is 1024. We use rule embeddings of size 128, node type embeddings of size 64, and a hidden size of 512 inside the LSTM [51] with a dropout of 0:21 inside the decoder.

Hyperparameter

Light-RAT-SQL reuses the same hyperparameters of RAT-SQL.

The code was implemented in Pytorch [56] and Adam [57] as an optimizer with default hyperparameters. We use a batch size of 20 and train for up to 40,000 steps.

As a syntactic parser, we use the SpaCy (https://spacy.io/) tool to create syntactic information as previously done by [2].

Results

For experimentation purposes, we design another model with 2 relation types and we call it Light RAT-SQL-2. When 2 tokens have a pre-existing relation (Syntactic dependency, Name base Linking, Value-Based Linking, Schema Linking), the relation name is "Linked" and "None" else. We call then our model either Light RAT-SQL or Light RAT-SQL-7.

Table 2. Accuracy of the Spider Dataset [52]

Model (Without any enhancement of one Pretrained Language Model: BERT, ELECTRA, etc.)	**Dev**	**Test**
IRNet [24]	53.2	46.7
Global-GNN [28]	52.7	47.4
IRNet V2 [24]	55.4	48.5
RAT-SQL [1]	62.7	57.2
Our model (Light RAT-SQL)	-	60.25

We run 5 random experimentations of RAT-SQL, Light RAT-SQL-7, and Light RAT-SQL-2 without any pre-trained Language to measure the change done in each model relation structure. Table 3 shows the confidence interval of each model, and we fail to reject H_0 for RAT-SQL and Light RAT-SQL-7 but there is strong evidence against H_0 ($p<0.001$) for RAT-SQL and Light RAT-SQL-2.

In addition, apply the *algorithm 1* spreading relations through heads improve the model's accuracy.

Table 3. Test Set Accuracy (and 95% Confidence Interval) of RAT-SQL, Light RAT-SQL, Light RAT-SQL without Applying "Spreading Algorithm" and Light RAT-SQL-2

Model	Accuracy
Light RAT-SQL	58.99 ± 1.04
RAT-SQL	58.90 ± 0.50
LRS w/o Spreading Algo	58.32 ± 0.80
Light RAT-SQL-2	53.37 ± 1.30

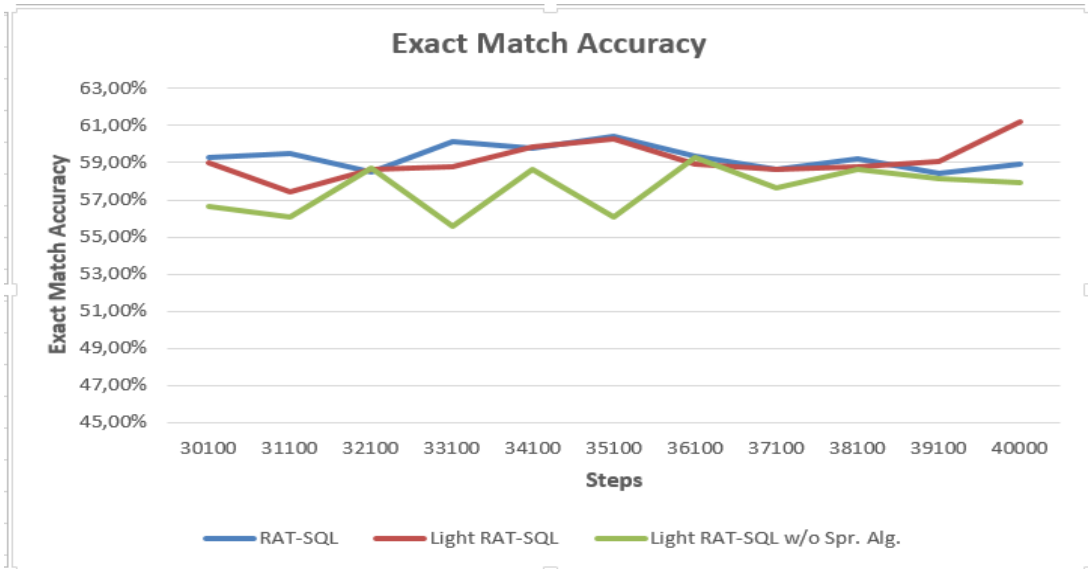


Figure 2. Exact Match Accuracy of RAT-SQL, Light RAT-SQL and Light RAT-SQL Without Speading Relations through Heads

We can see in **Figure 2.** that Light RAT-SQL-7 and RAT-SQL converge to almost the same accuracy.

Discussion

Alignment

The alignment produced by RAT-SQL, Light RAT-SQL-7, and Light RAT-SQL-2 is displayed in Figure 1. The three words that refer to columns (cylinders, model, and horsepower) are misaligned by Light RAT-SQL-2, but Light RAT-SQL-7 correctly identifies the appropriate columns. These three keywords have a significant impact on the alignments of subsequent words, leading in a sparse span-to-column alignment, for example, "biggest horsepower" to horsepower. The word "cars" contains an implicit reference to the table's car data and cars' names. Using the fact that these two tables have the three mentioned fields in Light RAT-SQL-7 and not in Light RAT-SQL-2, the alignment matrix successfully infers that they should be used instead of car makers.

Need to Spread Relations through Heads

Making each head of the model specialize in at least one embedding allows the relations data to contain distinct information per head, which is complementary to each other. Because

duplicate information may make the model difficult to train.

Figure 2 shows our ablation study where we trained the same model without spreading relations through heads. The result shows how Light RAT-SQL performs better with specialized heads.

Conclusion

This paper presents a Light version of RAT-SQL, a Light RAT-SQL. This latter is designed with only 7 relation types (since we consider "*None*" as trivial relation type) and has almost the same performance as the original RAT-SQL, which has 55 embedding relation types.

Apart from the ablation study performed by RAT-SQL, this study confirms the importance of the pre-existing relations in RAT-SQL. Light RAT-SQL-2 has a poor performance compared to RAT-SQL and Light RAT-SQL. The pre-existing relation can lift the model accuracy and improve alignment between the Question and the Schema. We show empirically the good way to reduce the number of relation types is to constrain this compared with the number of heads in the model. This makes the pre-existing relations more abstract during the design of the model. However, the relations of the Light RAT-SQL-2 (Linked and None) are more and more abstract and perform poorly. It is better to have more abstraction in relation types and have the number of relations close to the number of heads of the model. In future work, we will use the same relation structure and improve the relation-aware self-attention to optimally fit the pre-existing relations.

Limitations

Light RAT-SQL was not enhanced with a pre-trained language model such as BERT or ELECTRA since this requires large GPU resources. Investigation into enhancing our proposed model with Electra can be crucial to determine its effectiveness with other competitive models such as S^2SQL.

Acknowledgments

We would like to thank Jean-Marie TSHIMULA and Christian BOPE for the talks that shaped this work. We also thank anonymous reviewers for their crucial input.

Conflict of Interest Statement

All authors declare that they have no conflicts of interest.

References

[1] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson (2020): "RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers, . [Online]. Available: https://github.com/Microsoft/rat-sql.

[2] B. Hui et al (Mar. 2022): "S^{2}SQL: Injecting Syntax to Question-Schema Interaction Graph Encoder for Text-to-SQL Parsers, [Online]. Available: http://arxiv.org/abs/2203.06958.

[3] T. Scholak, R. Li, D. Bahdanau, H. de Vries, and C. Pal (Oct. 2020): "DuoRAT: Towards Simpler Text-to-SQL Models, doi: 10.18653/v1/2021.naacl-main.103.

[4] T. Yu et al (Sep. 2020), "GraPPa: Grammar-Augmented Pre-Training for Table Semantic Parsing, [Online]. Available: http://arxiv.org/abs/2009.13845.

[5] Z. Lan et al (2020): "Albert: A Lite Bert For Self-Supervised Learning Of Language Representations. [Online]. Available: https://github.com/google-research/ALBERT.

[6] J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova Bert-Ppt (2018): "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (Bidirectional Encoder Representations from Transformers).

[7] E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov (2019), "Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting,

the Rest Can Be Pruned," [Online]. Available: https://github.com/.
[8] E. F. Codd (1974), "Recent Investigations in Relational Data Base Systems," in IFIP Congress.
[9] A. Suhr, S. Iyer, Y. Artzi, and P. G. Allen (2018), "Learning to Map Context-Dependent Sentences to Executable Formal Queries. [Online]. Available: https://github.com/clic-lab/atis.
[10] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer (Apr. 2017): "Learning a Neural Semantic Parser from User Feedback, [Online]. Available: http://arxiv.org/abs/1704.08760.
[11] J. Herzig and J. Berant (Apr. 2018): "Decoupling Structure and Lexicon for Zero-Shot Semantic Parsing, [Online]. Available: http://arxiv.org/abs/1804.07918.
[12] A. Kamath and R. Das (Dec. 2018), "A Survey on Semantic Parsing", [Online]. Available: http://arxiv.org/abs/1812.00978.
[13] L. Dong and M. Lapata (May 2018): "Coarse-to-Fine Decoding for Neural Semantic Parsing", [Online]. Available: http://arxiv.org/abs/1805.04793.
[14] L. Dong and M. Lapata (Jan. 2016): "Language to Logical Form with Neural Attention", [Online]. Available: http://arxiv.org/abs/1601.01280.
[15] O. Goldman, V. Latcinnik, U. Naveh, A. Globerson, and J. Berant (Nov. 2017): "Weakly-supervised Semantic Parsing with Abstract Examples,", [Online]. Available: http://arxiv.org/abs/1711.05240.
[16] P. Yin and G. Neubig (Oct. 2018): "TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation", [Online]. Available: http://arxiv.org/abs/1810.02720.
[17] P. Yin and G. Neubig (Apr. 2017), "A Syntactic Neural Model for General-Purpose Code Generation", [Online]. Available: http://arxiv.org/abs/1704.01696.
[18] C. Xiao, M. Dymetman, and C. Gardent (2016): "Sequence-based Structured Prediction for Semantic Parsing," [Online]. Available: https://github.com/percyliang/sempre.
[19] J. Krishnamurthy, P. Dasigi, and M. Gardner (2017): "Neural Semantic Parsing with Type Constraints for Semi-Structured Tables.
[20] V. L. Shiv and C. Quirk (2019): "Novel positional encodings to enable tree-based transformers," in Advances in Neural Information Processing Systems, vol. 32.
[21] Q. He, J. Sedoc, and J. Rodu (Dec. 2021): "Trees in transformers: a theoretical analysis of the Transformer's ability to represent trees, [Online]. Available: http://arxiv.org/abs/2112.11913.
[22] N. Kitaev, Ł. Kaiser, and A. Levskaya (Jan. 2020): "Reformer: The Efficient Transformer", [Online]. Available: http://arxiv.org/abs/2001.04451.
[23] A. Vaswani et al (Jun. 2017): "Attention Is All You Need", [Online]. Available: http://arxiv.org/abs/1706.03762.
[24] J. Guo et al (2019): "Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation.
[25] R. Cao, L. Chen, Z. Chen, Y. Zhao, S. Zhu, and K. Yu (Jun. 2021): "LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations, [Online]. Available: http://arxiv.org/abs/2106.01093.
[26] X. V. Lin, R. Socher, and C. Xiong (Dec. 2020): "Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing", [Online]. Available: http://arxiv.org/abs/2012.12627.
[27] A. Gur, S. Yavuz, Y. Su, and X. Yan, "DialSQL: Dialogue Based Structured Query Generation.
[28] B. Bogin, M. Gardner, and J. Berant (2019): "Global Reasoning over Database Structures for Text-to-SQL Parsing.
[29] B. Bogin, M. Gardner, and J. Berant (Apr. 08, 2022): "Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing," pp. 4560–4565, 2019, Accessed [Online]. Available: https://github.com/benbogin/.
[30] C. Xu, W. Zhou, T. Ge, F. Wei, and M. Zhou (2020), "BERT-of-Theseus: Compressing BERT by Progressive Module Replacing . [Online]. Available: https://en.wikipedia.org/wiki/Ship_.
[31] M. Shoeybi, M. Patwary, R. Puri, P. Legresley, J. Casper, and B. Catanzaro, "Megatron-LM (2020): Training Multi-Billion Parameter Language Models Using Model Parallelism," [Online]. Available: https://github.com/.

[32] V. Sanh, L. Debut, J. Chaumond, and T. Wolf (Oct. 2019): “Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter, [Online]. Available: http://arxiv.org/abs/1910.01108.

[33] M. Lewis et al (Oct. 2019): “BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension, [Online]. Available: http://arxiv.org/abs/1910.13461.

[34] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever (2018): “Improving Language Understanding by Generative Pre-Training, [Online]. Available: https://gluebenchmark.com/leaderboard.

[35] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever (2019): “Language Models are Unsupervised Multitask Learners, [Online]. Available: https://github.com/codelucas/newspaper.

[36] Y. Liu et al (Jul. 2019): “RoBERTa: A Robustly Optimized BERT Pretraining Approach,” [Online]. Available: http://arxiv.org/abs/1907.11692.

[37] T. Wolf et al (Oct. 2019): “HuggingFace’s Transformers: State-of-the-art Natural Language Processing” [Online]. Available: http://arxiv.org/abs/1910.03771.

[38] T. B. Brown et al (2020): “Language Models are Few-Shot Learners. [Online]. Available: https://commoncrawl.org/the-data/.

[39] W. Fedus, B. Zoph, and N. Shazeer (2022): “Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity.

[40] K. Clark, M.-T. Luong, Q. v. Le, and C. D. Manning (Mar. 2020): “Electra: Pre-training Text Encoders as Discriminators Rather Than Generators, [Online]. Available: http://arxiv.org/abs/2003.10555.

[41] T. Scholak, N. Schucher, and D. Bahdanau (Sep. 2021): “PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models [Online]. Available: http://arxiv.org/abs/2109.05093.

[42] C. Raffel et al (Oct. 2019): “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, [Online]. Available: http://arxiv.org/abs/1910.10683.

[43] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. Eisenschlos (2020): “TaPas: Weakly Supervised Table Parsing via Pre-training, . doi: 10.18653/v1/2020.acl-main.398.

[44] L. Zhao, H. Cao, and Y. Zhao (Jan. 2021) “GP: Context-free Grammar Pre-training for Text-to-SQL Parsers,” [Online]. Available: http://arxiv.org/abs/2101.09901.

[45] X. Deng, A. H. Awadallah, C. Meek, O. Polozov, H. Sun, and M. Richardson (Oct. 2020), “Structure-Grounded Pretraining for Text-to-SQL,” doi: 10.18653/v1/2021.naacl-main.105.

[46] P. Yin, G. Neubig, W. Yih, and S. Riedel (May 2020): “TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data,” pp. 8413–8426, doi: 10.48550/arxiv.2005.08314.

[47] P. Shaw, J. Uszkoreit, G. Brain, and A. Vaswani (2018): “Self-Attention with Relative Position Representations.

[48] O. Vinyals, M. Fortunato, and N. Jaitly (Jun. 2015): “Pointer Networks,” [Online]. Available: http://arxiv.org/abs/1506.03134.

[49] M. Schuster and K. K. Paliwal (1997): “Bidirectional recurrent neural networks,” IEEE Transactions on Signal Processing, vol. 45, no. 11, doi: 10.1109/78.650093.

[50] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio (2014): “On the properties of neural machine translation: Encoder–decoder approaches,” in Proceedings of SSST 2014 - 8th Workshop on Syntax, Semantics and Structure in Statistical Translation. doi: 10.3115/v1/w14-4012.

[51] S. Hochreiter and J. Schmidhuber (1997): “Long Short-Term Memory,” Neural Comput, vol. 9, no. 8, doi: 10.1162/neco.1997.9.8.1735.

[52] T. Yu et al (Sep. 2018): “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task,” [Online]. Available: http://arxiv.org/abs/1809.08887.

[53] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. Mcclosky (2014), “The Stanford CoreNLP Natural Language Processing Toolkit.

[54] J. Pennington, R. Socher, and C. D. Manning (2014): “GloVe: Global Vectors for Word Representation, [Online]. Available: http://nlp.

[55] Y. Gal and Z. Ghahramani (2016): “A Theoretically Grounded Application of Dropout in Recurrent Neural Networks.

[56] A. Paszke et al (2019): “PyTorch: An Imperative Style, High-Performance Deep Learning Library.
[57] D. P. Kingma and J. Ba (Dec. 2014): “Adam: A Method for Stochastic Optimization, [Online]. Available: http://arxiv.org/abs/1412.6980.