# AIRI NLP Team at EHRSQL 2024 Shared Task: T5 and Logistic Regression to the Rescue

**Oleg Somov**[1,2], **Aleksei Dontsov**[1], **Elena Tutubalina**[1,3,4]

[1]AIRI, [2]MIPT, [3]Sber AI, [4]Kazan Federal University

**Correspondence:** somov@airi.net

## Abstract

This paper presents a system developed for the Clinical NLP 2024 Shared Task, focusing on reliable Text-to-SQL modeling on Electronic Health Records (EHRs). The goal is to create a model that accurately generates SQL queries for answerable questions while avoiding incorrect responses and handling unanswerable queries. Our approach comprises three main components: a query correspondence model, a Text-to-SQL model, and an SQL verifier. For the query correspondence model, we trained a logistic regression model using hand-crafted features to distinguish between answerable and unanswerable queries. As for the text-to-SQL model, we utilized T5-3B as a pre-trained language model, further fine-tuned on pairs of natural language questions and corresponding SQL queries. Finally, we applied the SQL verifier to inspect the resulting SQL queries. During the evaluation stage of the shared task, our system achieved an accuracy of 68.9% (metric version without penalty), positioning it at the fifth-place ranking. While our approach did not surpass solutions based on large language models (LMMs) like ChatGPT, it demonstrates the promising potential of domain-specific specialized models that are more resource-efficient. The code is publicly available at https://github.com/runnerup96/EHRSQL-text2sql-solution.

## 1 Introduction

Electronic health records (EHRs) play a critical role in storing comprehensive medical histories within hospital settings, capturing everything from patient admissions to treatment and discharge. However, efficiently retrieving relevant information from these records remains a significant challenge, particularly from complex medical relational databases.

This paper focuses on enhancing a text-to-SQL system specifically designed for the medical domain. The aim is to improve the retrieval process of patient information and enable better clinical decision-making. The objective is to develop a model capable of accurately generating SQL queries for answerable questions while effectively handling unanswerable queries and avoiding incorrect responses. In other words, when faced with unanswerable questions, the model should refrain from generating any SQL prediction and indicate the absence of an answer by returning NULL.

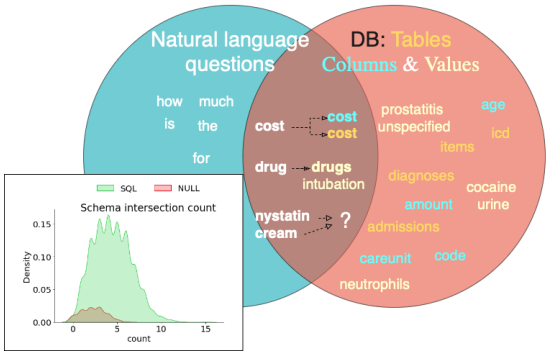


Figure 1: The schema intersection algorithm. We match normalized n-grams of an input natural language question "How much is the cost for the drug nystatin cream?" against normalized database (DB) content. As we can see on schema intersection count distribution, the NULL questions have much less schema intersection elements in comparison to SQL questions. For more details, please refer to Section 3.

To conduct experiments, we utilize the Text-to-SQL benchmark (Lee et al., 2022) provided by the organizers of the Clinical NLP 2024 task (Lee et al., 2024). This benchmark consists of pairs of input utterances and expected SQL queries, including cases where generating an SQL query is impossible for a given question. This dataset is linked to two open-source EHR databases—MIMIC-III (Johnson et al., 2016) and eICU (Pollard et al., 2018). This benchmark includes questions that address the actual needs of a hospital and incorporate various time expressions crucial to daily healthcare work.

In this paper, we describe our solution for the



*Proceedings of the 6th Clinical Natural Language Processing Workshop*, pages 431–438

Clinical NLP 2024 shared task on reliable Text-to-SQL. As shown in Figure 2, our system consists of three components - query correspondence model, Text-to-SQL model, and SQL result inspector. To sum up, the system takes the user's query as input and goes through the following steps: feature extraction, query scoring using the query correspondence model and alignment check, SQL generation using a Text-to-SQL model with question and schema input representation, SQL results inspection, execution of the generated query, and checking if the execution result meets the requirements. If the requirements are met, the system returns the result to the user.

The paper is organized as follows. Section 2 presents related work on Text-to-SQL corpora and state-of-the-art (SoTA) models. We describe our model with three components in Section 3. Experiments with baselines and our model are presented in Section 4. Finally, we discuss errors and conclude the work in Sections 5 and 6, respectively.

## 2 Related work

Text-to-SQL currently is one of the most developing and promising research areas in the field of semantic parsing. Well-known public leaderboard Spider (Yu et al., 2018) popularized the task, and Text-to-SQL domain developed many directions and specializations. Spider dataset is a complex and cross-domain Text-to-SQL dataset which consists of 10181 questions with 5693 SQL queries on 200 databases. The main goal of the dataset is to generalize to new databases. However, the Spider dataset does not contain unanswerable questions. Spider also gave rise to more complex datasets like BIRD (Li et al., 2024), which paid attention not only to SQL query complexity (introduction of window functions, etc.) but also to the optimality of the generated query. BIRD databases are close to real-world examples, with tables consisting of millions of data rows; hence, the optimal SQL is required.

Another dataset named CoSQL (Yu et al., 2019) raised questions about ambiguity and the system's ability to handle such questions. It is a dialogue-based Text-to-SQL benchmark, which consists of the following dialogue acts - answering user questions with SQL, double checking the user intent if the questions are ambiguous, or the system reminder to the user that the question is not related to the database.

Spider leaderboard gave rise to specialized Text-to-SQL architectures. Naturally, SoTA solutions adapted the following Text-to-SQL solutions - schema linking stage, encoding of question and schema, and subsequent decoding. Starting from most notable solutions like BRIDGE, which induced database content into training process (Lin et al., 2020) and RAT-SQL, which modified transformer architecture for question with schema interaction and specialized grammar-based decoding process (Wang et al., 2021) coming to the fine-tuning approaches which reached its peak in RESDSQL (Li et al., 2023) approach and PICKARD (Scholak et al., 2021). LLMs are also present in the leaderboard in the form of in-context learning few-shot approaches(Gao et al., 2023a) and SQL debugging stages (Pourreza and Rafiei, 2024). Most solutions utilize ChatGPT-4 as a core model and experiment with different prompt strategies for stages of schema linking, query generation, and SQL debugging.

Increased attention towards Text-to-SQL domain detected the problem of generalization in semantic parsing. The Spider dataset focused on cross-domain generalization, but the work of (Suhr et al., 2020) made the challenges more visible, introducing the challenges of single database split compared to cross-database setting. Recently, Somov and Tutubalina (2023) evaluated the generalization capabilities of supervised models on the original, multilingual, and target length splits of the improved version of the Spider dataset called PAUQ (Bakshandaeva et al., 2022). Results indicate that the models can generalize well to unseen simple SQL's, while multilingual split shows that some models benefit from learning on the translated task.

Overall, the ongoing progress in dataset development and the advancement of specialized architectures have significantly contributed to a deeper understanding of the Text-to-SQL task and its applications across various domains, including medicine.

## 3 Main method

Our final solution consists of 3 components - query correspondence module, fine-tuned Text-to-SQL model, and SQL result inspector, which checks the result of the generated query. The system pipeline is presented in Figure 2. The system output can be NULL if the system considers the query unanswerable or results if the system can answer the query. This section will describe our validation schema

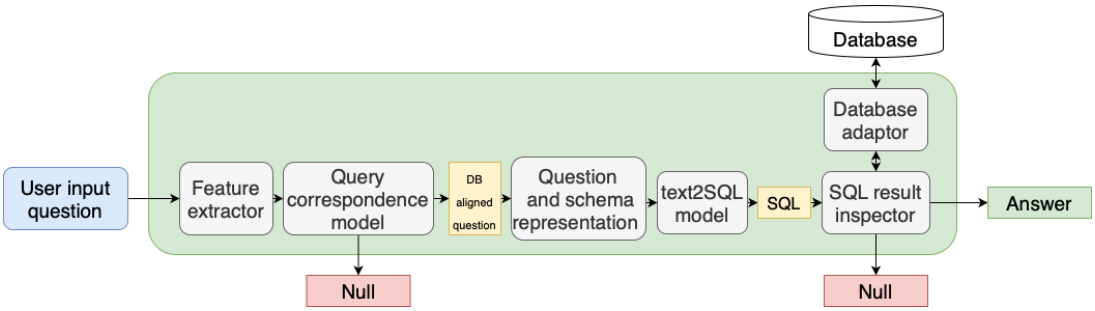


Figure 2: The system overview. The user query inputs into the Text-to-SQL system. The feature extractor extracts features for the query correspondence model. The query correspondence model scores the query by extracted features. If the question is aligned with our system, we pass the input question into Text-to-SQL generation model. It consists of question and schema input representation component and Text-to-SQL model. The generated query is passed to the SQL results inspector, which checks weather the query can be executed and checks the result of the execution. If the query execution result meets the requirement, we return the result to the user.

and all the system components in detail.

### 3.1 Validation schema

For our method evaluation, we have developed our validation schema. Our solution consists of two machine learning models - Query Correspondence model and Text-to-SQL model. The leader board submission of NULL revealed that the evaluation and test sets consist of approximately $20\%$ NULL's while our training set has approximately $9\%$ of NULL's. For the Query Correspondence model, we have prepared a similar test distribution - the training set has $10\%$ of NULLs while the validation set has $20\%$ of nulls. Since we do not observe the distribution shift for SQL question, for Text-to-SQL model, we prepared an i.i.d. splitting for evaluation.

### 3.2 Query correspondence model

The query correspondence model (QCM) is a component that analyzes input questions and discards them if they look like questions that can not be answered based on database content. It consists of two components - a feature extractor and a machine learning model. We get the input question and run preprocessing. The preprocessing steps include - punctuation cleaning, stop-word exclusion, lemmatization, and lowercase casting. Then, we extract 3 features from the processed question - schema intersection feature, first-word feature, and query length feature.

- Schema intersection feature is the number of elements from the database(attributes, tables, values) in the question. We extract and preprocess database content with punctuation cleaning, lemmatization, and lowercase casting. We merge attributes, tables, and values into one set. The processed question is tokenized by spaces and transformed into another set. The intersection between these two sets is the result feature value. On the public test set, the feature for detection of NULL scored $18.77\%$, detecting $94\%$ of NULL questions. Since the feature proved to be important for NULL question discarding, we later used it in our experiment as a decision component with a manually selected threshold. The schema intersection algorithm and corresponding intersection count distribution is presented in Figure 1.

- We examined the intersection of the processed NULL questions beginning (first 2 words) with processed SQL questions beginning and found out that there is only an $8\%$ intersection between sentence beginnings. Therefore, we matched all the NULL first 2 words against the input question. If the input processed sentence is matched against processed null sentence beginnings, the feature value is True and False otherwise.

- We have analyzed NULL question length and SQL question length and saw that the average length of NULL questions is 11 ($\sigma = 3$), and the average length of SQL questions is 15 ($\sigma = 6$). Due to such differences, we also utilized question process length as a feature.

During our experiments, we have also used other features, like pre-trained language model maximum entropy score, SVM classifier (Vapnik and

Chervonenkis, 1974) score based on TF-IDF encoder, pre-trained Transformer (Devlin et al., 2018) encoder-based retrieval features (distance to closest question with SQL, distance to closest question with NULL, number of NULL candidates @5/@10/@100 - but these features made our results only worse on public test set, so we have discarded them.

We pass these selected features through normalization and then pass them to a logistic regression model. We trained this model on the binary task on standardized extracted features and predicted SQL vs. NULL for every question. We evaluate our solution based on two metrics: sensitivity (Se) and specificity (Sp).

$$Se = \frac{TP}{TP + FN}, \quad Sp = \frac{TN}{TN + FP}$$

On our split, the model gets the average of sensitivity and specificity equal to 0.91 on our validation set with a 0.5 threshold. If the question is predicted as NULL, we do not pass the question further.

### 3.3 Text-to-SQL model

The next step is the Text-to-SQL model. We chose the T5-3B (Raffel et al., 2020) model, and our evaluation showed its high performance. If the query correspondence model evaluates the question as answerable, we pass the question to the text-to-SQL model. We wanted the model to learn only Text-to-SQL task. Therefore, we have trained the model on only text-to-SQL pairs. On our validation set, the execution match from the benchmark evaluation script with this model was Acc0 = 99%, Acc5= 92%, Acc10=86%, AccN=−501% on Text-to-SQL pairs only. We use classic input representation 1 as a concatenation of database name, question, and linearized schema representation of tables $T$ and columns $C$ (Shaw et al., 2021).

$$X = \textit{Database name} : \textit{Question} \mid [T_1] : [C_{11}], ...[C_{1|T_1|}] \mid [T_2] : [C_{21}], ... \quad (1)$$

We normalized the target SQL query with classic Spider Text-to-SQL fine-tuned model preprocessing as in RESDSQL (Li et al., 2023). The target representation during training was the following:

$$Y = \textit{Database name} \mid \textit{Query} \quad (2)$$

We have trained our T5-3B model for 16 epochs(approximately 4000 iterations) with a training batch size of 2 and gradient accumulation batch size of 8. The learning rate was 5e-5. Our input maximum length was 800, and the target length was 514. As demonstrated in Figure 3 we see that the model successfully converged and reached decent exact match accuracy.

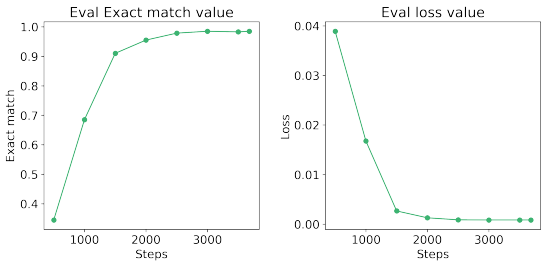


Figure 3: T5-3B training process on Text-to-SQL custom i.i.d. validation split from section 3.1. On the right exact match training plot, we see that the model decently learns to correctly align novel questions to SQL queries as the validation loss decreases.

### 3.4 SQL result inspector

After generation, we pass the result SQL to the SQL inspector. We rely on the hypothesis that the user must be very specific in his question to correctly match the elements of the schema in his question and get an answer to his question. Therefore, if the query fails and returns a None or 0 value for aggregate queries - we treat it as a false and exit with NULL. We examined the training SQL query outputs and discovered that approximately 90% of training queries return some meaningful result, meaning not None or 0 value for aggregate queries. We evaluate the SQL inspector on the EHRSQL train set - we run T5 prediction through it and evaluate how many generated SQL queries the inspector will discard and how many will approve. As in the Query Correspondence model, we measure sensitivity and specificity as in Equations 3.2. We get the average of sensitivity and specificity of 96%.

## 4 Experiments

This section describes our most successful attempts on the test leaderboard. Solutions feature schema intersection algorithm explained in Sec. 3.2 and SQL inspector explained in Sec. 3.4. In Table 1, we present official evaluation scores[1]: Accuracy0, Accuracy5, Accuracy10, AccuracyN. These metrics differ in penalty strategy for wrong predictions. In particular, Accuracy0 does not penalize any mis-

[1]For details, see https://www.codabench.org/competitions/1889/

| | Method | Accuracy0 | Accuracy5 | Accuracy10 | AccuracyN |
|---|---|---|---|---|---|
| 1 | Schema intersection@2 + ChatGPT ICL 5-shot | 55 | -41.8 | -138.6 | -22545 |
| 2 | Schema intersection@2 + T5-3B + ChatGPT debugger + SQL inspector | 53.3 | -27.2 | -107.8 | -18746.7 |
| 3 | Schema intersection@2 + T5-3B + SQL inspector | 64.4 | 53.3 | 42.2 | **-2535.6** |
| 4 | QCM + T5-3B + SQL inspector | **68.9** | **56.5** | **44** | -2831.1 |

Table 1: Experimental results of our systems on the official test set.

takes, while Accuracy5 counts a -5 penalty for each mistake result.

### 4.1 ChatGPT: in-context learning with few-shot examples

In 2020, the paradigm of in-context learning, introduced by Radford et al. (2019), emerged as a powerful technique that enables Language Model Models (LLMs) to solve problems without requiring fine-tuning. We effectively leverage the potential of few-shot learning by exposing the model to a few examples from the training set along with their corresponding solutions. To facilitate this process, we create an index of training questions with corresponding SQL query by extracting embedding of the question using SentenceBERT[2] (Reimers and Gurevych, 2019). To identify the most similar matches, we calculate the Euclidean distance between the index question vectors and the embedding of the natural language question. These selected questions, along with their corresponding SQL queries, are included in the prompt.

Furthermore, to provide the LLM with an understanding of the database's structure, we append a textual representation of the entire database schema and question at the end of the prompt. This approach mirrors the methodology employed in the DAIL-SQL technique (Gao et al., 2023b).

The final prompt is further passed into OpenAI API[3], model version `gpt-3.5-turbo`.

After gathering the results, we filtered out queries that did not pass our schema intersection manual threshold of 2.

### 4.2 ChatGPT debugger

Recent advances on the Spider leaderboard showed that the ChatGPT can not only work as an in-context learning algorithm but can also refine a given query. We have utilized the DIN-SQL (Pourreza and Rafiei, 2024) approach for self-correction. To address this, DIN-SQL proposed a self-correction module where the model is instructed to correct those minor mistakes. This is achieved in a zero-shot setting, where only the buggy code is provided to the model, and it is asked to fix the bugs. DIN-SQL proposed two different prompts for the self-correction module: generic and gentle. The generic prompt, DIN-SQL requests the model to identify and correct the errors in the "BUGGY SQL". The gentle prompt, on the other hand, does not assume the SQL query is buggy; instead, it asks the model to check for any potential issues and provides some hints. Since our Text-to-SQL T5-3B model performed well on our validation split, we have utilized a gentle approach for the model to fix potential bugs. We have used the original implementation [4]. Also, DIN-SQL experiments showed that a gentle prompt is more effective for the GPT-4 model, which proved to be better at this task. After gathering the results, we filtered out queries that did not pass our schema intersection manual threshold of 2. The query debugger algorithm resulted in quality deterioration in comparison to our final solution. The GPT-4 debugger of generated T5 queries usually just deleted some comparisons or conditions from the final query, making more false positive predictions.

### 4.3 RESDSQL fine-tuning

We have also tried to fine-tune the RESDSQL solution to the EHRSQL task. RESDSQL is fine-tuned SoTA on the Spider leaderboard. It consists of two training phases - cross-encoder classifier for question-relevant columns and tables detection and query generation stage via a pre-trained language model. During training in the query generation stage, the decoder input is prefixed with SQL skeleton, forcing the model to generate a correct SQL template and then fill it with schema elements and values. Although the cross-encoder component had a validation AUC score for detection of tables and columns 97.7%, the result execution accuracy0 on

[2] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[3] https://platform.openai.com

[4] https://github.com/MohammadrezaPourreza/Few-shot-NL2SQL-with-prompting

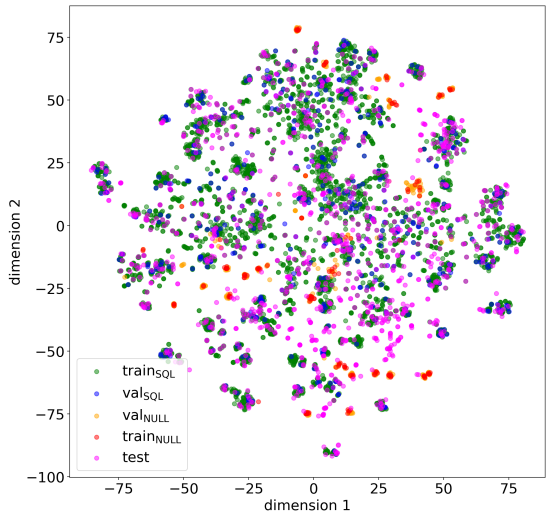


Figure 4: The distribution of embedding in two-dimensional space via t-SNE of our custom split of original training EHRSQL data of train and validation and along with test questions as in 3.1 for evaluation of Query Correspondence Model. **train**$_{SQL}$ stands for train question embeddings which have SQL, **train**$_{NULL}$ which do not.**val**$_{SQL}$ stands for val question embeddings which have SQL, **val**$_{NULL}$ which do not. **test** stands for EHRSQL test question embeddings.

our Text-to-SQL validation set was 77.5%. We suspect the problem is the inconsistency of the SQL skeleton with the target SQL query.

For example, almost every second SQL EHRSQL query contains `strftime` function, which includes two attributes; however, the RESD-SQL SQL skeleton, which is prefixed in the query generation model, contained only one attribute.

### 4.4 Experimental results

As shown in Table 1, we have conducted 4 submission experiments. All of our experiments use the schema intersection feature. The first three use it as a decision feature, while the final solution model uses it as a feature in QCM. We see that the first experiment was the worst. There was no SQL inspector phase, and ChatGPT itself generated incorrect queries. Then, we enhanced our solution with the SQL inspector component and used ChatGPT as a debugger for our T5 predictions. Unfortunately, the debug mode worsened SQL predictions, but due to the SQL inspector, we had fewer false positive predictions, as we can see in the accuracies of a penalty. We concentrated on purely T5-3B and other components in our following experiments. At first, we used only the schema intersection feature to discard unanswerable queries, but after careful exploratory data analysis, we found more features to be a good signal for our decision - so we developed a query correspondence model, which gave us the highest score.

Although we have good accuracy across all of our components on our validation split, we have much lower results on the test leaderboard. In Figure 4, we have plotted reduced SentenceBERT questions embeddings via the t-SNE algorithm on a coordinate plane. We see that the testing questions are shifted relatively to training data in terms of SQL and NULL questions. Although we also mimic data drift in our validation schema as in section 3.1, our validation questions are still closer to training questions than test questions. The solution to that problem might be running a solution in production mode with activated data markup for online and offline metrics alignment.

## 5 Error analysis

We manually checked the errors of our final system components on our validations sets from 3.1. The Text-to-SQL T5-3B errors mostly consist of regular errors - `ASC` to `DESC` mismatch, wrong column, missed comparison expression. Sometimes the model shows the overfitting signs - looping the prediction output, adding wrong syntactic constraints (like adding not needed `GROUP BY`) or extra symbol to value (`'10-31'` → `'10-31\'`).

The query correspondence errors come mostly from the starting word feature - although it helps to identify questions with starting phrases that were in the training set, it does not help to combat novel starting phrases that occur in the test set.

As we pointed out, we evaluated our query inspector on sensitivity and specificity metrics. Specificity is 99%, and sensitivity is 94%. We can see that we are stricter than necessary to generated queries, and sometimes correct SQL can return the result of None or 0, but we will not return it to the user.

## 6 Discussion and conclusion

In this work, we have built a reliable Text-to-SQL solution. We have developed our validation schema for model evaluation and submitted our final system results to the EHRSQL leaderboard. Our solution consists of 3 components - query correspondence model, Text-to-SQL generation model, and SQL inspector. During validation, we measured our performance based on sensitivity and specificity

metrics to account for NULL queries and execution match for the query generation model. The sensitivity and specificity metrics for the query correspondence model and SQL inspector are $81\%/99\%$ and $94\%/99\%$ accordingly. The execution accuracy of Text-to-SQL model is $99\%$. Our components are independent of each other; therefore, we can calculate the product probability that the NULL question will be discarded is $98\%$, while the probability that the SQL question will be answered correctly is $75\%$.

The advantages of our system are the following:

- The solution discards unanswerable queries with high precision while keeping a decent execution accuracy.
- Our components can be independently optimized.
- The solution is interpretable because every component has its single responsibility.
- Our model can be used on-premise without confidential data leaks to external language models.

The disadvantages of the system are:

- The cascading effect of the system leads to lower execution accuracy.
- Weak out-of-distribution robustness.
- We employ a heavy Text-to-SQL T5-3B pretrained language model, which needs significant resources for deployment.

As a future work direction, we see the necessity of developing reliable, robust, and lightweight specialized solutions. These solutions can be run and maintained on-premise without exposing personal data to external LLMs.

## Acknowledgments

The work of E.T. has been supported by the Russian Science Foundation grant # 23-11-00358. We would also like to thank the anonymous reviewers for their comments on this paper.

## Ethics Statement

One limitation of using databases for retrieval is that these sources may not be complete and can include errors. T5-3B, like any language model, may be subject to representation biases and potentially misleading results, which is a critical concern in the healthcare domain.

All pre-trained language models and datasets used in this work are publicly available for research purposes.

We honor and support the ACL Code of Ethics.

## References

Daria Bakshandaeva, Oleg Somov, Ekaterina Dmitrieva, Vera Davydova, and Elena Tutubalina. 2022. Pauq: Text-to-sql in russian. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 2355–2376.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023a. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023b. Text-to-sql empowered by large language models: A benchmark evaluation.

Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 2016. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9.

Gyubok Lee, Hyeonji Hwang, Seongsu Bae, Yeonsu Kwon, Woncheol Shin, Seongjun Yang, Minjoon Seo, Jong-Yeup Kim, and Edward Choi. 2022. Ehrsql: A practical text-to-sql benchmark for electronic health records. *Advances in Neural Information Processing Systems*, 35:15589–15601.

Gyubok Lee, Sunjun Kweon, Seongsu Bae, and Edward Choi. 2024. Overview of the ehrsql 2024 shared task on reliable text-to-sql modeling on electronic health records. In *Proceedings of the 6th Clinical Natural Language Processing Workshop*, Mexico City, Mexico. Association for Computational Linguistics.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4870–4888.

Tom J Pollard, Alistair EW Johnson, Jesse D Raffa, Leo A Celi, Roger G Mark, and Omar Badawi. 2018. The eicu collaborative research database, a freely available multi-center database for critical care research. *Scientific data*, 5(1):1–13.

Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

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. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional generalization and natural language variation: Can a semantic parsing approach handle both? In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 922–938, Online. Association for Computational Linguistics.

Oleg Somov and Elena Tutubalina. 2023. Shifted pauq: Distribution shift in text-to-sql. In *Proceedings of the 1st GenBench Workshop on (Benchmarking) Generalisation in NLP*, pages 214–220.

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.

Vladimir Vapnik and Alexey Chervonenkis. 1974. Theory of pattern recognition.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2021. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers.

Tao Yu, Rui Zhang, He Yang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, et al. 2019. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. *arXiv preprint arXiv:1909.05378*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.