TECHNISCHE UNIVERSITÄT ILMENAU
Fakultät für Informatik und Automatisierung

Master Thesis

A Review on State-of-the-art Text-To-SQL Solutions

presented by

Shahriar Yazdipour
Matrikel 62366

Supervisor:

Prof. Dr.-Ing. Patrick Mäder
M. Sc. Martin Hofmann

Ilmenau, April 1, 2023

Dedication

Dedicated to my family and friends for their support and encouragement throughout my academic journey.

I dedicate this thesis to the brave and heroic Iranian women who have stood up against oppression and fought for their rights and freedoms. These women, often at significant personal risk, have courageously spoken out against the injustices they have faced and have worked tirelessly to bring about positive change in their country.

Their tireless efforts and dedication to the cause of gender equality and social justice have inspired me and countless others worldwide. I am deeply grateful for their unwavering commitment to making the world a better place for all.

This thesis is also dedicated to the memory of those who have lost their lives in the struggle for equality and justice. Their sacrifice will never be forgotten, and their legacy will inspire future generations to fight for a more just and equitable world.

Also, I express my love to my parents, who have always been my biggest supporters and have believed in me throughout my academic journey. Their love, guidance, and encouragement have been invaluable to me.

I am immensely grateful to my advisors, Prof. Patrick Mäder and also Martin Hofmann, who has been excellent mentor and guide throughout the process of writing this thesis. Their expertise and support have been instrumental in helping me to complete this work.

Contents

1 Introduction

Data retrieval in databases is typically done using SQL (Structured Query Language). Text-to-SQL machine learning models are a recent development in state-of-the-art research. The technique is an attractive alternative for many natural language problems, including complex queries and extraction tasks. The text is converted into a SQL query that can be executed on the database. This technique can save time and effort for both developers and end-users by enabling them to interact with databases through natural language queries. Machine learning and knowledge-based resources aid in converting text language to SQL.

Text-to-SQL allows the elaboration of structured data with information about the natural language text in several domains, such as healthcare, customer service, and search engines. It can be used by data analysts, data scientists, software engineers, and end users who want to explore and analyze their data without learning SQL. It can be used in a variety of ways:

- Data analysts can use it to generate SQL queries for specific business questions, such as "What are the top ten products sold this month?"

- Data scientists can use it to generate SQL queries for machine learning experiments, such as "How does the price of these products affect their sales?"

- Businesses can use this technique to automate data extraction and improve efficiency.

- End-users who want to explore and analyze their data without learning SQL can use it by clicking on a button on any table or chart in a user interface.

Although these Text-to-SQL models may provide a partial solution to this complex problem, humans still have challenges to overcome. Even experienced database administrators and developers can need help with the task of dealing with unfamiliar schema when working on database migration projects. This is often due to the fact that they have never seen the schema before and therefore need to learn how to read and interpret it correctly. Furthermore, it can take time to determine how to make the necessary changes in order to migrate the data from one database to another successfully. In spite of these challenges, it is possible to successfully complete a database migration project with the help of a text-to-SQL model, as long as the model is carefully implemented and the proper steps are taken.

This research study will examine the various natural language processing (NLP) technologies that have been utilized in recent years to convert text language into Structured Query Language (SQL). Specifically, it will explore and compare the most commonly used NLP technologies and review their effects on the effectiveness of the conversion process. Moreover, this study will also analyze the representative datasets and evaluation metrics that are utilized in the current solutions for this challenging task. By doing so, it is our hope that this research study will provide valuable insights into how NLP technologies can be effectively and efficiently utilized in the conversion of text language into SQL.

Additionally, we will undertake a comprehensive study of the SEOSS (Software Engineering Dataset for Text-to-SQL and Question Answering Tasks) dataset from our esteemed researchers at the university. We will then evaluate the execution of this dataset using the most advanced Text-to-SQL model currently available. This will enable us to understand the capabilities of the SEOSS dataset better and help us to make informed decisions.

1.1 Challenges

Text-to-SQL is an intricate task, given the complexity and diversity of natural language and the structure and regulations of SQL. One of the most challenging aspects is to decipher the intent and significance of the natural language input, as it can be ambiguous or have varied interpretations. This can result in mistakes when building the corresponding SQL query, like selecting the incorrect table or columns or not recognizing the conditions for filtering or sorting the data. Additionally, the natural language input may contain typos or unknown words, which can complicate the mapping process. Moreover, the query generated may not be in the optimal form, as it has to take into account the various data types, operations, and constraints of the underlying database. Therefore, it is crucial to develop models and algorithms that can accurately map natural language to SQL queries.

Another challenge is dealing with the diverse and dynamic nature of databases, as the schema and data may change over time, and there may be variations in naming conventions and conventions across different databases. This can make it difficult for the model to correctly map the natural language input to the appropriate SQL elements, such as table and column names, and to handle variations in the structure of the SQL queries generated. Additionally, many real-world scenarios require integration with external knowledge bases and ontologies, which can be challenging to handle, especially when the external knowledge needs to be completed or consistent. Furthermore, the system must be robust to different types of user input, such as colloquial or informal language or input that needs to be completed or clarified. Additionally, Text-to-SQL systems must be able to handle errors in the input, such as typos, as well as rare edge cases that may not have been encountered during the training process. Finally, Text-to-SQL systems must be robust to the presence of out-of-vocabulary words and rare edge cases, which can be challenging to handle without significant amounts of labeled data, as well as the need to make accurate predictions with limited training data.

1.2 Thesis Outline

In this section, we provide an outline of our thesis.

- Chapter 1 of the thesis provides an introduction to the topic of Text-to-SQL and discusses the challenges and contributions of the research.

- Chapter 2 provides technical background on Text-to-SQL, including early approaches, recent approaches, and important terminologies such as LSTM, encoder, decoder, transformers, BERT, semantic parsing, baseline model and incremental decoding.

- Chapter 3 describes the benchmark datasets used for evaluating Text-to-SQL methods, including the ATIS, GeoQuery, IMDb, Advising, WikiSQL, and Spider datasets.

- Chapter 4 presents an overview of SOAT Text-to-SQL solutions, including Seq2SQL, SQLNet, SyntaxSQLNet, IRNet, EditSQL, RAT-SQL, and PICARD.

- Chapter 5 explains the evaluation metrics used to assess the performance of Text-to-SQL systems, including exact string matching, exact set matching, and distilled test suites.

- Chapter 6 presents an experiment and case study using the SEOSS dataset and the T5 PICARD model, with a focus on the metrics and evaluation results. The thesis also includes a section on EZ-PICARD and its Microservices practices.

- Chapter 7 concludes the thesis by summarizing the findings and exploring emerging challenges such as conversational Text-to-SQL.

2   Technical Background

In this chapter, we provide background information about the technical concepts related to the main topics of this thesis, which focus on natural language understanding and text generation. We focus on early and recent approaches and the terminology needed to understand the basics of this thesis.

The text-to-SQL problem, or NL2SQL, is defined as the following: Given a Natural Language Query (NLQ) on a Relational Database (RDB), produce a SQL query equivalent to the NLQ. Several challenges include ambiguity, schema linking, vocabulary gaps, and user errors. It has been a holy grail for the database community for over 30 years to translate user queries into SQL.

Early approaches to Text-to-SQL relied on rule-based and template-based methods, while recent approaches use neural networks and machine learning techniques. This allows them to handle a wide range of natural language inputs and generate more accurate SQL queries, which we will discuss further.

2.1   Early Approaches

Early approaches to Text-to-SQL focused on rule-based methods and template-based methods. These approaches relied on predefined templates and a set of predefined rules to generate SQL queries. These methods were based on the idea that a fixed set of templates and rules could be used to generate SQL queries for a wide range of natural language inputs. However, these methods were limited by their reliance on predefined templates and were not able to handle a wide range of natural language inputs.

2.1.1   Rule-based methods

In the case of rule-based methods, a set of predefined rules were used to map the natural language input to the corresponding SQL query. These rules were based on predefined grammar and were used to identify the SQL constructs present in the input text. These methods were able to generate simple SQL queries, but they were not able to handle more complex queries or handle variations in natural language inputs.

Early research in Text-to-SQL includes work by researchers such as Warren and Pereira in 1982[WP82], who proposed a rule-based method for generating SQL queries from natural language text. Their system used a set of predefined rules to map natural language constructs to SQL constructs and was able to generate simple SQL queries. Another example of a rule-based method is the work by Zelle and Mooney in 1996, who proposed CHILL parser[ZM96], a system that used a predefined grammar to identify the SQL constructs present in the input text and generate the corresponding SQL query. However, these rule-based methods were limited by their reliance on predefined templates and grammar rules, making them incapable of handling complex natural language inputs.

2.1.2 Template-based methods

Template-based methods, on the other hand, relied on predefined templates to generate SQL queries. These templates were based on a predefined set of SQL constructs and were used to map the natural language input to the corresponding SQL query. These methods were able to handle a limited set of natural language inputs, but they were not able to handle variations in the input or generate more complex queries. One of the very first systems that used predefined templates to map natural language inputs to SQL queries was able to handle a limited set of natural language inputs. The system was called LUNAR and was developed by Woods et al. in 1972[**?**]. In summary, early approaches to Text-to-SQL were limited by their reliance on predefined templates and rules, which made them unable to handle a wide range of natural language inputs and generate complex SQL queries. The rule-based and template-based methods were two of the most common early approaches used in Text-to-SQL, each with their own strengths and limitations.

2.2 Recent Approaches

Recent approaches to Text-to-SQL have focused on using neural networks and machine learning techniques to generate SQL queries. These methods use large amounts of training data to learn the relationship between natural language and SQL and can generate SQL queries for a wide range of inputs. These methods are able to handle a wide range of natural language inputs and are not limited by predefined templates or rules. Additionally, recent approaches leverage pre-trained models such as BERT, GPT-2, and T5, which have been pre-trained on a large corpus of text, to fine-tune text-to-SQL tasks, which enables them to understand the natural language inputs better and generate more accurate SQL queries.

One popular approach is the use of encoder-decoder architecture, which uses an encoder to encode the natural language input and a decoder to generate the corresponding SQL query. The encoder is a pre-trained language model such as BERT, which is fine-tuned on the task of text-to-SQL, and the decoder is a neural network that generates the SQL query. This architecture has been shown to be effective in generating accurate SQL queries for a wide range of natural language inputs.

Another recent approach is the use of reinforcement learning to generate SQL queries, where a neural network generates a sequence of SQL tokens and is trained using a reward signal based on the quality of the generated query. This approach has been shown to be adequate in generating more complex SQL queries and handling variations in natural language inputs.

In recent years, the Transformer architecture has had a significant impact on natural language processing and machine learning, including in the field of Text-to-SQL. The Transformer architecture, presented in the paper "Attention Is All You Need" by Vaswani et al. in 2017, is a neural network architecture that uses self-attention mechanisms to process sequences of data, such as natural language text.

One of the key advantages of the Transformer architecture is its ability to handle long-term dependencies in sequences of data, making it well-suited for tasks such as natural language understanding and text generation. This has led to the development of pre-trained Transformer models, such as BERT, GPT-2, and T5, that have been trained on a large corpus of text and can

be fine-tuned on specific tasks such as Text-to-SQL.

The use of pre-trained Transformer models such as BERT in Text-to-SQL has shown to be effective in improving the performance of the models. The pre-trained models have a good understanding of the natural language, which enables them to understand the input text better and generate more accurate SQL queries. The Transformer architecture and pre-trained models such as BERT have had a significant impact on recent studies in the field of Text-to-SQL. The ability of the Transformer architecture to handle long-term dependencies in sequences of data and the pre-trained models' good understanding of natural language has made it possible to generate more accurate SQL queries for a wide range of natural language inputs.

In summary, recent approaches to Text-to-SQL leverage neural networks and machine learning techniques, such as the use of encoder-decoder architecture and reinforcement learning. These approaches use large amounts of training data and pre-trained models such as BERT to generate accurate SQL queries for a wide range of natural language inputs.

2.3 Terminology

Here is an updated list of key terminology and vocabulary that you may need to know before studying Text-to-SQL language models:

2.3.1 Natural Language Processing (NLP)

The field of study focused on the interaction between human language and computers, which ranges from understanding spoken language to generating natural language text.

2.3.2 Tokenization

The process of breaking up a sentence into individual words or phrases is necessary for tasks such as machine translation and text summarization.

2.3.3 WordPiece embeddings

WordPiece[WSC$^+$16] is a tokenization approach used in natural language processing (NLP) to break down words into smaller units, also known as pieces. It is an extension of the original word2vec parameter learning algorithm and is used to address out-of-vocabulary (OOV) words, which are words that did not appear in the training data. This technique divides each word into a series of subword units learned during the training phase based on their frequency and consistency within words. These subword units are stored in a shared vocabulary, dubbed the WordPiece vocabulary, and can be used for multiple words. This system can represent rare or unseen words as a combination of more common subword units, which are more likely to be in the vocabulary. As a result, the model can handle OOV words more efficiently and reduce the vocabulary size, leading to a more economical representation of the language. In NLP models, words are usually portrayed as dense vectors referred to as word embeddings. WordPiece embeddings extend this representation by breaking words down into subword units and representing each piece as a dense vector. These subword embeddings are then combined to represent the whole word. The use of WordPiece embeddings has various advantages in NLP models. Firstly, it enables the model to treat OOV words more effectively by representing them as a combination of more common subword units. Secondly, it decreases the vocabulary size, resulting in a more succinct representation of the language. Finally, it enhances the model's capability to learn fine-grained representations of words and their meanings, resulting in improved performance in NLP tasks.

2.3.4 Word2Vec

Word2Vec[Ron14] is a well-known word embedding approach in NLP that encodes words as dense vectors in an unending, high-dimensional area. This technique is designed to capture the significance and context of words, providing an improved representation of words compared to classic one-hot encoding. The fundamental concept behind Word2Vec is to train a neural network to anticipate the context words about a target word, given the target word. As the model is trained, the weights of the neural network are adjusted in such a way that the dot

product of the input layer (representing the target word) and the output layer (representing the context words) closely estimate the probability distribution of the context words given the target word. Word2Vec can be trained to utilize two different algorithms: Continuous Bag-of-Words (CBOW) and Skip-gram. CBOW predicts the target word given the context words, while Skip-gram predicts the context words given the target word. The algorithm selection relies on the particular NLP task and the data available for training.

2.3.5 Encoder-Decoder Architecture

A powerful neural network architecture that utilizes an encoder[CvMG$^+$14] to transform the input data into a compact and meaningful representation and a decoder to generate the desired output from that representation. This architecture has been widely used in many applications such as language translation, image captioning, and text summarization to produce high-quality results. Furthermore, the encoder-decoder architecture has the advantage of being able to learn complex relationships between input and output, making it a suitable tool for many challenging tasks.

2.3.6 Transformers

The architecture introduced in the paper "Attention Is All You Need" by Vaswani et al. in 2017[VSP$^+$17], known as Transformers, is a revolutionary breakthrough in the way sequences of data are processed. By utilizing self-attention mechanisms, the model is able to achieve improved efficiency and accuracy, while also being much simpler to implement and deploy. This makes it particularly appealing for a wide range of applications, from natural language processing to computer vision. Furthermore, due to their scalability, Transformers are able to accommodate large data sets, enabling them to be used to tackle more complex tasks. As such, Transformers are becoming increasingly popular in the field of machine learning and artificial intelligence, with more and more research being done to further explore its capabilities.

There were many excellent works around 2015 on learning word vectors to continuous representations for words where the identity of a word was mapped to a fixed-length vector which ideally encoded some meaning about the word in a continuous space and for a long time.

That has been an essential part of the NLP pipeline, especially for deep learning models where these pre-trained word vectors were used, typically trained using an unsupervised objective, and new models were fed and trained on top of them.

An important paper in 2017 that helped researchers change their way of thinking towards the transfer learning paradigm was the unsupervised sentiment neuron paper from people at OpenAI [RJS17], which essentially showed that by just training a language model on a purely unsupervised objective, the model could learn concepts that were potentially useful for downstream tasks.

In 2018, the NLP community had a couple of super important papers, including the ULMFiT[HR18], which took the recipe from semi-supervised sequence learning, added some tweaks, figured out how to get it working better, and got some noble results with a similar pipeline, pre-training a language model, fine-tuning on a downstream task.

And then, ELMo[HR18] showed that we could get significantly better performance by using

a bi-directional language model.

Then GPT1 [RN18] came along, saying that instead of using analyst TM, we can get good performance by using a transformer with a language model.

Finally, in 2018, BERT [DCLT19] showed that a bi-directional transformer could get outstanding performance, and by the end of 2018, many researchers were convinced that this was the path forward given all of the impressive results that these papers and a few others showed.

Following these researches, there has been a burst of work on transfer learning for NLP, working on various methods, different pre-training ideas, datasets, and different benchmark tasks.

In Google T5, it is tried to use all the new studies in transfer learning and combine the best selection of these studies to achieve state-of-the-art results on many benchmarks covering summarization, classification, question answering, and more.

2.3.7 Self-attention

Self-attention[VSP$^+$17] is a mechanism used in the transformer architecture that enables the model to identify the significance of different elements of the input sequence, so as to be able to generate an output that is more accurate and effective. This mechanism allows the model to take into account the relationships between different parts of the input sequence and to factor those relationships into its output. Additionally, self-attention allows the model to capture patterns from the input sequence and to use those patterns to generate more meaningful output. It is this combination of factors that makes self-attention such an important tool for deep learning models.

2.3.8 Pre-training and Fine-tuning

Pre-training refers to training a model on a large dataset and then fine-tuning it on a smaller dataset for a specific task, which helps to improve the model's performance on the specific task.

2.3.9 Long Short-Term Memory (LSTM)

A type of recurrent neural network that has been designed to store information over a longer period of time than traditional neural networks, allowing it to better capture long-term dependencies[HS97]. This makes it especially well-suited for tasks such as language modeling and text generation, where it can take into account the context of the text in order to generate more accurate outputs. In addition, LSTM networks are able to identify patterns in the data that would be difficult for traditional networks to capture. This makes them ideal for tasks such as sequence prediction and classification, where they can identify patterns that would otherwise be too subtle for traditional networks to detect.

2.3.10 Bidirectional Encoder Representations from Transformers(BERT)

A pre-trained Transformer model that has been trained on a large corpus of text, with the primary aim of pre-training language representations for use in natural language processing tasks[DCLT19]. This pre-training helps to give BERT a strong understanding of the language structure and helps in faster training times for downstream tasks. BERT can be fine-tuned for various applications, such as Text-to-SQL, where it can provide better performance than non-specialized models. By leveraging the already learned representations from the pre-trained model, BERT is able to quickly adjust to the task at hand, resulting in faster training times.

2.3.11 SQL Constructs

The elements of SQL language such as SELECT, FROM, WHERE, JOIN, are used to build queries and retrieve data from a database.

2.3.12 Evaluation Metrics

Measures used to evaluate the performance of Text-to-SQL models, such as accuracy, F1-score, and Exact Match score, are used to compare different models and determine the best-performing model.

2.3.13 Baseline Model

A model that serves as a reference point or starting point for comparison, providing a baseline for performance against which other models can be evaluated.

2.3.14 Incremental decoding

A decoding strategy where the model generates a sequence of tokens one at a time, at each step conditioned on the previous tokens, the input, and the context of the sentence. This approach allows for a more dynamic and flexible generation of output, as it takes into account a variety of factors when making decisions about the next token. This strategy also helps the model avoid repeating itself, providing more diverse and unique outputs. Furthermore, incremental decoding helps the model to better capture the nuance of the language as it is able to build upon previous decisions and refine its output as it progresses[HM10].

2.3.15 Semantic parsing

Semantic parsing[KDG17] is an area of natural language processing that involves extracting the meaning or intent from text. One type of Semantic Parsing, Text-to-SQL, involves the conversion of natural language problems into SQL query statements. This is a challenging task, one that requires the use of advanced machine learning and natural language processing algorithms. As such, the research conducted in this field seeks to explore the various solutions and practices

that have been employed by researchers in order to effectively tackle this problem. Furthermore, it is also important to note that this problem is not just limited to the conversion of natural language into SQL query statements, as there are other applications of Semantic Parsing that have been explored, such as Natural Language Generation (NLG). Overall, by understanding the various techniques used for Semantic Parsing, we can gain a better understanding of the complexities involved in this task and how best to approach it.

3   Benchmark Dataset

Datasets play a crucial role in developing and evaluating Text-to-SQL models for semantic parsing of natural language phrases. A variety of benchmark datasets are available, each with unique characteristics and features. Examples of early datasets include ATIS[DBB+94], GeoQuery[TM01], and Yelp[YWDD17], which focus on a single topic and database. More recent datasets, such as WikiSQL[ZXS] and Spider[YZY+18], are larger and cover a broader range of domains.

Additionally, new datasets include more advanced queries to assess the generalization capabilities of models. These benchmark datasets provide a standardized testbed for evaluating the performance of Text-to-SQL models and are widely used in the research community. They vary in complexity, size, and annotation, allowing researchers to evaluate models' performance at different levels and under different scenarios. This chapter will review the top benchmark datasets used in the Text-to-SQL Semantic Parsing community and discuss their significance for the research community.

3.1   Single-Domain

3.1.1   ATIS (Air Travel Information System) and GeoQuery

ATIS (Air Travel Information System)[DBB+94] and GeoQuery[TM01] are two datasets that are frequently utilized for semantic parsing, a technique for converting natural language inquiries into a structured meaning representation. The ATIS dataset consists of audio recordings and hand transcripts of individuals using automated travel inquiry systems to search for information regarding flights. It is structured using a relational schema to organize data from the official airline guide, with 25 tables containing information concerning fares, airlines, flights, cities, airports, and ground services. All questions concerning this dataset can be answered using a single relational query. This makes it an ideal choice for training deep learning models, as it is designed for a specific domain and the queries are relatively straightforward.

Furthermore, the questions in the ATIS dataset are mainly limited to select and project queries. On the other hand, GeoQuery is made up of seven tables from the US geography database and 880 natural languages to SQL pairings. It includes geographic and topographical characteristics such as capitals, populations, and landforms. While both datasets are regularly employed to train deep learning models, GeoQuery is more comprehensive and provides a wider range of queries than ATIS. This includes JOIN and nested queries, as well as grouping and order queries, which are absent in the ATIS dataset. As a result, GeoQuery is better equipped to answer more complex queries, making it a better choice for training AI models.

3.1.2   IMDb Dataset

The IMDb dataset is a well-known dataset in the machine learning community. It contains 50,000 reviews from IMDb and has a limit of 30 reviews per movie[MDP+11]. It is noteworthy that the dataset is balanced in terms of positive and negative reviews, which are equally represented. When creating the dataset, reviews with a score of 4 out of 10 were considered negative

and those with a score of 7 out of 10 were considered positive. Neural reviews were excluded to maintain the quality of the dataset. The dataset is divided into training and testing datasets, each with an equal portion. To ensure fairness and accuracy in the results, the dataset creators have taken special care to keep the training and testing datasets balanced.

3.1.3 Advising Dataset

The Advising dataset[FDKZ+18] was created in order to propose improvements in Text-to-SQL systems. The creators of the dataset compare human-generated and automatically generated questions, citing properties of queries that relate to real-world applications. The dataset consists of questions from the University of Michigan students about courses that lead to particularly complex queries. The data is obtained from a fictional student database which includes student profile information such as recommended courses, grades, and previous courses. Moreover, in order to obtain the data for the dataset, academic advising meetings were conducted where students were asked to formulate questions they would ask if they knew the database. After obtaining the questions, the creators of the dataset compared the query results with those from other datasets such as ATIS, GeoQuery, and Scholar. Many of the queries in the Advising dataset were the same as those found in the other datasets.

3.1.4 MAS (Microsoft Academic Search)

MAS, or Microsoft Academic Search[RCM+13], is a database of academic and social networks and a collection of queries. It has a total of 17 tables in its database, as well as 196 natural languages to SQL pairs. MAS can handle join, grouping, and nested queries but does not support ordering queries.

There are a few limitations to be aware of when using natural language queries within MAS. Firstly, all-natural language questions must begin with the phrase "return me" and can not include an interrogative statement or a collection of keywords. Additionally, all queries must follow the proper grammatical conventions.

3.1.5 SEDE

Stack Exchange Data Explorer (SEDE)[HMB21] is from a popular online question-and-answer platform with more than 3 million questions, and it recently released a benchmark dataset of SQL queries containing 29 tables and 211 columns. This dataset comprises real-world questions from the Stack Exchange website, such as published posts, comments, votes, tags, and awards.

Although these datasets contain a variety of real-world challenges, they still need to be more tricky to parse semantically due to the complexity of the questions they contain. After further analysis of the 12,023 questions (clean) asked on the platform, a total of 1,714 have been verified by humans, which makes it an ideal choice for training and validating the model. This benchmark dataset is highly valuable and helpful for research in natural language processing, as it provides an extensive list of real-world challenges that have rarely been seen in other semantic parsing datasets.

3.1.6 SEOSS

SEOSS dataset is a compilation of natural language expressions with seven alternative phrasings, each linked to a single SQL query. In total, 166 questions (expressions) were organized. The natural language expressions were mainly obtained from existing literature and modified to match the data identified in the issue tracking system (ITS) and version control system (VCS) of an existing software project (namely Apache Pig). This data was extracted and saved into an SQLite database by Rath et al. [RM19].

Expressions are labeled into two different tags, development and research. Eighty-one queries with a focus on software needs of stakeholders and developers or from typical use cases' queries of issue tracking systems were labeled as 'development,' and 63 queries containing issue tracking systems information or version control systems were labeled as 'research.' Also, 22 records were generated from the content in questions stakeholders asked within the comment sections of issues of type bug, enhancement/improvement, new feature/feature request, and tasks of 33 open-source Apache projects, which were extracted and stored into databases by Rath and Mäder[RM19].


```
Q: Return the issue ids of issues of type Bug
SQL (Easy):
SELECT issue_id FROM issue
WHERE type = 'Bug'

Q: Return the issue id, type, description of issues that have component "impl"
SQL (Medium):
SELECT T1.issue_id, T1.type, T1.description FROM issue AS T1
JOIN issue_component AS T2 ON T1.issue_id = T2.issue_id
WHERE T2.component = "impl"

Q: In which fix version were most issues fixed
SQL (Hard):
SELECT fix_version FROM issue_fix_version
GROUP BY fix_version
ORDER BY Count(*) DESC LIMIT 1

Q: Return  the maximum number of file paths of modified files which can be
associated with issue ids of issues of type 'Improvement'
SQL (Extra Hard):
SELECT Count(file_path) FROM code_change AS T1
JOIN change_set_link AS T2 ON T1.commit_hash = T2.commit_hash
JOIN issue AS T3 ON T2.issue_id = T3.issue_id WHERE T3.type = 'Improvement'
GROUP BY T3.issue_id
ORDER BY Count(file_path) DESC LIMIT 1
```

Figure 1: Examples of queries with different levels of complexity in SEOSS-Queries [THM22]

In SEOSS-Queries[THM22] research, they experienced RatSQL and SQLNet methods on the SEOSS dataset and released their evaluation steps. In this research, we will use the same dataset to evaluate state-of-the-art models currently available in the literature and used in SPIDER for this dataset.

3.2 Large Scale Cross-Domain

3.2.1 WikiSQL

WikiSQL[HYPS19] consists of 80,654 natural language questions and corresponding SQL queries on 24,241 tables extracted from Wikipedia. Neither the train nor development sets contain the database in the test set. Databases and SQL queries have simplified the dataset's creators' assumptions. This dataset consists only of SQL labels covering a single SELECT column and aggregation and WHERE conditions. Furthermore, all the databases contain only one table.

The datasets in the test set are not present in the train or development sets in the WikiSQL problem definition. Further, the task needs to accept input from several table schemas. The model must therefore be generalized to new databases. However, they used oversimplified assumptions about the SQL queries and databases in order to generate questions and SQL pairings for 24241 databases. They provide WHERE conditions, a single SELECT column, and aggregation in their SQL labels. Additionally, each database only has one table, with no mention of JOIN, GROUP BY, or ORDER BY.

Prior to the release of SPIDER, this dataset was considered to be a benchmark dataset. Using WikiSQL has been the subject of a great deal of research. WikiSQL's "WHERE" clause has been recognized as one of the most challenging clauses to parse semantically, and SQLNet and SyntaxSQL were previous state-of-the-art models.

Table:

Player	Country	Points	Winnings ($)
Steve Stricker	United States	9000	1260000
K.J. Choi	South Korea	5400	756000
Rory Sabbatini	South Africa	3400	4760000
Mark Calcavecchia	United States	2067	289333
Ernie Els	South Africa	2067	289333

Question: What is the points of South Korea player?

SQL: SELECT Points WHERE Country = South Korea

Answer: 5400

Figure 2: Example from WikiSQL dataset[HYPS19]

One example of a state-of-the-art Text-to-SQL solution in the WikiSQL benchmark is the Seq2SQL model, which uses a sequence-to-sequence learning framework to map natural language input to SQL queries. The model uses an attention mechanism to align the input and output sequences and a pointer network to handle SQL queries with complex structural dependencies. We will discuss this model in more detail in the next section.

3.2.2 SPIDER
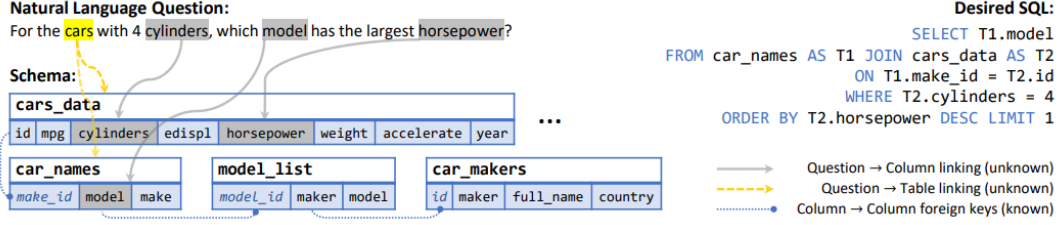
Figure 3: A difficult text-to-SQL task from the Spider dataset.[WSL$^+$]

The SPIDER database contains 10K questions and 5K+ complex SQL queries covering 138 different domains across 200 databases. As opposed to previous datasets (most of which used only one database), this one incorporates multiple datasets. Creating this dataset took 11 Yale University students, 1,000 man-hours in total.

Spider contains queries with a lot of intricate SQL elements. In comparison to the sum of the previous Text-to-SQL datasets, Spider comprises around twice as many nested queries and ten times as many ORDER BY (LIMIT) and GROUP BY (HAVING) components.

Creating this corpus was primarily motivated by the desire to tackle complex queries and generalize across databases without requiring multiple interactions.
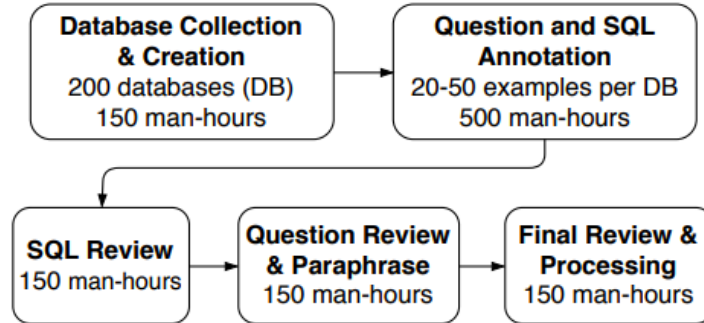
Figure 4: Process of creating SPIDER dataset[YZY$^+$18]

Creating a dataset involves three main aspects: SQL pattern coverage, SQL consistency, and question clarity. Several databases from WikiSQL are included in the dataset. The table is complex as it links several tables with foreign keys. In SPIDER, SQL queries include: SELECT with multiple columns and aggregations, WHERE, GROUP BY, HAVING, ORDER BY, LIMIT, JOIN, INTERSECT, EXCEPT, UNION, NOT IN, OR, AND, EXISTS, LIKE.

The complexity of the dataset increases and the accuracy of solutions drops as the number of foreign keys in the database increases. This is mainly due to the difficulty in selecting the relevant column and table names from a complex database schema. Furthermore, complex database schemas present a major challenge for the model to accurately capture the relationship between different tables which involve foreign keys. SQL queries with a higher number of foreign keys tend to join more tables, suggesting a need for more effective methods to encode the connection between tables with foreign keys.

SQL Hardness Criteria

In order to gain a better understanding of how the model performs on different queries, we have divided SQL queries into four difficulty levels: easy, medium, hard, and extra hard. This classification is based on the number of SQL components, selections, and conditions. Queries that contain multiple SQL keywords (e.g., GROUP BY, ORDER BY, INTERSECT, nested sub-queries, column selections, aggregators) are generally considered more complex. For example, a query is considered hard if it includes more than two SELECT columns, more than two WHERE conditions, and GROUP BY two columns, or contains EXCEPT or nested queries. If it contains even more additions on top of that, it is considered extra hard.

Complex question	What are the name and budget of the departments with average instructor salary greater than the overall average?
Complex SQL	SELECT T2.name, T2.budget FROM instructor as T1 JOIN department as T2 ON T1.department_id = T2.id GROUP BY T1.department_id HAVING avg(T1.salary) > (SELECT avg(salary) FROM instructor)

Figure 5: Example of Question-Query set from SPIDER[YZY$^+$18]

SPIDER's exact matching accuracy6 was 12.4% compared to existing state-of-the-art models. As a result of its low accuracy, SPIDER presents a strong research challenge. Current SPIDER accuracy is above 75.5% with an exact set match without values (refers to values in the WHERE clause) and above 72.6% with values using PICARD4.3.8.

The SPIDER challenge is a research competition dedicated to developing cutting-edge Text-to-SQL and Semantic Parsing solutions. In this challenge, participants strive to develop algorithms that can automatically generate structured SQL queries from natural language input, to improve the performance and accuracy of Text-to-SQL models.

In this challenge, numerous state-of-the-art Text-to-SQL solutions have been proposed, such as the Spider model. This model uses a combination of recurrent and convolutional neural networks to learn the mapping between natural language and SQL queries. This model also has a hierarchical structure, which allows it to process the natural language input more effectively, thereby allowing it to handle complex queries and variations in language with greater precision and accuracy. This model successfully generates accurate and efficient SQL queries from natural language inputs.

One difference between the SPIDER and WikiSQL challenges is the specific dataset that is used for evaluation. The SPIDER challenge uses a dataset of complex SQL queries and natural language questions derived from real-world databases, while the WikiSQL challenge uses a dataset of more straightforward SQL queries and natural language questions derived from Wikipedia articles. This difference in the dataset can affect the performance and accuracy of the models on the different tasks.

Another difference is in the evaluation metrics used. The SPIDER challenge evaluates the models using execution accuracy and natural language understanding metrics, while the WikiSQL challenge evaluates the models using only execution accuracy. This difference in the

evaluation metrics can affect how the models are trained and their performance on the tasks. We will discuss the evaluation metrics used in the SPIDER challenge in more detail in the next section 6.

3.2.3 Multi-Lingual Large Scale Datasets

In this study, we are only focusing on English datasets. Nevertheless, Researchers have produced several large-scale text-to-SQL datasets in diverse languages, such as CSpider[MSZ19], TableQA Sun et al. [SYL20], DuSQL Wang et al. (2020c) [WZW+20] in Chinese, ViText2SQL Tuan Nguyen et al. [TNDN20] in Vietnamese, and PortugueseSpider José and Cozman in Portuguese[JC21]. Human specialists primarily annotate these datasets based on the English Spider dataset, given that human translation is more accurate than machine translation Min et al. (2019a)[MSZ19]. As such, these datasets have the potential to evolve into valuable resources in multi-lingual text-to-SQL studies.

In this chapter, we have reviewed various datasets widely used in the Text-to-SQL Semantic Parsing community. These datasets vary in complexity, size, and annotation, providing a standardized testbed for evaluating the performance of Text-to-SQL models. We have discussed their unique characteristics and features from early datasets such as ATIS and GeoQuery to more recent datasets such as WikiSQL and Spider. The datasets discussed in this chapter are a valuable resource for the research community to evaluate the progress and performance of Text-to-SQL models. The continued development and improvement of these datasets will be necessary for advancing the field of Text-to-SQL Semantic Parsing. The table 1 below provides an overview of the datasets mentioned in this chapter, including the number of queries and questions sorted by year.

Dataset	Year	DBs	Tables	Utterances	Queries	Domain
ATIS	1994	1	32	5280	947	Air Travel Information
GeoQuery	2001	1	6	877	247	US geography database
Academic	2014	1	15	196	185	Microsoft Academic Search
IMDB	2015	1	16	131	89	Internet Movie Database
Scholar	2017	1	7	817	193	Academic Publications
Yelp	2017	1	7	128	110	Yelp Movie Website
WikiSQL	2017	26.521	26,521	80,654	77,840	Wikipedia
Advising	2018	1	10	3,898	208	Student Course Information
Spider	2018	200	1,020	10,181	5,693	138 Different Domains
SEDE	2021	1	29	12,023	11,767	Stack Exchange
SEOSS	2022	1	13	1,162	116	Project ITS and VSC

Table 1: Comparison of datasets (Sort by Year)
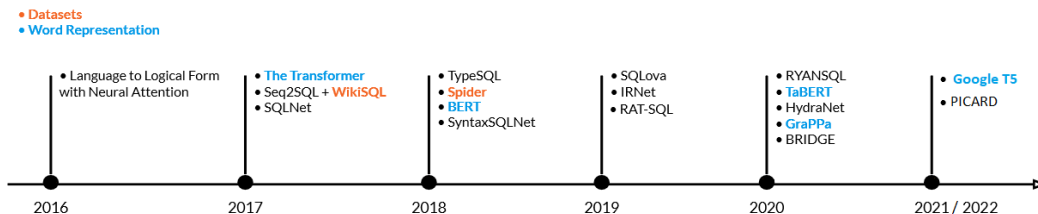
4 State-of-the-art Text-To-SQL Methods

Figure 6: Timeline of the deep learning process for Text-to-SQL.

This section will discuss existing cross-domain state-of-the-art (SOTA), text-to-SQL models, beginning with a broad overview and moving on to individual modules. This will provide a clear picture of the progress made in text-to-SQL research. Experiments have shown that pre-trained embeddings improve models because they construct better schema linking and a more accurate SQL structure.

An efficient text-to-SQL solution requires state-of-the-art natural language processing techniques. As a result of the neural network's ability to handle only numerical inputs and not raw text, word embedding has been used to represent numerical words. Aside from that, in the past few years, language models have become increasingly popular as a solution for increasing performance in natural language processing tasks. Assuming that words have numerical representations that differ from those of other words, word embeddings aim to map each word to a multidimensional vector, incorporating valuable information about the word. In addition to the brute-force creation of one-hot embeddings, researchers have developed highly efficient methods for creating representations that convey a word's meaning and relationships with other words. In most, if not all, Text-to-SQL systems, word embedding techniques such as Word2Vec[Ron14], and WordPiece embeddings[WSC+16] are used.

Recently Language models have been shown to excel at NL tasks as a new type of pre-trained neural network. It is important to note that language models are not a replacement for word embeddings since they are neural networks and need a way to transform words into vectors. Depending on the specific problem they want to solve, researchers can adapt the pre-trained model's inputs and outputs and train it for an additional number of epochs on their dataset. Thus, we can achieve state-of-the-art performance without complex architectures [DCLT18]. Recent neural network architectures, like the Transformer[VSP+17], have been used to achieve such performance by these models, which excel at handling NL and sequences of NL that are characterized by connections between words. Several language models have been used to handle the text-to-SQL task, including BERT [DCLT18]. BERT is a pre-trained language model that has been shown to achieve state-of-the-art performance in a variety of NLP tasks. BERT is a Transformer-based model that uses a bidirectional encoder to learn the representation of a word based on the context in which it appears. BERT has been used in several text-to-SQL models, such as BRIDGE [LSX] and RAT-SQL [WSL+].
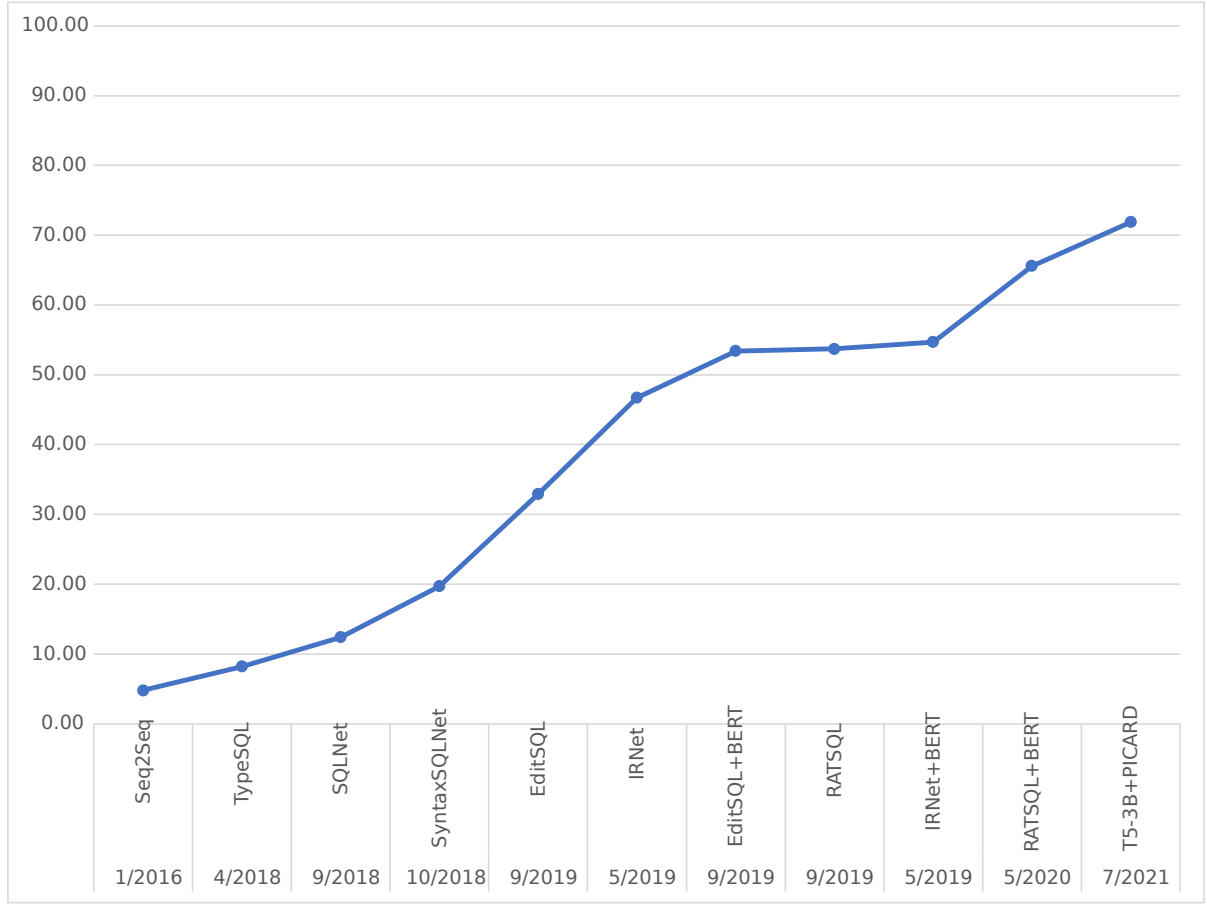
Figure 7: SPIDER benchmark Exact Match Results in 2022
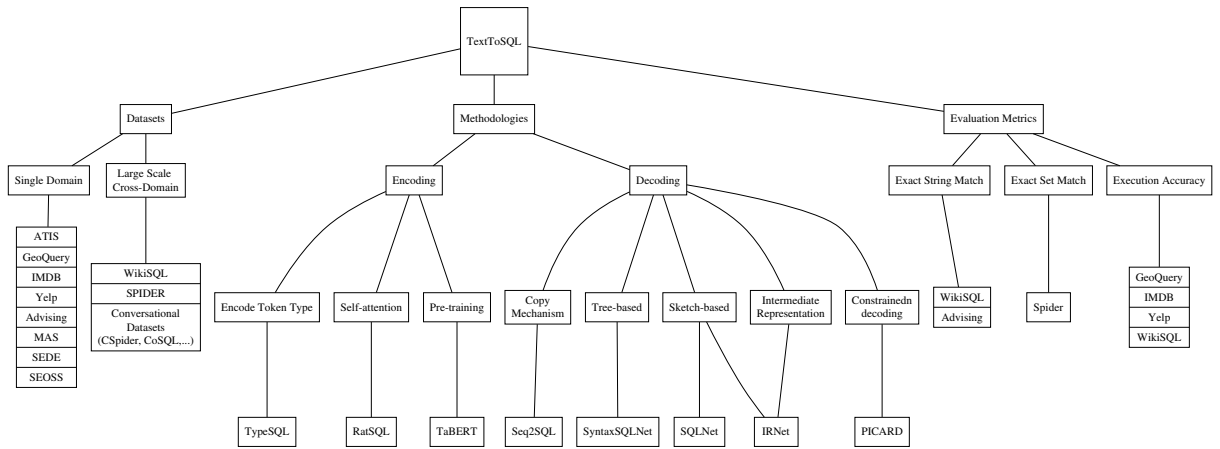
Figure 8: Text-to-SQL state-of-the-art Topology

4.1 Data Augmentation

Data augmentation has emerged as a valuable method to enhance the performance of text-to-SQL models by helping them tackle complex or previously unseen questions (Zhong et al., 2020b; Wang et al., 2021b), achieve state-of-the-art results with less supervised data (Guo et al., 2018), and attain robustness towards different question types (Radhakrishnan et al., 2020).

Common data augmentation strategies include paraphrasing questions and using pre-defined templates to increase data diversity. Iyer et al. (2017) utilized the Paraphrase Database (PPDB) (Ganitkevitch et al., 2013) to generate paraphrased training questions. Additionally, researchers have used neural models to create natural utterances for sampled SQL queries, thus expanding the available data. For instance, Li et al. (2020a) fine-tuned the pre-trained T5 model (Raffel et al., 2019) on WikiSQL. They used the SQL query as input to predict natural utterances and then synthesized SQL queries from WikiSQL tables to generate corresponding natural utterances with the tuned model.

The quality of augmented data is crucial, as low-quality data can negatively impact model performance (Wu et al., 2021). Various techniques have been employed to improve augmented data quality. Zhong et al. (2020b) used an utterance generator to produce natural utterances and a semantic parser to convert these utterances into SQL queries, filtering out low-quality data by retaining only instances with generated SQL queries that matched the sampled ones. Wu et al. (2021) adopted a hierarchical SQL-to-question generation process to obtain high-quality data, decomposing SQL queries into clauses, translating each clause into a sub-question, and combining the sub-questions into a complete question.

To further diversify augmented data, Guo et al. (2018) added a latent variable to their SQL-to-text model to promote question diversity. Radhakrishnan et al. (2020) improved the WikiSQL dataset by simplifying and compressing questions to mimic the colloquial query behavior of end-users. Wang et al. (2021b) used a probabilistic context-free grammar (PCFG) to explicitly model the composition of SQL queries, which fostered the sampling of compositional SQL queries. These data augmentation techniques collectively contribute to the improvement of text-to-SQL models, enabling them to better handle a wider range of questions and adapt to previously unseen data.
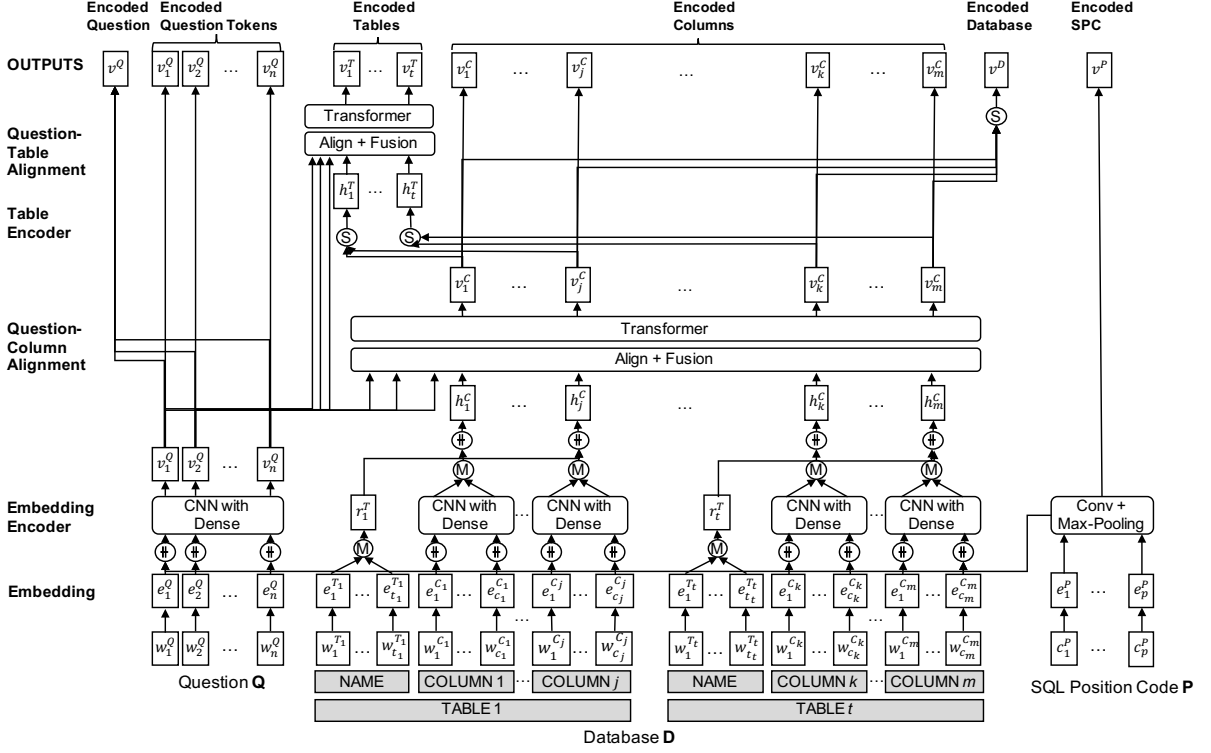
Figure 9: Network architecture of the proposed input encoder. ⊕ represents vector concatenation, Ⓜ represents max-pooling and Ⓢ represents self-attention.[CSKS]

4.2 Encoding

Encoders[KNNV22] are a crucial component in natural language processing tasks and consist of a multi-layered assembly of recurrent elements, such as Long Short-Term Memory (LSTM) units, Gated Recurrent Units (GRUs), or other similar structures. These recurrent elements work in tandem to process an input sequence, with each unit being responsible for handling a single element within the sequence, capturing the pertinent information for that specific element, and subsequently propagating this information forward to the next recurrent unit in the stack.

The primary function of an encoder is to systematically transform textual data into a suitable numerical or vector representation that retains the inherent relationships and dependencies among words, phrases, and sentences[CvMG+14]. This is achieved through a combination of techniques, such as tokenization, embedding, and the use of attention mechanisms, which together facilitate the encoding process.

Tokenization serves to break down the input text into smaller, manageable units, such as words or subwords, while embeddings assign a dense vector representation to each token, thus allowing machines to efficiently process and compare these tokens. Attention mechanisms, on the other hand, enable encoders to weigh the importance of different input elements and selectively focus on the most relevant parts of the input sequence when generating the final encoded representation.

By effectively converting the textual data into a machine-understandable format, encoders

play a pivotal role in empowering machines to recognize intricate patterns, relationships, and contextual cues within the text. Consequently, this ability to accurately discern the context of sentences and phrases forms the foundation for a wide array of natural language processing tasks, ranging from machine translation and sentiment analysis to text summarization and question-answering systems.

Several approaches have been explored to address the challenges of representing the meaning of questions, capturing the structure of database schemas, and establishing connections between database content and questions in the text-to-SQL domain[DCZ22]. These methods play a crucial role in facilitating the understanding of the complex relationships between natural language questions and their corresponding SQL queries.

One of the main challenges in text-to-SQL research is effectively representing the meaning of questions. Various encoding methods have been used to capture the semantics of natural language questions, ranging from traditional word embeddings like Word2Vec and GloVe to more advanced contextualized representations like BERT and its variants. These encoding techniques aim to produce meaningful vector representations of questions that models can use to understand and generate accurate SQL queries.

Another important aspect is representing database schemas, which serve as blueprints for organizing and structuring databases. Researchers have used various strategies to encapsulate database schema information, such as graph-based, tree-structured, and sequence-based encodings. These approaches enable text-to-SQL models to understand the hierarchical relationships and dependencies among various database elements. This allows for more accurate and efficient query generation.

Linking database content to questions is a vital task for text-to-SQL systems[DCZ22]. It involves the identification and mapping of relevant entities and attributes from the question to the database schema. To achieve this, various methods have been employed, including attention mechanisms, entity-linking techniques, and schema-agnostic encodings. These approaches help models identify relevant portions of the database schema and generate SQL queries that accurately reflect the intended meaning of the natural language questions.

Encoding methods and encoders play a crucial role in addressing the challenges of representing question semantics, encapsulating database schema structures, and linking database content to questions in the text-to-SQL domain. The exploration of diverse encoding techniques has led to significant advancements in the development of more accurate and efficient text-to-SQL models, furthering the field's understanding of the complex relationships between natural language questions and SQL queries[DCZ22].

4.2.1 Encode Token Types

Token Type Encoding is an innovative technique introduced in TypeSQL[YLZ+18], which focuses on transforming natural language queries into SQL-like structures by annotating each token with its corresponding SQL token type. This method has demonstrated impressive performance in accurately encoding intricate queries that involve multiple subqueries and join operations, thereby providing a more effective means of bridging the gap between natural language and SQL.

Methods	Adopted by	Applied datasets	Addressed challenges
Encode token type	TypeSQL	WikiSQL	Representing question meaning
Graph-based	GNN Global-GCN IGSQL RAT-SQL LEGSQL SADGA ShawdowGNN S2SQL	Spider Spider Sparc, CoSQL Spider Spider Spider Spider Spider	Representing question and DB schemas in a structured way and Schema linking
Self-attention	X-SQL SQLova RAT-SQL DuoRAT UnifiedSKG	WikiSQL WikiSQL Spider Spider WikiSQL, Spider	Representing question and DB schemas in a structured way and Schema linking
Adapt PLM	X-SQL SQLova Guo HydraNet	WikiSQL WikiSQL WikiSQL WikiSQL	Leveraging external data to represent question and DB schemas
Pre-training	TaBERT GraPPA GAP	Spider Spider Spider	Leveraging external data to represent question and DB schemas

Table 2: Methods used for encoding in text-to-SQL [DCZ22]

$$\textbf{SELECT } \texttt{\$AGG \$SELECT\_COL WHERE \$COND\_COL \$OP \$COND\_VAL}$$
$$(\textbf{AND } \texttt{\$COND\_COL \$OP \$COND\_VAL})\texttt{*}$$

Figure 10: SQL Sketch. The tokens starting with "$" are slots to fill. "*" indicates zero or more **AND** clauses.[YLZ+18]

TypeSQL

In TypeSQL, the core methodology involves a knowledge-based type system and a type-aware neural network model[YLZ+18]. The knowledge-based type system is responsible for identifying and disambiguating the types of entities and values mentioned in the input text. This system uses a combination of pre-defined types, a type dictionary, and a type hierarchy. The type dictionary maps entities and values to their corresponding types, while the type hierarchy defines the relationships between different types, allowing the model to infer implicit relationships between entities in the input text. TypeSQL, similar to SQLNet, adopts a sketch-based approach and treats the task as a slot-filling problem. As shown in Figure 10, the model is tasked with predicting the slots starting with '$'.

The results of TypeSQL demonstrate its effectiveness in generating SQL queries from natural language inputs. When tested on the WikiSQL dataset without considering the content of databases, TypeSQL surpasses the previous best model by 5.5% in execution accuracy. TypeSQL's simpler yet effective approach of encoding column names and logically grouping model components leads to significant improvements in the challenging WHERE clause sub-task. Moreover, the incorporation of word types enables TypeSQL to better encode rare entities and numbers. When given complete access to the database, TypeSQL achieves an execution accuracy of 82.6%, outperforming the previous content-aware system, SQLNet by a remarkable 17.5%.

Despite its success, the Token Type Encoding approach in TypeSQL does have certain limitations, particularly when it comes to handling queries with ambiguous or uncommon syntax patterns. Additionally, this methodology encounters challenges when faced with queries containing nested subqueries or intricate aggregation operations, as it may struggle to accurately capture the relationships and dependencies among the various tokens.

Another drawback of TypeSQL's Token Type Encoding technique is its susceptibility to issues when dealing with out-of-vocabulary words. In these cases, the method may generate incorrect or incomplete SQL encodings, which can adversely impact the overall effectiveness of the system. Nonetheless, the introduction of Token Type Encoding in TypeSQL has laid the foundation for further research and development in the field of neural text-to-SQL generation, providing valuable insights and paving the way for more advanced techniques to address these limitations and improve the overall performance of natural language to SQL conversion systems.

4.2.2 Graph-based Methods

Graph-based methods are an effective approach for encoding the structural information found in database schemas. They have become particularly important as DBs have grown more complex, such as those found in the Spider dataset. These methods involve using graphs to represent

Model	EMA
TypeSQL[YLZ+18]	8.0
EditSQL [TNDN20]	36.4
GNN [BBG19]	40.7
Global-GNN [BBG19]	52.7
RATSQL [WSL+]	69.7
ShadowGNN[CCZ+21]	72.3
LGESQL[CCC+21]	75.1
S²SQL[HGW+22]	76.4

Table 3: The exact match accuracy on the Spider dev set.

the DB schema structure, with nodes representing tables and columns, and edges representing relationships between them, such as primary and foreign key constraints.

Bogin et al.[BBG19] were among the first to propose using graphs in this manner, utilizing Graph Neural Networks (GNNs)[LTBZ17] to encode the graph structure. Also, They employed Graph Convolutional Networks (GCNs) and gated GCNs for capturing DB structures and selecting relevant information for SQL generation. RAT-SQL[WSL+] added more relationships to the DB schema graphs, such as "both columns are from the same table."

In addition to encoding DB schema information, graph-based methods have been used to represent natural language (NL) questions alongside the schema. Various types of graphs have been used to capture semantics in NL and facilitate linking between NL and table schema. For example, LGESQL[CCC+21] utilized line graphs to capture multi-hop semantics using meta-paths, while SADGA[CYXH] adopted a graph structure to provide a unified encoding for both natural language utterances and DB schemas, assisting in question-schema linking.

In order to improve the generalization of graph-based methods for unseen domains, ShadowGNN [CCZ+21] takes a unique approach. It disregards the names of tables or columns in the database and instead employs abstract schemas within the graph projection neural network. This results in delexicalized representations of questions and DB schemas, allowing the model to better handle new or previously unseen domains. S2SQL[HGW+22] integrated syntax dependency among question tokens into the graph to enhance model performance even more.

In summary, graph-based methods have proven to be valuable for encoding structural information in DB schemas, bridging the gap between natural language questions and schema elements, and enhancing the performance of models in context-dependent text-to-SQL tasks. Upon examining the results from the Spider benchmark $Table3$, it is evident that there has been a significant overall performance improvement when comparing graph-based methods.

4.2.3 Self-attention

Self-attention is a fundamental component in natural language processing (NLP) models, particularly those based on the Transformer architecture. It serves as the primary building block of the transformer structure, as mentioned in the works of X-SQL[HMCC19], SQLova[HYPS19], and UnifiedSKG[XWS+22]. These models employ the original self-attention mechanism by default.

The self-attention mechanism allows the model to weigh and aggregate different words or

tokens in a sequence based on their relative importance[VSP+17]. In essence, it helps the model to focus on the most relevant parts of a given input while processing it. This is accomplished by computing attention scores between each pair of tokens in the input, which are then used to produce a weighted sum of the input tokens. The mechanism is particularly effective in handling long-range dependencies within the text.

However, the original self-attention mechanism can be modified to cater to specific tasks or address particular challenges. One such modification is relation-aware self-attention, employed by RAT-SQL[WSL+] and DuoRAT[SLB+21]. This variation of self-attention is designed to take advantage of the relationships between tables and columns when working with structured data.

Relation-aware self-attention extends the original self-attention by incorporating information about the structure and relations in the input data. This additional information is used to adjust the attention scores, allowing the model to focus on the most relevant relationships between different elements in the input. As a result, models equipped with relation-aware self-attention can better handle tasks involving structured data, such as SQL query generation or table-based reasoning.

4.2.4 Adapt PLM

Adapt PLM (Pre-trained Language Models) methods aim to utilize the knowledge encapsulated in pre-trained language models, such as BERT[DCLT18], to improve their performance on text-to-SQL tasks. These methods modify or extend the original PLMs to better align with the specific requirements of the task.

One common approach is to encode both the natural language questions and the database schemas using PLMs. For instance, SQLova[HYPS19] and RYANSQL[CSKS21] concatenate the question words and schema words as input to the BERT encoder. This approach allows the model to learn representations that capture the relationships between questions and the underlying schema.

Some methods go a step further by adjusting the embeddings produced by the PLMs. X-SQL[HMCC19] proposes the replacement of the segment embeddings from the pre-trained encoder with column-type embeddings for the WikiSQL dataset. Guo and Gao [GG20] introduce an approach that encodes additional feature vectors for matching between question tokens and table cells, as well as column names. These feature vectors are then concatenated with the BERT embeddings of questions and DB schemas.

HydraNet[LCH+20] uses BERT to encode the question and individual columns, an approach that is more aligned with the tasks BERT is pre-trained on. After obtaining BERT representations for all columns, the model selects the top-ranked columns for SQL prediction

Examining the WikiSQL benchmark results in Table 4, we can observe a significant overall performance improvement when employing pre-trained language models (PLMs) compared to previous methods. This enhancement can be attributed to the ability of PLMs, such as BERT, to capture complex linguistic patterns and relationships within the input data. By leveraging the knowledge encapsulated in these models and adapting them to the text-to-SQL task, researchers have been able to achieve better alignment with the specific requirements of the problem domain. As a result, PLM-based approaches have demonstrated superior performance in generat-

Model	Execution Accuracy
Seq2SQL[ZXS]	60.8
TypeSQL[YLZ+18]	74.5
SQLova[HYPS19]	90.2
Guo[GG20]	91.1
HydraNet[LCH+20]	92.4

Table 4: The execution accuracy on the WikiSQL dev set.

ing accurate SQL queries from natural language questions, surpassing traditional methods and showcasing the potential of PLMs in addressing complex language understanding tasks.

4.2.5 Pre-training

Pre-training methods are a crucial part of training transformer-based models for text-to-SQL tasks. These methods aim to align the models with the required tasks by using various objectives and pre-training data. Some popular pre-training methods include TaBERT[YNYR], Grappa[YWL+20], and GAP[SNW+20], which utilize different strategies to achieve their goals.

TaBERT [YNYR] focuses on tabular data for pre-training. It employs two main objectives: masked column prediction and cell value recovery. These objectives help the model better understand the structure and semantics of tables, allowing it to generate SQL queries more effectively.

Grappa [YWL+20] generates synthetic question-SQL pairs over tables for pre-training. It uses BERT and relies on two primary objectives: masked language modeling (MLM) and predicting column presence and SQL operations. MLM helps the model understand the language structure in the context of SQL queries while predicting column presence and SQL operations allowing it to learn how columns and operations are related to the questions.

GAP [SNW+20] pre-trains BART [LLG+20] on both synthesized text-to-SQL and tabular data. It utilizes four objectives: MLM, column prediction, column recovery, and SQL generation. The combination of these objectives enables the model to understand language patterns and table structures while also learning to generate SQL queries that match the given text.

In summary, various pre-training methods have been proposed to better align transformer-based models with text-to-SQL tasks. A glance at the SPIDER Benchmark results $Table5$ highlights the performance improvement when incorporating advanced pre-training methods with RATSQL. The integration of GAP, which leverages multiple objectives, leads to a more effective understanding of text-to-SQL tasks. A further enhancement is observed when combining RATSQL with GraPPa, a method that uses synthesized question-SQL pairs. These findings emphasize the value of using cutting-edge pre-training techniques to boost transformer-based models like RATSQL in text-to-SQL tasks.

Model	EMA Dev.
RATSQL	62.7
RATSQL + GAP	71.8
RATSQL + GraPPa	73.4

Table 5: The exact match accuracy on the Spider dev set.

4.3   Decoding

Decoders [CvMG⁺14] form an integral part of sequence-to-sequence models in natural language processing tasks, and they are constructed as a multi-layered architecture of recurrent elements, such as Long Short-Term Memory (LSTM) units, Gated Recurrent Units (GRUs), or other analogous structures. The primary responsibility of a decoder is to generate an output sequence by predicting an output, denoted as y, for each time step. This output sequence can be a series of words, phrases, or even entire sentences, depending on the specific problem being addressed.

At each time step, the current recurrent unit within the decoder receives a hidden state from the preceding recurrent unit. This hidden state encapsulates the information gathered up to that point and serves as a vital input for the current recurrent unit to make an informed prediction. Moreover, decoders can also incorporate attention mechanisms to help focus on the most relevant parts of the input sequence when generating the output. This is particularly useful in tasks that require the decoder to selectively attend to different input elements during the decoding process.

Decoders are commonly employed in a wide range of natural language processing applications [KNNV22], including but not limited to, machine translation, text summarization, question-answering systems, and dialogue generation. In question-answering tasks, for instance, the output sequence generated by the decoder is often a collection of words.

Numerous approaches have been suggested to enhance the decoding process for more precise and efficient SQL generation, ultimately bridging the divide between natural language and SQL query formulation. As illustrated in the table below, we have classified these techniques into five primary categories, along with additional methodologies[DCZ22].

4.3.1   Tree-based

In the realm of text-to-SQL research, tree-based decoders have emerged as a popular approach for generating logical forms or abstract syntax trees (ASTs) from input text. Two key papers in this area are Dong and Lapata (2016) [DL16] with their Seq2Tree model and Yin and Neubig (2017) [YN17] with their Seq2AST model. While these works do not specifically focus on text-to-SQL datasets, they inspire the development of tree-based decoding methods within the text-to-SQL context, such as SyntaxSQLNet by Yu et al. (2018b) [YYY⁺18].

The Seq2Tree model by Dong and Lapata (2016) [DL16] creates logical forms through a top-down approach, where components of the sub-tree are generated based on their parent nodes, independently from the input question. This model learns the syntax of the logical forms implicitly, without any explicit guidance. On the other hand, the Seq2AST model by Yin and Neubig (2017) [YN17] explicitly integrates syntax into the generation process through the use of an AST. This approach decodes the target programming language by constructing an AST that adheres to the language's syntax rules.

Taking inspiration from these approaches, SyntaxSQLNet by Yu et al. (2018b) [YYY⁺18] adapts the tree-based decoding method to SQL syntax. This model employs a recursive structure that calls various modules to predict different SQL components, ultimately generating a valid SQL query. The model's tree-based decoding technique is tailored to the SQL language and

Methods	Adopted by	Applied datasets	Addressed challenges
Tree-based	Seq2Tree Seq2AST SyntaxSQLNet	- - Spider	Hierarchical decoding
Sketch-based	SQLNet Coarse2Fine IRNet RYANSQL	WikiSQL WikiSQL Spider Spider	Hierarchical decoding
Bottom-up	SmBop	Spider	Hierarchical decoding
Self-Attention	Seq2Tree Seq2SQL	- WikiSQL	Synthesizing information
Bi-attention	BiSQL	Spider	Synthesizing information
Relation-aware Self-attention	DuoRAT	Spider	Synthesizing information
Copy Mechanism	Seq2AST Seq2SQL SeqGenSQL	- WikiSQL WikiSQL	Synthesizing information
Intermediate Representation	IncSQL IRNet ValueNet	WikiSQL WikiSQL Spider	Bridging the gap between natural language and SQL query
Constrained decoding	PICARD	Spider	Fine-grained decoding

Table 6: Methods used for decoding in text-to-SQL [DCZ22]

facilitates a more structured prediction process.

In summary, tree-based decoders in text-to-SQL research offer a structured way to generate logical forms or ASTs, making use of either implicit or explicit syntax learning. By adapting these methods to the specific requirements of SQL syntax, researchers can develop more effective models for translating natural language questions into SQL queries.

4.3.2 Sketch-based

Sketch-based decoders have gained attention in text-to-SQL research as an approach that simplifies the generation of SQL queries by leveraging predefined query structures, or "sketches." These sketches follow SQL grammar and allow the model to focus on filling in the slots rather than predicting the output grammar and content simultaneously.

SQLNet by Xu et al. (2017) [XLS] is an example of a sketch-based model that aligns with SQL grammar. The sketch captures dependencies between predictions, which means that the prediction for each slot is conditioned only on the slots it depends on. This approach effectively avoids issues arising from equivalent serializations of the same SQL query.

Dong and Lapata (2018) [DL18] further refine the sketch-based approach by decomposing the decoding process into two stages. The first decoder predicts a rough sketch, while the second decoder fills in the low-level details based on the input question and the sketch. This coarse-to-fine decoding has been adopted in other works, such as IRNet by Guo et al. (2019) [GZG+19].

To handle complex SQL queries with nested structures, RYANSQL by Choi et al. (2021) [CSKS21] introduces a recursive method for generating SELECT statements. This model employs sketch-based slot filling for each of the SELECT statements, enabling the generation of more intricate queries.

In summary, sketch-based decoders simplify the text-to-SQL generation process by providing predefined query structures that follow SQL grammar. This approach enables models to focus on filling in content slots, captures dependencies between predictions, and allows for the handling of complex queries with nested structures. By decomposing the decoding process into multiple stages, sketch-based decoders can efficiently translate natural language questions into accurate SQL queries.

4.3.3 Bottom-up

Bottom-up decoders offer an alternative approach to tree-based and sketch-based decoding mechanisms, which are typically top-down in nature. One example of a bottom-up decoder is the method employed by Rubin and Berant (2021) [RB21].

In a bottom-up decoding mechanism, the model starts with a set of K trees, each of height t. The decoder then scores trees of height t+1, which are constructed based on SQL grammar from the current set of trees in the beam. The K highest-scoring trees are retained, and a new representation of these trees is generated and placed in the new beam. This process iteratively builds the trees from the bottom up until a complete SQL query is formed.

This bottom-up approach contrasts with top-down methods, where trees or sketches are generated from the root node or a coarse representation, and then progressively filled in or expanded by adding more details or subtrees. The bottom-up method focuses on constructing trees by iteratively expanding them based on the current best candidates, effectively narrowing down the search space and improving the efficiency of the decoding process.

In summary, bottom-up decoders present an alternative to top-down methods for generating SQL queries from natural language input. By iteratively expanding and scoring trees based on the current beam, these decoders can efficiently generate accurate SQL queries while maintaining a manageable search space.

4.3.4 Attention Mechanism

Attention mechanism decoders play a critical role in integrating encoder-side information during the decoding process. By computing attention scores and multiplying them with hidden vectors from the encoder, a context vector is generated, which is then used to produce an output token.

Various attention structures have been employed to enhance the decoder's performance and effectively propagate the information encoded from questions and database schemas. One such example is SQLNet (Xu et al., 2017) [XLS], which introduces the concept of column attention. This technique involves using hidden states from columns and multiplying them by embeddings for the question to calculate attention scores for a given column. The attention scores are then used to help the model focus on relevant columns when generating the SQL query.

Another approach, proposed by Guo and Gao (2018) [GG20], incorporates bi-attention over a question and column names for SQL component selection. This method enables the model to simultaneously attend to both the question and column names, which can improve the model's ability to identify and select relevant SQL components.

Wang et al. (2019) [WTL19] adopt a structured attention mechanism [KDHR17] that computes marginal probabilities to fill in the slots of their generated abstract SQL queries. This approach allows the model to better capture the structure of SQL queries and enhances the overall generation process.

DuoRAT [SLB+21] implements a relation-aware self-attention mechanism in both its encoder and decoder components. This attention mechanism accounts for relationships between different elements within the input data, thus improving the model's ability to comprehend and generate accurate SQL queries.

Other works, such as those by Scholak et al. PICARD (2021b) [SSB21] and Unified-SKG by Xie et al. (2022) [XWS+22], use sequence-to-sequence transformer-based models or decoder-only transformer-based models that incorporate the self-attention mechanism by default. The self-attention mechanism allows the model to weigh the significance of each input token concerning other tokens in the sequence, which can enhance the quality and coherence of the generated output.

In summary, attention mechanism decoders have been an essential aspect of Text-to-SQL research, with various structures designed to improve the propagation of information and the generation of accurate SQL queries. By continuously refining and adapting these attention mechanisms, researchers aim to further enhance the performance of Text-to-SQL models.

4.3.5 Copy Mechanism

The Copy mechanism is a vital component in various Text-to-SQL models, as it facilitates the direct copying of specific words or tokens from the input sequence to the generated output. Several research papers have implemented this mechanism to improve the performance of their models.

Seq2AST by Yin and Neubig (2017) [YN17] and Seq2SQL by Zhong et al. (2017) [ZXS] both employ the pointer network, introduced by Vinyals et al. (2015) [VFJ17], to calculate the probability of copying words from the input sequence. The pointer network is a type of neural network that can learn to point to specific positions in the input data, allowing the model to copy tokens directly from the input when generating output sequences.

Wang et al. (2018a) [WBS17] take a different approach to the copy mechanism by using types, such as columns, SQL operators, and constants from questions, to explicitly restrict the locations in the query that can be copied from. This method helps the model focus on copying only relevant tokens to generate coherent and accurate SQL queries. Additionally, they develop a new training objective that encourages the model to only copy from the first occurrence of a token in the input sequence, which can prevent potential redundancies in the generated output.

Furthermore, the copy mechanism has been adopted in the context-dependent text-to-SQL task, as demonstrated by Wang et al. (2020b) [WLC20]. In this scenario, the copy mechanism is particularly beneficial for models that need to handle complex input data, such as multiple

questions or queries, and generate output sequences that accurately reflect the context.

In summary, the copy mechanism plays a crucial role in various Text-to-SQL models by allowing them to copy specific tokens from the input sequence directly, enhancing the accuracy and coherence of the generated SQL queries. By adopting different techniques and refining the copy mechanism, researchers continue to improve the performance of their models in the Text-to-SQL domain.

4.3.6   Intermediate Representations

Intermediate representations (IRs) are employed in Text-to-SQL research to bridge the gap between natural language and SQL queries. By using IRs, researchers can simplify and abstract SQL queries, making it easier for models to learn and generate an accurate output.

IncSQL by Shi et al. (2018)[STC⁺18] is one such approach that defines actions for different SQL components, allowing the decoder to decode these actions instead of raw SQL queries. This method reduces the complexity of the decoding process and can improve the overall performance of the model.

IRNet by Guo et al. (2019) [GZG⁺19] introduces SemQL, an intermediate representation for SQL queries designed to cover most of the challenging Spider benchmark. SemQL simplifies SQL queries by removing the JOIN ON, FROM, and GROUP BY clauses and merging the HAVING and WHERE clauses. ValueNet by Brunner and Stockinger (2021) [BS21] builds upon SemQL by introducing SemQL 2.0, which extends the original representation to include value representation. Additionally, NatSQL by Gan et al. (2021c) [GCX⁺21] modifies SemQL by removing set operators, such as INTERSECT, which combine the results of two or more SELECT statements.

Suhr et al. (2020) [LB99] implement SemQL as a mapping from SQL to a representation with an under-specified FROM clause, which they call SQLUF. Rubin and Berant (2021) employ a relational algebra augmented with SQL operators as intermediate representations, offering another approach to simplifying SQL queries.

However, one of the main challenges with intermediate representations is that they are typically designed for specific datasets and cannot be easily adapted to others. To address this issue, Herzig et al. (2021) [HSC⁺21] propose a more generalized intermediate representation by omitting tokens in the SQL query that do not align with any phrase in the natural language utterance.

The success of intermediate representations in Text-to-SQL tasks has inspired researchers to explore their use in other executable language domains, such as SPARQL for database systems. Works by Saparina and Osokin (2021) [SO21] investigate the potential of intermediate representations for SPARQL queries.

In conclusion, intermediate representations play an essential role in Text-to-SQL research by simplifying and abstracting SQL queries, making it easier for models to learn and generate an accurate output. The exploration of various intermediate representation techniques continues to improve the performance of Text-to-SQL models and inspire advancements in other related domains.

4.3.7 Constrained decoding

Constrained decoding methods are employed in natural language processing tasks, such as text-to-SQL, to improve the quality of generated outputs by imposing certain constraints or utilizing auxiliary models during the decoding process. These methods aim to prevent the generation of invalid tokens, exclude non-executable partial SQL queries, or facilitate the generation of complete SQL queries.

PICARD (Scholak et al., 2021b) [SSB21] is an example of a method that sets constraints on the decoder to avoid generating invalid tokens. Other methods, such as those proposed by Wang et al. (2018b) [WTB+18] and Hwang et al. (2019) [HYPS19], adopt an execution-guided decoding mechanism that eliminates non-executable partial SQL queries from the output candidates.

Some approaches, like Global-GNN (Bogin et al., 2019b) [BGB19] and Kelkar et al. (2020) [KRB+20], use separately trained discriminative models to rerank the top-K SQL queries in the decoder's output beam. This technique allows the model to reason about complete SQL queries rather than considering each word and database schema in isolation.

Chen et al. (2020b) [CSLJ20] employ a gating mechanism to select between the output sequence encoded for the question and the output sequence from the previous decoding steps at each step for SQL generation. This approach helps in generating more accurate and coherent SQL queries.

Müller and Vlachos (2019) [MV19] draw inspiration from machine translation and apply byte-pair encoding (BPE) (Sennrich et al., 2016) [SHB16] to compress SQL queries into shorter sequences, guided by abstract syntax trees (AST). This technique reduces the difficulties in SQL generation, leading to improved performance in text-to-SQL tasks.

4.3.8 T5 + PICARD

After the release of Google T5, researchers have been using it to improve the accuracy of text-to-SQL models instead of BERT. New solutions have been released, such as the PICARD with T5-3B model, that significantly improved the SPIDER challenge's accuracy and are motivating researchers to use T5 in their work with innovative approaches since 2021.

T5

In Transfer Learning, we start by training our model in an unsupervised fashion on unlabeled data. Then fine-tuning it on a labeled dataset some tasks that we care about, which we call the downstream tasks. For instance, in our unsupervised free training task, we take some text, drop out some of the words, and train the model to predict the missing words. Next, we will fine-tune it on a supervised task like sentiment analysis classifying movie reviews as a given label. This way of training has become an incredible recipe for natural language processing.

T5 Model implemented by Raffel et al. [RSR+19] uses the BERT encoder-decoder architecture proposed by Vaswani et al. (2017)devlin-etal-2019-bert and they showed in their studies that it will outperform decoder-only language models. Originally T5 was introduced with five pre-trained models — Small (60 million parameters), Base(220 million parameters), Large(770 million parameters), 3B(3 billion parameters), and 11B(11 billion parameters)[RSR+19].

Model	Parameters	# layers	d_{ff}
Small	60M	6	2048
Base	220M	12	3072
Large	770M	24	4096
3B	3B	24	16384
11B	11B	24	65536

Figure 11: T5 models with their Nr. of parameters, layer and feed-forward parameters[RSR+19]

To pre-train the T5 model, we start with clean text and drop some words to corrupt the text. Each dropped-out span will be replaced with a unique sentinel token, so if multiple words in a row get dropped out, they will be replaced with a single token. The words are dropped out independently uniformly at random so for an inviting get replaced by a single Sentinel token. Then the model is trained to output Sentinel tokens to delineate the dropped-out text corresponding to the text that was dropped out in the input and then each span of dropped-out text.

This method is pretty similar to the span BERT objective. It tried to come up with an objective that was not too different from standard practice.

Google T5's basic idea is that it models every NLP problem and every text problem as a text-to-text task that takes the text as input and produces text as output.

So fundamentally, it is in a sequence-to-sequence framework; hence, T5 is perfectly suitable for transfer learning machine translation. T5 can handle various tasks, and it can be fine-tuned for different NLP tasks, such as summarization, COLA (Corpus on linguistic acceptability),
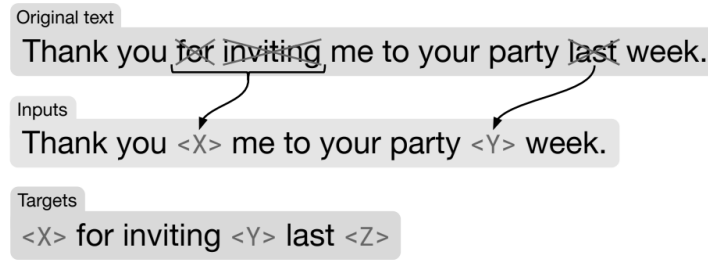
Figure 12: Pre-training by Replace Corrupted Spans [RSR+19]

classification, multiple text translation, also regression problems like STSB that predict how similar two sentences are. And in our case Text-to-SQL.
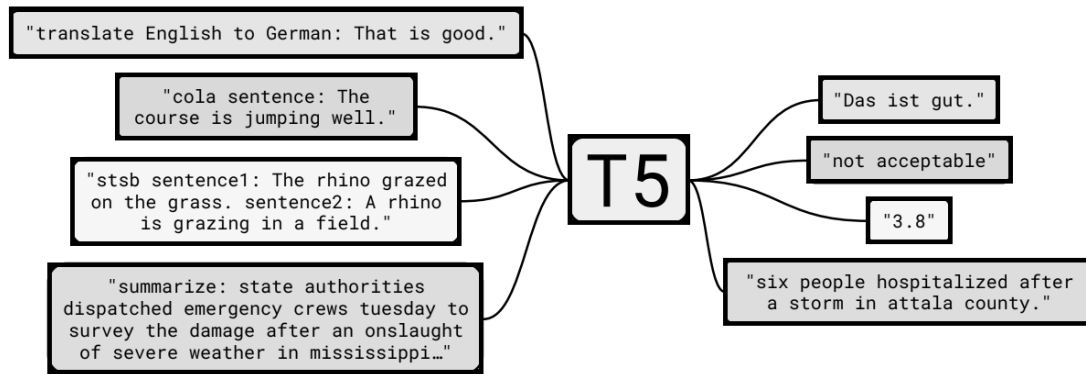
Figure 13: Each task uses text as input in the model and generates target text. In this way, the same model, loss function, and hyper-parameters are used across various diverse tasks, including translation. [RSR+19]

Further, because the same model is used for many tasks, the model understands which tasks to perform by prepending a prefix that will also be text. Therefore, By the end of fine-tuning, T5 will have "n" different models where "n" is the number of tasks. It starts with the same base pre-trained model, and then it is fine-tuned on task A, and then separately, on task B and task C. In our work, we are essentially adding another task to the T5 to handle SQL translation.

C4 (Colossal Clean Crawled Corpus)

The T5 model is pre-trained on C4 Dataset[RSR+19], so its results are quite realistic. The C4 is an unlabeled dataset gathered and filtered from Common Crawl Dataset, a non-commercial crawler that saves snapshots of the web every month. And web content is dumped out on the order of 20 terabytes.

The cleaning process included deduplication, discarding incomplete sentences, and removing offensive or noisy content. The filtering led to more reliable results on downstream tasks, and the added size let the model size grow without over-fitting when pre-training. C4 is about 750 gigabytes of clean-ish data and is accessible in Tensorflow Datasets Library.

Beam Search

Before understanding the PICARD, let us first understand the concept of Beam Search:
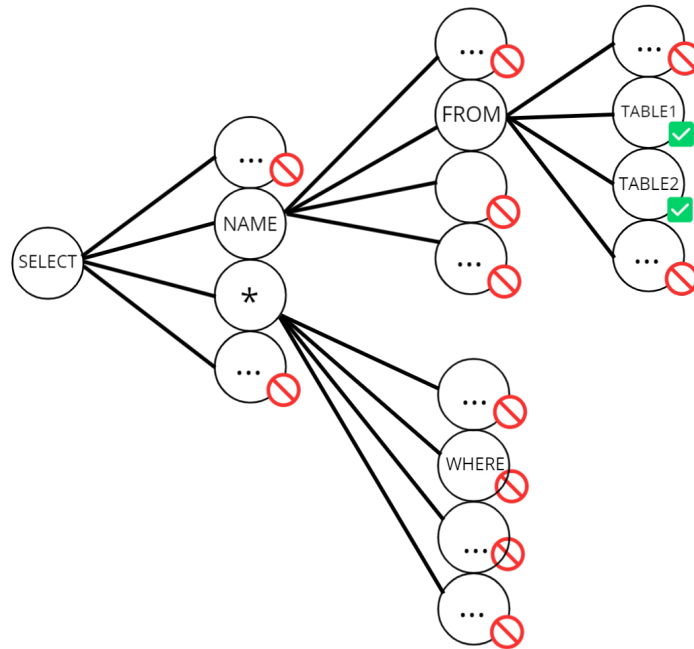
Figure 14: 4-Beam Search

Beam search is a widely used search algorithm in natural language processing and machine learning. It is beneficial in sequence-to-sequence (seq2seq) models, which generate output sequences based on input sequences. Beam search is used to find the most likely sequence of output words given an input sequence.

The basic idea behind beam search is to maintain a set of the most likely sequences at each step of the decoding process. This set of sequences, called the "beam," is initially set to the starting point of the decoding process, and at each step, new sequences are generated by considering all the following possible words. The new sequences are then ranked based on their likelihood, and the highest-ranking sequences are added to the beam. The process is repeated until a stopping criterion is met [MLGftADNI19]. Beam search is handy in seq2seq models because it allows the model to generate multiple output sequences rather than just a single sequence. This is important because, in many cases, there may be multiple valid outputs for a given input sequence. By generating multiple outputs, beam search allows the model to explore the space of possible outputs and find the most likely sequences.

One of the critical advantages of beam search is that it is computationally efficient. Because it only considers a small number of sequences at each step, it can quickly find the most likely sequences without exploring the entire space of possible outputs. This makes it well-suited for use in applications with limited computational resources, such as on mobile devices or in real-time systems. Another advantage of beam search is that it can be used with other techniques, such as attention mechanisms, to improve the performance of seq2seq models. Attention mechanisms allow the model to focus on specific parts of the input sequence when generating the output, which can help to improve the quality of the generated sequences.

In conclusion, Beam Search is a robust algorithm widely used in natural language processing

and machine learning, particularly in the context of sequence-to-sequence (seq2seq) models. It allows the model to generate multiple output sequences rather than just a single sequence and is computationally efficient, making it well-suited for use in applications where computational resources are limited. Additionally, it can be combined with other techniques, such as attention mechanisms, to improve the performance of seq2seq models.

PICARD

PICARD[SSB21], short for "Parsing Incrementally for Constrained Auto-Regressive Decoding," is a method that can be used in conjunction with any language model decoder or vocabulary that utilizes auto-regressive language modeling.

PICARD is a technique that utilizes standard beam search, commonly used in natural language processing, to generate executable code by ensuring the output of the language model is both syntactically and semantically correct. It works by expanding a beam of hypotheses step by step and discarding any tokens that are not valid at each decoding step. This method can be applied to any language model that generates a sequence of tokens, including character, subword, and word-level models, without requiring unique recovery methods.

It effectively improves the performance of existing models and achieves state-of-the-art performance on tasks such as text-to-SQL translation. Warps model prediction scores and integrates trivially with existing greedy and beam search algorithms used in auto-regressive decoding from language models.

At each generation step, Picard first restricts prediction to the top-k highest probability tokens and then assigns a score of negative infinity to those that fail Picard's numerous checks.

PICARD has four modes that control the level of comprehensiveness of its checking process: off, lexing, parsing without guards, and parsing with guards, with the latter being the most comprehensive. In lexing mode, PICARD checks if the current token is a valid keyword or identifier. In parsing guard mode, it checks if the current token is a valid keyword or identifier, a valid SQL keyword, and a valid SQL identifier.

Picard can detect spelling errors in keywords or reject table and column names that are invalid for the given SQL schema. "Out-of-distribution compositional generalization and natural language variation" refers to the ability of a natural language processing (NLP) system to handle novel combinations of words and phrases that it has not seen before while also being able to handle variations in language usage. Compositional generalization refers to the ability of an NLP system to understand and generate novel combinations of words and phrases by using its knowledge of the meanings and relationships of individual words and phrases. This is an essential aspect of NLP because it allows the system to understand and generate language flexibly and adaptively.

The concept of natural language variation refers to the multiple ways people can express the same ideas or concepts using natural language. This can include variations in dialect, style, or tone, which can make it difficult for NLP systems to understand and generate language accurately.

Together, out-of-distribution compositional generalization and natural language variation represent fundamental challenges in the field of NLP. They require NLP systems to handle a

wide range of language input and output in order to be effective.

PICARD can be applied as an optional feature during inference but is not necessarily included in pre-training or fine-tuning, and for text-to-SQL translation, it works directly on the output of the language model. PICARD has been shown to have state-of-the-art performance on complex Spider text-to-SQL translation tasks, achieving an accuracy of 75.1%.

Picard warps model prediction scores and integrates trivially with existing greedy and beam search algorithms. In addition to the token ids of the current hypothesis, the model's language modeling head also predicts the log-softmax scores for each vocabulary token. Additionally, Picard has access to SQL schema information, including table and column names and which column resides in which table.

Motivated by the success of Shaw et al. [SCPT21], who demonstrated that a pre-trained T5-Base or T5-3B model could effectively learn the text-to-SQL task, generalize to never-before-seen databases, and even rival the state-of-the-art methods of Choi et al.[CSKS21] without any modifications to the model itself, the researchers opted to use T5 as the baseline for all their experiments. The results from Shaw et al.[SCPT21] suggest that T5-based models had the potential to improve the field of natural language processing significantly. Therefore, the researchers sought to take advantage of the capabilities of T5 in order to gain new insights into how natural language can be effectively utilized to solve complex tasks.

5 Learning Techniques

The advancement of Text-to-SQL research has been driven by various learning techniques that address specific challenges in the field. This chapter provides a comprehensive overview of these learning techniques, focusing on both fully supervised and weakly supervised methods.

5.1 Fully Supervised Learning Techniques

Fully supervised learning techniques rely on labeled data to train models. We discuss several state-of-the-art methods that have been proposed to improve Text-to-SQL generation.

5.1.1 Active Learning (Ni et al., 2020)

Active learning aims to minimize the amount of labeled data required for training by selectively choosing the most informative examples. Ni et al. (2020) proposed an active learning framework that leverages uncertainty estimation to identify samples that would benefit most from human annotations.

5.1.2 Interactive/Imitation Learning (Yao et al., 2019)

Interactive or imitation learning focuses on learning from demonstrations, where a model attempts to mimic expert behavior. Yao et al. (2019) introduced an interactive learning approach that incorporates user feedback to refine the model's understanding of complex SQL queries.

5.1.3 Meta-learning (Huang et al., 2018)

Meta-learning involves training models to learn how to learn efficiently. Huang et al. (2018) proposed a meta-learning technique for Text-to-SQL tasks, which allows the model to adapt quickly to new tasks or domains with limited labeled data.

5.1.4 Multi-task Learning (Chang et al., 2020)

Multi-task learning involves training a single model on multiple related tasks simultaneously. Chang et al. (2020) explored a multi-task learning framework for Text-to-SQL generation, demonstrating that sharing information across tasks can lead to improved performance.

5.2 Weakly Supervised Learning Techniques

Weakly supervised learning techniques use weak or noisy labels for training, often reducing the need for extensive human annotation.

5.2.1 Reinforcement Learning (Zhong et al., 2017)

Reinforcement learning focuses on learning by trial and error, with models receiving feedback through rewards or penalties. Zhong et al. (2017) applied reinforcement learning to Text-to-SQL generation, showing that such an approach can effectively learn from weak supervision.

5.2.2 Meta-learning and Bayesian Optimization (Agarwal et al., 2019)

Agarwal et al. (2019) combined meta-learning and Bayesian optimization for weakly supervised Text-to-SQL tasks. This approach allows models to adapt to new tasks more efficiently while leveraging limited supervision, ultimately reducing the need for large amounts of labeled data.

6 Evaluation Metrics

The F1 score is a metric used to evaluate the performance of many machine learning tasks. It is computed as the harmonic mean of precision and recall, where precision is the ratio of true positive (TP) predictions to the total positive predictions, and recall is the ratio of true positive predictions to the total actual positive values. The F1 score is between 0 and 1, with higher values representing better performance. Precision indicates the accuracy of the classifier's prediction of the positive class. It is calculated by taking the number of correct positive predictions (True Positive) and dividing it by the total number of positive predictions (True Positive and False Positive). A higher precision value means that the classifier is less likely to identify a negative instance as a positive incorrectly. Recall measures the classifier's ability to identify all positive instances. It is determined by dividing the number of True Positive predictions by the total number of positive predictions (True Positive and False Negative). A higher recall value indicates that the classifier is less likely to miss a positive instance.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F1 = \frac{2 * (Precision * Recall)}{Precision + Recall} \tag{3}$$

Text-to-SQL tasks are usually evaluated by multiple methods such as Component Matching, Accurate matching rate and Execution accuracy rate. Predicted SQL statements are compared with standard statements to determine how accurate the match is. By splitting the predicted SQL statement and definitive statement into multiple clauses according to keywords, we can solve the problem of matching errors caused by the order of the where clause. The matching is successful as long as the elements in both sets are the same.

$$Accuracy = \frac{Successful\ matching\ of\ predicted\ SQL\ statements}{total number\ of\ questions} \tag{4}$$

When using the correct predicted SQL statements, the correct execution rate refers to the proportion of questions that can receive the correct answers from the database.

Evaluation Setup

The evaluation methods for Text-to-SQL systems have advanced over time to assess different aspects of model performance and adaptability. Early datasets generally used a standard train/dev/test split to randomly divide question-SQL pairs across sets as proposed by Iyer et al. (2017), which provided a baseline for evaluation but did not specifically target model generalization capabilities.

To assess the ability of models to generalize to new SQL query structures within a given domain, Finegan-Dollak et al. (2018) introduced the SQL query split method. This method

ensures that SQL queries are only present in one set among the train, dev, and test sets, offering a better understanding of how well models can adapt to new query structures. This approach allows researchers to more accurately evaluate the robustness of their models when faced with novel SQL queries.

In addition to the SQL query split, Yu et al. (2018c) proposed a database split method, which aims to evaluate the generalization capabilities of Text-to-SQL models across different databases. By withholding databases in the test set from the training phase, this method assesses the ability of models to handle unseen databases, emphasizing the importance of real-world adaptability in Text-to-SQL systems.

Other splitting methods have also been proposed to support various research objectives. For example, Shaw et al. (2021) and Chang et al. (2020) have introduced alternative evaluation setups to address specific challenges and research questions in the Text-to-SQL domain. These methods, along with the standard train/dev/test split, SQL query split, and database split, contribute to the diverse range of evaluation strategies employed in Text-to-SQL research, enabling a more comprehensive understanding of model performance and generalization capabilities.

6.1   Naiive Execution Accuracy

Naïve Execution Accuracy (NEA) has become a popular evaluation metric for Text-to-SQL systems because it can measure the accuracy of generated SQL queries in a way that considers both syntax and meaning. Unlike other metrics that mainly focus on the syntax of the queries, NEA examines the practical impact of the queries by evaluating their execution results. Therefore, it provides a more complete view of a model's performance, allowing researchers to better understand how well their algorithms perform in real-world situations.

By applying NEA to commonly used datasets like GeoQuery, IMDB, Yelp, and WikiSQL, researchers can gain a more nuanced understanding of their models' strengths and weaknesses. For instance, GeoQuery is a dataset that contains geographical questions and corresponding SQL queries. By using NEA to evaluate this dataset, researchers can determine not only whether their model generates syntactically correct SQL queries but also whether these queries return the correct geographical data when executed.

Similarly, NEA provides valuable insight into the performance of Text-to-SQL models when generating queries related to movie and business information in the context of the IMDB and Yelp datasets, respectively. By measuring the accuracy of the execution results, NEA helps researchers identify potential improvement areas, whether in the natural language understanding component or the SQL generation process.

The WikiSQL dataset is another significant example where NEA has been used as an evaluation metric. WikiSQL is a massive dataset derived from Wikipedia's SQL-like tables and contains over 24,000 questions and corresponding SQL queries. Evaluating the performance of Text-to-SQL models on WikiSQL can be challenging due to the dataset's size and complexity. However, NEA enables researchers to assess the performance of their models more, accounting for both the syntax of the generated queries and the accuracy of the data they return when executed.

In summary, the adoption of Naïve Execution Accuracy as an evaluation metric for Text-to-SQL systems has proven to be valuable in recent studies involving datasets such as GeoQuery,

IMDB, Yelp, and WikiSQL. By evaluating the practical impact of generated SQL queries, NEA provides a more comprehensive understanding of a model's performance than traditional metrics that concentrate solely on syntax. Consequently, NEA enables researchers to identify potential improvement areas more effectively, ultimately advancing Text-to-SQL technology.

6.2 Exact String Matching

Exact Matching[XLS], a popular metric for assessing the effectiveness of Text-to-SQL models, but it has drawbacks because it can yield erroneous negative results when the semantic parser can produce innovative syntactic structures. The predicted SQL query is compared against the corresponding reference SQL query. The model is considered to have produced the proper SQL query and is given a score of 1.0 if the predicted query is an exact duplicate of the reference query. The model is deemed to have generated an invalid query and obtains a score of 0.0 if the predicted query does not match the reference query. This metric aids in evaluating the overall syntactic and semantic accuracy of the generated query, but it ignores the query's constituent parts. This measure is a reliable evaluation technique because it verifies the entire SQL query. It is, therefore, a more stringent evaluation metric because it only deems a query correct if it exactly matches the reference question, down to the capitalization, spacing, and word order.

6.3 Exact Set Matching

Exact Set Matching compares the set of predicted SQL queries with the set of corresponding reference SQL queries, regardless of the elements' order, to assess the performance of a model. If every element from the set of predicted queries is included in the reference query, it returns a score of 1.0; otherwise, it returns a score of 0.0.

Generally, Exact Set Matching is more forgiving than Exact Matching, as the former does not take the order of elements or capitalization into account. On the other hand, Exact Matching is more stringent as it requires a perfect match including the order of words, capitalization and spaces, thus making it a reliable evaluation method.

6.4 Component Matching

Component matching[YZY+18] involves comparing the elements of the generated SQL query (e.g., the specified columns and tables) to the elements of the reference SQL query. Evaluation is based on the number of components that match correctly between the produced and reference queries, with a higher amount indicating improved performance. This metric assists in measuring the precision of the model's capability to create the correct SQL query components, but it does not factor in the full syntactic or semantic correctness of the query. Furthermore, it is utilized to assess the performance of various models on the same dataset.

6.5 Test Suite Accuracy (Execution Accuracy)

The execution accuracy metric[YZY$^+$18] is a commonly used measure to evaluate the performance of text-to-SQL models. It determines the percentage of correctly generated SQL queries that can be successfully executed on the relevant database. In other words, it evaluates how well a model can convert text written in natural language into a SQL query that can successfully access the desired data from a database.

Execution accuracy is typically reported as a percentage, and higher values denote better performance. It is also important to remember that this metric only considers how correctly the generated SQL queries are syntactically and semantically and ignores how relevant or comprehensive the information is that is returned. Consequently, it is frequently combined with other metrics, such as informativeness, which assesses the accuracy and completeness of the retrieved data.

7 Experiments

Since SEOSS Dataset[RM19] was only evaluated and trained with SQLNet and RatSQL in this section, we decided to investigate further by experimenting with this dataset using state-of-the-art solutions currently proposed for the SPIDER challenge. In order to determine the effectiveness of these methods, we compared the results obtained with those of SQLNet**??** and RatSQL**??** from the SEOSS-Queries research paper[THM22]. The results of these experiments are presented in the following section, and they will demonstrate the potential of modern solutions for solving the SPIDER task.

7.1 Limitations

Our experiment requires a lot of computational resources as we mainly leverage the T5 model. We used a single Nvidia RTX 3070 16GB GPU with 40GB Memory for our experiment, which unfortunately limited us to smaller models with more restrictions. Despite these limitations, we were still able to achieve excellent results. If we had used a larger T5 model, such as T5-3B, we would have been able to reach much higher scores. Therefore, investing in a more powerful GPU for our experiment is something that we must consider in order to maximize our results.

7.2 SEOSS evaluation with T5 PICARD

After studying the SEOSS dataset, we decided to experiment with the PICARD model4.3.8 to evaluate its performance against that of SQLNet and RatSQL. We decided to use the T5Base model for our experiment, as it is smaller than the T5-3B and T5-11B models used by most state-of-the-art studies. To ensure a fair comparison between the models, we used two beam sizes of 2 and 4 and the same evaluation metrics as SEOSS-SQLNet and SEOSS-RatSQL, which is "exact matching accuracy". We wanted to see if the PICARD model could achieve similar results to those of SQLNet and RatSQL, so we conducted our experiment with our findings. The results of our experiment are discussed in the following section and can be used to compare the performance of the PICARD model to the models used in the SEOSS study. [1]

Model	Picard Mode	Beams	Exact Matching Accuracy	Execution Accuracy
T5-base	parse with guards	2	0.3297	0.3576
T5-base	lex	4	0.3071	0.3039
T5-base	parse with guards	4	0.3286	0.3512
T5-large	parse with guards	4	0.4274	0.4822

Table 7: Experrmiment Accuracy Results

The table shows the results of various configurations of T5-base and T5-large models for natural language processing tasks. The configurations are differentiated by the Picard mode parse with guards or lex and the number of beams used in the beam search process 2 or 4.

Comparing the results, we can observe that:

---

[1]Link to the Github Page: https://github.com/yazdipour/text-to-sql-seoss-t5

- The T5-large model generally performs better than the T5-base model in both exact matching accuracy and execution accuracy.

- The parse with guards Picard mode performs better than the lex Picard mode in both models.

- Using four beams instead of 2 in the beam search process improves the performance for both models and Picard modes.

- The highest exact matching accuracy is achieved by the T5-large model with parse with guards Picard mode and four beams 0.4274.

- The highest execution accuracy is also achieved by the T5-large model with parse with guards Picard mode and four beams 0.4822.

- Increasing beam size does not have a significant effect compared to changing the model and mode.

Exact Match Accuracy	easy 392	medium 378	hard 77	extra hard 84	all 931
PICARD + T5Base + 4Beam	0.446	0.254	0.182	0.012	0.307
PICARD + T5Large + 4Beam	0.571	0.410	0.182	0.060	0.427

Table 8: Comparison between Exact Match Accuracy

The table compares the exact match accuracy of various models that are not fine-tuned for our dataset. The models are evaluated on five difficulty levels: easy, medium, hard, extra hard, and all.

Comparing the results, we can observe that:

- The PICARD + T5Large + 4Beam model has the highest exact match accuracy among all models and difficulty levels, with a maximum value of 0.571; this shows that this model is more generalized and can handle unseen databases quite well compared to other solutions.

- The PICARD + T5Base + 4Beam model performs better than the other models, with a maximum value of 0.446.

- The RatSQL models with Glove and Bert embeddings perform similarly, with a maximum value of 0.309 and 0.201, respectively.

- The SQLNet model performs the worst among all models, with a maximum value of 0.023.

- The extra hard and all difficulty levels generally have lower exact match accuracy values compared to the easy and medium levels. Amazingly T5 PICARD was still able to solve complex problems with a low percentage, yet better than the other models.

- Research in[THM22] shows that the trained RatSQL were able to achieve a high exact match accuracy. This is a good sign that the PICARD model can achieve high accuracy with a little fine-tuning.

F1 Scores

Here, we can observe the F1 scores of each SQL Keyword for the PICARD T5-Large 4-Beam experiment on the SEOSS dataset. We can see that PICARD has managed to attain a very impressive F1 score for the SEOSS dataset without even having to be specifically trained for our dataset. This is a very encouraging result and indicates that the model is able to generalize accurately across different domains. Moreover, it is essential to note that the F1 score obtained by the PICARD model was obtained without any additional fine-tuning. This is a testament to the robustness and capability of the model and further highlights its ability to generalize to different datasets.

We experimented with a variety of different parameters, including beam size, modes and model sizes, and spent multiple hours for each evaluation. These experiments have been carefully documented in the Appendix 10 of this thesis, where you can find the results in detail.
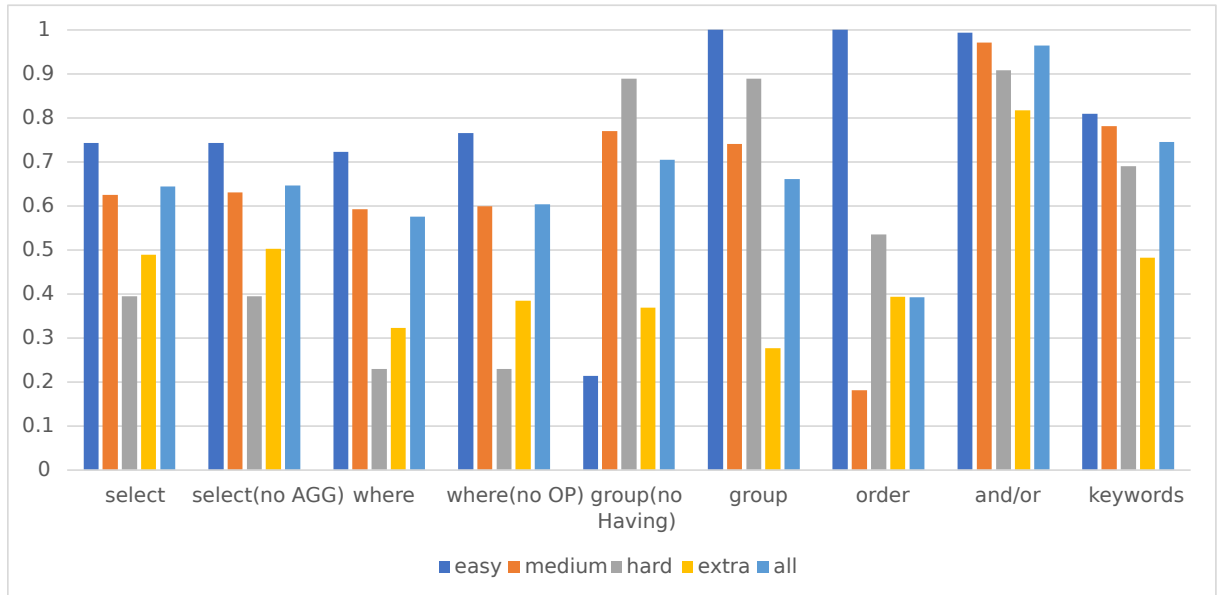
Figure 15: F1 Scores of Component Matching - PICARD T5-Large 4-Beam

7.3   SEOSS evaluation with GPT 3.5

Model	Version	Exact Matching Accuracy	Execution Accuracy
GPT	3.5-turbo	0.423	0.447

Table 9: Expermiment Accuracy Results

7.4   Spider evaluation with GPT 3.5

Exact Match Accuracy	easy 392	medium 378	hard 77	extra hard 84	all 931
SQLNet	0.023	0.000	0.000	0.000	0.010
RatSQL + Glove	0.309	0.214	0.091	0.000	0.224
RatSQL + Bert	0.161	0.201	0.065	0.012	0.156
PICARD + T5Base + 4Beam	0.446	0.254	0.182	0.012	0.307
PICARD + T5Large + 4Beam	0.571	0.410	0.182	0.060	0.427
GPT 3.5-turbo	0.551	0.460	0.130	0.190	0.447

Table 10: Comparison between Exact Match Accuracy

7.5   EZ-PICARD - Microservices Practices
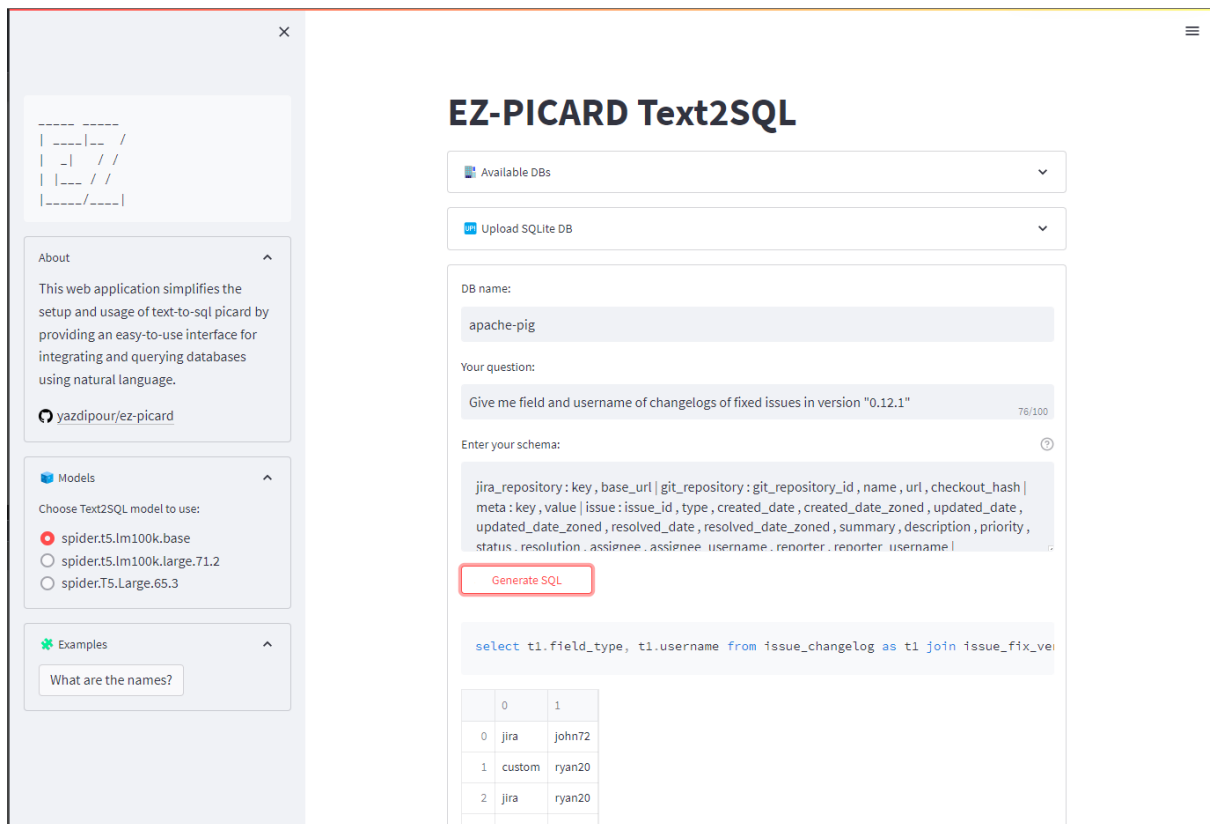
Figure 16: EZ-PICARD Web Application

For software engineering practices and to make PICARD setup easier for engineers, researchers, and users, a microservice web service with a web application has been created and open-sourced to the community[1]. This application consists of a web user interface that gives users the ability to upload their databases and enter their natural language questions and receive queries from our model with values from the database if available. Additionally, a REST API exists for further expansion and usage within the application, providing users with a more versatile and powerful tool for their needs. This web service and application is designed to make the usage of PICARD easier and more accessible for everyone and to allow for the development of new applications and services that utilize its powerful capabilities.

PICARD is a method for constrained inference on top of an existing model, but it is not a model itself. Currently, the PICARD parser and the supporting software are not supported for PostgreSQL, MySQL and others, which would require changes to the PICARD parser, translation of Spider databases and text-to-SQL data, and retraining models to produce MSSQL code. To use the Picard Method, a complex toolchain of Haskell code is built with CABAL and requires a complicated toolchain for the Facebook Thrift library.

The thrift library is used for communication between the parser and the beam search algorithm. The parser, written in the efficient and powerful Haskell programming language, is

---

[1]Link to the Github Page: https://github.com/yazdipour/ez-picard/

used in combination with the hf transformers, which is a Python package. To further expand the scope of the system, new SQL engines can be supported by adding a parser for each one.

These parsers also need to be written in Haskell, as the existing SQLite parser is of limited use in this regard, as it has been written to work best on Spider's subset of SQLite and only supports part of the SQLite specification. This means that more advanced parsers must be created to maximize the system's capabilities. Additionally, these parsers need to be written with a high level of precision in order to ensure that the system can effectively communicate with various engines and databases.

With EZ-PICARD, we can have an adapter layer between SQLite DB and any other database engine, such as MySQL. This layer can be implemented independently from PICARD itself using Python instead of Haskell and can provide a wide range of features, such as automatically translating queries from SQLite to the target database engine and mirroring the schema to the SQLite DB. This adapter layer can provide further advantages by allowing developers to use the same codebase to support multiple database engines, thus reducing the need for additional development and maintenance costs.
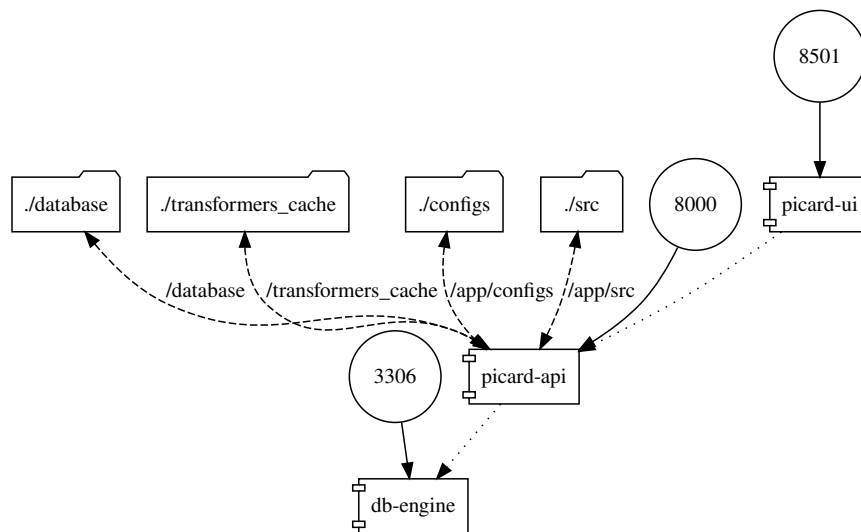
Figure 17: EZ-PICARD Architecture

8 Conclusion

In this thesis, we discussed the state-of-the-art text-to-SQL solutions from a cross-domain perspective, providing a comprehensive overview of the current progress in the field. We demonstrated the effectiveness of pre-trained embeddings in improving schema linking and SQL structure accuracy through experimental results. We hope this study will shed light on the key similarities as well as differences between older models and more recent approaches.

We also explored the impact of the dataset on the performance of the text-to-SQL models. We showed that the Spider dataset is a challenging benchmark for the text-to-SQL task. We also demonstrated the challenging SEOSS dataset and worked on some experiments on it with state-of-the-art models. Our comparison of different models for Text-to-SQL tasks shows that the PICARD + T5 model is a promising choice. However, the potential for even better results exists through fine-tuning the PICARD + T5 model. This process could lead to even more accurate results, but it would likely require access to high-end computing resources. These results demonstrate the importance of considering both model architecture and computational resources when evaluating the performance of NLP models.

Investigating new solutions and the need for more robust evaluation metrics now need to be addressed and further explored in future research. Additionally, with the growth of research in the transformer and language models field, new challenges, such as the Conversation-to-SQL task, have emerged and warrant further research directions.

In conclusion, text-to-SQL has witnessed significant progress over the past few years due to the development of cutting-edge datasets, models, and evaluation metrics. This field offers a wide variety of possibilities for ongoing research and technological advancement.

9 Discussion and Future Directions

The field of text-to-SQL is a rapidly growing area of research, with numerous systems and approaches proposed to generate SQL queries from natural language text. However, there are still several areas that require further exploration and improvement.

A promising avenue for further research is cross-domain text-to-SQL. Incorporating domain-specific knowledge into models trained on existing datasets would enable them to be more adaptable and applicable in different domains. Furthermore, this would facilitate their capacity to handle scenarios where the tables are corrupted or unavailable. In addition, advanced handling of user inputs that are different from the existing datasets and providing database administrators with the ability to manage database schemas and update content are key real-world applications of text-to-SQL. Additionally, multilingual text-to-SQL and creating a database interface for the disabled are noteworthy directions for future research.

Incorporating Text-to-SQL into a broader range of research areas, such as constructing a question-answering system for databases or a dialog system with knowledge from databases, could promote progress in the field. Investigating the interconnection between SQL and other logical forms, as well as generalized semantic parsing, would yield a more comprehensive comprehension of the topic and facilitate the development of more adjustable and generalizable systems.

Regarding more focused strategies, prompt learning could be utilized to improve the robustness of text-to-SQL, and existing text-to-SQL systems could be evaluated and compared to identify their advantages and disadvantages.

In sum, the domain of text-to-SQL has much potential for growth and progress, with numerous significant realistic applications and prospects for integration with related fields.

10 Appendix

- T5 base

- Mode: Lex

- maximum tokens to check: 2

- number of beams: 2

	easy	medium	hard	extra	all
select	0.873	0.624	0.469	0.500	0.717
where	0.882	0.824	0.308	0.421	0.780
group	0.000	0.805	0.969	0.556	0.726
order	0.000	0.409	0.469	0.400	0.431
and/or	1.000	0.929	0.896	0.598	0.927
keywords	0.885	0.867	0.672	0.455	0.829

Table 11: PARTIAL MATCHING ACCURACY

	easy	medium	hard	extra	all
select	0.597	0.373	0.390	0.131	0.447
where	0.756	0.429	0.229	0.104	0.477
group	0.000	0.520	0.886	0.119	0.490
order	0.000	0.321	0.429	0.095	0.267
and/or	0.992	1.000	0.986	0.961	0.993
keywords	0.834	0.513	0.506	0.119	0.545

Table 12: PARTIAL MATCHING RECALL

	easy	medium	hard	extra	all
select	0.709	0.467	0.426	0.208	0.551
where	0.814	0.564	0.262	0.167	0.592
group	1.000	0.632	0.925	0.196	0.585
order	1.000	0.360	0.448	0.154	0.329
and/or	0.996	0.963	0.939	0.737	0.959
keywords	0.859	0.644	0.578	0.189	0.658

Table 13: PARTIAL MATCHING F1

- T5 base

- Mode: parse with guards

- maximum tokens to check: 2

- number of beams: 2

	easy	medium	hard	extra	all
select	0.873	0.624	0.469	0.500	0.717
where	0.882	0.824	0.308	0.421	0.780
group	0.000	0.805	0.969	0.556	0.726
order	0.000	0.409	0.469	0.400	0.431
and/or	1.000	0.929	0.896	0.598	0.927
keywords	0.885	0.867	0.672	0.455	0.829

Table 14: PARTIAL MATCHING ACCURACY

select	0.597	0.373	0.390	0.131	0.447
where	0.756	0.429	0.229	0.104	0.477
group	0.000	0.520	0.886	0.119	0.490
order	0.000	0.321	0.429	0.095	0.267
and/or	0.992	1.000	0.986	0.961	0.993
keywords	0.834	0.513	0.506	0.119	0.545

Table 15: PARTIAL MATCHING RECALL

	easy	medium	hard	extra	all
select	0.709	0.467	0.426	0.208	0.551
where	0.814	0.564	0.262	0.167	0.592
group	1.000	0.632	0.925	0.196	0.585
order	1.000	0.360	0.448	0.154	0.329
and/or	0.996	0.963	0.939	0.737	0.959
keywords	0.859	0.644	0.578	0.189	0.658

Table 16: PARTIAL MATCHING F1

- T5 base

- Mode: Lex

- maximum tokens to check: 2

- number of beams: 4

	easy	medium	hard	extra	all
select	0.858	0.623	0.483	0.571	0.713
where	0.863	0.804	0.333	0.474	0.771
group	0.000	0.812	0.968	0.500	0.699
order	0.000	0.391	0.484	0.444	0.431
and/or	1.000	0.921	0.870	0.602	0.921
keywords	0.836	0.833	0.673	0.476	0.797

Table 17: PARTIAL MATCHING ACCURACY

	easy	medium	hard	extra	all
select	0.538	0.341	0.377	0.143	0.409
where	0.714	0.418	0.229	0.117	0.460
group	0.000	0.469	0.857	0.095	0.448
order	0.000	0.321	0.429	0.095	0.267
and/or	0.995	1.000	0.971	0.980	0.994
keywords	0.789	0.462	0.481	0.119	0.505

Table 18: PARTIAL MATCHING RECALL

	easy	medium	hard	extra	all
select	0.661	0.441	0.423	0.229	0.520
where	0.782	0.550	0.271	0.188	0.576
group	1.000	0.594	0.909	0.160	0.546
order	1.000	0.353	0.455	0.157	0.329
and/or	0.997	0.959	0.918	0.746	0.956
keywords	0.812	0.595	0.561	0.190	0.618

Table 19: PARTIAL MATCHING F1

- T5 base

- Mode: parse with guards

- maximum tokens to check: 2

- number of beams: 4

	easy	medium	hard	extra	all
select	0.757	0.537	0.417	0.469	0.618
where	0.779	0.746	0.250	0.433	0.695
group	0.000	0.658	0.971	0.391	0.561
order	0.000	0.269	0.457	0.526	0.407
and/or	1.000	0.910	0.883	0.619	0.919
keywords	0.742	0.785	0.667	0.422	0.729

Table 20: PARTIAL MATCHING ACCURACY

	easy	medium	hard	extra	all
select	0.653	0.442	0.390	0.274	0.511
where	0.690	0.464	0.171	0.169	0.475
group	0.000	0.571	0.971	0.214	0.552
order	0.000	0.250	0.457	0.238	0.314
and/or	0.992	0.997	1.000	1.000	0.995
keywords	0.823	0.605	0.571	0.226	0.610

Table 21: PARTIAL MATCHING RECALL

	easy	medium	hard	extra	all
select	0.701	0.485	0.403	0.346	0.560
where	0.732	0.572	0.203	0.243	0.564
group	1.000	0.612	0.971	0.277	0.556
order	1.000	0.259	0.457	0.328	0.355
and/or	0.996	0.951	0.938	0.765	0.956
keywords	0.780	0.684	0.615	0.295	0.665

Table 22: PARTIAL MATCHING F1

- T5 large

- Mode: parse with guards

- maximum tokens to check: 2

- number of beams: 4

	easy	medium	hard	extra	all
select	0.781	0.672	0.400	0.552	0.684
where	0.739	0.689	0.269	0.396	0.642
group	0.000	0.756	0.865	0.391	0.675
order	0.000	0.250	0.528	0.542	0.462
and/or	1.000	0.947	0.831	0.699	0.937
keywords	0.796	0.846	0.735	0.574	0.792

Table 23: PARTIAL MATCHING ACCURACY

	easy	medium	hard	extra	all
select	0.709	0.585	0.390	0.440	0.608
where	0.708	0.520	0.200	0.273	0.523
group	0.000	0.726	0.914	0.214	0.649
order	0.000	0.143	0.543	0.310	0.343
and/or	0.987	0.997	1.000	0.983	0.992
keywords	0.823	0.725	0.649	0.417	0.704

Table 24: PARTIAL MATCHING RECALL

	easy	medium	hard	extra	all
select	0.743	0.625	0.395	0.490	0.644
where	0.723	0.593	0.230	0.323	0.576
group	1.000	0.741	0.889	0.277	0.661
order	1.000	0.182	0.535	0.394	0.393
and/or	0.994	0.971	0.908	0.817	0.964
keywords	0.809	0.781	0.690	0.483	0.746

Table 25: PARTIAL MATCHING F1

- GPT

- Mode: 3.5 Turbo

	easy	medium	hard	extra	all
select	0.924	0.816	0.788	0.763	0.862
where	0.759	0.816	0.321	0.400	0.719
group	0.154	0.863	1.000	0.000	0.719
order	0.000	0.714	0.000	0.889	0.667
and/or	1.000	0.984	0.909	0.863	0.973
keywords	0.868	0.911	0.469	0.543	0.838

Table 26: PARTIAL MATCHING ACCURACY

	easy	medium	hard	extra	all
select	0.653	0.529	0.338	0.345	0.549
where	0.768	0.658	0.257	0.182	0.590
group	0.286	0.360	0.114	0.000	0.266
order	0.000	0.357	0.000	0.190	0.171
and/or	0.934	0.968	0.972	0.945	0.952
keywords	0.903	0.574	0.195	0.226	0.573

Table 27: PARTIAL MATCHING RECALL

	easy	medium	hard	extra	all
select	0.765	0.642	0.473	0.475	0.671
where	0.763	0.729	0.286	0.250	0.648
group	0.200	0.508	0.205	1.000	0.389
order	1.000	0.476	1.000	0.314	0.273
and/or	0.966	0.976	0.940	0.902	0.962
keywords	0.885	0.704	0.275	0.319	0.680

Table 28: PARTIAL MATCHING F1

Bibliography

[BBG19]   Ben Bogin, Jonathan Berant, and Matt Gardner. Representing schema struc-
             ture with graph neural networks for text-to-SQL parsing. In *Proceedings of
             the 57th Annual Meeting of the Association for Computational Linguistics*,
             pages 4560–4565, Florence, Italy, July 2019. Association for Computational
             Linguistics.

[BGB19]   Ben Bogin, Matt Gardner, and Jonathan Berant. Global reasoning over
             database structures for text-to-SQL parsing. In *Proceedings of the 2019 Con-
             ference on Empirical Methods in Natural Language Processing and the 9th
             International Joint Conference on Natural Language Processing (EMNLP-
             IJCNLP)*, pages 3659–3664, Hong Kong, China, November 2019. Associa-
             tion for Computational Linguistics.

[BS21]   Ursin Brunner and Kurt Stockinger. Valuenet: A natural language-to-sql
             system that learns from database information, 2021.

[CCC+21]   Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu.
             LGESQL: Line graph enhanced text-to-SQL model with mixed local and
             non-local relations. In *Proceedings of the 59th Annual Meeting of the Asso-
             ciation for Computational Linguistics and the 11th International Joint Con-
             ference on Natural Language Processing (Volume 1: Long Papers)*, pages
             2541–2555, Online, August 2021. Association for Computational Linguis-
             tics.

[CCZ+21]   Zhi Chen, Lu Chen, Yanbin Zhao, Ruisheng Cao, Zihan Xu, Su Zhu, and Kai
             Yu. ShadowGNN: Graph projection neural network for text-to-SQL parser.
             In *Proceedings of the 2021 Conference of the North American Chapter of
             the Association for Computational Linguistics: Human Language Technolo-
             gies*, pages 5567–5577, Online, June 2021. Association for Computational
             Linguistics.

[CSKS]   DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol
             Shin. RYANSQL: Recursively applying sketch-based slot fillings for com-
             plex text-to-SQL in cross-domain databases. version: 1.

[CSKS21]   DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol
             Shin. RYANSQL: Recursively Applying Sketch-based Slot Fillings for Com-
             plex Text-to-SQL in Cross-Domain Databases. *Computational Linguistics*,
             47(2):309–332, 07 2021.

[CSLJ20]   Sanxing Chen, Aidan San, Xiaodong Liu, and Yangfeng Ji. A tale of two
             linkings: Dynamically gating between schema linking and structural linking
             for text-to-SQL parsing. In *Proceedings of the 28th International Conference
             on Computational Linguistics*, pages 2900–2912, Barcelona, Spain (Online),
             December 2020. International Committee on Computational Linguistics.

[CvMG+14]   Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bah-
             danau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning

phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[CYXH]   Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. SADGA: Structure-aware dual graph aggregation network for text-to-SQL.

[DBB⁺94]   Deborah A. Dahl, Madeleine Bates, Michael Brown, William Fisher, Kate Hunicke-Smith, David Pallett, Christine Pao, Alexander Rudnicky, and Elizabeth Shriberg. Expanding the scope of the ATIS task: The ATIS-3 corpus. In *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994.

[DCLT18]   Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[DCLT19]   Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[DCZ22]   Naihao Deng, Yulong Chen, and Yue Zhang. Recent advances in text-to-sql: A survey of what we have and what we expect, 2022.

[DL16]   Li Dong and Mirella Lapata. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany, August 2016. Association for Computational Linguistics.

[DL18]   Li Dong and Mirella Lapata. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[FDKZ⁺18]   Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[GCX⁺21]   Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John Drake, and Qiaofu Zhang. Natural SQL: Making SQL easier to infer from natural language specifications. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2030–2042, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[GG20]   Tong Guo and Huilin Gao. Content enhanced bert-based text-to-sql generation, 2020.

[GZG+19]   Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Towards complex text-to-sql in cross-domain database with intermediate representation. *CoRR*, abs/1905.08205, 2019.

[HGW+22]   Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Bowen Li, Jian Sun, and Yongbin Li. S^{2}sql: Injecting syntax to question-schema interaction graph encoder for text-to-sql parsers, 2022.

[HM10]   Liang Huang and Haitao Mi. Efficient incremental decoding for tree-to-string translation. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 273–283, Cambridge, MA, October 2010. Association for Computational Linguistics.

[HMB21]   Moshe Hazoom, Vibhor Malik, and Ben Bogin. Text-to-sql in the wild: A naturally-occurring dataset based on stack exchange data. *CoRR*, abs/2106.05006, 2021.

[HMCC19]   Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. X-sql: reinforce schema representation with context, 2019.

[HR18]   Jeremy Howard and Sebastian Ruder. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146, 2018.

[HS97]   Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[HSC+21]   Jonathan Herzig, Peter Shaw, Ming-Wei Chang, Kelvin Guu, Panupong Pasupat, and Yuan Zhang. Unlocking compositional generalization in pretrained models using intermediate representations, 2021.

[HYPS19]   Wonseok Hwang, Jinyeung Yim, Seunghyun Park, and Minjoon Seo. A comprehensive exploration on wikisql with table-aware word contextualization. *CoRR*, abs/1902.01069, 2019.

[JC21]   Marcelo Archanjo José and Fábio Gagliardi Cozman. mrat-sql+gap: A portuguese text-to-sql transformer. *CoRR*, abs/2110.03546, 2021.

[KDG17]   Jayant Krishnamurthy, Pradeep Dasigi, and Matt Gardner. Neural semantic parsing with type constraints for semi-structured tables. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1516–1526, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

[KDHR17]   Yoon Kim, Carl Denton, Luong Hoang, and Alexander M. Rush. Structured attention networks. In *International Conference on Learning Representations*, 2017.

[KNNV22]   Ayush Kumar, Parth Nagarkar, Prabhav Nalhe, and Sanjeev Vijayakumar. Deep learning driven natural languages text to sql query conversion: A survey, 2022.

[KRB$^+$20]   Amol Kelkar, Rohan Relan, Vaishali Bhardwaj, Saurabh Vaichal, Chandra Khatri, and Peter Relan. Bertrand-dr: Improving text-to-sql using a discriminative re-ranker, 2020.

[LB99]   Jeong Oog Lee and Doo Kwon Baik. Semql: A semantic query language for multidatabase systems. In *International Conference on Information and Knowledge Management, Proceedings*, International Conference on Information and Knowledge Management, Proceedings, pages 259–266. ACM, 1999. Proceedings of the 1999 8th International Conference on Information Knowledge Management (CIKM'99) ; Conference date: 02-11-1999 Through 06-11-1999.

[LCH$^+$20]   Qin Lyu, Kaushik Chakrabarti, Shobhit Hathi, Souvik Kundu, Jianwen Zhang, and Zheng Chen. Hybrid ranking network for text-to-SQL. 2020.

[LLG$^+$20]   Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.

[LSX]   Xi Victoria Lin, Richard Socher, and Caiming Xiong. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing.

[LTBZ17]   Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks, 2017.

[MDP$^+$11]   Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[MLGftADNI19]   P. J. Moore, T. J. Lyons, J. Gallacher, and for the Alzheimer's Disease Neuroimaging Initiative. Random forest prediction of alzheimer's disease using pairwise selection from time series data. *PLOS ONE*, 14(2):1–14, 02 2019.

[MSZ19]   Qingkai Min, Yuefeng Shi, and Yue Zhang. A pilot study for Chinese SQL semantic parsing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3652–3658, Hong Kong, China, November 2019. Association for Computational Linguistics.

[MV19]   Samuel Müller and Andreas Vlachos. Byte-pair encoding for text-to-sql generation, 2019.

[RB21]   Ohad Rubin and Jonathan Berant. SmBoP: Semi-autoregressive bottom-up semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 311–324, Online, June 2021. Association for Computational Linguistics.

[RCM+13]   Senjuti Basu Roy, Martine De Cock, Vani Mandava, Swapna Savanna, Brian Dalessandro, Claudia Perlich, William Cukierski, and Ben Hamner. The microsoft academic search dataset and kdd cup 2013 - workshop for kdd cup 2013. ACM - Association for Computing Machinery, August 2013.

[RJS17]   Alec Radford, Rafal Józefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *CoRR*, abs/1704.01444, 2017.

[RM19]   Michael Rath and Patrick Mäder. The seoss 33 dataset — requirements, bug reports, code history, and trace links for entire projects. *Data in Brief*, 25:104005, 2019.

[RN18]   Alec Radford and Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[Ron14]   Xin Rong. word2vec parameter learning explained. *CoRR*, abs/1411.2738, 2014.

[RSR+19]   Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[SCPT21]   Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 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, August 2021. Association for Computational Linguistics.

[SHB16]   Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[SLB+21]   Torsten Scholak, Raymond Li, Dzmitry Bahdanau, Harm de Vries, and Chris Pal. DuoRAT: Towards simpler text-to-SQL models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1313–1321, Online, June 2021. Association for Computational Linguistics.

[SNW+20]   Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. Learning contextual representations for semantic parsing with generation-augmented pre-training, 2020.

[SO21]   Irina Saparina and Anton Osokin. SPARQLing database queries from intermediate question decompositions. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8984–8998, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[SSB21]   Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 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. Association for Computational Linguistics, November 2021.

[STC+18]   Tianze Shi, Kedar Tatwawadi, Kaushik Chakrabarti, Yi Mao, Oleksandr Polozov, and Weizhu Chen. Incsql: Training incremental text-to-sql parsers with non-deterministic oracles, 2018.

[SYL20]   Ningyuan Sun, Xuefeng Yang, and Yunfeng Liu. Tableqa: a large-scale chinese text-to-sql dataset for table-aware SQL generation. *CoRR*, abs/2006.06434, 2020.

[THM22]   Mihaela Todorova Tomova, Martin Hofmann, and Patrick Mäder. Seoss-queries - a software engineering dataset for text-to-sql and question answering tasks. *Data in Brief*, 42:108211, 2022.

[TM01]   Lappoon R. Tang and Raymond J. Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In Luc De Raedt and Peter Flach, editors, *Machine Learning: ECML 2001*, pages 466–477, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[TNDN20]   Anh Tuan Nguyen, Mai Hoang Dao, and Dat Quoc Nguyen. A pilot study of text-to-SQL semantic parsing for Vietnamese. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4079–4085, Online, November 2020. Association for Computational Linguistics.

[VFJ17]   Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks, 2017.

[VSP+17]   Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[WBS17]   Chenglong Wang, Marc Brockschmidt, and Rishabh Singh. Pointing out sql queries from text. Technical Report MSR-TR-2017-45, November 2017.

[WLC20]   Huajie Wang, Mei Li, and Lei Chen. PG-GSQL: Pointer-generator network with guide decoding for cross-domain context-dependent text-to-SQL generation. In *Proceedings of the 28th International Conference on Computational*

Linguistics, pages 370–380, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics.

[WP82]   David H. D. Warren and Fernando C Pereira. An efficient easily adaptable system for interpreting natural language queries. *Am. J. Comput. Linguistics*, 8:110–122, 1982.

[WSC⁺16]   Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[WSL⁺]   Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers.

[WTB⁺18]   Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. Robust text-to-sql generation with execution-guided decoding, 2018.

[WTL19]   Bailin Wang, Ivan Titov, and Mirella Lapata. Learning semantic parsers from denotations with latent structured alignments and abstract programs. 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 3774–3785, Hong Kong, China, November 2019. Association for Computational Linguistics.

[WZW⁺20]   Lijie Wang, Ao Zhang, Kun Wu, Ke Sun, Zhenghua Li, Hua Wu, Min Zhang, and Haifeng Wang. DuSQL: A large-scale and pragmatic Chinese text-to-SQL dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6923–6935, Online, November 2020. Association for Computational Linguistics.

[XLS]   Xiaojun Xu, Chang Liu, and Dawn Song. SQLNet: Generating structured queries from natural language without reinforcement learning.

[XWS⁺22]   Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, Victor Zhong, Bailin Wang, Chengzu Li, Connor Boyle, Ansong Ni, Ziyu Yao, Dragomir Radev, Caiming Xiong, Lingpeng Kong, Rui Zhang, Noah A. Smith, Luke Zettlemoyer, and Tao Yu. Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models, 2022.

[YLZ⁺18]   Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir R. Radev. Type-sql: Knowledge-based type-aware neural text-to-sql generation. *CoRR*, abs/1804.09769, 2018.

[YN17]   Pengcheng Yin and Graham Neubig. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada, July 2017. Association for Computational Linguistics.

[YNYR]   Pengcheng Yin, Graham Neubig, Wen-tau Yih, and Sebastian Riedel. TaBERT: Pretraining for joint understanding of textual and tabular data. version: 1.

[YWDD17]   Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. Sqlizer: Query synthesis from natural language. *Proc. ACM Program. Lang.*, 1(OOPSLA), oct 2017.

[YWL⁺20]   Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir R. Radev, Richard Socher, and Caiming Xiong. Grappa: Grammar-augmented pre-training for table semantic parsing. *CoRR*, abs/2009.13845, 2020.

[YYY⁺18]   Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir R. Radev. Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task. *CoRR*, abs/1810.05237, 2018.

[YZY⁺18]   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. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. 2018.

[ZM96]   John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *AAAI/IAAI, Vol. 2*, 1996.

[ZXS]   Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. version: 6.