A Survey on Text-to-SQL Parsing: Concepts, Methods, and Future Directions

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, Yongbin Li

Abstract—Text-to-SQL parsing is an essential and challenging task. The goal of text-to-SQL parsing is to convert a natural language (NL) question to its corresponding structured query language (SQL) based on the evidences provided by relational databases. Early text-to-SQL parsing systems from the database community achieved a noticeable progress with the cost of heavy human engineering and user interactions with the systems. In recent years, deep neural networks have significantly advanced this task by neural generation models, which automatically learn a mapping function from an input NL question to an output SQL query. Subsequently, the large pre-trained language models have taken the state-of-the-art of the text-to-SQL parsing task to a new level. In this survey, we present a comprehensive review on deep learning approaches for text-to-SQL parsing. First, we introduce the text-to-SQL parsing corpora which can be categorized as single-turn and multi-turn. Second, we provide a systematical overview of pre-trained language models and existing methods for text-to-SQL parsing. Third, we present readers with the challenges faced by text-to-SQL parsing and explore some potential future directions in this field.

Index Terms—Text-to-SQL Parsing, Semantic Parsing, Natural Language Understanding, Table Understanding, Deep Learning

1 Introduction

With the popularity of electronic devices, tables have become the mainstream to store large structural data from various resources (e.g., webpages, databases and spreadsheets), which represent the data as a grid-like format of rows and columns so that users can easily inquire the patterns and discover insights from data. Although the tables can be efficiently accessed by skilled professionals via the handcrafted structured query languages (SQLs), a natural language (NL) interface can facilitate the ubiquitous relational data to be accessed by a wider range of non-technical users [1]. Therefore, text-to-SQL parsing, which aims to translate NL questions to machine-executable SQLs, has attracted noticeable attention from both industrial and academic communities. It can empower non-expert users to effortlessly query tables and plays a central role in various real-life applications such as intelligent customer service, question answering, and robotic navigation.

Early text-to-SQL parsing work [2] from the Database community made a noticeable progress with the cost of heavy human engineering and user interactions with the systems. It is difficult, if not impossible, to design SQL templates in advance for various scenarios or domains. In recent years, recent advances of deep learning and the availability of large-scale training data have significantly improve text-to-SQL parsing by neural generation models. A typical neural generation method is the sequence-to-sequence (Seq2Seq) [3] model, which automatically learns a mapping function from the input NL question to the output SQL under encoder-decoder schemes. The key idea is to construct an encoder to understand the input NL questions together with related table schema and leverage a grammar-based neural decoder to predict the target SQL. The Seq2Seq based approaches have become the mainstream for text-to-SQL parsing mainly because they can be trained in an end-to-end way and reduce the need for specialized domain knowledge.

So far, various neural generation models have been developed to improve the encoder and the decoder respectively. On the encoder side, several general neural networks are widely used to globally reason over natural language query and database schema. IRNet [4] encoded the question and the table schema separately with bi-directional LSTM [5] and self-attention mechanism [6]. RYANSQL [7] employed convolutional neural network [8] with dense connection [9] for question/schema encoding. With the advance of pre-trained language models (PLMs), SQLova [10] first proposed to leverage the pre-trained language models (PLMs) such as BERT [11] as the base encoder. RATSQL [12], SADGA [13] and LGESQL [14] adopted graph neural network to encode the relational structure between the database schema and a given question. On the decoder side, there are two categories of SQL generation approaches including the sketch-based methods and the generation-based methods. Specifically, the sketch-based methods [15, 10, 16] decompose the SQL generation procedure into sub-modules, where each sub-module corresponds to the type of the prediction slot to be filled. These sub-modules are later gathered together to generate the final SQL query. To enhance the performance of the generated SQL logic form,

people using graph

sketch-based methods

- *B. Qin, L. Wang and M. Yang are with Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, China, 518055. B. Qin and L. Wang are also with University of Chinese Academy of Sciences, Beijing, China, 101408. E-mail: {lh.wang1, bw.qin, min.yang}@siat.ac.cn*
- *B. Hui, B. Li, R. Geng, R. Gao, J. Sun, L. Si, F. Huang and Y. Li are with Alibaba Group, Beijing, China. E-mail: {binyuan.hby, binhua.lbh, ruiying.gry, caorongyu.cry, jian.sun, luo.si, f.huang, shuide.lyb}@alibaba-inc.com*
- *J. Li is with The University of Hong Kong, Hong Kong. E-mail: jl0725@connect.hku.hk*

Min Yang and Yongbin Li are corresponding authors.

generation -based methods

the generation-based methods [4, 12, 14, 17] usually decoded the SQL query as an abstract syntax tree in the depth-first traversal order by employing an LSTM [5] decoder.

In parallel, PLMs have proved to be powerful in enhancing text-to-SQL parsing and yield state-of-the-art performances, which benefit from the rich linguistic knowledge in large-scale corpora. However, as revealed in previous works, there are intrinsic differences between the distribution of tables and plain texts. Directly fine-tuning the PLMs trained on large-scale plain texts to downstream text-to-SQL parsing hinders the models from effectively modeling the relational relational structure in question/schema, and thus leads to sub-optimal performances. Current studies to alleviate the above limitation attempt to build Tabular Language Models (TaLMs) by directly encoding tables and texts, which show improved results on downstream text-to-SQL parsing tasks. For example, TaBERT [18] jointly encoded texts and tables with masked language modeling (MLM) and masked column prediction (MCP) respectively, which was trained on a large corpus of tables and their corresponding English contexts. TaPas [19] extended BERT [11] by using additional positional embeddings to encode tables. In addition, two classification layers are applied to choose table cells and aggregation operators which operate on the table cells. Grappa [20] introduced a grammar-augmented pre-training framework for table semantic parsing, which explores the schema linking in table semantic parsing by encouraging the model to capture table schema items which can be grounded to logical form constituents. Grappa achieved the state-of-the-art performances for text-to-SQL parsing.

Contributions of this survey. This manuscript aims at providing a comprehensive review of the literature on text-to-SQL parsing as shown in Fig. 1. By providing this survey, we hope to provide a useful resource for both academic and industrial communities. First, we introduce the experimental datasets and present a taxonomy that classifies the representative text-to-SQL approaches. Moreover, we present readers with the challenges faced by text-to-SQL parsing and explore some potential future directions in this field.

This manuscript is organized as follows. In Section 2, we define the text-to-SQL parsing formally and introduce the official evaluation metrics. Section 3 presents the main scenarios (single-turn and multi-turn utterances) and the corresponding datasets for text-to-SQL parsing. We introduce the representative pre-training, encoding and decoding techniques for text-to-SQL parsing in Section 4 and Section 5 respectively. Section 6 concludes this manuscript and outlines the future directions, followed by the references.

2 Background

In this section, we first provide a formal problem definition of text-to-SQL parsing. Then, we describe the official evaluation metrics for verifying the text-to-SQL parsers. Finally, we introduce the benchmark corpora used for training the neural text-to-SQL parsers.

2.1 Task Formulation

Text-to-SQL (T2S) parsing aims to convert a natural language (NL) question under database items to its corresponding structured query language (SQL) that can be executed against a relational database. As shown in Table 1, we provide formal notation to normalise task definitions. Generally, existing T2S parsing approaches can be categorized into single-turn (context-independent) and multi-turn (context-dependent) settings. Formally, for the single-turn T2S parsing setting, given a NL question Q and the corresponding database schema $\mathcal{S} = \langle \mathcal{T}, \mathcal{C} \rangle$, our goal is to generate a SQL query Y. To be specific, the question $Q = \{q_1, q_2, \cdots, q_{|Q|}\}$ is a sequence of $|Q|$ tokens. The database schema consists of $|\mathcal{T}|$ tables $\mathcal{T} = \{t_1, t_2, \cdots, t_{|\mathcal{T}|}\}$ and $|\mathcal{C}|$ columns $C = \{c_1, c_2, \cdots, c_{|\mathcal{C}|}\}$. Each table t_i is described by its name that contains multiple words $[t_{i,1}, t_{i,2}, \cdots, t_{i,|t_i|}]$. Each column $c_j^{t_i}$ in table t_i is represented by words (a phrase) $[c_{j,1}^{t_i}, c_{j,2}^{t_i}, \cdots, c_{j,|c_j^{t_i}|}^{t_i}]$. We denote the whole input as $X = \langle \mathcal{Q}, \mathcal{S} \rangle$.

For the multi-turn T2S parsing setting, we aims to convert a sequence of NL questions to the corresponding SQL queries, where the NL questions may contain ellipsis and anaphora that refers to earlier items in the previous NL questions. Formally, let $U = \{U_1, \ldots, U_T\}$ denote a sequence of utterances with T turns, where $U_t = (X_t, Y_t)$ represents the t-th utterance which is the combination of a NL question X_i and a SQL query Y_i. In addition, there is corresponding database schema $\mathcal{S}$. At the t-th turn, the goal of multi-turn T2S parsing is to produce the SQL query Y_t conditioned on the current NL question X_t, the historical utterances $\{U_i\}_{i=1}^{t-1}$, and the database schema $\mathcal{S}$.

2.2 Evaluation metrics

The text-to-SQL (T2S) parsers are generally evaluated by comparing the generated SQL queries against the ground-truth SQL answers. Concretely, there are two types of evaluation metrics that are used for evaluating the single-turn T2S setting, including exact set match accuracy (EM) and execution accuracy (EX) [23]. For the multi-turn T2S setting, question match accuracy (QM) and interaction match accuracy (IM) [31] are commonly employed.

2.2.1 *Single-turn T2S Evaluation*

Exact Set Match Accuracy (EM) The exact set match accuracy (without values) is calculated by comparing the ground-truth SQL query and the predicted SQL query. Both ground-truth and predicted queries are parsed into normalized data structures which have the following SQL clauses such as `SELECT` • `GROUP BY` • `WHERE` • `ORDER BY` • `KEYWORDS` (including all SQL keywords without column names and operators).

We treat the predicted SQL query as correct only if all of the SQL clauses are correct by a set comparison as follows:

$$score(\hat{Y}, Y) = \begin{cases} 1, & \hat{Y} = Y \\ 0, & \hat{Y} \neq Y \end{cases} \quad (1)$$

where $\hat{Y} = \{(\hat{k}^i, \hat{v}^i), i \in (1, m)\}$ and $Y = \{(k^i, v^i), i \in (1, m)\}$ denote the component sets of the predicted SQL

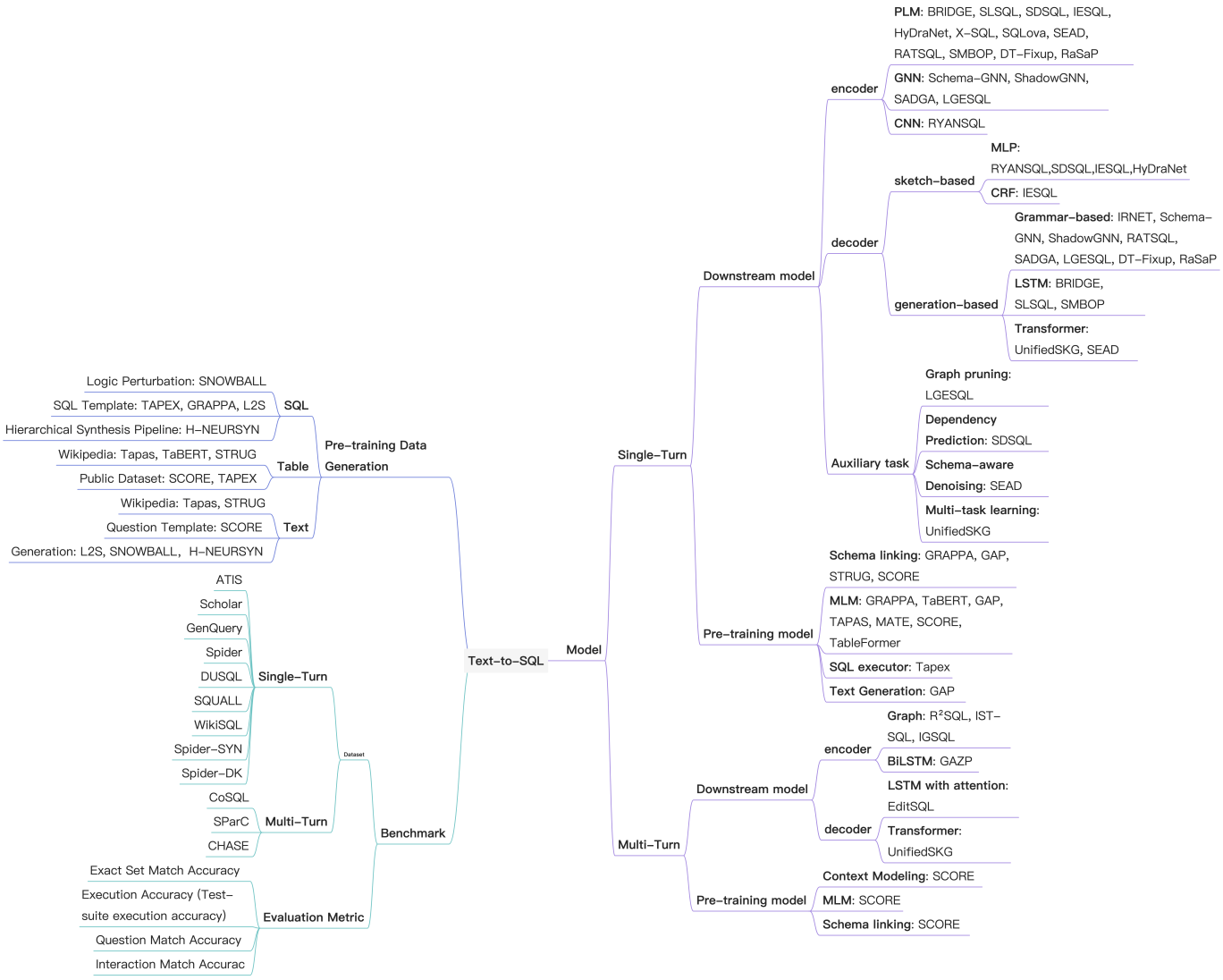

Fig. 1. The comprehensive overview of the text-to-SQL parsing datasets, the pre-training tabular language models, the downstream text-to-SQL parsing approaches.

TABLE 1
The notations used in this manuscript.

Symbol	Description
$\mathcal{S}$	Sequence of database schema tokens, which consists of tables and columns.
$\mathcal{T}$	Sequence of table tokens.
$\mathcal{C}$	Sequence of column tokens.
$\mathcal{Q}$	Sequence of question tokens.
q	Question token.
t	Table token.
c	Column token.
$\|\|$	Length of tokens.
X	Input of text-to-SQL model, which consists of question and schema.
Y	Output of text-to-SQL model, referring to SQL query.
I	Input sequence of encoder, which consists of special token, question token and schema tokens.
`[CLS],[SEP]`	Special token of PLMs.
$\mathbf{u}$	Graph node embedding vector.
$\mathbf{r}$	Relation embedding vector.
W_K	Weight matrix of key vectors, which are used to calculate the attention score.
W_Q	Weight matrix of query vectors, which are used to calculate the attention score.

query and the ground-truth query respectively. Here, k stands for a SQL clause and v is the corresponding value of the clause. m is the number of components. Formally, the

TABLE 2
The statistics of the T2S datasets. "#" denotes the number of the corresponding units.

Dataset	Single-Turn	Multi-Turn	Cross-domain	Robustness	Languages	#Question	#SQL	#DB	#Domain	#Table
GenQuery [2]	✓				en	880	247	1	1	6
Scholar [21]	✓				en	817	193	1	1	7
WikiSQL [22]	✓				en	80654	77840	26521	-	1
Spider [23]	✓		✓		en	10181	5693	200	138	1020
Spider-SYN [24]	✓		✓	✓	en	7990	4525	166	-	876
Spider-DK [25]	✓		✓	✓	en	535	283	10	-	48
Spider-SSP [26]	✓				en	-	-	-	-	-
CSpider [27]	✓		✓		zh	10181	5693	200	138	1020
SQUALL [27]	✓		✓		en	15620	11276	2108	-	2108
DuSQL [28]	✓		✓		zh	23797	23797	200	-	820
ATIS [29, 30]		✓			en	5418	947	1	1	27
SparC [31]		✓	✓		en	4298	12726	200	138	1020
CoSQL [32]		✓	✓		en	3007	15598	200	138	1020
CHASE [33]		✓	✓		zh	5489	17940	280	-	1280

exact set match accuracy is calculated by:

$$\text{EM} = \frac{\Sigma_{n=1}^{N} score(\hat{Y}_n, Y_n)}{N}, \tag{2}$$

where N denotes the total number of samples. EM evaluates the model performance by strictly comparing differences in SQL, but human's SQL annotations are often biased since a NL question may correspond to multiple SQL queries.

Execution Accuracy (EX) Execution accuracy (with values) is calculated by comparing the output results of executing the ground-truth SQL query and the predicted SQL query on the database contents shipped with the test set. We treat the predicted query as correct only if the results of executing the predicted SQL query $\hat{V}$ and the ground-truth SQL query V are same:

$$score(\hat{V}, V) = \begin{cases} 1, & \hat{V} = V \\ 0, & \hat{V} \neq V \end{cases} \tag{3}$$

Similarly with EM, the EX is calculated by:

$$\text{EX} = \frac{\Sigma_{n=1}^{N} score(\hat{Y}_n, Y_n)}{N}, \tag{4}$$

To avoid false positives and false negatives caused by SQL execution on finite size databases, the test-suite execution accuracy [34] extends the execution to multiple database instances per schema. Concretely, the test-suite distills a small database from random generated databases to achieve high code coverage. In this way, we can provide the best approximation of semantic accuracy.

2.2.2 Multi-turn T2S Evaluation

Given a multi-turn setting, there are a total of P question sequences, where each sequence contains O rounds and a total of $M = P \times O$ questions.

Question Match Accuracy (QM) The question match accuracy is calculated as the EM score over all questions. Its value is 1 for each question only if all predicted SQL clauses are correct. We first calculate the EM score for each question as follows:

$$score(\hat{Y}, Y) = \begin{cases} 1, & \hat{Y} = Y \\ 0, & \hat{Y} \neq Y \end{cases} \tag{5}$$

where $\hat{Y}$ and Y are the predicted and ground-truth SQL queries, respectively. Then, the question match accuracy is calculated by:

$$\text{QM} = \frac{\Sigma_{m=1}^{M} score(\hat{Y}_m, Y_m)}{M}, \tag{6}$$

where M denotes the total number of questions.

Interaction Match Accuracy (IM) The interaction match accuracy is calculated as the EM score over all interactions (question sequences). The score of each interaction is 1 only if all the questions within the interaction are correct. Formally, the score for each interaction is defined as follows:

$$interaction = \begin{cases} 1, & \prod_{i=1}^{o} score(\hat{Y}_i, Y_i) = 1 \\ 0, & \prod_{i=1}^{o} score(\hat{Y}_i, Y_i) = 0 \end{cases} \tag{7}$$

where o is the number of turns in each interaction. Then, the IM score is calculated by:

$$\text{IM} = \frac{\Sigma_{p=1}^{P} interaction_p}{P}. \tag{8}$$

where P is the total number of interactions.

2.3 Datasets

High-quality corpora are essential for learning and evaluating the text-to-SQL (T2S) parsing systems. In the following, we summarize extensively-used datasets into two primary categories: the single-turn T2S corpora with single-turn (stand-alone) questions and the multi-turn T2S corpora with multi-turn sequential questions.

2.3.1 Single-Turn T2S Corpora

GenQuery The GenQuery [2] dataset is a collection 880 NL questions for querying a database of US geographical facts (denoted as Geobase). A relational database schema and SQL queries are constructed over Geobase for 700 questions. Afterwards, the remaining NL questions are further annotated by [21], following the widely used 600/280 training/test split [35].

Scholar The Scholar [21] dataset is a collection of 816 NL questions annotated with SQL queries, where 600 NL questions are used for training and 216 questions are used for testing. An academic database consisting of academic papers is provided to executed the SQL queries.

WikiSQL The original WikiSQL [22] dataset is a collection of 80,654 hand-crafted NL question and SQL query pairs along with the corresponding SQL tables extracted from 24,241 HTML tables on Wikipedia. In particular, for each selected table six SQL queries are generated following the SQL templates and rules. Then, for each SQL query, a crude NL question is annotated using templates via crowdsourcing on Amazon Mechanical Turk. The WikiSQL dataset contains much more instances and tables than ATIS [29, 30], GenQuery [2] and Scholar [21]. In addition, the WikiSQL dataset is more challenging than previous T2S corpora since WikiSQL spans over a large number of tables and the text-to-SQL parsers should generalize to not only new queries but also new table schema. The instances in WikiSQL can be randomly split into training/validation/testing sets, such that each table is involved in exactly one set.

Spider The Spider [23] dataset is a large-scale benchmark for cross-domain text-to-SQL parsing. Spider contains 10,181 NL questions and 5,693 unique SQL queries over 200 databases belonging to 138 different domains. Different from the prior T2S datasets that contain tables from the same domain, the Spider dataset contains complex NL questions and SQL queries spanning over multiple databases and domains. In addition, the SQL queries in the Spider dataset can be further divided into four levels based on the difficulty of the SQL queries: easy, medium, hard, extra hard, which can be used to better evaluate the model performance on different queries. Finally, the Spider dataset is randomly split into 7,000 instances for training, 1,034 instances for development, and 2,147 instances for testing.

Spider-Syn The Spider-Syn [24] dataset is another challenging variant of Spider [23], which modifies the NL questions from Spider [23] by replacing the schema-related words with the corresponding synonyms. In this way, the explicit alignments between the words in NL questions and the tokens in table schemas are eliminated, which make the schema linking more challenging for text-to-SQL parsing. Spider-Syn [24] is composed of 7000 training instances and 1034 development instances. Note that Spider-Syn [24] has no test set since the original Spider [23] does not release the test set publicly.

Spider-DK The Spider-DK [25] dataset is a challenging variant of the Spider [23] development set, which can be used to better investigate the generalization ability of existing text-to-SQL models in understanding the domain knowledge. Spider-DK is constructed by adding domain knowledge that reflects real-world question paraphrases to some NL questions from the Spider development set. Concretely, it consists of 535 NL-SQL pairs, where 265 NL-SQL pairs are modified by adding domain knowledge while the rest 270 NL-SQL pairs remain the same as in the original Spider dataset. It is noteworthy that Spider-DK is smaller than the original Spider development set since not every instance can be easily modified to add domain knowledge.

Spider-SSP The Spider-SSP [26] refers to the compositional generalization version of the Spider [23] dataset. This ability to generalize to novel combinations of the elements observed during training is referred to as compositional generalization. A new train and test split of the Spider dataset is proposed based on Target Maximum Compound Divergence (TMCD) [26]. Spider-SSP consists of $3,282$ training instances and $1,094$ testing instances, and the databases are shared between the training and testing instances.

CSpider The CSpider [27] dataset is a Chinese variant of Spider [23] by translating the English NL questions in Spider into Chinese. Similar to Spider, CSpider consists of the same question-SQL pairs as in the Spider dataset.

SQUALL The SQUALL [36] dataset is an extension of WIKITABLEQUESTIONS [37], which enriches the $11,276$ samples from the training set of WIKITABLEQUESTIONS by providing hand-crafted annotations including both SQL queries and the labeled alignments between NL question tokens as well as the corresponding SQL fragments. In total, SQUALL contain $15,620$ instances, which are split into $9,030$ instances for training, $2,246$ instances for validation, and $4,344$ instances for testing.

DuSQL The DuSQL [28] dataset is a larges-scale Chinese corpus for cross-domain text-to-SQL parsing, which consists of $23,797$ NL-SQL pairs along with 200 databases and 813 tables belonging to more than 160 domains. Different from most previous corpora that are manually annotated, the SQL queries in DuSQL are automatically generated via production rules from the grammar.

2.3.2 Multi-Turn T2S Corpora

ATIS The original ATIS [29, 30] dataset is a collection of user questions asking for flight information on airline travel inquiry systems along with a relational database that contains information about cities, airports, flights, and so on. Most of the posted questions can be answered by querying the database with SQL queries. Since the original SQL queries are inefficient to be executed by using the IN clauses, the SQL queries are further modified by [21] while keeping the output of the SQL queries unchanged. In total, there are 5,418 NL utterances with corresponding executable SQL queries, where 4,473 utterances for training, 497 for development and 448 for testing.

SParC The SParC [31] dataset is a large-scale cross-domain context-dependent text-to-SQL corpus, which contains about 4.3k question sequences including 12k+ question-SQL pairs along with 200 complex databases belonging to 138 domains. SParC is built on the Spider [23], where each question sequence is based on a question from Spider by asking inter-related questions. After obtaining the sequential questions, a SQL query is manually annotated for each question. Following Spider, SParC is split into training, development and test sets with a ratio of 7:1:2, such that each database appears in only one set.

CoSQL The CoSQL [32] dataset is the first large-scale cross-domain conversational text-to-SQL dataset created under

the WOZ setting, which consists of about 3k dialogues including 30k+ turns and 10k+ corresponding SQL queries along with 200 complex databases belonging to 138 domains. In particular, each conversation simulates a DB query scenario where the annotators working as DB users issue NL questions to retrieve answers with SQL queries. The sequential NL questions can be used to clarify historical ambiguous questions or notify users of unanswerable questions. Similar to Spider [23] and SParC [31] , CoSQL is also split into training, development and test sets with a ratio of 7:1:2, such that each database appears in only one set.

CHASE The CHASE [33] dataset is a large-scale context-dependent chinese text-to-SQL corpus, which is composed of $5,459$ coherent question sequences including $17,940$ questions with their SQL queries. The context-dependent question-SQL pairs span over 280 relational databases. CHASE has to variants: CHASE-C and CHASE-T. Specifically, CHASE-C collects 120 Chinese relational databases from DuSQL [28] and creates 2003 question sequences as well as their SQL queries from scratch. CHASE-T is created by translating the 3456 English questions sequences and 160 databases from SParC [31] into Chinese. The CHASE dataset is split into $3,949/755/755$ samples for training, validation and testing, such that a database appears in solely one set.

3 Single-turn T2S Parsing Approaches

Deep learning has long been dominant in the field of text-to-SQL parsing, yielding state-of-the-art performances. In this manuscript, we provide a comprehensive review of recent neural network-based approaches for text-to-SQL parsing. A typical neural text-to-SQL method is usually based on the sequence-to-sequence (Seq2Seq) model [3], in which an encoder is devised to capture the semantics of the NL question with a real-valued vector and a decoder is proposed to generate the SQL query token by token based on the encoded question representation. As illustrated in Table 3, we divide the downstream text-to-SQL parsing methods into several primary categories based on the **encoder** and the **decoder**. Next, we describe each category of the text-to-SQL parsing methods in detail.

3.1 Encoder

The first goal of the encoder is to learn **input representation**, jointly representing the NL question and table schema representations. The second goal of the encoder is to perform **structure modelling**, since the text-to-SQL parsing task is in principle a highly structured task.

3.1.1 Input Representation

As stated in Section 2.1, there are two types of input information to be considered for text-to-SQL parsing: the NL question and the table schemas, which are jointly represented by the encoder. Generally, the input representation learning methods can be divided into two primary categories, including LSTM-based [5, 49] and Transformer-based [6] methods.

LSTM-based Methods Motivated by the significant success in text representation learning, LSTM-based methods [5, 49] are widely used to learn contextualized representations of input NL question and table schema, which are then passed into the decoder for generating SQL query. TypeSQL [38], Seq2SQL [22] and SyntaxSQLNet [39] work on the stand-alone question-SQL pairs and adopt the bidirectional LSTM (Bi-LSTM) to learn semantic representations of the input sequence which is the concatenation of the NL question and the column names. IRNet [4] encodes the NL question and the table schema by using two separate Bi-LSTM encoders. In particular, the two Bi-LSTM encoders take as input the word embeddings and the corresponding schema linking type embeddings, where the schema linking type embeddings are obtained by applying n-gram string matching to identify the table and column names mentioned in the NL question.

Transformer-based Methods Recently, the Transformer-based [6] models have shown state-of-the-art performances on text representation learning for multiple natural language processing (NLP) tasks. There are also several text-to-SQL parsing methods such as SQLova [10] and SLSQL [50] that extend BERT [11] and RoBERTa [51] for encoding the NL question together with the table and column headers. Generally, the Transformer-based encoders follow a three-step procedure.

First, the NL question and the database schema are concatenated and taken as the integrated input sequence of the encoder. Formally, the input sequence can be formulated as $\mathcal{I} = ([\text{CLS}]; q_1; \ldots; q_{|Q|}; [\text{SEP}]; s_1; [\text{SEP}]; \ldots; [\text{SEP}]; s_{|\mathcal{T}|+|\mathcal{C}|})$, where $[\text{CLS}]$ and $[\text{SEP}]$ indicate the pre-defined special tokens as in [11]. The input sequence can be extended to the multi-turn setting by sequentially concatenating current questions, dialog history and schema items [52].

Second, the pre-trained language models (PLMs) such as BERT [11] and RoBERTa [51] can significantly boost parsing accuracy by enhancing the generalization of the encoder and capturing long-term token dependencies. In general, for question tokens, the output hidden states from the final layer of the Transformer block in the BERT [11] or RoBERTa models are considered as the contextualized representations of question tokens. For each database schema item, the output hidden state of its front special token $[\text{SEP}]$ is regarded as the table or column header representation.

Third, leveraging flexible neural architecture on the top of PLMs can further enhance the encoder's output representations with strong expressive ability. For example, SQLova [10] and SDSQL [16] further stack two Bi-LSTM layers on the top of output representations of BERT [11]. GAZP [53] proposes an additional self-attention layer [6] on the top of a Bi-LSTM layer to compute the intermediate representations. RYANSQL [7] sequentially employs the convolutional neural network [54] with dense connection [9] and a scaled dot-product attention layer [6] on the top of BERT [11] to align question tokens with columns. It is noteworthy that the parameters of convolutional neural network are shared across the NL question and columns. BRIDGE [43] encodes the input sequence with BERT [11] and lightweight subsequent layers (i.e., two Bi-LSTM layers). In addition, dense look-up features are applied to represent meta-data information of the table schema such as primary key, foreign key and

TABLE 3
The representative downstream text-to-SQL parsing approaches. EM denotes the exact match accuracy on the Spider [23] data for the latest submissions. The columns denote the most important architecture decisions (SL - Schema Linking, DSL - Domain Specific Language).

Model	Encoder				Decoder							EM Dev	EM Test	EX Dev	EX Test
	SL	LSTM	Transformer	GNN	LSTM	Transformer	Grammar	Sketch	DSL	Constrained Decoding	Re-Ranking				
Seq2Seq baseline [23]		✓			✓							1.8	4.8	-	-
TypeSQL[38]	✓	✓						✓				8.9	8.2	-	-
SyntaxSQLNet [39]				✓	✓		✓					25.0	-	-	
GNN [40]	✓			✓	✓		✓					51.3	-	-	-
EditSQL[41]		✓	✓		✓							57.6	53.4	-	-
Bertrand-DR[42]		✓	✓		✓						✓	58.5	-	-	-
IRNet[4]	✓				✓		✓		✓			61.9	54.7	-	-
RYANSQL[7]								✓				66.6	58.2	-	-
BRIDGE[43]	✓		✓		✓		✓					70.0	65.0	70.3	68.3
RATSQL[12]	✓		✓		✓		✓					69.7	65.6	-	-
SMBOP [44]	✓		✓			✓	✓					69.5	71.1	75.0	71.1
ShadowGNN [45]	✓		✓		✓		✓		✓			72.3	66.1	-	-
RaSaP[17]	✓		✓			✓	✓					74.7	69.0	-	70.0
SADGA[13]	✓			✓	✓		✓					73.1	70.1	-	-
DT-Fixup[46]	✓		✓			✓						75.0	70.9	-	-
T5-Picard[47]			✓	✓		✓				✓		75.5	71.9	79.3	75.1
LGESQL[14]	✓			✓	✓		✓					75.1	72.0	-	-
S^2SQL[48]	✓			✓	✓		✓					76.4	72.1	-	-

datatype. These meta-data features are further fused with the BERT [11] encoding of the schema component via a feed-forward layer.

3.1.2 Structure Modelling

The development of large cross-domain datasets such as WikiSQL [22] and Spider [23] results in the realistic generalization challenge to deal with unseen table schemas. Each NL question corresponds to a multi-table database schema. The training and testing sets do not share overlapped databases. The challenge of the generalization requires the text-to-SQL parsing methods to encode the NL question and the table schema into representations with powerful expressive ability from three aspects.

- First, the encoder should be able to recognize NL tokens used to refer to tables and columns either implicitly or explicitly, which is called schema **linking structure** – aligning entity mentions in the NL question to the mentioned schema tables or columns.
- Second, the encoded representations should be aware of the **schema structure** information such as the primary keys, the foreign keys, and the column types.
- Third, the encoder should be able to perceive complex variations in the NL question, i.e., **question structure**.

Graph are the best form of data to express the complex structure in the text-to-SQL parsing task. Recently, several graph-based methods [40, 12, 14] have been proposed to reason over the NL question tokens and schema entities, and model the complex input representation. These methods consider the NL question tokens and schema items as multi-typed nodes, and the structural relations (edges) among the nodes can be pre-defined to express diverse intra-schema relations, question-schema relations and intra-question relations.

Linking Structure As illustrated in Figure 2, schema linking aims at identifying references of columns, tables and condition values in NL questions [55]. The text-to-SQL parsers should learn to detect table or column names mentioned in NL questions by matching question tokens with the schema, and the identified tables or columns are then utilized to generate SQL queries. Intuitively, schema linking facilitates both cross-domain generalizability and complex SQL generation, which have been regarded as

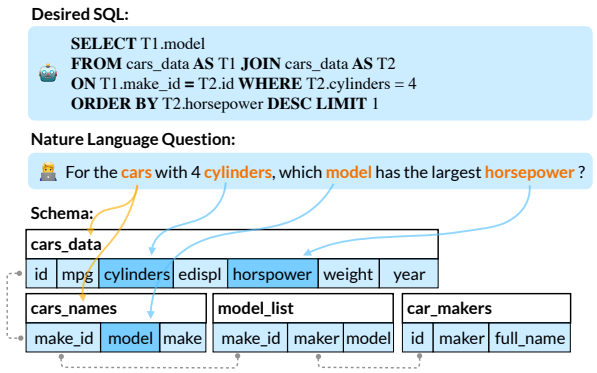

Fig. 2. Example of schema linking structure used in [12].

the current bottleneck of text-to-SQL parsing. [50] demonstrates that more accurate schema linking conclusively leads to better text-to-SQL parsing performance. Conventional schema linking can be extracted by means of rules or string matching. For example, IRNET [4] takes the extracted linking information directly as input. SDSQL [16] leverages the extracted linking information as a label for multi-task learning. IESQL [56] employs a conditional random filed (CRF) layer [57] for the question segment, which transforms the linking task into a sequence tagging task. However, the aforementioned schema linking methods cannot capture the comprehensive semantic relationship between the NL question and table schema. Another popular approach [41] proposes the cross-attention to implicitly learn relationships between the NL question and schema representations.

Recently, the graph-based linking approaches [12, 44, 17, 45, 14] have been proposed to reason over the NL question tokens and schema entities, and model the complex input representation. These methods consider the NL question tokens and schema items as multi-typed nodes, and the structural relations (edges) among the nodes can be pre-defined to express diverse intra-schema relations, question-schema relations and intra-question relations. In particular, in RATSQL [12], the graph is constructed based on two kinds of relations (i.e., name-based linking and value-based linking), where the name-based linking refers to partial or exact occurrences of table/column names in the same NL question and the value-based linking refers

what do they mean by values here?

to the question-schema alignment that occurs when the NL question mentions any values appearing in the schema and the desired SQL. The relation-aware self-attention mechanism [58] is then proposed for graph representation learning, which exploits global reasoning over the constructed graph. Specifically, given a sequence of token representations, the relational-aware self-attention computes a scalar similarity score between each pair of token representations $e_{ij} \propto \mathbf{u}_i W_Q \left(\mathbf{u}_j W_K + \mathbf{r}_{ij}^K\right)$, where $\mathbf{u}_i$ and $\mathbf{u}_j$ denote the graph nodes, and the term $\mathbf{r}_{ij}^K$ denotes an embedding that represents a relation between $\mathbf{u}_i$ and $\mathbf{u}_j$ from a closed set of possible relations. To enhance the model generalization capability for unseen or rare schemas, ShadowGNN [45] alleviates the impact of the domain information by abstracting the representations of the NL question and the SQL query before applying the relation-aware graph computation [12]. In addition, several works have been devoted to tackle the challenge of heterogeneous graph encoding for the text-to-SQL parsing. LGESQL [14] constructs an edge-centric graph from the node-centric graph as in RATSQL [12], which explicitly considers the topological structure of edges. The information propagates more efficiently by considering both the connections between nodes and the topology of directed edges. Two relational graph attention networks (RGANs) [59] are devised to model the structure of the node-centric graph and the edge-centric graph respectively, mapping the input heterogeneous graph into token representations.

Schema Structure It is intuitive to leverage a relational graph neural network (GNN) to model the relations in the relational database, propagating the node information to its neighbouring nodes. The GNN-based methods help to aggregate feature information of neighboring nodes, making the obtained input representation more powerful. SchemaGNN [40] first converts the database schemas to a graph by adding three types of edges: the foreign-primary key relation, the column-in-table relation and the table-own-column relation. The constructed graph is softly pruned conditioned on the input question, which is then fed into gated GNNs [60] to learn schema representations being aware of the global schema structure. Furthermore, GlobalGNN [61] proposes a similar approach by employing a graph convolutional network (GCN) to learn schema representations, where a relevance probability conditioned on the question is computed for every schema node. Some advanced studies, such as RAT-SQL [12] and LGESQL [14], also learn the structure of the schema as a unique edge in the graph, demonstrating the indispensability of the schema structure for text-to-SQL parsing.

pass db schema as a graph by encoding entity relations in the edges

Question Structure S^{2}SQL [48] investigates the importance of syntax in text-to-SQL encoder, and proposes a flexible and robust injection method. It leverages three induct dependency types, i.e., `Forward`, `Backward`, `NONE`, which stack multi-layer transformers to implicitly model complex question structure. In addition, a decoupling constraint is employed to induce the diverse relation embedding. SADGA [13] constructs the question graph conditioned on the dependency structure and the contextual structure of the NL question sequence, and builds the schema graph conditioned on the schema structure. Specifically, three different types of links are defined for question tokens to construct the graphs: the 1-order word dependency (i.e., the relation between two consecutive words), the 2-order word dependency, and the parsing-based dependency that captures syntactic relations among the NL question words. Then, the structure-aware aggregation approach is proposed to capture the alignment between the constructed graphs through two-stage linking. The unified representations are learned by aggregating the information via a gated-based mechanism.

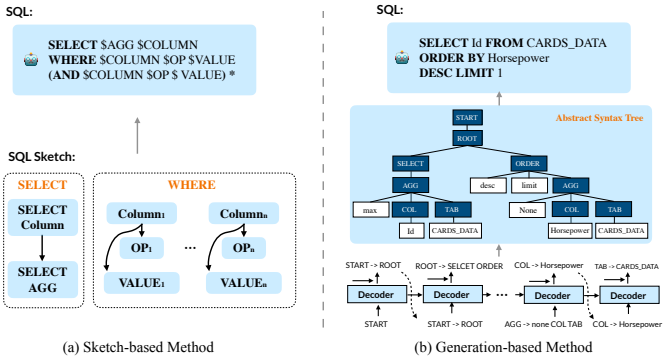

Fig. 3. Example of SQL sketch used in [15].

3.2 Decoder

The decoder used in existing Text-to-SQL parsing models can be divided into two categories: **sketch-based methods** and **generation-based methods**. In this section, we provide a comprehensive overview of the two types of decoder architectures.

3.2.1 *Sketch-based Methods*

The sketch-based methods decompose the SQL generation procedure into sub-modules, e.g., `SELECT` column, `AGG` function, `WHERE` value. For example, SQLNet [15] employs the SQL sketch. The tokens `SELECT`, `WHERE` and `AND` indicate the SQL keywords, and the following components indicate the types of prediction slots to be filled. For example, the `AGG` slot indicates the slot to be filled with either an empty token or one of the aggregation operators such as `SUM` and `MAX`; the `VALUE` slot needs to be filled with a sub-string of the question; the `COLUMN` slot needs to be filled with a column name; the `OP` slot needs to be filled with operations such as $>$, $<$, $=$). These slots are later gathered together and interpreted to generate the final SQL query. Each slot has separate model which does not share their trainable parameters and is responsible for predicting a part of the final SQL independently. Specifically, for the `COLUMN` slot, a column attention mechanism [15] is applied to reflect the most relevant information in NL questions when prediction is made on a particular column. For the `OP` slot, predicting its value is a 3-way classification task ($>$, $<$, $=$). For the `VALUE` slot, [15] employs a Seq2Seq structure to generate the sub-string of the NL question.

SQLova [10] and SDSQL [16] modify the syntax-guided sketch used in [15]. The proposed sketch-based decoder consists of six prediction modules, including `WHERE-NUMBER`, `WHERE-COLUMN`, `SELECT-COLUMN`,

SELECT-AGGREGATION, WHERE-OPERATOR, and WHERE-VALUE. The specific role of each action is described as follows:

- SELECT-COLUMN identifies the column in "SELECT" clause from the given NL question.
- SELECT-AGGREGATION identifies the aggregation operator for the given select-column prediction.
- WHERE-NUMBER predicts the number of "WHERE" conditions in SQL queries.
- WHERE-COLUMN calculates the probability of generating each columns for the given NL question.
- WHERE-OPERATOR identifies the most probable operators given where-column prediction among three possible choices (>, =, <).
- WHERE-VALUE identifies which tokens of a NL question correspond to condition values for the given "WHERE" columns.

executing guided filtering strategy?

In the SQL query generation stage, an execution-guided decoding strategy [62] is utilized to exclude the non-executable partial SQL queries from the output candidates. TypeSQL [38] further improves the above approach by declining the number of modules. TypeSQL chooses to combine the select-column module and the where-column module into a single module since their prediction procedures are similar, and the where-column module depends on the output of the select-column module. In addition, the where-operator and where-value modules are combined together because the predictions of these two modules depend on the outputs of the where-column module. Generally, the sketch-based approaches are fast and guaranteed to conform to correct SQL syntax rules. However, it is difficult for these approaches to handle complex SQL statements such as multi-table JOINs, nested queries, and so on. Thus, the sketch-based approaches are popular on the WikiSQL [22] dataset, but are difficult to be applied on the Spider [23] dataset which involves complex SQLs. Only RYANSQL [7] implements complex SQL generation by recursively applying the sketch method.

3.2.2 Generation-based Methods

On the other hand, the generation-based approaches are based on the Seq2Seq model to decode SQL, which are more preferable for complex SQL scenarios than the sketch-based approaches. For example, Bridge [43] uses a LSTM-based pointer-generator [63] with multi-head attention and copy mechanism as the decoder which is initiated with the final state of the encoder. At each decoding step, the decoder performs one of the following actions: generating a token from the vocabulary V, copying a token from the question Q, or copying a schema component from the database schema S.

Since the above generation-based approaches may not generate SQL queries with correct grammar, some advanced methods [4, 12] generate the SQL as an abstract syntax tree (AST) [64] in the depth-first traversal order [65]. In particular, these methods employ an LSTM decoder to perform a sequence of three types of actions that either expand the last generated node into a grammar rule, called APPLY-RULE action or choose a column/table from the schema when completing a leaf node, called SELECT-COLUMN action and SELECT-TABLE action respectively. These actions can construct the corresponding AST of the target SQL query. Specifically, APPLY-RULE applies a production rule to the current derivation tree of a SQL query and expands the last generated node with the grammar rule. The probability distribution is computed by a softmax classification layer for the pre-defined abstract syntax description language (ASDL) rules. SELECT-COLUMN and SELECT-TABLE complete a leaf node by selecting a column c or a table t from the database schema respectively by directly copying the table and column names from database schema via copy mechanism [63].

There are also several works [46, 66, 67] which neglect the SQL grammar during the decoding process, by leveraging the powerful large scale pre-trained language model like T5 [68] finetuned on the text-to-SQL training set for SQL query generation. Formally, the transformer-based decoder follows the standard text generation process and produces the hidden state in step t for generating the t-th token as described in [6]. An affine transformation is then applied on the learned hidden state to obtain prediction probability over the target vocabulary V for each word. In addition, as revealed in [42], for some cases, although the best generated SQL is in the candidate list of beam search, it is not at the top of the candidate list. Therefore, a discriminative re-ranker (**Re-Ranking**) strategy is introduced to extract the best SQL query from the candidate list predicted by the text-to-SQL parser. The re-ranker is constructed as a BERT fine-tuned classifier that is independent of schema, and the probability of the classifier is utilized as the score for the query to re-rank. Some studies [4, 45] introduce a domain specific language (**DSL**) serving as an intermediate representation to bridge the NL question and the SQL query. The methods reveal that there is a invariable mismatch between the intentions conveyed in natural language and the implementation details in SQL when the resulting SQL is processed to tree-structured form. Hence, the key idea of DSL is to omit the implementation details of the intermediate representations. IRNet [4] presents a grammar-based neural model to generate a SemQL query as the intermediate representation bridging the NL question and the SQL query, and a SQL query is then inferred from the generated SemQL query with domain knowledge. Furthermore, a **constrained decoding** strategy is proposed in Picard [47] via incremental parsing, which facilitates the parser to identify valid target sequences by rejecting inadmissible tokens at each decoding step.

4 Multi-turn T2S Parsing Approaches

Compared to the single-turn T2S setting, the multi-turn T2S setting emphasizes the usage of contextual information (historical information), which can be incorporated in both the encoder and the decoder. Next, we describe how the contextual information is leveraged.

4.1 Encoder

The encoder processes the contextual information for input representation learning. In addition, the linking structure and the schema structure are considered during the encoding phase.

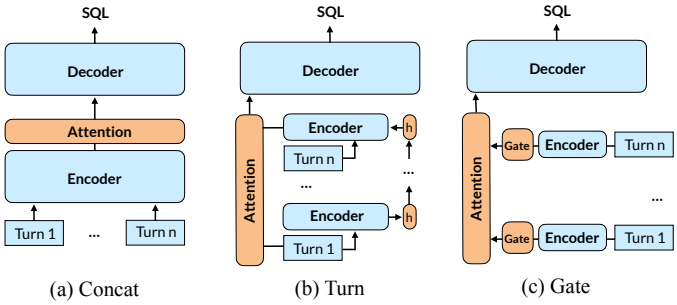

Fig. 4. Different contextual NL questions encoding strategies for multi-turn T2S parsing [69].

4.1.1 Multi-turn Input Representation

With regard to the multi-turn representation learning, previous works mainly focus on two aspects, including (i) how to learn high-quality contextual question and schema representations and (ii) how to effectively encode the historical SQL queries.

For the contextual question and schema representation learning, as shown in Fig. 4, [69] investigates the impact of different contextual information encoding methods on the multi-turn T2S parsing performance, including (a) contacting all the NL questions within each question sequence as input, (b) using a turn-level encoder to deal with each question, and (c) devising a gate mechanism to balance the importance of each historical question. Typically, EditSQL [41] utilizes two separate Bi-LSTMs [5] for encoding the NL questions and the table schema respectively. Specifically, for the question at each turn, EditSQL first utilizes a Bi-LSTM to encode the question tokens. The output hidden states are then fed into a dot-product attention layer [70] over the column header embeddings. The relationships among the multi-turn questions are captured by the turn attention mechanism. At the current turn, the dot-product attention between the current question and previous questions in the history is computed, and the weighted average of previous question representations is then added to the current question representation to form the context-aware question representation. For each column header, the table name and the column name are concatenated and passed into a Bi-LSTM layer. The output hidden states are then fed into a self-attention layer [6] to better capture the internal structure of the table schemas such as foreign keys. In addition, the self-attention vector and the question attention vector are concatenated and fed into the second Bi-LSTM layer to obtain the final column header representation. [71] learns a context memory controller to maintain the memory by keeping the cumulative meaning of the sequential NL questions and using an external memory to represent contextual information. [72] decouples the multi-turn T2S parsing into two pipeline tasks: question rewriting and single-turn T2S parsing. The question rewriting (QR) module aims to generate semantic-completion question based on the dialogue context, which concatenates the dialogue history and the latest question as the input of the QR module. The goal of the QR module is to generate a simplified expression based on the latest question and the dialogue history. The single-turn T2S parser predicts the corresponding SQL query with the simplified expression generated by the QR module, which chooses a pre-trained BART as question rewriter and RAT-SQL [12] as the single-turn text-to-SQL parser.

For the historical SQL input encoding, HIE-SQL [73] treats the logic-forms of the SQL query as an another modality for the NL question and incorporates additional SQL encoders to capture the semantic connection between the SQL query and the NL question. It produces comprehensive representations of the previously predicted SQL queries to improve the contextual representation.

4.1.2 Multi-turn Structure Modelling

Different from the single-turn setting, the multi-turn setting requires the integration of contextual inductive bias into the structural modelling.

Linking Structure R^2SQL [74] focuses on the uniqueness of contextual linking structures, introducing a novel dynamic graph framework to efficiently model contextual questions, database schemas, and the complex linking structures between them. A decay mechanism is applied to mitigate the impact of historical links on the current turn.

Schema Structure [41] exploits the conversation history by editing the previous predicted SQL to improve the generation quality. It focuses on taking advantages of previous question texts and previously predicted SQL query to predict the SQL query at the current turn. Concretely, the previous question is first encoded into a series of tokens, and the decoder then predicts a switch to alter it at the token level. This sequence editing approach simulates the changes at token level and is hence resistant to error propagation. IGSQL [75] points out that it should not only take historical user inputs and previously predicted SQL query into consideration but also utilize the historical information of database schema items. Therefore, IGSQL proposes a database schema interaction graph encoder to learn database schema items together with historical items, keeping context consistency for context-dependent text-to-SQL parsing. The cross-turn schema interaction graph layer and intra-turn schema graph layer update the schema item representations by using the previous turn and the current turn respectively. IST-SQL [76] deals with the multi-turn text-to-SQL parsing task inspired by the task-oriented dialogue generation task. IST-SQL defines, tracks and utilizes the interaction states for multi-turn text-to-SQL parsing, where each interaction state is updated by a state update mechanism based on the previously predicted SQL query.

4.2 Decoder

For the multi-turn setting, most previous methods [41, 77, 74] employ a LSTM decoder with attention mechanisms to produce SQL queries conditioned on the historical NL questions, the current NL question, and the table schema. The decoder takes the encoded representations of the current NL question, SQL-states, schema-states, and last predicted SQL query as input and apply query editing mechanism [41] in the decoding progress to edit the previously generated SQL query while incorporating the context of the NL questions and schemas. To further alleviate the challenge that the tokens from the vocabulary may be completely irrelevant

to the SQL query, separate layers are used to predict SQL keywords, table names and question tokens. A softmax operation is finally used to generate the output probability distribution.

5 Pre-training for Text-to-SQL Parsing

Pre-trained language models (PLMs) have proved to be powerful in enhancing text-to-SQL parsing and yield impressive performances, which benefit from the rich knowledge in large-scale corpus. However, as revealed in previous works [18, 20], there are intrinsic differences between the distribution of tables and plain texts, leading to sub-optimal performances of general PLMs such as BERT [11] in text-to-SQL parsing. Recently, several studies have been proposed to alleviate the above limitation and build tabular language models (TaLMs) by simultaneously encoding tables and texts, which show improved results on downstream text-to-SQL parsing tasks. In this section, we provide a comprehensive review of existing studies on pre-training for text-to-SQL parsing from the perspectives of pre-training data construction, input feature learning, pre-training objectives and the backbone model architectures.

5.1 Pre-training Data Construction

Insufficient training data is an important challenge to learn powerful pre-trained tabular language models. The quality, quantity and diversity of the pre-training data have significant influence on the general performance of the pre-trained language models when applied to downstream text-to-SQL parsing tasks. Although it is easy to collect a large amount of tables from Web (e.g., Wikipedia), obtaining high-quality NL questions and their corresponding SQL queries over the collected tables is a labor-intensive and time-consuming process. Recently, there have been plenty of studies to generate pre-training data for text-to-SQL parsing manually or automatically. Next, we discuss the previous pre-training data construction methods from three perspectives: table collection, NL question generation, and logic form (SQL) generation.

5.1.1 Table Collection

We briefly introduce several sources that have been extensively used for table collection. WikiTableQuestion [37] is a representative corpus which is composed of 22,033 question-answer pairs on 2,108 tables, where the tables are randomly collected from Wikipedia with at least five columns and eight rows. The tables are pre-processed by omitting all the non-textual information, and each merged cell is duplicated to keep the table valid. In total, there are 3,929 distinct column headers (relations) among the 13,396 columns. WikiSQL [22] is a collection of 80,654 hand-crafted question-SQL pairs along with 24,241 HTML tables collected from Wikipedia. The tables are collected from [78], and the small tables that have less than five columns or five rows are filtered. WDC WebTables [79] is a large-scale table collection, which contains over 233 million tables and has been extracted from the July 2015 version of the CommonCrawl. Those tables are classified as either relational (90 million), entity (139 million), or matrix (3 million). WikiTables [80] contains 1.6 million high-quality relational Wikipedia tables constructed by extracting all HTML tables from Wikipedia which had the class attribute "wikitable" (used to easily identify data tables) from the November 2013 XML dump of English Wikipedia. ToTTo [81] is an open-domain English table-to-text dataset with over 120,000 training examples which contains 83,141 corresponding web tables automatically collected from Wikipedia using heuristics with various schema spanning various topical categories. WebQueryTable [82] is composed of 273,816 tables by using 21,113 web queries, where each query is used to search the Web pages and the relevant tables are obtained from the top ranked Web pages. Spider [23] is a large-scale text-to-SQL dataset, which contains 200 databases belonging to 138 different domains. In particular, the databases come from three resources: (i) 70 complex databases are collected from SQL tutorials, college database courses, and textbook examples, (ii) 40 databases are collected from the DatabaseAnswers[1], (iii) 90 databases are selected from WikiSQL [22].

5.1.2 Natural Language Question Annotation

So far, different question annotation methods have been introduced to annotate natural language questions based on the collected databases. Generally, the methods can be divided into three categories: sampling-based methods, template-based methods, and generation-based methods.

Sampling-based Methods Many works produce the NL questions in pre-training data by extracting text-table pairs from Wikipedia. Concretely, TaPas [19] creates the pre-training corpus by collecting text-table pairs from Wikipedia, where there are about 6.2M tables and 21.3M text snippets. In particular, the table captions, segment titles, article descriptions, article titles, and textual table segments are extracted as the text snippets of the corresponding tables. StruG [83] directly collects about 120k NL web tables and corresponding text descriptions from the ToTTo dataset [81], which extracts the text-table pairs from Wikipedia by employing three heuristics: number matching, cell matching, and hyperlinks.

Template-based Methods There are several works that generate the NL questions automatically by using templates or rules. Grappa [20] constructs question-SQL templates by extracting entity mentions of SQL operations and database schemas. By leveraging the created templates on randomly sampled tables, a large amount of question-SQL pairs can be synthesized automatically. SCoRe [84] leverages only 500 samples from the development set of SParC [31] to derive utterance-SQL generation grammars consisting of a list of synchronous question-SQL templates and follow-up question templates. Finally, about 435k text-to-SQL conversations are synthesized for context-dependent text-to-SQL pre-training.

Generation-based Methods Several works have been introduced to generate NL questions from entity sequences automatically with text generation models. For example, [85] proposes a cross-domain neural model, which accepts a table schema and samples a sequence of entities to be appear

1. http://www.databaseanswers.org/

TABLE 4
The pre-training data construction.

	Table								Question		
	WIKITABLEQUESTIONS	WIKISQL [22]	WIKITABLE	ToTTo	Spider [23]	WebTable	Wikipedia	GITHUB	ToTTo	Wikipedia	Spider [23]
TaBERT[18]						✓	✓				
TAPAS [19]							✓			✓	
GRAPPA [20]		✓	✓		✓						
GAP								✓			✓
STRUG				✓					✓		
MATE							✓			✓	
TAPEX	✓										
TableFormer							✓			✓	
SCORE		✓	✓		✓					✓	

in the NL question, to transform the entity sequence to the NL question. Specifically, the T5 model [68] is first fine-tuned on a small corpus containing entity-question pairs and then applied to generate NL questions given entity sequences. For example, given the input entity sequence "*department management : head name text — head age number — head born state text*", the T5 model is likely to output the NL question "*List the name, born state and age of the heads of departments ordered by age.*". In addition, [86] introduces a generative model to generate utterance-SQL pairs, which leverages the probabilistic context-free grammar (PCFG) to model the SQL queries and then employs a BART-based translation model to transform the logical forms to NL questions. For example, for a given input SQL sequence "*select area where state_name = 'texas'*", the generative model outputs the NL question "*what is the area of Texas?*". GAZP [53] also generates utterances corresponding to these logical forms using the generative model. In addition, the input and output consistency of the synthesized utterance is verified. Specifically, we parse the generated queries into logical forms and keep the queries whose parses are equivalent to the corresponding original logical forms.

There are also some studies that invite annotators to manually create natural questions and corresponding SQL queries without leveraging templates or rules, such that the generated NL questions are natural and diverse. In particular, Spider [23] generates 10,181 NL questions and 5,693 unique SQL queries over 200 databases by considering three primary aspects: SQL pattern coverage ensuring that enough SQL samples are obtained to cover all common SQL patterns, SQL consistency ensuring that the semantically equivalent NL questions share the same SQL query, and question clarity ensuring that the vague or too ambiguous questions are not included.

5.1.3 SQL Annotation

Generally, the SQL annotation methods can be divided into three primary categories: logic perturbation, SQL template instantiation and hierarchical synthesis pipeline.

Logic Perturbation Due to the expensive process of obtaining SQL queries, the logic perturbation-based approaches have been proposed to augment the SQL queries by performing random logic perturbation according to hand-tuned rules [87]. In particular, [87] generally enumerates the perturbations of each given SQL query based on hand-tuned rules that follow three kinds of logic inconsistencies: (i) logic shift aiming to generate the questions and logical forms that are logically distinct from the original ones, (ii) the phrase and number changes aiming to modify the appointed numerical values and phrases in logical forms, and (iii) entity insertion, swapping and deletion that ignores the entity mention in logical forms, inserts new entities into logical forms, or swaps any two entities within in a logical form. There are three reasons for automatically generating more SQL queries by perturbing the logical forms. First, the regular structures of logical forms make the procedure of logical corruption controllable. Second, we can easily validate the perturbed logical forms with corresponding grammar checker and parser. Third, it is easy to obtain the corresponding questions of the generated SQL queries with minor modification given the original question-SQL pair.

SQL Template Instantiation There are several studies that apply the SQL template instantiation methods to automatically generate SQL queries based on existing templates [88, 53] or self-defined synchronous context-free grammar (SCFG) [20]. [88] utilizes the production rules of the SQL grammar defined in the SUQALL dataset [36] for SQL annotation. Given a SQL template, the headers and cell values of the tables are uniformly select to fill the template. GAZP [53] samples logical forms by leveraging a grammar, such as the SQL grammar over the database schema. First, GAZP creates coarse templates by pre-processing the SQLs via the SQL grammar and further replacing the mentions of columns with typed slots. Then, the slots of each coarse templates are filled-in by new database contents. Instead of completely relying on the existing templates, GRAPPA [20] learns from the examples in Spider [23] and designs a new SCFG which is then applied on a large number of existing tables to produce new SQL queries. The key idea behind this method is to define a set of non-terminal types for operations, table names, cell values, column names, and then substitute the entities with corresponding non-terminal symbols in the SQL query to form a SQL production rule. The SQL template instantiation methods often heavily depend on limited templates, and it is hard for them to generate diverse SQL queries with new compositions.

Hierarchical Synthesis Pipeline Different from the above mentioned approaches that synthesize new SQL queries based on hand-crafted or induced rules and templates, the hierarchical synthesis pipeline approaches are based on the large-scale pre-trained language models (PLMs) [68], which are motivated by the fact that PLMs can improve model generalization by incorporating additional diverse samples into the training corpus without labour-intensive manual

works. Concretely, [85] proposes a neural approach without grammar engineering but achieves high semantic parsing performance. To this end, the pre-trained text generation models such as T5 [68], which is fine-tuned on the text-to-SQL data, are implemented to map entity sequences sampled from the table schema to NL questions [85]. Then, the learned semantic parsers are then applied on the generated NL questions to produce the corresponding SQL queries. The overall data synthesis pipeline is easy to implement and achieves great diversity and coverage due to the usage of the large PLMs.

5.2 Input Encoding

In the text-to-SQL parsing task, the input often involves two parts: NL questions and table schemas, and the output would be the SQL queries. However, the textual data, tabular data and SQL queries are heterogeneous, which have different structures and formats. To be specific, the tabular data is generally distributed in two-dimensional structures with numerical values and words, while the SQL queries are usually composed of SQL keywords (e.g., "SELECT", "UPDATE", "DELETE", "INSERT INTO") and schema entities. Hence, it is non-trivial to develop a joint reasoning framework over the three types of data. In this section, we review the recent studies on heterogeneous input encoding of pre-training for text-to-SQL parsing.

5.2.1 Textual Data Encoding

Text encoding can be divided into dynamic and static types based on the word encoding in natural language processing. Some methods applied GloVe [89] to initialize the word embedding of each input item by looking up an embedding dictionary without the context such as RATSQL [12] and LGESQL [14]. However, the static embedding methods are still limited. Neither of the static methods are able to tackle the polysemy problem. In addition, the learned features are restricted by the pre-defined window size. With the development of the pre-train language models, some studies attempt to encode textual data with PLMs instead of static word embeddings. In particular, plenty of methods (e.g., TABERT [18], TAPAS [19], MATE [90], STRUG [83]) utilize the pre-trained BERT [11] as the encoder to get the contextualized word-level representations, and the parameters of BERT [11] are updated along with the training process. GRAPPA [20] uses ROBERTA [51] as the encoder. TAPEX [88] leverages both BART [91] encoder and decoder, while GAP [92] merely utilizes the BART encoder.

5.2.2 Tabular Data Encoding

Different from textual data, the tabular data is distributed in two-dimensional (2-D) structures. The table pre-training approaches need to first convert the 2-D table data into linearized 1-D sequence input before feeding the tabular data into language models. A common serialization method is to flatten the table data into a sequence of tokens in the row-by-row manner and then concatenate the question tokens before the table tokens for tabular pre-training, such as TAPAS [19], MATE [90], and TABLEFORMER [93]. TABERT [18] proposes content snapshots to encode a subset of table content which is most relevant to the input utterance. This strategy is then combined with a vertical attention mechanism, sharing the information among the cell representations in different rows. There are also some studies (e.g., STRUG [83], GRAPPA [20] and UnifiedSKG [94]) which only take the headers of tables as input without considering the data cells.

While the NLP models generally take the 1-D sequences as input, the positional encoding becomes crucial for tabular data to facilitate the neural models to better capture the structure information. Most previous pre-training methods such as TABERT [18], GRAPPA [20], and TAPEX [88] explored the global positional encoding strategy on the flattened tabular sequences. Nevertheless, in addition to the 1-D sequential positions, the tables have structured columns and rows which consists of critical two-dimensional and hierarchical information. The works such as TAPAS [19] and MATE [90] encode the row and column content based on column/row IDs. TABLEFORMER [93] decides whether two cells are in the same column/row and the column header, rather than considers the absolute order information of columns and rows in the tables.

5.3 Pre-training Objectives

Most existing pre-training models for text-to-SQL parsing employ either a single Transformer or a Transformer-based encoder-decoder framework as the backbone, and adopt different kinds of pre-training objectives to capture the characteristics of the text-to-SQL parsing task. As illustrated in Table 5, the pre-training objectives can be divided into five primary categories, including masked language modelling (MLM) [18, 19, 20, 92, 90, 93, 84], schema linking [20, 92, 83, 84], SQL executor [88], text generation [92] and context modelling [84]. Next, we will introduce the implementation details of each primary pre-training objective.

5.3.1 Masked Language Modelling

Existing works often explore different variants of masked language modeling (MLM) to guide the language models to learn better representations of both natural language and tabular data. Concretely, the MLM objectives can be divided into three primary categories: reconstructing the corrupted NL sentences, reconstructing the corrupted table headers or cell values, and reconstructing the tokens from the corrupted NL sentences and tables.

In particular, most pre-training models [18, 19, 20, 92, 90, 93, 84] adopt masked language modelling by randomly masking a part of the input tokens from NL sentences or table headers and then predicting the masked tokens. Then, the MLM loss is calculated by minimizing the cross-entropy loss between the original masked tokens and the predicted masked tokens. In addition, TABERT [18] also proposed a Masked Column Prediction (MCP) and a Cell Value Recovery (CVR) to learn the column representations of tables, where the MCP objective predicts the names and data types of masked columns and the CVR objective attempts to predict the original value of each cell in the masked column given its cell vector. GAP [92] devised a Column Recovery (CRec) objective to recovery the corresponding column name conditioned on a sampled cell value.

TABLE 5
The pre-training objectives for text-to-SQL parsing.

Models \ Objectives	Masked Language Modelling	Schema Linking	SQL Executor	Text Generation	Context Modelling
TaBERT [18]	✓ (MLM, MCP, CVR)				
TAPAS [19]	✓ (MLM)				
GRAPPA [20]	✓ (MLM)	✓ (SSP)			
GAP [92]	✓ (MLM, CRec)	✓ (CPred)		✓ (GenSQL)	
STRUG [83] []		✓ (CG, VG)			
MATE [95]	✓ (MLM)				
Tapex [88]			✓		
TableFormer [93]	✓ (MLM)				
SCORE [84]	✓ (MLM)	✓ (CCS)			✓ (TCS)

5.3.2 Schema Linking

Schema linking is a key component in text-to-SQL parsing, which learns the alignment between NL questions and given tables. In particular, schema linking aims at identifying the references of columns, tables and condition values in NL questions. It is especially important for complex SQL generation and cross-domain generalization, where the text-to-SQL parser should be aware of what tables and columns are involved in the NL question even when referring against with tables from arbitrary domains and modelling complex semantic dependencies between NL questions and SQL queries.

Recently, several pre-training objectives [20, 92, 83, 84] are devised to model the schema linking information by learning the correlations of NL questions and tables. GRAPPA [20] proposed a SQL Semantic Prediction (SSP) objective, which aims at predicting whether a column name appears in the SQL query and which SQL operation is triggered conditioned on the NL question and given table headers. The SSP objective is implemented by converting the SQL sequence labeling into operation classification for each column, which results in 254 possible operation classes. Similar to GRAPPA[20], SCORE [84] proposed a Column Contextual Semantics (CCS) objective, aiming to predict what operation should be performed on the given column. STRUG [83] proposed three structure-grounded objectives to learn the text-table alignment, including Column Grounding (CG), Value Grounding (VG) and Column-Value mapping (CV). Concretely, the CG objective is a binary classification task, aiming at predicting whether a column is mentioned in the NL question or not. The VG objective is also transformed to binary classification, which aims at predicting where a token is a part of a grounded value conditioned on the NL question and the table schema. To further align the grounded columns and values, the CG objective is devised to match the tokens in the NL question and the columns. Similar to the CG objective, GAP [92] also developed a Column Prediction (CPred) objective to predict whether a column is used in the NL question or not.

5.3.3 SQL Executor

Modelling structured tables plays a crucial role in text-to-SQL pre-training. TAPEX [88] proposed a SQL executor objective by pre-training a neural model to mimic a SQL executor on tables. Specifically, the neural SQL executor is learned to execute the SQL query and output the corresponding correct result, which requires the model to have deep understanding of the SQL queries and tables.

5.3.4 SQL Generation

The goal of text-to-SQL parsing is to translate NL questions into SQL queries that can be executed on the given tables. Therefore, it can be beneficial to incorporate the SQL generation objective into the pre-training methods so as to further enhance the downstream tasks. GAP [92] proposes a SQL generation objective to generate specific SQL keywords or column names in the appropriate positions rather than merely predict whether a column is mentioned or not.

5.3.5 Turn Contextual Switch

The above pre-training objectives primarily model standalone NL questions without considering the context-dependent interactions, which result in sub-optimal performance for context-dependent text-to-SQL parsing. SCORE [84] is the first representative pre-training method for context-dependent Text-to-SQL parsing. In particular, SCORE designs a turn contextual switch (TCS) objective to model the context flow by predicting the context switch label (from 26 possible operations) between two consecutive user utterances. Despite its effectiveness, the TCS objective ignores the complex interactions of context utterances, and it is difficult to track the dependence between distant utterances.

6 FUTURE DIRECTIONS

Despite the remarkable progress of previous methods, there remain several challenges for developing high-quality text-to-SQL parsers. Based on the works in this manuscript, we discuss several directions for future exploration in the field of text-to-SQL parsing.

6.1 Effective High-quality Training Data Generation

The current benchmark datasets for text-to-SQL parsing are still limited by the quality, quantity and diversity of training data. For instance, WikiSQL [22] contains a large amount of simple question-SQL pairs and single tables, which neglects the quality and diversity of the training instances. In particular, WikiSQL [22] has several limitations which are described as follows. First, the training, development and testing sets in WikiSQL [22] share the same domains, without concerning the cross-domain generalization ability of

the text-to-SQL parsers. Second, the SQL queries in WikiSQL [22] are simple, which do not contain complex operations, such as "ORDER BY", "GROUP BY", "NESTED" and "HAVING". Third, each database in WikiSQL [22] contains only one table, simplifying the text-to-SQL parsing. Spider [23] is a cross-domain and complex benchmark dataset for text-to-SQL parsing, which contains complicated SQL queries and databases with multiple tables in different domains. Spider [23] is proposed to investigate the ability of a text-to-SQL parser to generalize to not only new SQL queries and database schemas but also new domains. However, Spider [23] merely has about ten thousand samples, which is not large enough to build a high-quality text-to-SQL parser. There are also many data generation methods that apply rule-based methods to produce a large amount of question-SQL pairs. However, these automatically generated samples are of inferior quality and usually lose diversity. Therefore, how to construct text-to-SQL corpora with high quality, large-scale quantity and high diversity is an important future exploration direction.

6.2 Handling Large-scale Table/Database Morphology

The tables used in current benchmark corpora usually contain less than ten rows and columns by considering the input length limitation of current neural text-to-SQL models. However, in many real-world applications, the involved tables usually consist of thousands of rows and columns, which pose a big challenge to the memory and computational efficiency of existing neural text-to-SQL models. In particular, when the number or size of involved tables becomes too large, how to encode the table schemas and retrieve appropriate knowledge from large tables are challenging. Thus, more future efforts should be made on how to (i) develop effective text-to-models that can encode a long sequence of table schemas, and (ii) improve the execution efficiency of the generated SQL when involving a large-scale database.

6.3 Structured Tabular Data Encoding

Different from textual data, the tabular data involved in text-to-SQL parsing is distributed in 2-D structures. Most text-to-SQL parsing methods first convert the 2-D table data into the linearized 1-D sequence, which is then fed into the input encoder. Such linearization methods cannot capture the structural information of tables. In addition, most previous works primarily focus on web tables, while more other kinds of tabular data that contain hyperlinks, visual data, spreadsheet formulas, and quantities are not considered. It is non-trivial to learn high-quality representations from such diverse data types by directly encoding a linearized input sequence. It is worth exploring text-to-SQL methods for effectively encoding structural information of such 2-D tabular data, so that more comprehensive input representations can be learned to facilitate the SQL generation.

6.4 Heterogeneous Information Modeling

Existing text-to-SQL datasets, such as Spider [23], mainly contain textual and numeric data from NL questions and tables. However, many real-life applications contain more types of data (e.g., images) over heterogeneous forms. Using only homogeneous information source cannot satisfy the demands of some real-life applications, such as E-commerce and metaverse. For example, [96] takes the first step towards this direction by proposing a challenging question answering dataset that requires joint reasoning over texts, tables and images. When applying the text-to-SQL models in the E-commerce and metaverse domains, it is necessary to process multimodal data and aggregate information from heterogeneous information sources for obtaining the correct returned results.

6.5 Cross-domain Text-to-SQL Parsing

Most existing works have worked on in-domain text-to-SQL parsing, where the training and testing sets share the same domains. However, no matter how much data is collected and applied to train a text-to-SQL parser, it is difficult to cover all possible domains of databases. Thus, when deployed in practice, a well-trained text-to-SQL parser cannot generalize to new domains and often performs unsatisfactorily. Although some cross-domain text-to-SQL datasets (e.g., Spider [23], DK [25] and SYN [24]) have been constructed for the challenging cross-domain settings, there are no text-to-SQL methods that design tailored algorithms to deal with the out-of-distribution (OOD) generalization problem where the testing data distribution is unknown and different from the training data distribution. The supervised learning method is fragile when being exposed to data with different distributions. Therefore, how to explore the OOD generalization of the text-to-SQL parser is a promising future direction for both academic and industry communities.

6.6 Robustness of Text-to-SQL Parsing Models

The robustness of text-to-SQL parsing models pose a prominent challenge when being deployed in real-life applications. Small perturbations in the input may significantly reduce the performance of text-to-SQL parsing models. High performance requires robust performance on noisy inputs. Recently, Spider-SYN [24] investigates the robustness of text-to-SQL parsing models to synonym substitution by removing the explicit schema-linking correspondence between NL questions and table schemas. Spider-DK [25] investigates the generalization of text-to-SQL parsing models by injecting rarely observed domain knowledge into the NL questions so as to evaluate the model understanding of domain knowledge. The experimental results from both [24] and [25] demonstrated that the performance of text-to-SQL models are inferior when facing the small perturbations (synonym substitution and domain knowledge injection) in the input, even though the training and test data shares similar distributions. Hence, it is necessary to stabilize the neural text-to-SQL parsing models, making the models more robust to different perturbations There is very few exploration for improving the robustness of text-to-SQL models, thus more efforts should be paid to this research direction.

6.7 Zero-shot and Few-shot Text-to-SQL Parsing

The "pre-training+fine-tuning" paradigm has been widely used for text-to-SQL parsing, yielding state-of-the-art performances. Although the text-to-SQL parsing models that

are trained via the "pre-training+fine-tuning" paradigm can obtain impressive performance, they still require a large-scale annotated dataset for fine-tuning the downstream tasks. These neural models could show poor performance in zero-shot setting without any task-specific annotated training data. One possible solution is to adopt pre-trained language models (e.g., GPT-3 [97] and Codex [98]) for zero-shot transfer to downstream tasks without the fine-tuning phase. [99] and [94] revealed that large pre-trained language models can achieve competitive performance on text-to-SQL parsing without fine-tuning. For example, Codex [98] achieved the execution accuracy up to 67% on the Spider [23] development set. Therefore, zero-shot and few-shot text-to-SQL parsing is a promising direction for future exploration.

6.8 Pre-Training for Context-dependent Text-to-SQL Parsing

Context-dependent text-to-SQL parsing need to effectively process context information so as to effectively generate the current SQL query since users may omit previously mentioned entities as well as constraints and introduce substitutions to what has already been stated. The key challenge is how to track and explore the interaction states in history utterances to assist the models to better understand the current NL utterance. Nevertheless, most prior TaLMs primarily model stand-alone NL utterances without considering the context-dependent interactions, which result in sub-optimal performance. Although SCoRe [84] model the turn contextual switch by predicting the context switch label between two consecutive user utterances, it ignores the complex interactions of context utterances and cannot track the dependence between distant utterances. Inspired by the dialogue state tracking [76, 100, 101] which keeps track of user intentions in the form of a set of dialogue states (e.g., slot-value pairs) in task-oriented dialogue systems, it is beneficial to keep track of schema states (or user requests) of context-dependent SQL queries in the pre-training process. How to explore and make good use of context information to enrich question and schema representations is critical for the deployment of text-to-SQL generalization methods.

6.9 Interpretability of Text-to-SQL Models

Deep neural networks (DNNs) have achieved the state-of-the-art performance on text-to-SQL parsing. However, the neural models usually lack of interpretability and are perceived as black boxes. For the sensitive domains such as finance or healthcare, the interpretability of neural text-to-SQL models and returned results is necessary, which is as important as the SQL generation performance. Instead of exploring the implicit encoding and reasoning strategies making the predicted results uninterpretable to humans, how to take advantage of both the representation ability of DNNs and the explicit reasoning ability of symbolic approaches is a promising future direction for text-to-SQL parsing, especially for the applications in the highly sensitive domains.

6.10 Data Privacy-preserving

Many text-to-SQL models have been deployed on the cloud to process user data by the small businesses. Due to the sensitivity of user data, data privacy-preserving could be an essential but challenging task in text-to-SQL parsing. In particular, most text-to-SQL parsing models require identifying patterns as well as relations from large-scale corpora and related database schemas and gathering all information into a central site for feature representation learning. However, the data privacy issue may prevent the neural models from building a centralized architecture given the fact that the relations or patterns among different database schemas may be distributed among several custodians, none of which are allowed to transfer their user data to other sites. How to effectively transform the NL questions as well as database schema to privatized input representations on the local devices and then upload the transformed representations to the service provider would be also a promising exploration direction.

7 Conclusion

This manuscript aims at providing the first comprehensive survey for text-to-SQL parsing from the NLP community. We provide an overview of the existing datasets, current neural text-to-SQL parsing models, tailored pre-training methods, and possible future exploration directions. First, we introduce available single-turn and multi-turn text-to-SQL corpora and provide a tabular to summarize these resources. Second, we review the state-of-the-art neural text-to-SQL parsing models based on varying encoders and decoders. Third, we present the well-known tabular pre-training methods for text-to-SQL parsing and elaborate on existing pre-training objectives. Finally, we discuss the future directions in the field of text-to-SQL parsing. We hope that this survey can provide some insightful perspectives and inspire the widespread implementation of real-life text-to-SQL parsing systems.

Acknowledgments

Min Yang was partially supported by National Natural Science Foundation of China (No. 61906185), Shenzhen Basic Research Foundation (No. JCYJ20210324115614039 and No. JCYJ20200109113441941), Shenzhen Science and Technology Innovation Program (Grant No. KQTD20190929172835662), Youth Innovation Promotion Association of CAS China (No. 2020357). This work was supported by Alibaba Group through Alibaba Innovative Research Program.

References

[1] L. Wang, B. Qin, B. Hui, B. Li, M. Yang, B. Wang, B. Li, F. Huang, L. Si, and Y. Li, "Proton: Probing schema linking information from pre-trained language models for text-to-sql parsing," *arXiv preprint arXiv:2206.14017*, 2022.

[2] J. M. Zelle and R. J. Mooney, "Learning to parse database queries using inductive logic programming," in *Proceedings of the national conference on artificial intelligence*, 1996.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Proc. of NeurIPS*, 2014.
[4] J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J.-G. Lou, T. Liu, and D. Zhang, "Towards complex text-to-SQL in cross-domain database with intermediate representation," in *Proc. of ACL*, 2019.
[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, 1997.
[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. of NeurIPS*, 2017.
[7] D. Choi, M. C. Shin, E. Kim, and D. R. Shin, "RYAN-SQL: Recursively applying sketch-based slot fillings for complex text-to-SQL in cross-domain databases," *Computational Linguistics*, 2021.
[8] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *ArXiv preprint*, 2015.
[9] D. Yoon, D. Lee, and S. Lee, "Dynamic self-attention: Computing attention over words dynamically for sentence embedding," *ArXiv preprint*, 2018.
[10] W. Hwang, J. Yim, S. Park, and M. Seo, "A comprehensive exploration on wikisql with table-aware word contextualization," *ArXiv preprint*, 2019.
[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. of AACL*, 2019.
[12] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers," in *Proc. of ACL*, 2020.
[13] R. Cai, J. Yuan, B. Xu, and Z. Hao, "Sadga: Structure-aware dual graph aggregation network for text-to-sql," *Proc. of NeurIPS*, 2021.
[14] R. Cao, L. Chen, Z. Chen, Y. Zhao, S. Zhu, and K. Yu, "LGESQL: Line graph enhanced text-to-SQL model with mixed local and non-local relations," in *Proc. of ACL*, 2021.
[15] X. Xu, C. Liu, and D. Song, "Sqlnet: Generating structured queries from natural language without reinforcement learning," *ArXiv preprint*, 2017.
[16] B. Hui, X. Shi, R. Geng, B. Li, Y. Li, J. Sun, and X. Zhu, "Improving text-to-sql with schema dependency learning," *ArXiv preprint*, 2021.
[17] J. Huang, Y. Wang, Y. Wang, Y. Dong, and Y. Xiao, "Relation aware semi-autoregressive semantic parsing for nl2sql," *ArXiv preprint*, 2021.
[18] P. Yin, G. Neubig, W.-t. Yih, and S. Riedel, "TaBERT: Pretraining for joint understanding of textual and tabular data," in *Proc. of ACL*, 2020.
[19] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. Eisenschlos, "TaPas: Weakly supervised table parsing via pre-training," in *Proc. of ACL*, 2020.
[20] T. Yu, C. Wu, X. V. Lin, B. Wang, Y. C. Tan, X. Yang, D. R. Radev, R. Socher, and C. Xiong, "Grappa: Grammar-augmented pre-training for table semantic parsing," in *Proc. of ICLR*, 2021.
[21] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer, "Learning a neural semantic parser from user feedback," in *Proc. of ACL*, 2017.
[22] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *ArXiv preprint*, 2017.
[23] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in *Proc. of EMNLP*, 2018.
[24] Y. Gan, X. Chen, Q. Huang, M. Purver, J. R. Woodward, J. Xie, and P. Huang, "Towards robustness of text-to-SQL models against synonym substitution," in *Proc. of ACL*, 2021.
[25] Y. Gan, X. Chen, and M. Purver, "Exploring underexplored limitations of cross-domain text-to-SQL generalization," in *Proc. of EMNLP*, 2021.
[26] P. Shaw, M.-W. Chang, P. Pasupat, and K. Toutanova, "Compositional generalization and natural language variation: Can a semantic parsing approach handle both?" in *Proc. of ACL*, 2020.
[27] Q. Min, Y. Shi, and Y. Zhang, "A pilot study for Chinese SQL semantic parsing," in *Proc. of EMNLP*, 2019.
[28] L. Wang, A. Zhang, K. Wu, K. Sun, Z. Li, H. Wu, M. Zhang, and H. Wang, "DuSQL: A large-scale and pragmatic Chinese text-to-SQL dataset," in *Proc. of EMNLP*, 2020.
[29] P. J. Price, "Evaluation of spoken language systems: the ATIS domain," in *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*, 1990.
[30] D. A. Dahl, M. Bates, M. Brown, W. Fisher, K. Hunicke-Smith, D. Pallett, C. Pao, A. Rudnicky, and E. 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.
[31] T. Yu, R. Zhang, M. Yasunaga, Y. C. Tan, X. V. Lin, S. Li, H. Er, I. Li, B. Pang, T. Chen, E. Ji, S. Dixit, D. Proctor, S. Shim, J. Kraft, V. Zhang, C. Xiong, R. Socher, and D. Radev, "SParC: Cross-domain semantic parsing in context," in *Proc. of ACL*, 2019.
[32] T. Yu, R. Zhang, H. Er, S. Li, E. Xue, B. Pang, X. V. Lin, Y. C. Tan, T. Shi, Z. Li, Y. Jiang, M. Yasunaga, S. Shim, T. Chen, A. Fabbri, Z. Li, L. Chen, Y. Zhang, S. Dixit, V. Zhang, C. Xiong, R. Socher, W. Lasecki, and D. Radev, "CoSQL: A conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases," in *Proc. of EMNLP*, 2019.
[33] J. Guo, Z. Si, Y. Wang, Q. Liu, M. Fan, J.-G. Lou, Z. Yang, and T. Liu, "Chase: A large-scale and pragmatic Chinese dataset for cross-database context-dependent text-to-SQL," in *Proc. of ACL*, 2021.
[34] R. Zhong, T. Yu, and D. Klein, "Semantic evaluation for text-to-sql with distilled test suite," in *Proc. of EMNLP*, 2020.
[35] L. S. Zettlemoyer and M. Collins, "Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars," *Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence*, 2005.
[36] T. Shi, C. Zhao, J. Boyd-Graber, H. Daumé III, and L. Lee, "On the potential of lexico-logical alignments

for semantic parsing to SQL queries," in *Proc. of EMNLP Findings*, 2020.

[37] P. Pasupat and P. Liang, "Compositional semantic parsing on semi-structured tables," in *Proc. of ACL*, 2015.

[38] T. Yu, Z. Li, Z. Zhang, R. Zhang, and D. Radev, "TypeSQL: Knowledge-based type-aware neural text-to-SQL generation," in *Proc. of AACL*, 2018.

[39] T. Yu, M. Yasunaga, K. Yang, R. Zhang, D. Wang, Z. Li, and D. Radev, "SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task," in *Proc. of EMNLP*, 2018.

[40] B. Bogin, J. Berant, and M. Gardner, "Representing schema structure with graph neural networks for text-to-SQL parsing," in *Proc. of ACL*, 2019.

[41] R. Zhang, T. Yu, H. Er, S. Shim, E. Xue, X. V. Lin, T. Shi, C. Xiong, R. Socher, and D. Radev, "Editing-based SQL query generation for cross-domain context-dependent questions," in *Proc. of EMNLP*, 2019.

[42] A. Kelkar, R. Relan, V. Bhardwaj, S. Vaichal, C. Khatri, and P. Relan, "Bertrand-dr: Improving text-to-sql using a discriminative re-ranker," *ArXiv preprint*, 2020.

[43] X. V. Lin, R. Socher, and C. Xiong, "Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing," in *Proc. of EMNLP Findings*, 2020.

[44] O. Rubin and J. Berant, "SmBoP: Semi-autoregressive bottom-up semantic parsing," in *Proc. of AACL*, 2021.

[45] Z. Chen, L. Chen, Y. Zhao, R. Cao, Z. Xu, S. Zhu, and K. Yu, "ShadowGNN: Graph projection neural network for text-to-SQL parser," in *Proc. of AACL*, 2021.

[46] P. Xu, D. Kumar, W. Yang, W. Zi, K. Tang, C. Huang, J. C. K. Cheung, S. J. Prince, and Y. Cao, "Optimizing deeper transformers on small datasets," in *Proc. of ACL*, 2021.

[47] T. Scholak, N. Schucher, and D. Bahdanau, "PICARD: Parsing incrementally for constrained auto-regressive decoding from language models," in *Proc. of EMNLP*, 2021.

[48] B. Hui, R. Geng, L. Wang, B. Qin, B. Li, J. Sun, and Y. Li, "S ^{2} sql: Injecting syntax to question-schema interaction graph encoder for text-to-sql parsers," *ArXiv preprint*, 2022.

[49] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE transactions on Signal Processing*, 1997.

[50] W. Lei, W. Wang, Z. Ma, T. Gan, W. Lu, M.-Y. Kan, and T.-S. Chua, "Re-examining the role of schema linking in text-to-SQL," in *Proc. of EMNLP*, 2020.

[51] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *ArXiv preprint*, 2019.

[52] Y. Gou, Y. Lei, L. Liu, Y. Dai, and C. Shen, "Contextualize knowledge bases with transformer for end-to-end task-oriented dialogue systems," in *Proc. of EMNLP*, 2021.

[53] V. Zhong, M. Lewis, S. I. Wang, and L. Zettlemoyer, "Grounded adaptation for zero-shot executable semantic parsing," in *Proc. of EMNLP*, 2020.

[54] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*, 2017.

[55] Q. Liu, D. Yang, J. Zhang, J. Guo, B. Zhou, and J.-G. Lou, "Awakening latent grounding from pretrained language models for semantic parsing," *arXiv preprint arXiv:2109.10540*, 2021.

[56] J. Ma, Z. Yan, S. Pang, Y. Zhang, and J. Shen, "Mention extraction and linking for SQL query generation," in *Proc. of EMNLP*, 2020.

[57] J. D. Lafferty, A. McCallum, and F. C. N. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *Proc. of ICML*, 2001.

[58] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," in *Proc. of AACL*, 2018.

[59] K. Wang, W. Shen, Y. Yang, X. Quan, and R. Wang, "Relational graph attention network for aspect-based sentiment analysis," in *Proc. of ACL*, 2020.

[60] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," in *Proc. of ICLR*, 2016.

[61] B. Bogin, M. Gardner, and J. Berant, "Global reasoning over database structures for text-to-SQL parsing," in *Proc. of EMNLP*, 2019.

[62] C. Wang, P.-S. Huang, A. Polozov, M. Brockschmidt, and R. Singh, "Execution-guided neural program decoding," in *ICML workshop on Neural Abstract Machines and Program Induction v2 (NAMPI)*, 2018.

[63] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," in *Proc. of ACL*, 2017.

[64] D. C. Wang, A. W. Appel, J. L. Korn, and C. S. Serra, "The zephyr abstract syntax description language." in *DSL*, 1997.

[65] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, 1972.

[66] P. Shaw, M.-W. Chang, P. Pasupat, and K. Toutanova, "Compositional generalization and natural language variation: Can a semantic parsing approach handle both?" in *Proc. of ACL*, 2021.

[67] K. Xuan, Y. Wang, Y. Wang, Z. Wen, and Y. Dong, "Sead: End-to-end text-to-sql generation with schema-aware denoising," *ArXiv preprint*, 2021.

[68] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *ArXiv preprint*, 2019.

[69] Q. Liu, B. Chen, J. Guo, J.-G. Lou, B. Zhou, and D. Zhang, "How far are we from effective context modeling ? an exploratory study on semantic parsing in context," in *Proc. of IJCAI*, 2020.

[70] T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. of EMNLP*, 2015.

[71] P. Jain and M. Lapata, "Memory-Based Semantic Parsing," *Transactions of the Association for Computational Linguistics*, 2021.

[72] Z. Chen, L. Chen, H. Li, R. Cao, D. Ma, M. Wu, and K. Yu, "Decoupled dialogue modeling and seman-

tic parsing for multi-turn text-to-sql," *arXiv preprint arXiv:2106.02282*, 2021.

[73] Y. Zheng, H. Wang, B. Dong, X. Wang, and C. Li, "Hie-sql: History information enhanced network for context-dependent text-to-sql semantic parsing," *arXiv preprint arXiv:2203.07376*, 2022.

[74] B. Hui, R. Geng, Q. Ren, B. Li, Y. Li, J. Sun, F. Huang, L. Si, P. Zhu, and X. Zhu, "Dynamic hybrid relation network for cross-domain context-dependent semantic parsing," *ArXiv preprint*, 2021.

[75] Y. Cai and X. Wan, "IGSQL: Database schema interaction graph based neural model for context-dependent text-to-SQL generation," in *Proc. of EMNLP*, 2020.

[76] R.-Z. Wang, Z.-H. Ling, J. Zhou, and Y. Hu, "Tracking interaction states for multi-turn text-to-sql semantic parsing," in *Proc. of AAAI*, 2021.

[77] R.-Z. Wang, Z.-H. Ling, J.-B. Zhou, and Y. Hu, "Tracking interaction states for multi-turn text-to-sql semantic parsing," in *Proc. of AAAI*, 2021.

[78] C. S. Bhagavatula, T. Noraset, and D. Downey, "Methods for exploring and mining tables on wikipedia," in *Proc. of KDD*, 2013.

[79] O. Lehmberg, D. Ritze, R. Meusel, and C. Bizer, "A large public corpus of web tables containing time and context metadata," in *Proceedings of the 25th International Conference Companion on World Wide Web*, 2016.

[80] C. S. Bhagavatula, T. Noraset, and D. Downey, "Tabel: Entity linking in web tables," in *Proc. of ISWC*, 2015.

[81] A. Parikh, X. Wang, S. Gehrmann, M. Faruqui, B. Dhingra, D. Yang, and D. Das, "ToTTo: A controlled table-to-text generation dataset," in *Proc. of EMNLP*, 2020.

[82] Y. Sun, Z. Yan, D. Tang, N. Duan, and B. Qin, "Content-based table retrieval for web queries," *Neurocomputing*, 2019.

[83] X. Deng, A. H. Awadallah, C. Meek, O. Polozov, H. Sun, and M. Richardson, "Structure-grounded pretraining for text-to-SQL," in *Proc. of AACL*, 2021.

[84] T. Yu, R. Zhang, A. Polozov, C. Meek, and A. H. Awadallah, "Score: Pre-training for context representation in conversational semantic parsing," in *Proc. of ICLR*, 2021.

[85] W. Yang, P. Xu, and Y. Cao, "Hierarchical neural data synthesis for semantic parsing," *ArXiv preprint*, 2021.

[86] B. Wang, W. Yin, X. V. Lin, and C. Xiong, "Learning to synthesize data for semantic parsing," in *Proc. of AACL*, 2021.

[87] C. Shu, Y. Zhang, X. Dong, P. Shi, T. Yu, and R. Zhang, "Logic-consistency text generation from semantic parses," in *Proc. of ACL Findings*, 2021.

[88] Q. Liu, B. Chen, J. Guo, M. Ziyadi, Z. Lin, W. Chen, and J. guang Lou, "Tapex: Table pre-training via learning a neural sql executor," 2021.

[89] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proc. of EMNLP*, 2014.

[90] J. Eisenschlos, M. Gor, T. Müller, and W. Cohen, "MATE: Multi-view attention for table transformer efficiency," in *Proc. of EMNLP*, 2021.

[91] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proc. of ACL*, 2020.

[92] P. Shi, P. Ng, Z. Wang, H. Zhu, A. H. Li, J. Wang, C. N. dos Santos, and B. Xiang, "Learning contextual representations for semantic parsing with generation-augmented pre-training," in *Proc. of AAAI*, 2021.

[93] J. Yang, A. Gupta, S. Upadhyay, L. He, R. Goel, and S. Paul, "Tableformer: Robust transformer modeling for table-text encoding," *ArXiv preprint*, 2022.

[94] T. Xie, C. H. Wu, P. Shi, R. Zhong, T. Scholak, M. Yasunaga, C.-S. Wu, M. Zhong, P. Yin, S. I. Wang *et al.*, "Unifiedskg: Unifying and multi-tasking structured knowledge grounding with text-to-text language models," *ArXiv preprint*, 2022.

[95] B. Wang, M. Lapata, and I. Titov, "Meta-learning for domain generalization in semantic parsing," in *Proc. of AACL*, 2021.

[96] A. Talmor, O. Yoran, A. Catav, D. Lahav, Y. Wang, A. Asai, G. Ilharco, H. Hajishirzi, and J. Berant, "Multimodalqa: complex question answering over text, tables and images," in *Proc. of ICLR*, 2021.

[97] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, 2020.

[98] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[99] N. Rajkumar, R. Li, and D. Bahdanau, "Evaluating the text-to-sql capabilities of large language models," 2022.

[100] W. He, Y. Dai, M. Yang, J. Sun, F. Huang, L. Si, and Y. Li, "Unified dialog model pre-training for task-oriented dialog understanding and generation," in *Proc. of SIGIR*, 2022.

[101] W. He, Y. Dai, Y. Zheng, Y. Wu, Z. Cao, D. Liu, P. Jiang, M. Yang, F. Huang, L. Si *et al.*, "Galaxy: A generative pre-trained model for task-oriented dialog with semi-supervised learning and explicit policy injection," in *Proc. of AAAI*, 2022.