**UniSAr: a unified structure-aware autoregressive language model for text-to-SQL semantic parsing**

Longxu Dou1 · Yan Gao2 · Mingyang Pan1 · Dingzirui Wang1 · Wanxiang Che1 · Jian-Guang Lou2 · Dechen Zhan1

Received: 19 December 2022 / Accepted: 5 June 2023 / Published online: 5 July 2023
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2023

**Abstract**

Existing text-to-SQL semantic parsers are typically designed for particular settings such as handling queries that span multiple tables, domains, or turns which makes them ineffective when applied to different settings. We present UniSAr (Unified Structure-Aware Autoregressive Language Model), which benefits from directly using an off-the-shelf language model architecture and demonstrates consistently high performance under different settings. Specifically, UniSAr extends existing autoregressive language models to incorporate two non-invasive extensions to make them structure-aware: (1) adding structure mark to encode database schema, conversation context, and their relationships; (2) constrained decoding to decode well-structured SQL for a given database schema. On seven well-known text-to-SQL datasets covering multi-domain, multi-table, and multi-turn, UniSAr demonstrates highly comparable or better performance to the most advanced specifically-designed text-to-SQL models.

**Keywords** Text-to-SQL · Semantic parsing · Natural language interfaces to databases · Natural language processing · Constrained decoding

**1 Introduction**

Text-to-SQL semantic parsing translates a user’s natural language question into the corresponding executable SQL query [1–4]. This greatly reduces the entry barrier of data analysis to lay users daunted by the technical nuances of SQL. As text-to-SQL techniques matured, enhancements have been proposed to tackle different settings. These enhancements can be roughly organized into three settings of research (Fig. 1): (1) multi-domain where a parser must generalize to databases in various domains [1, 5]; (2) multi-table where the parser must understand the database structure and generate complex SQL query bridging multiple tables [2, 6]; and (3) multi-turn where a parser must understand the dialog history, often requiring co-reference resolution and ellipsis recovery [4, 7, 8].

To address the unique challenges of different settings, researchers have proposed various neural architectures, such as grammar-based decoder for SQL generation [3, 9, 10], GNN for database structure modeling [11], and stacked interaction-layer for context modeling [12]. However, these architectures are prone to overfitting specific datasets, making them non-trivial to adapt to others. For example, the
In this work, we present a simple yet effective text-to-SQL parser: UniSAr (Unified Structure-Aware Autoregressive Language Model). Compared with the specifically-designed models (invasive), UniSAr is simple as it does not need any specifically designed DNN modules other than a pre-trained language model (non-invasive). Such a simple architecture gives UniSAr the opportunity to be easily adapted to different datasets, and thus UniSAr enjoys high generalizability. Besides, benefiting from our proposed non-invasive extensions, UniSAr achieves consistently high performance across different settings, which is very effective. Concretely, we propose two non-invasive extensions to make an off-the-shelf autoregressive language model structure-aware. First, we encode the structure information (e.g., database schema, conversation context, and their relationships) by inserting some special tokens named Structure Mark into the serialized schema and question as inputs. Then, we adopt Prefix Tree based Constrained Decoding, dynamically constraining the decoding process to filter the invalid tokens (e.g., synonyms of schema) during SQL generation. Importantly, our UniSAr is model-agnostic, such that other core model advances in text-to-SQL can also adopt our non-invasive extensions to further enhance performance.

To prove the effectiveness and generalizability of our UniSAr, we conduct experiments on seven popular text-to-SQL datasets covering multi-domain, multi-table, and multi-turn. With a simple and unified architecture, our model achieves comparable or even better performance against task-specific models. Importantly, the simple architecture enables different tasks to share the sample training protocol. UniSAr could be easily improved by multi-task training.

To summarize, our contributions are three-fold:

- To deeply understand the challenges of different text-to-SQL scenarios, we first systematically summarize three representative text-to-SQL settings, then propose a unified seq2seq framework UniSAr based on PLMs to replace the existing task-specific parsers.
- To further advance the unified framework UniSAr, we propose two non-invasive extensions: Structure Mark and Prefix Tree based Constrained Decoding, which make the PLMs structure-aware in both encoding phrase and decoding phrase.
- To empirically prove the effectiveness and generalizability of UniSAr, we conduct extensive experiments on seven popular text-to-SQL datasets, which demonstrate that UniSAr outperforms many specifically-designed parsers under a unified architecture.

The rest parts of this paper are organized as follows: Sect. 2 summarizes the most relevant research work to us and illustrates the necessity of a unified parser; Sect. 3 introduces the task formulation of text-to-SQL and the advancements of UniSAr beyond vanilla PLMs; Sect. 4 demonstrates the experiment setup and the implementation details of UniSAr; Sect. 5 lists the experimental results, ablation studies of different components of UniSAr, and explores four interesting research questions; Sect. 6 summarizes our work and discusses potential future work.

2 Related work

2.1 Invasive approaches for text-to-SQL

The existing text-to-SQL models usually adopt various task-specific modules to meet the different requirements in specific scenarios. We refer to them as invasive approaches since they need to modify the architecture once the text-to-SQL setting changes. While these task-specific frameworks achieve promising performance in the specific setting, they could not be easily generalized to other settings, due to the discrepancy of data distribution across settings.

For example, an additional context encoder [8, 12, 13] is required to model the dialogue history in multi-turn setting, as shown in the part (a) of Fig. 2. However, this module would cause the overfitting problem if we leverage this multi-turn-specific model to single-turn scenario as discussed in Table 7. While for the design of the decoder, as shown in part (b) of Fig. 2, previous work usually employ slot-filling decoder for simple non-nested SQL [14–16] (better decoding efficiency), and adopt grammar-based decoder [3, 8, 10] for complex nested SQL (higher decoding accuracy). Obviously, it’s non-trivial to merge these two approaches to be both fast and accurate then work in both single-table and multi-table settings. To summarize, these invasive approaches would prohibit the employment of...
text-to-SQL in the real scenario, where we expect to develop and maintain one model but fit all scenarios.

Therefore, to improve the generalizability of the text-to-SQL approach, we build UniSAr on top of pre-trained language models which reveal excellent generalizability across different NLP tasks. UniSAr provides a unified way to encode the structure information and decode valid SQL that fits all text-to-SQL settings, which could serve as a backbone model for many downstream applications. It is extremely simple as it does not need specifically designed modules (non-invasive) and is also incredibly effective.

2.2 Non-invasive approaches for text-to-SQL

Recent research works [17–20] explore that directly employ the pre-trained language model like BART and T5 for semantic parsing tasks. Basically, they formulate the parsing tasks (structure prediction) as seq2seq generation tasks. Despite achieving the promising results [17], the paradigm of writing rules (e.g., SCFG grammars) to generate valid logic forms, still involves huge human efforts. In the following, we will compare UniSAr with two popular non-invasive approaches: UnifiedSKG [19] and PICARD [18].

UnifiedSKG [19] investigated the utilization of pre-trained language models (PLMs) in various tasks that involve structured information as input (e.g., KBQA, TableQA, Table-based Fact Verification, text-to-SQL). However, it only explores vanilla structure modeling (i.e., linearized the structural input into a sequence), disregarding the essential aspect of modeling structural information. In comparison, our UniSAr advances the PLMs as it considers the structure information in both encoding and decoding phrases. Our additional experiments (Sect. 5.3.1) demonstrate that the use of structure marks further enhances the performance of PICARD, indicating that our model makes the pre-trained language model more attuned to structural considerations.

Fig. 2 Comparison between the task-specific modules of existing text-to-SQL models and the unified parser employed in this work. a Task-Specific Encoder for encoding the database structure (relation-aware transformer and graph neural network) and the context information (historical SQL encoder). b Task-Specific Decoder for decoding the simple SQL (slot-filling decoder) and complex SQL (coarse-to-fine decoder and grammar-based decoder). c Unified Parser first flattens the input (question and tables) and then outputs the SQL following the general seq2seq generation fashion.
2.3 Expanding pre-trained language model with knowledge

The vanilla PLMs like BART [21] are inevitably static in knowledge storing and weak in knowledge representation. KG-BART [22] augmented the pre-trained language model with a knowledge graph to promote the ability of commonsense reasoning for text generation. Concretely, it encompassed the complex relations of concepts through the extra knowledge graph and then produced more logical and natural output than vanilla BART.

Compared with their work which solely considers factual knowledge, our work also focuses on structure knowledge and grounding knowledge. Thus, UniSAr is orthogonal with theirs, which could extend their work to more NLP tasks that require structure modeling.

3 Methodology

Overall, we employ pre-trained autoregressive language models as the backbone of UniSAr, since they exhibit excellent adaptability and generalizability in many NLP tasks. We propose two non-invasive extensions to make PLMs become structure-aware: (1) we encode the structure information (e.g., database schema, conversation context and their linking relationships) by inserting some special tokens named structure marks into the serialized schema and question as inputs; (2) we adopt constrained decoding to decode well-structured SQL via simply filtering invalid tokens (e.g., synonyms of schema) during beam search.

3.1 Preliminaries

In this section, we will describe the necessary background knowledge. First, we formally define the text-to-SQL problem and its components. Then, we present how to formulate text-to-SQL as a seq2seq task. Finally, we introduce two famous pre-trained language models: BART [21] and T5 [23], which are the backbone models in our framework.

3.1.1 Problem definition of text-to-SQL

Given a natural language question $Q$ and a schema $S = (C, T)$ for a relational database, the goal of text-to-SQL is to generate the corresponding SQL $P$. Here the question $Q = q_1 ... q_{|Q|}$ is a sequence of words, and the schema consists of columns $C = \{c_1, ..., c_{|C|}\}$ and tables $T = \{t_1, ..., t_{|T|}\}$. Each column name $c_i$ contains words $c_{i,1}, ..., c_{i,|c_i|}$. Each table name $t_i$ contains words $t_{i,1}, ..., t_{i,|t_i|}$.

Formally, we represent the database schema as a directed graph $G = (\mathcal{V}, \mathcal{E})$. Its nodes $\mathcal{V} = C \cup T$ are the columns and tables of the schema, each labeled with the words in its name. Its edges $\mathcal{E}$ are defined by the pre-existing database relations, such as $(c_1, \text{SAME_TABLE}, c_2), (c_1, \text{BELONGS_TO_TABLE}, t_1), (t_1, \text{HAS_COLUMN}, c_i)$.

We systematically categorize the existing text-to-SQL settings into three groups (Fig. 1): (1) multi-domain where a parser must generalize to databases in various domains [1, 5]; (2) multi-table where the parser must understand the database structure and generate complex SQL query bridging multiple tables [2, 6]; and (3) multi-turn where a parser must understand the dialog history, often requiring co-reference resolution and ellipsis recovery [4, 7, 8].

3.1.2 Algorithm: modeling text-to-SQL as Seq2Seq task

In this work, we formulate text-to-SQL as a sequence-to-sequence task. Encoding the question is relatively straightforward while encoding the table is non-trivial since it exhibits underlying structures. In practice, we linearize the schema into a flattened sequence so that it can be fed into the language model.

Concretely, by inserting several special tokens to indicate the table boundaries, the linearized database schema $S_{seq}$ can be represented as

$$S_{seq} = [\text{TABLE}] t_1 ... [\text{COLUMN}] c_1 ... c_{|C|},$$

where $[\text{TABLE}]$ and $[\text{COLUMN}]$ are special tokens to separate the region of table names and column names. And ‘|’ is the delimiter between the columns or the tables. To generate the executable SQL, we also attach the $j$-th values of $i$-th column $c_i$ as $v_{ij}$ behind $c_i$ and separate each value with the symbol ‘\&’.

For brevity, we omit these parts in the above formula.

For multi-turn settings, we concatenate the dialogue history $Q_1, ..., Q_{i-1}$ and current $i$-th question $Q_i$ in the chronological order to form the linearized questions $Q_{seq}$ as

$$Q_{seq} = [\text{HISTORY}] Q_1 ... [\text{QUESTION}] Q_i$$

where $[\text{HISTORY}]$ and $[\text{QUESTION}]$ are special tokens to separate the region of the history question and the current question. And ‘|’ is the delimiter between the questions. Please refer to Fig. 3 to find a concrete case.

Finally, we concatenate the linearized database schema $S_{seq}$ and the linearized questions $Q_{seq}$ to form the input as

$$[\text{TABLE}] t_1 ... [\text{COLUMN}] c_1 ... c_{|C|}$$

$[\text{HISTORY}] Q_1 ... [\text{QUESTION}] Q_i$,

then we feed them to the language model. The output of our algorithm is the SQL sequence $P$. 
3.1.3 Backbone model: pretrained language model

In this work, we implement our model UniSAr on the top of two existing pre-trained language models: BART [21] and T5 [23]. They are widely used pre-trained encoder-decoder models, which follow the standard sequence-to-sequence Transformer architecture [24].

T5 reframes all NLP tasks into a unified text-to-text format where the input and output are always text strings. In contrast to BERT-style models, this text-to-text framework allows us to use the same model, loss function, and hyperparameters on any NLP task. On the contrary, BART modifies the ReLU activation functions of the Transformer to GeLU. It is pre-trained via corrupting sentences (e.g., randomly sampling spans and masking each one) and then optimizing a reconstruction loss.

We employ the English BART and T5 for English text-to-SQL benchmarks. We employ the mBART-CC25/mT5-Large for Chinese benchmarks. For brevity, we collectively call them BART/T5 in the following descriptions.

3.2 Encoding structure with structure mark

It’s well-known that the structure information plays an important role in text-to-SQL [10, 25]. In contrast to the specialized models, the recent research on prompt tuning provides a unified way to jointly learn different tasks. [26, 27] make a step by making prompts with additional marks (some special tokens) to encode various structure information. Inspired by this, we propose Structure Mark to advance the language model to be structure-aware.

As shown in Fig. 3, we roughly categorized the structure information into three classes: (1) schema property to expand the semantic information of schema; (2) database structure to aggregate the information from schema neighbors; (3) discourse structure to supply the conversation context in the history question. In the following, we would introduce (1) how to derive the structure mark? and (2) how to insert the mark into the flattening schema?

![Structure-aware annotator](image)
3.2.1 Schema linking

The vanilla flattened schema neglects the modeling of structure information. Therefore, the semantic representation could only rely on the surface name so it would easily lead to a disambiguation problem. Notably, the schema-linking could greatly improve the correctness of alignment between question and database and eventually boost the text-to-SQL performance [25, 28]. The schema-linking information includes (1) the name-based linking information between question and schema that serve as the prior of schema linking; (2) the value-based linking information that augments the column representation via leveraging the database content.

For name-based linking information, we first enumerate n-gram spans of both question and schema then select the matched span pairs (i.e., schema alias and entity mention). Concretely, we derive the schema-linking results using fuzzy string-match following [3, 10, 29]. In practice, we distinguish the exact-match and partial-match to realize the fine-grained alignment. We adopt [EM] for exact match and [PM] for partial match. For example, column PlAyer_id has partially overlapped with the token ‘Player’ in question, thus we attach the special token [PM] in front of the column PlAyer_id. For example, if table name $t_1$ and column name $c_1$ are linked during schema linking, we could derive the $S_{seq}$ as

$$[TABLE] [PM] t_1 \ldots [COLUMN] [PM] c_1 \ldots [PM] c_n,$$

where [PM] is the special token indicating the table and column are partial matched with the tokens in the question.

For value-based linking information, we first normalize the database content (e.g. uniform the TIME/DATE representation with heuristic rules) then compare them with the spans of question. Lastly, if any value of the column is grounded in the question, we insert the tag [VM] in front of this column. Notably, the scale of database contents (or values) of datasets evaluated in our paper is not very large (10 lines on average), thus we just enumerate them one by one with simple Regex matching. For realistic scenarios in industry, where we might meet very large tables (more than 1 million rows), we could adopt some open-source search engines like Lucene. For the sake of brevity, we collectively call these special tags ([EM], [PM], [VM]) as [M] in Fig. 3.

3.2.2 Database structure

The database structure could improve the representation of the schema via aggregating information from neighboring nodes. As shown in Fig. 3, the database structure refers to the tables relations (e.g., Ranking points to Matches). To insert this structure information into the model, we first design the template ‘head table, points to, tail table’. Then we replace the placeholder with the specific table name. Finally, we concatenate these pairs with the delimiter ‘|’ and add them to the input. We further adopt a special token [GRAPH] as the indicator between the database structure and other input parts (e.g., schema and question) as

$$[GRAPH] t_1 \rightarrow t_2 | t_2 \rightarrow t_3 \ldots,$$

where $\rightarrow$ is the special token indicating the head table points to tail table.

3.2.3 Discourse structure

Modeling the discourse structure in multi-turn text-to-SQL would greatly enhance the ability in co-reference resolution and ellipsis recovery [12, 13, 30]. Concretely, we derive the mentioned schema of the history question to track the dialogue flow explicitly. For example, if table name $t_1$ and column name $c_{[i]}$ are mentioned during the history, we could derive the linearized database schema $S_{seq}$ as

$$[TABLE] [H] t_1 \ldots [COLUMN] c_1 \ldots [H] c_{[i]},$$

where [H] is the special token indicating the column is history mentioned.

Note that we are facing a similar exposure-bias problem in neural machine translation. That is, the gap between training (with golden context) and inference (with predicted context) would cause a discrepancy in data distribution, and further lead to performance degradation during inference. Fortunately, we observe that UniSAR reveals excellent robustness to this problem, which mostly takes advantage of the underlying PLMs.

Moreover, we conduct the oracle experiments with the golden context in Sect. 5.3.3, to explore the upper bound in adopting the discourse structure mark. We have also explored directly inserting the SQL from the last turn following HIESQL [30], instead of inserting the structure mark. However, its performance does not exceed the structure mark. Thus, we adopt this simple-yet-effective structure mark in our experiments, which benefits from the shorter input length but keeps better performance.

3.3 Constrained decoding with prefix tree

Besides the semantically correct, it’s also important to generate the executable SQL which would eventually influence the execution accuracy. That is, the output SQL should strictly obey the SQL grammar and be faithful to the database schema. In our preliminary experiments, we observe that pre-trained language models (BART and T5) are already

---

1 https://lucene.apache.org/.
skilled at learning SQL grammar, which indicates their excellent grammar learning ability. However, the vanilla language models struggle in schema prediction especially when the target schema has rich synonyms (e.g., the model generates nation in the output SQL, instead of Citizenship in the schema).

To address this problem, we proposed a simple constrained decoding method with a schema-based prefix tree inspired by [31], which is both effective and efficient. We name it PTCD (Prefix T ree based Constrain Decoding) and will adopt this abbreviation in the following sections. Next, we will introduce (1) how to build the prefix tree based on database schema?; (2) how to employ prefix tree for constrained decoding? Moreover, we will further distinguish PTCD from PICARD [18], another popular constrained decoding method for text-to-SQL.

3.3.1 Prefix tree construction

We first construct the prefix tree (also known as trie [32]) based on the database schema. The nodes of the tree are annotated with the tokens from the vocabulary (i.e., BPE tokens for BART). For each node, its child nodes refer to all the allowed tokens, which are the prefix traversing the tree from the root to this node. We emphasize that the nodes of the prefix tree are built in BPE-level rather than the word-level. Thus, traversing the prefix tree could return the multi-word table names, where the space is modeled as the index defined in the vocabulary of BART/T5.

The prefix tree can be constructed in $O(M \times \log N)$ time, where $M$ is the average length of schema elements and $N$ is the total number of schema elements inserted in the tree. One advantage of the prefix tree is its internal lookup function only needs $O(M)$ time. And the storage requirements are also acceptable considering the limited number of schema elements in realistic applications. Take Spider for example, there are 5.1 tables in each database, with 3.6 columns in each table.

3.3.2 Constrained beam search

As shown in Fig. 4, we design three modes to dynamically constrain the generation:

- **SCHEMA_MODE** for generating the legal schema elements in SQL, where the allowed continuations are derived by the prefix tree according to the generated sub-string of the schema entity.
- **KEYWORD_MODE** for generating the legal keywords in SQL, where the allowed continuations are a closed set full of keyword tokens.
- **VALUE_MODE** (No constrain) for generating the literal strings (i.e., values) in SQL, where all the vocabulary tokens are allowed to generate. Notably, this mode might also generate the start token of each schema element.

These three modes are switched during the generation process by hand-crafted rules. When encountering the space character, we determine the specific mode based on the last generated span (LGS). As shown in Fig. 4, LGS could be (1) an in-completed schema; (2) a completed schema or (3) a keyword.

More specifically, if LGS is an in-completed schema, we keep the SCHEMA_MODE to find out which token is allowed to be generated in the next step. Otherwise, if LGS is a keyword, we adopt the VALUE_MODE to generate the value or the start tokens of the schema. There are several combinations of how to switch the mode under different situations, so please...
refer to our code for more implementation details. Notably, we employed the constraints masking the log probabilities of the invalid tokens and not their logits following [31].

### 3.3.3 Comparison with PICARD: simple and efficient

The most concurrent work with ours is PICARD [18], which employs a dedicated Haskell server to regulate the decoding process. Basically, PICARD introduces four different constrained decoding levels: off (no checking), lexing (lexical-level, first-order), parsing without guards (semantic-level, second-order), and parsing with guards (schema-level, third-order). In PICARD, the prediction that passes a higher level will always include a lower level.

We distinguish our PTCD from PICARD in terms of two aspects: (1) **Implementation**: the PTCD only needs to construct the prefix-tree (first-order, lexical-level) while PICARD requires to specify the SQL grammar (second-order, semantic-level). Notably, we argue it’s *unnecessary* to apply the second-order constrained since the PLMs could learn the SQL grammar very well in our preliminary studies; (2) **Decoding Efficiency**: The time complexity of PICARD is $O(N^2)$ or $O(N^3)$ where $N$ is the total number of schema elements. In contrast, the time complexity of PTCD is $O(M)$ where $M$ is the average length of schema elements as discussed above. Besides the above qualitative analysis, we also conduct the quantitative analysis in Sect. 5.3.4 which measures both the time–cost and accuracy of PICARD and PTCD.

## 4 Experimental setup

In this section, we will introduce (1) seven popular text-to-SQL datasets used in experiments; (2) evaluation metrics for each setting; (3) baseline models including both invasive and non-invasive approaches and (4) UniSAr implementation details.

### 4.1 Datasets

We compare UniSAr with previous task-specific parsers using seven popular text-to-SQL datasets. The dataset statistics are shown in Table 1. To systemically compare our unified parser with previous task-specific models, we categorize the datasets into three groups: (1) **Multi-Domain**: WikiSQL [1] and TableQA [5]; (2) **Multi-Table**: Spider [2] and DuSQL [6]; (3) **Multi-Turn**: CoSQL [7], SparC [4] and Chase [8].

Notably, five datasets (Spider, CoSQL, SparC, DuSQL, and Chase) are also under cross-domain settings (i.e., the evaluation data comes from the different domains of the training data). Their test sets are not publicly available so we conduct all these experiments on the dev set. Table 1 lists the dataset we used.

### 4.2 Evaluation metric

For WikiSQL and TableQA, we utilize logic form accuracy (LX) and execution accuracy (EX) as evaluation metrics following [1]. For Spider and DuSQL, we report exact set match accuracy (EM) following [2]. For SparC, CoSQL, and Chase, we report question match accuracy (QM) and interaction match accuracy (IM) following [4].

### 4.3 Baseline model

For each setting, we select the most representative invasive models and non-invasive models as our baselines. For non-invasive methods, we only list the PLM in *moderate size* to make a fair comparison.

**Multi-Domain** (1) SQLNet [15] is a sketch-based method. (2) SQLova [14] is a sketch-based method that integrates the pre-trained language model; (3) Coarse2Fine [33] first generates the SQL template and then fills the value. (4) X-SQL [34] enhances the structure schema representation with contextual embedding. (5) F-SQL [16] improves the representation of schema with table content. (6) HydraNet [35] uses column-wise ranking and decoding. (7) BRIDGE [36] further leverages the database content to augment the column representation. (8) SeaD [20] trains a seq2seq model with schema-aware denoising objectives.

**Multi-Table** (1) RYANSQL [37] recursively predicts nested queries with a sketch-based slot-filling algorithm; (2) IRNet [3] utilizes SemQL as an abstract representation of SQL queries; (3) RAT-SQL [10] utilizes a complete relational-aware neural network to handle various pre-defined relations; (4) IRNetExt [6] extends IRNet to parse calculation questions and predict values.

**Multi-Turn** (1) EditSQL [13] adopts an editing-based encoder-decoder model where the tokens are either

| Dataset        | Language | #Question | Len(SQL) | #Table | #DB   |
|----------------|----------|-----------|----------|--------|-------|
| WikiSQL [1]    | EN       | 80,654    | 10.6     | 26,521 | 26,521|
| TableQA [5]    | ZH       | 49,974    | 9.1      | 5291   | 5291  |
| Spider [2]     | EN       | 10,181    | 21.7     | 1020   | 200   |
| DuSQL [6]      | ZH       | 23,797    | 20.2     | 820    | 200   |
| CoSQL [7]      | EN       | 15,598    | 18.4     | 1020   | 200   |
| SparC [4]      | EN       | 12,726    | 17.8     | 1020   | 200   |
| Chase [8]      | ZH       | 17,940    | 20.9     | 1020   | 200   |

Len(SQL) means the average length of SQL, which somehow implies the complexity of SQL generation

*EN* English, *ZH* Chinese
generated from the vocab or copied from the history; (2) IGSQL [12] proposes a schema interaction graph encoder to utilize historical information of database schema items; (3) RAT-SQL-con [10] is the extension of RAT-SQL for multi-turn settings implemented by [8].

4.4 Implementation details

The training process for BART/T5 takes about 10–22 h (in different model sizes) with four NVIDIA Tesla V100-32G GPUs. Code is available at link.

Fine-tuning BART

We conduct the experiments using the Fairseq [38] to do data pre-processing, training, and inference. We adopt the BART-Large and set the task of Fairseq as TRANSLATION_FROM_PRETRAINED_BART. The learning rate is $1 \times 10^{-5}$. The max tokens are set to 1800. We adopt the polynomial_decay with 5000 warmup updates. Mixed precision (stage 3) is adopted to speed up the model training. The optimizer is Adam [40] with the default parameters. The max-update is set to 10,000. Empirically, the model obtained the best performance of about 7000 steps (about 10–15 epochs) in Spider, CoSQL, and SParC. Notably, Fairseq [38] dynamically tunes the batch size to realize higher GPU utilization.

Fine-tuning T5

We fine-tune T5 model [23] using Deepspeed framework [41] and Transformer library [42]. The batch size is set to 2048 following the setup of PICARD [18], which indicates the larger batch size leads to better performance and a more stable training process. Mixed precision (stage 3) is adopted to speed up the model training. For optimizer, since Deepspeed doesn’t support the Adafactor [43] (which PICARD adopted and indicates better performance for the T5 family model), we use the AdamW [44] instead.

Post-process: SQL completion

In our study, we find that the generated SQL statements often miss some JOIN components, since they are often not explicitly mentioned in natural language questions. To make the SQL complete, we need to find back the potential missing JOIN components based on the database schema. Concretely, we first construct a schema graph, where the nodes are tables or columns, and the edges are schema relationships. Then we try to find the tables and columns that are located in the shortest path of the existing tables and columns in an in-completed SQL. Take the case in Fig. 5 for example, the in-completed SQL that does not predict table MATCHES in JOIN clause. Notably, we infer the table MATCHES based on its neighbors: PLAYERS and RANKING, which are already predicted by UniSAR.

Concretely, we first construct a schema graph, where the nodes are tables or columns, and the edges are schema relationships. Then we try to find the tables and columns that are located in the shortest path of the existing tables and columns in an in-completed SQL. Take the case in Fig. 5 for example, the in-completed SQL that does not predict table MATCHES in JOIN clause. Notably, we infer the table MATCHES based on its neighbors: PLAYERS and RANKING, which are already predicted by UniSAR and MATCHES is located on the path of these two tables. We could also infer the missing column WINNER_ID, the primary key of MATCHES. Consequently, we amend the broken SQL predicted path by a simple heuristic rule rather than the complex constraint like PICARD [18].

Learning structure mark

For learning structure marks, like [EM], [PM], [VM], and [H], we investigate two approaches: (1) we directly treat these special tags as the normal tokens and then learn their representation from scratch; (2) we initialize these tags with the ‘special tokens’ mechanism in Fairseq with modifying the ‘dict.json’ and vocab reading code. The experimental results on Spider and DuSQL indicate these two approaches yield similar performances. Given the simplicity of the first approaches, we adopt this in our remaining experiments. For other tokens (like SQL keywords), we just treat them like normal tokens.

Handling lengthy inputs

To handle the lengthy inputs that exceed the maximum limits of PLMs, we employ heuristic rules to truncate the input. Lengthy inputs are a common issue for two primary reasons. Firstly, databases with numerous tables tend to have the large number of columns that require processing, which may result in exceeding the PLM’s input length limit. Secondly, including structure mark information in the input may introduce irrelevant data, further exacerbating this issue. For example, the ‘baseball’ database in Spider has 353 columns. To handle the lengthy inputs containing many columns, we only retain the columns that
(1) are mentioned during the schema linking process, or (2) are mentioned during the historical SQLs, or (3) have relationships with the columns in (1) or (2) such as being in the same table or linked tables. Consequently, this approach would shorten the [Table], [Column] and [Graph] parts as listed in Fig. 3.

5 Results and analysis

In this section, we will first discuss the main results over three settings as introduced in Sect. 4. Then, we will conduct the ablation studies to specify the effectiveness of each component of UniSAr. After that, we will explore four interesting questions to study extensibility, upper-bound, generalizability, and efficiency respectively. At last, we will discuss the limitations of UniSAr and propose potential advancements.

5.1 Main results

The results of multi-domain, multi-table, and multi-turn are listed in Tables 2, 3, and 4. Most results of previous models are reported by the cited papers respectively. For WikiSQL, we re-implement RAT-SQL with BERT-Large. For multi-turn settings, the results of the invasive model are reported by [8]. Basically, UniSAr achieves excellent performance under all settings, which demonstrates the effectiveness of two extensions.

In the following, we interpret the results of three settings separately to highlight the effectiveness of UniSAr.

Multi-domain As shown in Table 2, we could observe that (1) compared with invasive approaches, the moderate-size PLMs could obtain similar or better performance; (2) compared with non-invasive approaches (vanilla PLMs), UniSAr outperform them by 1.7% and 1.5% in BART and T5 setting

| Model            | WikiSQL | TableQA |
|------------------|---------|---------|
|                  | Dev     | Test    | Dev     | Test    |
|                  | LX EX   | LX EX   | LX EX   | LX EX   |
| Invasive         |         |         |         |         |
| SQLNet [15]      | – 69.8  | – 68.0  | – 61.4  | 67.2    |
| Coarse2Fine [33]| 72.5    | 71.7    | 72.6    | 76.7    |
| SQLova [14]      | 81.6    | 80.7    | 81.7    | 85.8    |
| X-SQL [34]       | 83.3    | 83.2    | 83.3    | 87.8    |
| F-SQL [16]       | – 85.6  | 91.4    | – 90.4  | 93.2    |
| HydraNet [35]    | 83.6    | 83.8    | – 89.2  | – –     |
| BRIDGE [36]      | 86.2    | 85.7    | 88.4    | 91.8    |
| Non-invasive     |         |         |         |         |
| T5-Large [23]    | 84.2    | 83.4    | 86.3    | 89.9    |
| BART-Large [21]  | 83.7    | 84.7    | 88.7    | 91.8    |
| SeaD [20]        | 84.9    | 90.2    | 87.2    | 90.2    |
| UNISAR_T5-Large  | 85.9    | 86.1    | 87.2    | 91.8    |
| UNISAR_BART-Large| **86.7**| **91.7**| **89.9**| **95.1**|

Note that we report the models without using Execution-Guided Decoding. Invasive stands for invasive approaches and non-invasive stands for non-invasive approaches.

The bold values represent the best scores for each metric.

5.2 Ablation studies

In this section, we will conduct the ablation studies to specify the effectiveness of each component of UniSAr. After that, we will explore four interesting questions to study extensibility, upper-bound, generalizability, and efficiency respectively. At last, we will discuss the limitations of UniSAr and propose potential advancements.

Multi-domain As shown in Table 2, we could observe that (1) compared with invasive approaches, the moderate-size PLMs could obtain similar or better performance; (2) compared with non-invasive approaches (vanilla PLMs), UniSAr outperform them by 1.7% and 1.5% in BART and T5 setting

| Model            | Spider | DuSQL |
|------------------|--------|-------|
| Invasive approaches |       |       |
| RYANSQL [37]     | 66.6   | –     |
| IRNet [3]        | 63.9   | 38.4  |
| IRNetExt [6]     | –      | 59.8  |
| RAT-SQL [10]     | 69.7   | –     |
| BRIDGE [36]      | 70.0   | –     |
| LGESQL [45]      | **75.1**| –     |
| Non-invasive approaches |       |       |
| T5-Large [23]    | 65.3   | 82.7  |
| BART-Large [21]  | 64.5   | 82.2  |
| T5-Large+PICARD [18] | 69.1 | –     |
| UNISAR_T5-Large  | 68.1   | 83.7  |
| UNISAR_BART-Large| **70.0**| **84.3**|

The performance of T5-Large is reported by PICARD [18]. The bold values represent the best scores for each metric.

5.3 Exploring interesting questions

In this section, we will conduct the ablation studies to specify the effectiveness of each component of UniSAr. After that, we will explore four interesting questions to study extensibility, upper-bound, generalizability, and efficiency respectively. At last, we will discuss the limitations of UniSAr and propose potential advancements.

Multi-domain As shown in Table 2, we could observe that (1) compared with invasive approaches, the moderate-size PLMs could obtain similar or better performance; (2) compared with non-invasive approaches (vanilla PLMs), UniSAr outperform them by 1.7% and 1.5% in BART and T5 setting

Note that we report the models without using Execution-Guided Decoding. Invasive stands for invasive approaches and non-invasive stands for non-invasive approaches.

The bold values represent the best scores for each metric.

5.4 Discussion on limitations and potential advancements

In this section, we will conduct the ablation studies to specify the effectiveness of each component of UniSAr. After that, we will explore four interesting questions to study extensibility, upper-bound, generalizability, and efficiency respectively. At last, we will discuss the limitations of UniSAr and propose potential advancements.

Multi-domain As shown in Table 2, we could observe that (1) compared with invasive approaches, the moderate-size PLMs could obtain similar or better performance; (2) compared with non-invasive approaches (vanilla PLMs), UniSAr outperform them by 1.7% and 1.5% in BART and T5 setting

Note that we report the models without using Execution-Guided Decoding. Invasive stands for invasive approaches and non-invasive stands for non-invasive approaches.

The bold values represent the best scores for each metric.

5.5 Conclusion

In this section, we will conduct the ablation studies to specify the effectiveness of each component of UniSAr. After that, we will explore four interesting questions to study extensibility, upper-bound, generalizability, and efficiency respectively. At last, we will discuss the limitations of UniSAr and propose potential advancements.

Multi-domain As shown in Table 2, we could observe that (1) compared with invasive approaches, the moderate-size PLMs could obtain similar or better performance; (2) compared with non-invasive approaches (vanilla PLMs), UniSAr outperform them by 1.7% and 1.5% in BART and T5 setting

Note that we report the models without using Execution-Guided Decoding. Invasive stands for invasive approaches and non-invasive stands for non-invasive approaches.

The bold values represent the best scores for each metric.
respectively; (3) compared with SeaD [20], which further pretrained the PLMs by schema-aware denoising objective, UniSAr achieves better performance with greater efficiency (only fine-tuning needed).

As shown in Table 3, we could observe that (1) UniSAr outperforms the vanilla PLMs by 4.1% in Spider and 1.6% in DuSQL, which demonstrates the effectiveness of our proposed methods; (2) both vanilla PLMs and UniSAr are still far behind the advanced invasive approaches like LGESQL [45] (with complex mechanisms in modeling database structure), which indicates that modeling the structure information is definitely a challenge; (3) T5-Large+PICARD [18] exceeds UNISAR_T5-Large by 1%, which implies that the well-designed constrained-decoding could inspire the better performance of vanilla PLMs. But our following experiments in Sect. 5.3.1 conclude that the structure mark could serve as a good complement to PICARD, which further improves the PICARD by 0.7%.

Multi-turn As shown in Table 4, we could observe that (1) vanilla PLMs (both T5 and BART) are not very effective compared with the multi-turn version RAT-SQL, which indicates that simply concatenating the question history could not fully encode the context information; (2) UniSAr (with structure mark) not only exceeds the vanilla PLMs but also outperforms the invasive models (with context-oriented modules), which implies that the discourse-level mark indeed provides the context information; (3) T5 achieves better performance than BART in multi-turn settings, which could be attributed to the that the pretraining task of T5 involves more context understanding tasks like reading comprehension and question answering.2

5.2 Ablation studies

To study the effect of two non-invasive approaches and SQL completion post-process, we conduct an ablation study on Spider, CoSQL, and SParC. As shown in Table 5, (1) the structure mark significantly boosts the performance of BART-Large, implying that it does effectively represent the structure knowledge, and the schema-linking mark is the most important part which improves the performance by 2.6%; (2) from the view of decoding, the constrained generation solution also boosts the performance by 2.3%, since it addresses the problem of generating invalid SQL; (3) the post-processing (SQL Completion) promotes the performance by 1% through mitigate the incompletely SQL generation.

5.3 Analysis

To conduct a comprehensive study about UniSAr, we explore the following research questions: RQ1 about

Table 5 The ablation study results of UniSAR_BART-Large without a different type of structure mark, constrained decoding, and SQL completion

| Model          | Spider | CoSQL | SParC |
|----------------|--------|-------|-------|
| UNISAR_BART-Large | 70.0   | 51.8  | 60.4  |
| Schema-linking mark | 67.2 (−2.8%) | 49.3 (−2.5%) | 57.8 (−2.6%) |
| Database mark   | 69.1 (−0.9%) | 51.1 (−0.7%) | 59.3 (−1.1%) |
| Discourse mark  | –      | 51.3 (−0.5%) | 59.7 (−0.7%) |
| All mark        | 66.9 (−3.1%) | 48.7 (−3.1%) | 57.4 (−3.0%) |
| Constrained decoding | 67.5 (−2.5%) | 50.1 (−1.7%) | 57.8 (−2.6%) |
| SQL completion  | 68.8 (−1.2%) | 50.8 (−1.0%) | 58.9 (−1.5%) |

The exact match score is reported
The bold values represent the best scores for each metric.

---

2 https://ai.googleblog.com/2020/02/exploring-transfer-learning-with-t5.html.
RQ2 about upper-bound: What’s the performance of larger PLMs with structure marks? RQ3 about generalizability: Would the UniSAr performance be further improved under unified training with different text-to-SQL datasets? RQ4 about efficiency: What’s the efficiency (i.e., time cost) of prefix tree based constrained decoding?

5.3.1 Under larger pre-trained language models

UniSAr is implemented based on BART-Large and mBART-CC25, which has 400M or 610M parameters respectively. They are both the middle-size pre-trained model. We further explore the structure mark with larger PLM to examine whether the improvement would be consistent as UniSAr.

Experimental results in Table 6 demonstrate that: (1) the different size T5 model receives the performance gain the structure mark overall, implying the explicit structure modeling about input data is still useful for large pre-trained encoder-decoder model; (2) structure mark could also improve the PICARD (the most excellent decode strategy so far) further, indicating the structure information should be considered from both encoding and decoding sides for text-to-SQL problems.

5.3.2 Under oracle structure mark

To explore the upper bound of performance boost that structure mark brings, we conduct experiments under the oracle setting. Concretely, we derive oracle schema-linking by human annotation from [25] for Spider, and oracle previous SQL for CoSQL, SParC, and Chase.

Experimental results demonstrate that (1) UniSAr gets a 1.2% performance boost in Spider; (2) UniSAr receives 5.9%, 8.7%, 11.7% boost in multi-turn datasets (SParC, CoSQL, Chase) respectively. These experiments indicate that UniSAr could be improved further if we could obtain higher-quality structure marks. For example, we could replace the fuzzy-match-based schema-linking with model-based grounding [28]. Concretely, the fuzzy-match-based schema-linking could only get a 65.1% F1-score in Spider dataset while ETA [28] could obtain 82.5% F1-score.

5.3.3 Explore the generalizability under unified training

UniSAr is easier to do multi-task training since its unified architecture enables different tasks to share the sample training protocol. To explore the generalizability of UniSAr, we conduct the multi-task training of single-turn (Spider) + multi-turn (CoSQL and SParC), and compare its performance with RAT-SQL [10].

Experimental results in Table 7 demonstrate that (1) both BART and UniSAr show a positive trend by jointly training; (2) RAT-SQL is degraded in the multi-task setting compared with the single-task setting. We argue that the data distribution across different settings is different, despite that these datasets are based on the same database and have the near size of training data. The divergence of RAT-SQL could be attributed to the task-specific modules (e.g. grammar-based decoder) which make the model easily overfit to a specific dataset (i.e., specific SQL distribution). To conclude, UniSAr and PLMs have better generalizability in the multi-task setting or unified training.

Table 6 Results of different size T5 model with structure mark (SM)

| Model      | Setting | Spider EM | CoSQL EM | IM |
|------------|---------|-----------|----------|----|
| T5-Base (220M) | Vanilla | 57.2      | 47.6     | 17.2 |
|            | SM      | 60.1      | 50.6     | 20.4 |
|            | PICARD  | 65.8      | 51.7     | 20.7 |
|            | SM + PICARD | 66.9     | 53.1     | 21.5 |
| T5-Large (770M) | Vanilla | 65.3      | 52.5     | 21.5 |
|            | SM      | 68.1      | 53.7     | 22.4 |
|            | PICARD  | 69.1      | 54.2     | 22.6 |
|            | SM + PICARD | 70.2     | 55.1     | 23.2 |
| T5-3B (3B)   | Vanilla | 69.9      | 53.8     | 21.8 |
|            | SM      | 72.2      | 55.7     | 22.3 |
|            | PICARD  | 75.5      | 56.9     | 24.2 |
|            | SM + PICARD | 75.9     | 57.8     | 25.8 |

Table 7 Logical Form Accuracy (LX) of the joint training across single-turn (Spider) and multi-turn (CoSQL and SParC)

| Model | Setting | Spider EM | CoSQL EM | SParC EM |
|-------|---------|-----------|----------|----------|
| RAT-SQL | Single | 69.7      | 50.8     | 60.1     |
|       | Joint  | 68.2      | 50.0     | 58.2     |
| Δ     | − 1.5% | − 0.8%    | − 1.9%   |          |
| BART  | Single | 64.5      | 47.1     | 55.0     |
|       | Joint  | 67.5      | 50.1     | 57.8     |
| Δ     | + 3.0% | + 3.9%    | + 2.8%   |          |
| UniSAr| Single | 70.0      | 51.8     | 60.4     |
|       | Joint  | 70.8      | 52.9     | 60.9     |
| Δ     | + 0.8% | + 1.1%    | + 0.5%   |          |

Δ indicates the performance difference between single and joint settings. RAT-SQL is trained with BERT-Large. UniSAr and BART are both built on top of BART-Large.
5.3.4 Decoding efficiency of PTCD

Besides discussing the implementation difference between PTCD and PICARD [18] (Sect. 3.3.3), we further conduct the quantities experiments to compare the decoding efficiency and accuracy. Concretely, the experiments are conducted with T5-3B model on Spider dataset with an NVIDIA A100-40GB GPU. Experimental results demonstrate (1) for the vanilla model, the decoding speed is 2.5 s per sample with 69.9% accuracy; (2) with PICARD, it’s 3.1 s per sample with 75.5% accuracy; (3) with PTCD, it’s 2.7 s per sample with 74.8% accuracy. We could observe that PTCD only leaves behind PICARD about 0.7% while reducing 33% decoding time cost. Simultaneously, PTCD is easier to implement in Python and expand for complex scenarios (e.g., nested SQL).

5.4 Case studies

In Fig. 6, we compare the SQL generated by UniSAr with those predicted by the vanilla BART. We notice that UniSAr performs better than the BART, especially on examples that involve multiple tables.

For example, in the first case where BART fails to identify the existence of table DOCUMENTS. For comparison, UniSAr successfully predicts the connection of two tables since the structure mark presents the database structure. In the second case, vanilla BART predicts a token (e.g., COMPETITOR) that does not exist in the database schema. This will cause an ill-formed SQL but UniSAr ensures the faithful generation with constrained decoding. Moreover, in the third case, we can find that SQL completion infers the missing table CAR_NAMES based on the matched table MODEL_LIST and CAR_DATA.

In the fourth case from the multi-turn setting, UniSAr still outperforms the BART in contextual modeling by effectively encoding the information of dialogue history. However, in the last case, our UniSAr is awkward to predict unnecessary table AIRPORTS. This error perhaps can be attributed to an inappropriate structure mark due to inaccurate schema-linking. A high-precision structure mark will alleviate this problem.

5.5 Limitations and potential advancements

The aforementioned experiments were conducted using benchmarks provided by the research community, which may not fully reflect the complexities of real-world scenarios. To illustrate this, we will examine two representative cases that pose challenges for UniSAr: (1) semantic-level schema linking and (2) fine-grained value matching. We will...
discuss the limitations of UniSAR in these challenges and propose potential advancements.

Semantic-level schema linking UniSAR may face limitations in accurately linking schemas at a semantic level. This could be challenging due to issues such as ambiguity in schema representations, lack of contextual information, or difficulties in identifying subtle semantic nuances. For instance, real-world scenarios often involve synonyms (such as actor and performer) and polysemous words (such as player, which can refer to both a competitor and a musician depending on the context). To address this, future improvements could involve incorporating more contextual and domain-specific knowledge, leveraging external resources, or employing advanced semantic parsing techniques to better capture the nuances of schema linking.

Fine-grained value matching UniSAR may struggle with fine-grained value matching, which requires precise identification and matching of specific values within a schema. This could be challenging due to variations in data formats, units, or representations, as well as inconsistencies or errors in the data. Future advancements could involve incorporating more sophisticated pattern matching algorithms, leveraging machine learning techniques to learn value representations, or integrating data validation and error correction mechanisms into UniSAR to improve its ability to accurately match fine-grained values.

6 Conclusion and future work

In this work, we propose UniSAR, a simple-yet-effective approach to address text-to-SQL across various settings in a unified framework. Concretely, we extend the pre-trained language model (e.g., BART and T5) with two non-invasive extensions (i.e., structure mark and prefix tree based constrained decoding) to make it structure-aware. Experimental results demonstrate the effectiveness of our UniSAR on seven well-known text-to-SQL datasets. UniSAR achieves comparable or better performance to the most advanced specifically-designed text-to-SQL models.

For future work, it is easy to switch UniSAR to giant language models if there are sufficient computing resources. Compared with the traditional model-centric approaches (i.e., design specific modules for different settings), we believe that the data-centric approaches like UniSAR will gradually be developed better given the recent advancements in Large Language Models (LLMs). Our work somehow proves the idea that simple task-specific data manipulation is pretty effective compared with designing complex task-specific modules.

Besides the research paradigm, it’s also interesting to explore how to model other structure knowledge (e.g., knowledge graph and hierarchical table) with structure marks to advance other structure-related NLP tasks. For instance, the structure mark would definitely improve the model like UnifiedSKG [19], which requires encoding the different types of structure information for better reasoning ability.

Acknowledgements We thank all anonymous reviewers for their constructive comments. Wanxiang Che was supported via the grant 2020AAA0106501, 62236004 and 61976072.

Data availability The dataset listed in Table 1 are publicly available. WikiSQL: https://github.com/salesforce/WikiSQL. TableQA: https://github.com/ZhuYiiTechnology/TableQA. Spider: https://yale-lily.github.io/spider. DuSQL: https://github.com/luge-ai/du-sql. CoSQL: https://yale-lily.github.io/cosql. SParC: https://yale-lily.github.io/sparc. Chase: https://sjtu-intsoft.github.io/chase/.

References

1. Zhong V, Xiong C, Socher R (2017) Seq2sql: generating structured queries from natural language using reinforcement learning. arXiv:1709.00103
2. Yu T, Zhang R, Yang K, Yasunaga M, Wang D, Li Z, Ma J, Li I, Yao Q, Roman S, Zhang Z, Radev D (2018) Spider: a large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. EMNLP2018. https://aclanthology.org/D18-1425
3. Guo J, Zhan Z, Gao Y, Xiao Y, Lou J-G, Liu T, Zhang D (2019) Towards complex text-to-SQL in cross-domain database with intermediate representation. Association for Computational Linguistics, Florence, Italy. https://aclanthology.org/P19-1444. Accessed 2 Aug 2019
4. Yu T, Zhang R, Yasunaga M, Tan YC, Lin XV, Li S, Er H, Li I, Pang B, Chen T, Ji E, Dixit S, Proctor D, Shim S, Kraft J, Zhang V, Xiong C, Socher R, Radev D (2019) SParC: cross-domain semantic parsing in context. Association for Computational Linguistics, Florence, Italy. https://aclanthology.org/P19-1443. Accessed 2 Aug 2019
5. Sun N, Yang X, Liu Y (2020) TableQA: a large-scale Chinese text-to-SQL dataset for table-aware SQL generation. arXiv:2006.06434
6. Wang L, Zhang A, Wu K, Sun K, Li Z, Wu H, Zhang M, Wang H (2020) DuSQL: a large-scale and pragmatic Chinese text-to-SQL dataset. Association for Computational Linguistics, Online. https://aclanthology.org/2020.emnlp-main.562. Accessed 20 Nov 2020
7. Yu T, Zhang R, Er H, Li S, Xue E, Pang B, Lin XV, Tan YC, Shi T, Li Z, Jiang Y, Yasunaga M, Shim S, Chen T, Fabbri A, Li Z, Chen L, Zhang Y, Dixit S, Zhang V, Xiong C, Socher R, Lasecki W, Radev D (2019) CoSQL: a conversational text-to-SQL challenge towards cross-domain natural language interfaces to databases. Association for Computational Linguistics, Hong Kong, China. https://aclanthology.org/D19-1204. Accessed 7 Nov 2019
8. Guo J, Si Z, Wang Y, Liu Q, Fan M, Lou J-G, Yang Z, Liu T (2021) Chase: a large-scale and pragmatic Chinese dataset for cross-database context-dependent text-to-SQL. Association for Computational Linguistics, Online. https://doi.org/10.18653/v1/2021.acl-long.180. https://aclanthology.org/2021.acl-long.180
9. Yin P, Neubig G (2018) TRAXN: a transition-based neural abstract syntax parser for semantic parsing and code generation. Association for Computational Linguistics, Brussels, Belgium. https://aclanthology.org/D18-2002. Accessed 4 Nov 2018
1. Wang B, Shin R, Liu X, Polozov O, Richardson M (2020) RAT-SQL: relation-aware schema encoding and linking for text-to-SQL parsers. Association for Computational Linguistics, Online. https://doi.org/10.18653/v1/2020.acl-main.677. https://aclanthology.org/2020.acl-main.677

10. Wang B, Shin R, Liu X, Polozov O, Richardson M (2020) RAT-SQL: relation-aware schema encoding and linking for text-to-SQL parsers. Association for Computational Linguistics, Online. https://doi.org/10.18653/v1/2020.acl-main.677. https://aclanthology.org/2020.acl-main.677

11. Bogin B, Gardner M, Berant J (2019) Global reasoning over database structures for text-to-SQL parsing. Association for Computational Linguistics, Hong Kong, China. https://aclanthology.org/D19-1378. Accessed 7 Nov 2019

12. Cai Y, Wan X (2020) IGSQL: database schema interaction graph based neural model for context-dependent text-to-SQL generation. Association for Computational Linguistics, Online. https://doi.org/10.18653/v1/2020.emnlp-main.560. https://aclanthology.org/2020.emnlp-main.560

13. Zhang R, Yu T, Er H, Shim S, Xue E, Lin XV, Shi T, Xiong C, Socher R, Radev D (2019) Editing-based SQL query generation for cross-domain context-dependent questions. Association for Computational Linguistics, Hong Kong, China. https://aclanthology.org/D19-1537. Accessed 7 Nov 2019

14. Huw W, Yim J, Park S, Seo M (2019) A comprehensive exploration on wikisql with table-aware word contextualization. arXiv:1902.01069

15. Xu X, Liu C, Song D (2017) Sqlnet: Generating structured queries from natural language without reinforcement learning. arXiv preprint. arXiv:1711.04436

16. Zhang X, Yin F, Ma G, Ge B, Xiao W (2020) F-sql: fuse table schema and table content for single-table text2sql generation. IEEE Access 8:136409–136420. https://doi.org/10.1109/ACCESS.2020.3011747

17. Shin R, Lin CH, Thomson S, Chen C, Roy S, Platanios EA, Pauls A, Klein D, Eisner J, Durme BV (2021) Constrained language models yield few-shot semantic parsers. CoRR. arXiv:2104.08768

18. Scholak T, Schucher N, Bahdanau D, Bahdanau D (2021) PICARD: parsing incrementally for constrained auto-regressive decoding from language models. EMNLP2021. https://aclanthology.org/2021.emnlp-main.779

19. Xie T, Wu CH, Shi P, Zhong R, Scholak T, Yasunaga M, Wu C-S, Zhong M, Yin P, Wang SL, Zhong V, Wang B, Li C, Boyle C, Ni A, Yao Z, Radev D, Xiong C, Kong L, Zhang R, Smith NA, Zettlemoyer L, Yu T (2022) UnifiedSKG: unifying and multi-tasking structured knowledge grounding with text-to-text language models. arXiv preprint. arXiv:2201.05966

20. Xin K, Wang Y, Wang Y, Wen Z, Dong Y (2021) Sead: End-to-end text-to-SQL generation with schema-aware denoising. arXiv.org. arXiv:2105.07911

21. Lewis M, Liu Y, Goyal N, Ghazvininejad M, Mohamed A, Levy O, Stoyanov V, Zettlemoyer L (2020) BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. Association for Computational Linguistics, Online. https://aclanthology.org/2020.acl-main.703. Accessed 10 Jul 2020

22. Liu Y, Wan Y, He L, Peng H, Yu PS (2020) KG-BART: knowledge graph-augmented BART for generative commonsense reasoning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol 35, no 7, pp 6418–6425

23. Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, Zhou Y, Li W, Liu P (2020) Exploring the limits of transfer learning with a unified text-to-text transformer. J Mach Learn Res 21(140):1–67

24. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Part of Advances in Neural Information Processing Systems 30 (NIPS 2017)

25. Lei W, Wang W, Ma Z, Gan T, Lu W, Kan M-Y, Chua T-S (2020) Re-examining the role of schema linking in text-to-SQL. Association for Computational Linguistics, Online. https://aclanthology.org/2020.emnlp-main.564. Accessed 20 Nov 2020

26. Aghajanyan A, Ohkonou D, Lewis M, Joshi M, Xu H, Ghosh G, Zettlemoyer L (2022) Hlm: hyper-text pre-training and prompting of language models. ICLR

27. Chen X, Zhang N, Xie X, Deng S, Yao Y, Tan C, Huang F, Si L, Chen H (2021) Knowprompt: knowledge-aware prompt-tuning with synergistic optimization for relation extraction. arXiv:2104.07650

28. Liu Q, Yang D, Zhang J, Guo J, Zhou B, Lou J-G (2021) Awakening latent grounding from pretrained language models for semantic parsing. In: Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021. Association for Computational Linguistics, pp 1174–1189, Online. https://aclanthology.org/2021.findings-acl.100

29. Sun Y, Tang D, Duan N, Ji J, Cao G, Feng X, Qin B, Liu T, Zhou M (2018) Semantic parsing with syntax- and table-aware SQL generation. Association for Computational Linguistics, Melbourne, Australia. https://aclanthology.org/P18-1034. Accessed 20 Jul 2018

30. Zheng Y, Wang H, Dong B, Wang X, Li C (2022) HIE-SQL: history information enhanced network for context-dependent text-to-SQL semantic parsing. ACL2022. arXiv:2203.07376

31. Cao ND, Izakard G, Riedel S, Petroni F (2021) Autoregressive entity retrieval. ICLR2021. https://openreview.net/forum?id=5k8F6U39V

32. Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms. The MIT Press, Cambridge

33. Dong L, Lapata M (2018) Coarse-to-fine decoding for neural semantic parsing. Association for Computational Linguistics, Melbourne, Australia. https://aclanthology.org/P18-1068. Accessed 20 Jul 2018

34. He P, Mao Y, Chakrabarti K, Chen W (2019) X-SQL: reinforce schema representation with context. arXiv:1908.08113

35. Lyu Q, Chakrabarti K, Hathi S, Kundu S, Zhang J, Chen Z (2020) Hybrid ranking network for text-to-SQL. arXiv:2008.04759

36. Lin XV, Socher R, Xiong C (2020) Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. Association for Computational Linguistics, Online. https://aclanthology.org/2020.findings-emnlp.438. Accessed 20 Nov 2020

37. Choi D, Shin M, Kim E, Shin DR (2020) RYAASQL: recursively applying sketch-based slot fillings for complex text-to-SQL in cross-domain databases. CoRR. arXiv:2004.03125

38. Ott M, Edunov S, Baevski A, Fan A, Gross S, Ng N, Grangier D, Auli M (2019) fairseq: A fast, extensible toolkit for sequence modeling. Association for Computational Linguistics, Minneapolis, Minnesota. https://aclanthology.org/N19-4009. Accessed 7 Jun 2019

39. Srivastava N, Hinton GE, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15:1929–1958

40. Kingma DP, Ba J (2015) Adam: a method for stochastic optimization. CoRR. arXiv:1412.6980

41. Rasley J, Rajbhandari S, Ruwase O, He Y (2020) DeepSpeed: system optimizations enable training deep learning models with over 100 billion parameters. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3394486.3406703

42. Wolf T, Debut L, Sanh V, Chaumond J, Delangue C, Moi A, Rault S, Louf R, Funtowicz M, Davison J, Shlifer S, von Platen P, Ma C, Jernite Y, Plu J, Xu C, Le Scao T, Gugger S, Drame M, Lhoest Q, Rush A (2020) Transformers: state-of-the-art natural language processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. Association for Computational Linguistics, pp 38–45. Online. https://aclanthology.org/2020.emnlp-demos.6
43. Shazeer N, Stern M (2018) Adafactor: adaptive learning rates with sublinear memory cost. PMLR. https://proceedings.mlr.press/v80/shazeer18a.html

44. Loshchilov I, Hutter F (2019) Decoupled weight decay regularization. ICLR 2019. https://openreview.net/forum?id=Bkg6RlCqY7

45. Cao R, Chen L, Chen Z, Zhao Y, Zhu S, Yu K (2021) LGESQL: line graph enhanced text-to-SQL model with mixed local and non-local relations. ACL 2021

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.