Combining Small Language Models and Large Language Models for Zero-Shot NL2SQL

Ju Fan
Renmin University of China
fanj@ruc.edu.cn

Zihui Gu
Renmin University of China
guzh@ruc.edu.cn

Songyue Zhang
Renmin University of China
zhangsongyue@ruc.edu.cn

Yuxin Zhang
Renmin University of China
zhangyuxin159@ruc.edu.cn

Zui Chen
MIT CSAIL
chenz429@mit.edu

Lei Cao
University of Arizona/MIT
lcao@csail.mit.edu

Guoliang Li
Tsinghua University
liguoliang@tsinghua.edu.cn

Samuel Madden
MIT CSAIL
madden@csail.mit.edu

Xiaoyong Du
Renmin University of China
duyong@ruc.edu.cn

Nan Tang
HKUST (Guangzhou) / HKUST
nantang@hkust-gz.edu.cn

ABSTRACT

Zero-shot natural language to SQL (NL2SQL) aims to generalize pre-trained NL2SQL models to new environments (*e.g.,* new databases and new linguistic phenomena) without any annotated NL2SQL samples from these environments. Existing approaches either use small language models (SLMs) like BART and T5, or prompt large language models (LLMs). However, SLMs may struggle with complex natural language reasoning, and LLMs may not precisely align schemas to identify the correct columns or tables. In this paper, we propose a ZeroNL2SQL framework, which divides NL2SQL into smaller sub-tasks and utilizes both SLMs and LLMs. ZeroNL2SQL first fine-tunes SLMs for better generalizability in SQL structure identification and schema alignment, producing an *SQL sketch.* It then uses LLMs's language reasoning capability to fill in the missing information in the SQL sketch. To support ZeroNL2SQL, we propose novel database serialization and question-aware alignment methods for effective sketch generation using SLMs. Additionally, we devise a multi-level matching strategy to recommend the most relevant values to LLMs, and select the optimal SQL query via an execution-based strategy. Comprehensive experiments show that ZeroNL2SQL achieves the best zero-shot NL2SQL performance on benchmarks, *i.e.,* outperforming the state-of-the-art SLM-based methods by 5.5% to 16.4% and exceeding LLM-based methods by 10% to 20% on execution accuracy.

PVLDB Reference Format:
Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Samuel Madden, Xiaoyong Du, and Nan Tang. Combining Small Language Models and Large Language Models for Zero-Shot NL2SQL . PVLDB, 17(11): 2750 - 2763, 2024.
doi:10.14778/3681954.3681960

[*] Nan Tang is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.
doi:10.14778/3681954.3681960

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/ruc-datalab/ZeroNL2SQL.

1 INTRODUCTION

Natural Language to SQL (**NL2SQL**), which translates a natural language question into an SQL query, makes it easier for non-technical users to access and analyze data, and thus can be useful for business intelligence, data analytics, and other data-driven applications.

Figure 1 shows how (a) a natural language question Q posed over (b) a database D can be translated into (c) an SQL query S.

Zero-shot NL2SQL. A practical scenario for NL2SQL is that oftentimes, for new databases or applications, high-quality training data (*i.e.,* annotated NL2SQL samples) is time-consuming and labor-intensive to acquire, and thus is not available. In this paper, we refer to the case of *zero* annotated NL2SQL samples as *zero-shot NL2SQL*. Thus, a natural question is: can we utilize existing annotated NL2SQL samples, either from public benchmarks (*e.g.,* Spider [48]) or within enterprises, to train an NL2SQL model that is *generalizable* to new test environments? An affirmative answer to this question has the potential to dramatically reduce the expert knowledge and human efforts for training data annotation.

However, the key obstacle to answer the question is that test environments for NL2SQL may be very different from existing annotated datasets, which may include the following cases. (1) *new databases*: an NL2SQL model trained on the Spider [48] benchmark may not perform well for domain-specific (*e.g.,* academic or financial) databases [16, 28]. (2) *new linguistic phenomena*: varying linguistic phenomena (*e.g.,* abbreviations, synonyms, etc.) in the test environments may lead to dramatic performance declines for existing NL2SQL models [4].

State of the Art: Strengths and Limitations. The state-of-the-art (SOTA) solutions for NL2SQL mainly rely on pre-trained language models, which fall into two categories: small language models (SLMs) such as BART [17] and T5 [32], and large language models (LLMs) such as GPT4 and PaLM [7]. We have conducted an in-depth analysis to gain insights into strengths and limitations of the SOTA

(a) A Text Question Q

Which course has the highest score for the student named timothy ward?

(b) Snippets of a Database D

Course

id	course	teacher
001	math	jordy wu
…	…	…

Student

id	given_name	last_name	score	course
1	timmy	ward	92	math
…	…	…	…	…

(c) The Ground-truth SQL Query S *w.r.t.* Q

```
SELECT course FROM Student
WHERE given_name = 'timmy' AND last_name = 'ward'
ORDER BY score LIMIT 1;
```

(d) An SQL query S' translated by an SLM

```
SELECT course FROM Student
WHERE given_name = 'timothy ward'
ORDER BY score LIMIT 1;
```

(e) An SQL query S'' translated by an LLM

```
SELECT Course.course, Student.score
FROM Student JOIN Course ON Student.id = Course.id
WHERE given_name = 'timothy' AND last_name = 'ward'
ORDER BY score LIMIT 1;
```

Figure 1: A sample NL2SQL translation. The incorrect portions are highlighted in red.

solutions for zero-shot NL2SQL, and report the error distributions of both categories in Figure 2.

SLM-based methods, such as RASAT [30], PICARD [37], and RESDSQL [18], have shown promise in generating accurate SQL queries on NL2SQL datasets through fine-tuning on numerous annotated NL2SQL samples. However, SLM-based methods may have *limited generalizability* in natural language reasoning in our zero-shot settings, which may dramatically degrade the performance of the methods [28]. Consider S' in Figure 1(d), given "student named timothy ward" in question Q, an SLM-based method only selects one column `given_name` that is similar to the word "named". This is because the annotated NL2SQL samples used to train the method do not differentiate between `given_name` and `last_name`. Although further fine-tuning on new databases or linguistic phenomena can alleviate this problem, it requires a significant amount of high-quality training data, such as annotated NL2SQL samples. Acquiring such data can be both time-consuming and labor-intensive, making it a challenging task.

LLMs like PaLM [7] and GPT4 [25], which are often accessed through API calls, have demonstrated remarkable reasoning abilities across a range of domains and tasks. Compared with SLMs, LLMs are capable of (complicated) language reasoning, including understanding question semantics, resolving ambiguities, and performing logical deductions, which are necessary for generating SQL queries in new environments. However, LLMs may not achieve precise schema alignment: as shown in Figure 2, 42% of the errors are caused by incorrect table/column selection. In particular, LLMs tend to choose more columns and tables to cover the input content, leading to incorrect execution results. Consider S'' in Figure 1(e): the LLM identifies incorrect columns (`score`) and tables (`course`) in `SELECT` and `FROM` clauses.

Our Proposal. We propose a decomposition-based approach that breaks down the NL2SQL task into smaller sub-tasks, such that each sub-task is more amenable to solve in our zero-shot setting. Naturally, the thought process of writing an SQL query can be broken down into four sub-tasks:

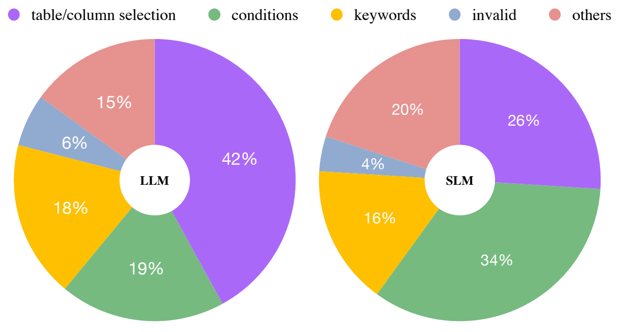
(1) Identifying query structure that consists of SQL reserved keywords, *e.g.*,`SELECT`, `FROM`, `WHERE`, and `ORDER BY`;
(2) Aligning relevant schema elements with the question, *i.e.*, columns and tables in `SELECT` and `FROM` clauses;
(3) Completing the SQL query by deducing conditions in `WHERE` clause, columns in `ORDER BY` or `GROUP BY` clauses, etc.
(4) Iteratively correcting the SQL query if there are syntax or execution errors.

By analyzing the behavior of SLMs and LLMs over many different data sets, our key observation is that, contrary to the intuitive observation that LLMs should outperform SLMs, we find that the two together complement each other and perform better than either alone when performing the above four steps. Specifically, a task-specific fine-tuned SLM can better understand the database schema and the SQL syntax, which enables it to excel in structure identification and schema alignment. In contrast, existing research reveals that LLMs often face an "hallucination" issue, generating text unrelated to the given instructions, due to their lack of domain knowledge [21]. However, a general LLM possesses strong language reasoning capabilities [2], making it well-suited for SQL completion that require complex condition reasoning. Moreover, LLM also exhibits excellent interaction capabilities, allowing it to perform error correction efficiently through appropriate feedback.

Based on the insight, we propose ZeroNL2SQL, a framework that uses SLMs and LLMs to solve different steps in our decomposition-based approach, combining the best of the two worlds to address the generalization challenge of zero-shot NL2SQL. ZeroNL2SQL consists of two key steps, as illustrated in Figure 3.

- **Step 1: SQL sketch generation** utilizes SLMs for SQL structure identification and schema alignment to generate an *SQL sketch*, with attributes to SELECT, tables included in FROM, and necessary keywords (*e.g.*,`ORDER BY`) for composing the SQL query.
- **Step 2: SQL query completion and correction** leverages LLMs to complete the missing information in the SQL sketch and generate complete SQL queries, *e.g.*, aligning with data values from the database. For example, although the question refers to "timothy", the database actually uses the abbreviation "timmy". Thus, "timmy" should be used in the SQL query.

Challenges and Solutions. The first challenge is how to develop an SQL sketch generation method that can *generalize* to new databases or linguistic phenomena. We introduce an encoder-decoder based SLM model that generates an SQL sketch. Specifically, to improve the generalizability of the model, we design a novel database serialization strategy to make the encoder more adaptive to new databases. Moreover, we propose a *question-aware aligner* to obtain the most relevant SQL sketches by inferring the semantics



(a) The error distribution of LLMs (b) The error distribution of SLMs

Figure 2: Stats of SLMs and LLMs on different error types.

`Step 1:` SQL sketch generation

```
SELECT course FROM Student
WHERE  some conditions
ORDER BY  some attributes  LIMIT  some number ;
```

$\Downarrow$ (an SQL sketch)

`Step 2:` SQL query completion and correction

```
SELECT course FROM Student
WHERE given_name = 'timothy' AND last_name = 'ward'
ORDER BY score LIMIT 1;
```

Figure 3: An illustration of ZeroNL2SQL.

of the question with new linguistic phenomena (*e.g.,* different questioning styles [4]: "Which airport has the most flights? List its code." v.s. "What is the airport code with the largest amount of flights?").

The second challenge is how to effectively guide an LLM (*e.g.,* GPT4) to complete SQL queries that are *aligned* with data values stored in the databases. For example, in Figure 1, the NL2SQL model needs to generate an SQL condition `given_name = 'timmy'` (rather than 'timothy'). Given to the length constraint of LLM's input, it is infeasible to include all values in the input context, which makes this *value alignment* problem challenging. To this end, we design a multi-level matching strategy to recommend the most relevant values to the LLM, and propose an execution-based selection strategy to select the best SQL query without syntax or execution errors.

Contributions. We summarize our contributions as follows.

(1) ***The ZeroNL2SQL framework.*** We study the problem of zero-shot NL2SQL, and introduce a novel framework, ZeroNL2SQL, that first generates SQL sketches using SLMs with optimized schema alignment methods, then completes SQL queries using LLMs with value alignment, to solve the generalization challenge of zero-shot NL2SQL in a step-by-step fashion (Section 3)

(2) ***Optimizations.*** We propose novel techniques to tackle the challenges of the following problems: SQL sketch generation (Section 4), SQL query completion and correction (Section 5).

(3) ***New SOTA Zero-shot NL2SQL Result.*** We conduct a comprehensive evaluation of zero-shot NL2SQL on two benchmarks: Dr.Spider [4] and KaggleDBQA [16], on a total of 18 test sets (Section 6). Our experimental results show that ZeroNL2SQL can improve the average execution accuracy of ChatGPT (API version: gpt-3.5-turbo-0613) on the NL2SQL tasks by 10% to 20%, and its performance surpasses the state-of-the-art SLM-based models and fine-tuned LLMs (Exp-1 & Exp-2), as well as in-context learning methods (Exp-3).

2 PRELIMINARIES

2.1 Zero-shot NL2SQL

Let Q be a natural language question, and D a relational database consisting of n tables $\{T_1, T_2, \ldots, T_n\}$, where c_{ij} denotes the j-th column of the i-th table T_i. The problem of **NL2SQL** is to generate an SQL query S, given the question Q and a provided database D.

Zero-shot NL2SQL [4, 16, 51] refers to the fact that the inference environment $\mathcal{D}_{\text{test}} = \{(D_i, Q_i, S_i)\}_{i=0}^{M}$ does not appear in the training set $\mathcal{D}_{\text{train}} = \{(D_j, Q_j, S_j)\}_{j=0}^{N}$, including three situations:

(1) Testing on new databases. The databases $\{D_i\}_{i=0}^{M}$ used for testing is different in terms of schema and instance from the databases $\{D_j\}_{j=0}^{N}$ used for training. For example, the training database contains column `name`, but the testing database contains the columns `given_name` and `last_name`. Another example is the training database is about science, while the testing database is about finance.

(2) Testing on new questions. The questions $\{Q_i\}_{i=0}^{M}$ used for testing are different in terms of linguistic phenomena from the questions $\{Q_j\}_{j=0}^{N}$ used for training. For example, questions in the training set explicitly mention column or table names in the database schema, while the words in the testing questions do not explicitly appear in the database schema or as synonyms (*e.g.,* the question in Figure 1 can also be expressed as *"What is timothy ward's best performing course?"*).

(3) Testing on new SQL. The SQL queries $\{S_i\}_{i=0}^{M}$ used for testing are different in terms of local semantics and complexity from the SQL queries $\{S_j\}_{j=0}^{N}$ used for training. For example, only a small number of training SQL queries contain nested clauses, while a large number of testing SQL queries contain nested clauses.

2.2 SQL Query and SQL Sketch

The **SQL queries** supported by ZeroNL2SQL belong to Data query language [1] (DQL) in the base grouping of SQL sub-languages, which includes the following components: `SELECT` with multiple columns and aggregations, `WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, `LIMIT`, `JOIN`, `INTERSECT`, `EXCEPT`, `UNION`, `NOT IN`, `OR`, `AND`, `EXISTS`, `LIKE` as well as nested queries.

An **SQL Sketch** consists of the following three components:

(1) **SELECT.** Attributes that need to be returned to the user, *e.g.,* `course` in Figure 3, as well as the aggregation function acting on the attributes (*e.g.,* AVG(score)).

(2) **FROM.** Tables used to obtain data, *e.g.,* `Student` in Figure 3.

(3) **KEYWORDs.** Keywords representing sub-clauses, *e.g.,* `SELECT`, `FROM`, `ORDER BY`, `LIMIT`, as shown in Figure 3.

2.3 Language Models

Language models (LMs), typically based on Transformers [43], aim to understand and generate human-like text. These models utilize sophisticated algorithms and vast amounts of training data to

[1]https://en.wikipedia.org/wiki/Data_query_language

learn the patterns, structures, and semantics of language. Technically, LMs are trained to model the generative likelihood of word sequences, thereby enabling them to estimate the probability of subsequent tokens based on the provided input context. They have proven to be invaluable in various applications, especially in natural language processing.

In this paper, we distinguish between two specific terms: small language models (SLMs) and large language models (LLMs).

- **SLMs** refer to the models that can be hosted locally by normal users and fine-tuned for different downstream tasks, such as BERT [9], BART [17], and GPT2 [31].
- **LLMs** refer to the models that achieve superior performance on a variety of downstream tasks by increasing model capacity, such as ChatGPT, PaLM [7], GPT4 [25] and LLaMA [40].

Although both SLMs and LLMs are pre-trained on a large amount of text data, the former mainly adapt to downstream tasks through fine-tuning, while the latter achieves zero-shot complex reasoning through *in-context learning* and *instruction following* [50], without changing the model parameters.

2.4 Language Models for NL2SQL

Recent work [22, 33] formulates the NL2SQL task as an end-to-end translation task, and leverages prompting P to guide LMs:

$$\mathsf{LM}(Q, D, P) \rightarrow S$$

The design of the prompt P is the key to the quality of generating SQL queries. Existing methods include using only task description as the prompt [22], or adding some manually annotated NL2SQL samples ($Question, SQL$) to the prompt [52]. Our approach provides fine-grained prompts for LLMs through SQL sketches (generated by SLMs) and value recommendation, aiming to support the accurate generation of SQL queries.

3 AN OVERVIEW OF ZERONL2SQL

In this paper, we introduce ZeroNL2SQL, which uses LLMs to produce precise SQL queries by providing them with customized SQL sketches and high-quality value recommendations. As illustrated in Figure 4, ZeroNL2SQL consists of the following two key steps.

1. **SQL sketch generation.** Given a user question Q and a database schema D_{schema}, an *SQL Sketch Generation* module generates a list of SQL sketch candidates $\mathcal{D}_{\text{sketch}}$ = (SELECT, FROM, Keywords) and passes it to an LLM.
2. **SQL query completion and correction.** Given a set of SQL sketch candidates $\mathcal{D}_{\text{sketch}}$, an *SQL Query Completion and Correction* module guides the LLM to complete SQL sketches in the details, then we recommend the relevant values in the database to calibrate the incorrect query (*e.g.,* from 'timothy' to 'timmy'). Finally, the best SQL query is automatically selected as final output.

To support the above framework, we develop two key modules, *SQL Sketch Generation* and *SQL Query Completion and Correction*, to guide the LLMs with fine-grained prompts.

SQL Sketch Generation. Given a user question Q and a database schema D_{schema}, this module generates a ranked list of SQL sketch candidates $\mathcal{D}_{\text{sketch}}$. A straightforward implementation of *SQL sketch generation* is to model it as a sequence-to-sequence generation problem and use an Encoder-Decoder based SLM to generate the SQL sketch, *i.e.,*

$$t = SLM_{\text{decode}}(SLM_{\text{encode}}(Q, D_{\text{schema}}))$$

where $SLM_{\text{encode}}(\cdot)$ is the SLM's encoder for converting the input (Q, D_{schema}) into a high-dimensional hidden vector x, and $SLM_{\text{decode}}(\cdot)$ is the SLM's decoder for generating an SQL sketch t based on x via beam search.

However, it is non-trivial to train such an *SQL Sketch generation* module to provide accurate SQL sketch in our zero-shot setting where test data differs from training data. To address this, we first use an SQL sketch learning framework equipped with a database-aware serialization strategy to guide the model to generate a list of valid SQL sketches in different test environments. We then introduce a question-aware aligner to further rank the SQL sketches based on the fine-grained semantics of user questions. More details of *SQL Sketch Generation* are given in Section 4.

SQL Query Completion and Correction. Given a ranked list of SQL sketches, this module leverages LLMs to complete these SQL sketches and selects the optimal SQL query as the final output. There are two main challenges in this module: (1) achieving value alignment, and (2) selecting the best SQL query.

To address the first challenge, we design a multi-level matching strategy to provide the LLM with appropriate value recommendations that are both in line with the original SQL query and grounded on the database. Specifically, this module takes the condition predicted by the LLM as input, denoted as $(column^0, value^0)$. We start from $column^0$ and gradually expand the matching range of $value^0$ to the entire database to obtain the best tuple $(column, value)$, which is then fed back to the LLM for rewriting the SQL query. In addition, we explore different similarity calculation methods based on the type of value, such as abbreviations or synonyms.

To address the second challenge, based on the observation that the results of SQL execution can reflect the quality of SQL, we design an execution-based selection strategy to choose the best SQL query that is executable and returns high-quality results. More details of the SQL query completion and correction module are given in Section 5.

4 SQL SKETCH GENERATION

Next, we present *SQL Sketch Generation* that generates a ranked list of SQL sketches, as illustrated in Figure 5. The main challenge is to achieve good generalization across new databases and new linguistic phenomena. To this purpose, we first introduce an SQL sketch learning framework equipped with a database-aware serialization strategy in Section 4.1, and then develop question-aware alignment in Section 4.2 to further rank the SQL sketches based on the semantics of the natural language question.

4.1 SQL Sketch Learning

Given a user question Q and a database schema D_{schema}, we need to generate three parts: SELECT, FROM, Keywords in the SQL sketch, which are called *sub-tasks* for ease of presentation. We formulate this problem as a sequence-to-sequence generation problem and adopt an Encoder-Decoder small language model (SLM) as the backbone. We enable the Encoder-Decoder SLM to learn to generate

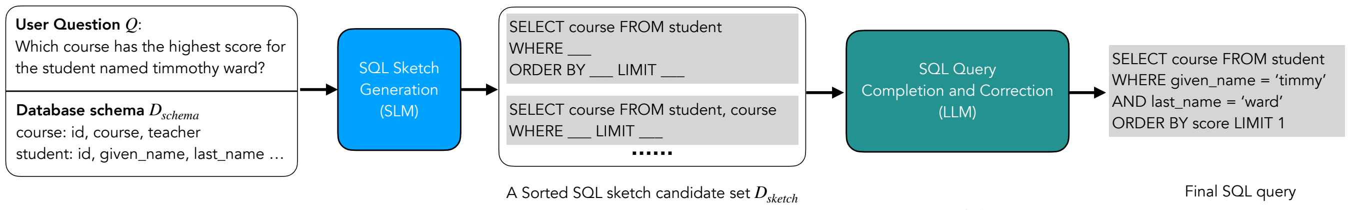

User Question Q:
Which course has the highest score for the student named timmothy ward?
Database schema D_{schema}
course: id, course, teacher
student: id, given_name, last_name …
SQL Sketch Generation (SLM)
SELECT course FROM student WHERE ___ ORDER BY ___ LIMIT ___
SELECT course FROM student, course WHERE ___ LIMIT ___
A Sorted SQL sketch candidate set D_{sketch}
SQL Query Completion and Correction (LLM)
SELECT course FROM student WHERE given_name = 'timmy' AND last_name = 'ward' ORDER BY score LIMIT 1
Final SQL query

Figure 4: ZeroNL2SQL Overview. Given a user question Q and a database schema D_{schema}, (1) an *SQL Sketch Generation* Module generates a list of SQL sketch candidates, and (2) an *SQL Query Completion* Module completes the SQL query.

these parts through multi-task learning, which is shown in Figure 5. Next, we introduce the two key steps of SQL Sketch Learning component: database-aware serialization and parameter learning.

Database-aware serialization. Given a user question Q and the database schema D_{schema}, we combine them with different instructions to construct specific inputs for each sub-task as:

$$[INS]\ \texttt{question}:[Q]\ \texttt{database}:\texttt{S}([D_{\text{schema}}])$$

where INS is the instruction for each sub-task, and $\texttt{S}(\cdot)$ is a serialization function for serializing the structured database schema D_{schema} into a text sequence.

Figure 5 shows the instructions corresponding to each sub-task, following the previous works [1, 36, 45], which are mainly composed of the task descriptions. For example, for FROM generation sub-task, the corresponding instruction is "Generate the relevant tables of this question according to the database". The main intuition is that different instructions can enable the SLM to understand different sub-tasks in order to achieve the desired output.

For database schema serialization, previous SLM-based works [18, 30, 37, 38] directly concatenate the table/column names and require the model to output these names to form an SQL query. However, SLM is obliged to generate valid table/column names that exist in the database. Previous methods [18, 30, 37, 38] cannot guarantee this when the databases or linguistic phenomena changes. Example 1 and Figure 6-(1) provides a detailed explanation.

Example 1 (Direct table/column name serialization). *Most existing works use the Spider dataset [48] as the training set, which has high column mention percentage in user questions [8]. We have an observation that the SLM often directly copies column/table names from the question Q during training, rather than selecting from the database. We design an experiment to explore the impact of data distribution shift on the SLM. We train the T5-3B model [32] on the Spider dataset by directly generating column/table names. Figure 6-(1) shows the test results of the fine-tuned T5-3B model, where column "attendance" does not exist in the database.*

To address this, we propose a simple yet effective database-aware serialization strategy to enable the SLM to choose the valid database tables or columns. Specifically, we achieve this by training the SLM to refer to the column/table in the database by their *indexes*. Specifically, given D_{schema} named D_{name} that contains n tables $\{T_0, T_1, \ldots, T_n\}$, and c_{ij} is the j-th column of the i-th table T_i, we use parentheses and indexes to the different parts to serialize the database, as shown below:

$$\begin{aligned}\texttt{S}([D_{\text{schema}}]) = [D_{\text{name}}]\ &\texttt{t0}:T_0(\texttt{c0}:c_{00}, \texttt{c1}:c_{01}, \texttt{c2}:\ldots)\\ &\texttt{t1}:T_1(\texttt{c0}:c_{10}, \texttt{c1}:c_{11}, \texttt{c2}:\ldots)\\ &\ldots\end{aligned}$$

For example, in Figure 5, the serialized representation of the database "car_1" is "car_1: t0: model_list (c0: modelid, c1: maker, c2: model) t1: continents (c0: contid, c1: continent) t2: car_names (…)". In addition, for tables containing foreign key relationships (*e.g.,* in Figure 5, column "id" of table "cars_data" has a foreign key "makeid" in table "car_names"), we append it in the form of "t4.c0 = t2.c0" after the serialized table "cars_data".

In this way, we enforce the SLM to choose the table/column index that best matches the user question rather than directly copying it from the question. Finally, the index is automatically translated back to the original column/table names. Example 2 and Figure 6-(2) illustrate our database-aware serialization strategy.

Example 2 (Database-aware serialization). *Continuing with Example 1, we use Spider as the training set. The difference is that we require the model to learn how to use indexes to refer to corresponding tables or columns. Figure 6-(2) illustrates the test result, the model first outputs* `SELECT t0.c2`*, which is then automatically translated into column and table names in the database:* `SELECT stadium.highest`.

Based on the comparison results, we can see that Database-aware serialization strategy can enable the model to select valid table and column names from the database, rather than directly copying words from the question.

As discussed previously, to alleviate the local optimal problem of beam search, we retain top-k hypotheses generated by the SLM as candidates instead of only considering the best one. Thus, for the three parts of an SQL sketch, our SQL sketch learning method produces $\mathcal{D}_{\text{SELECT}} = \{\text{SELECT}_i\}_{i=0}^{K_1}$, $\mathcal{D}_{\text{FROM}} = \{\text{FROM}_i\}_{i=0}^{K_2}$, and $\mathcal{D}_{\text{Keywords}} = \{\text{Keywords}_i\}_{i=0}^{K_3}$.

Parameter Learning. To train such a model to generate SQL sketches, we perform supervised fine-tuning on an Encoder-decoder SLM (*e.g.,* T5 [32] and BART [17]) on a set of annotated data. Specifically, we first extract training data for three sub-tasks from an annotated NL2SQL dataset (*e.g.,* Spider [48]). The obtained dataset $\mathcal{M} = \{(INS, Q, DB_{\text{schema}}, L)\}$ consists of instruction INS, database schema D_{schema} and user question Q for input, and label L for target. Note that the instructions and labels for different sub-tasks are different. The SLM is fine-tuned on three sub-tasks simultaneously and optimize parameters θ by minimizing the maximum likelihood:

$$\mathcal{L}(\theta) = -\mathbb{E}_{(INS,Q,D_{\text{schema}},L)\in\mathcal{M}} \log P_\theta(L|INS, Q, D_{\text{schema}})$$

4.2 Question-Aware Aligner

For SQL sketch learning, the SLM generates SQL sketches based on the database schema. However, it is still lack of fine-grained

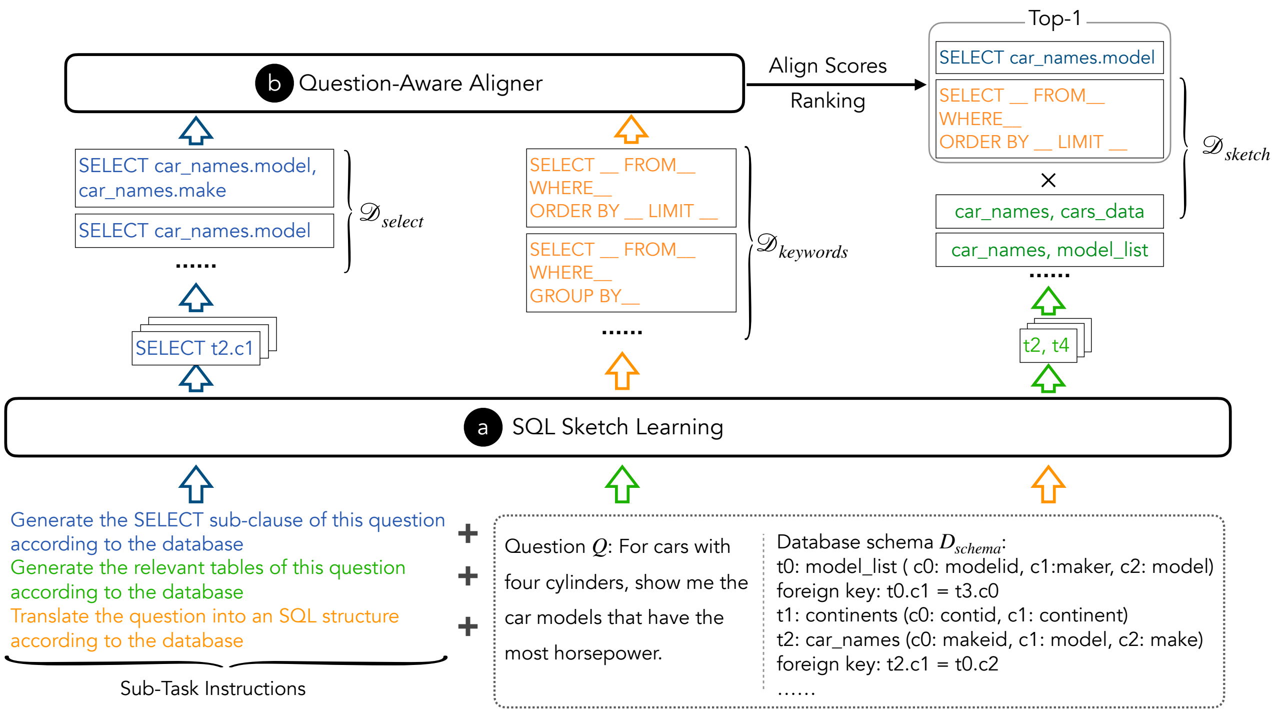
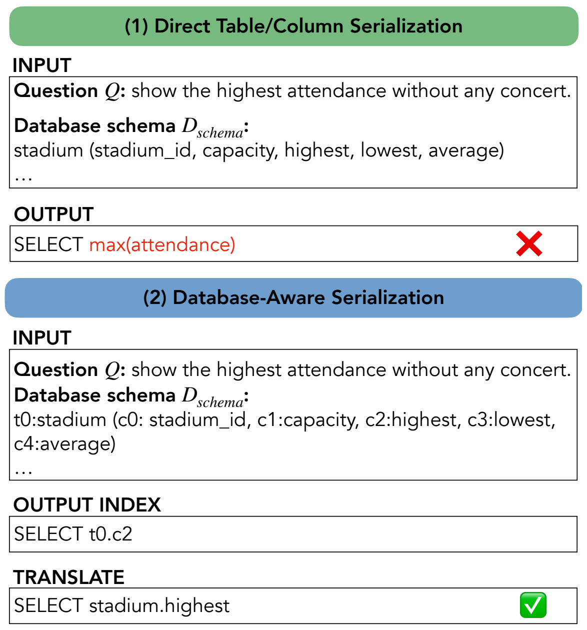

Top-1
b Question-Aware Aligner
Align Scores
Ranking
SELECT car_names.model
SELECT __ FROM__ WHERE__ ORDER BY __ LIMIT __
$\mathcal{D}_{sketch}$
×
car_names, cars_data
car_names, model_list
SELECT car_names.model, car_names.make
SELECT car_names.model
$\mathcal{D}_{select}$
SELECT __ FROM__ WHERE__ ORDER BY __ LIMIT __
SELECT __ FROM__ WHERE__ GROUP BY__
$\mathcal{D}_{keywords}$
SELECT t2.c1
t2, t4
a SQL Sketch Learning
Generate the SELECT sub-clause of this question according to the database
Generate the relevant tables of this question according to the database
Translate the question into an SQL structure according to the database
Sub-Task Instructions
Question Q: For cars with four cylinders, show me the car models that have the most horsepower.
Database schema D_{schema}:
t0: model_list (c0: modelid, c1:maker, c2: model)
foreign key: t0.c1 = t3.c0
t1: continents (c0: contid, c1: continent)
t2: car_names (c0: makeid, c1: model, c2: make)
foreign key: t2.c1 = t0.c2
……

Figure 5: An overview of the *SQL Sketch Generation* module. (a) A SLM is used to generate a list of candidates $\mathcal{D}_{\text{SELECT}}$, $\mathcal{D}_{\text{FROM}}$, and $\mathcal{D}_{\text{Keywords}}$ after SQL sketch learning (Section 4.1). (b) A Question-Aware Aligner is used to further rank the SELECT **and** Keywords **candidates based on fine-grained question semantics (Section 4.2). The top-1** SELECT **and** Keywords **are combined with $\mathcal{D}_{\text{from}}$ to form the ordered SQL sketch candidate set $\mathcal{D}_{\text{sketch}}$.**


(1) Direct Table/Column Serialization

INPUT

Question Q: show the highest attendance without any concert.

Database schema D_{schema}:
stadium (stadium_id, capacity, highest, lowest, average)
…

OUTPUT

SELECT max(attendance) ❌

(2) Database-Aware Serialization

INPUT

Question Q: show the highest attendance without any concert.
Database schema D_{schema}:
t0:stadium (c0: stadium_id, c1:capacity, c2:highest, c3:lowest, c4:average)
…

OUTPUT INDEX

SELECT t0.c2

TRANSLATE

SELECT stadium.highest ✅

Figure 6: Comparison of direct table/column name serialization and database-aware serialization.

optimization at the semantic-level for different parts. Specifically, for SELECT and Keywords parts, they need to be closely aligned with the question's intention and the question's requirements, respectively. For example, in Figure 4, as the question's intention is "which course", the corresponding SELECT part should be "SELECT course". In addition, as the question's requirements are "has the highest score", the corresponding Keywords part should include "ORDER BY".

To bridge this gap, we design a question-aware aligner to further prune SQL sketch candidates by selecting the best SELECT and Keywords. Note that we do not perform question-based filtering on FROM part here, because the FROM part also depends on complex foreign key connections and cannot be inferred directly based on the semantics of the question. Specifically, we take the Cartesian product of set $\mathcal{D}_{\text{SELECT}}$ and set $\mathcal{D}_{\text{Keywords}}$ to obtain the final candidate set $\mathcal{D} = \{(\text{SELECT}_i, \text{Keywords}_j) | \text{SELECT}_i \in \mathcal{D}_{\text{SELECT}}, \text{Keywords}_j \in \mathcal{D}_{\text{Keywords}}\}$. Then, for each candidate from $\mathcal{D}$, we concatenate it with the user question Q to form an input sequence, *i.e.,*

$$[\texttt{CLS}]\ \texttt{user question}: Q.\ \texttt{our solution}: \text{SELECT}_i, \text{Keywords}_j [\text{SEP}]$$

where [CLS] and [SEP] are special tokens that represent the start and end of the input sentence, respectively. The input sequence is converted to a high-dimensional representation $H_{[\texttt{CLS}]}$ through an Encoder-based SLM (*e.g.,* BERT [9], RoBERTa [24], and DeBERTa [14]). $H_{[\texttt{CLS}]}$ is further fed into a fully connected layer $\texttt{FC}(\cdot)$ to obtain the alignment score $a_{ij} = \sigma(FC(H_{[\texttt{CLS}]}))$ between $(\text{SELECT}_i, \text{Keywords}_j)$ and Q, where σ is the softmax function.

Algorithm 1: Multi-Level Matching

```
Input: database $D$, SQL $S'$, similarity threshold $r$
Output: Rewritten SQL $S''$
1  $feedback = []$;
2  $levels = [column, table, database]$;
3  for $(column^0, value^0)$ in extractCondition($D, S$) do
4      for $level$ in $levels$ do
5          getCandidateValues($level, D$) $\rightarrow \mathcal{V}$;
6          SimilarityCalculation($\mathcal{V}, value^0$) $\rightarrow$
             $(column', value')$;
7          if closeEnough($value^0, value', r$) then
8              break;
9          end
10     end
11     $feedback$.append($column', value'$);
12 end
13 $S'' \leftarrow$ feedbackToLLM($feedback$);
```

Finally, we combine $\mathcal{D}_{\text{FROM}}$ with the best (SELECT, Keywords) selected by the question-aware aligner to form the ranked SQL sketch candidate set $\mathcal{D}_{\text{sketch}}$.

Parameter Learning. We perform supervised fine-tuning on an Encoder-based SLM to train our question-aware aligner. For each question Q in the training set $\mathcal{M}$, we obtain the ordered candidate set $\mathcal{D}$ as above. For each sample $(\text{SELECT}_i, \text{Keywords}_j) \in \mathcal{D}$, if and only if both SELECT_i and Keywords_j are correct, the alignment label L_a is 1; otherwise the alignment label L_a is set to 0. In this way, we can convert D into a training set $A = \{(Q, \text{SELECT}, \text{Keywords}, L_a)\}$. Then, we optimize the parameters θ_{aligner} (including the parameters of the Encoder-based SLM and the fully connected layer) by minimizing the cross entropy loss on the training set A.

5 SQL QUERY COMPLETION & CORRECTION

In this section, we present our SQL query completion method that leverages LLM to fill the missing information in the SQL sketch. Our main intuition is to mimic human behaviors to iteratively correct SQL queries, including revising queries based on the execution results and the database content. To achieve this, we first introduce a multi-level matching strategy to help LLM complete SQL queries with correct value alignment (Section 5.1). Then, we design a selection strategy to obtain optimal SQL queries (Section 5.2).

5.1 Multi-Level Matching

ZeroNL2SQL aims to ensure consistency between the SQL query and the database, *i.e.,* the *value alignment* presented in the Introduction. To achieve this, we introduce a *multi-level matching* strategy to provide appropriate value recommendations to the LLM. In this process, we explore different similarity calculation methods.

Given the condition $(column^0, value^0)$ predicted by the LLM, there are two types of errors: incorrect predicted value and incorrect predicted column. For the former, we can directly find the value that is most similar to $value^0$ in $column^0$. On the other hand, for the latter, we need to expand the matching range to the entire database. However, directly setting the matching range as the entire database may introduce completely unrelated columns. Therefore, we consider gradually expanding the matching range in three levels. As illustrated in Algorithm 1, we match the value on the column, table, and database in sequential order (Line 2). In any level of matching range, if we find a value that is close enough to $value^0$ (Line 7), we terminate the matching process and directly return the value and its corresponding column as a recommendation to LLM. Note that we only consider conditions where $(column^0)$ equals a string value, as correcting conditions involving greater than or less than a numerical value is challenging due to the wide range of potential incorrect types. We will consider this in the future work.

We also find that, due to the imperfect accuracy of SQL sketches, this step can actually compensate for some errors introduced by SQL sketch. For instance, an SQL sketch might predict the absence of a certain table, but multi-level matching can provide feedback to LLM by searching for target values and columns in the database, enabling LLM to add the missing table based on this feedback.

Similarity Calculation. Given $(value^0, value')$, the typical methods for similarity calculation can be generally divided into two categories, namely character-based and semantic-based.

For character-based similarity calculation, the similarity score can be calculated by fuzzymatch [15], *i.e.,*

$$1 - \frac{\text{IndelDistance}(value^0, value')}{\min[\text{length}(value^0), \text{length}(value')]}$$

where IndelDistance() is the minimum number of insertions and deletions to convert one sequence into another sequence, and length() represents the length of a string.

For semantic-based similarity calculation, we first convert the two values into two high-dimensional vector-based representations. Then, we obtain the similarity score by calculating the inner product of these two representations. Specifically, to obtain the vector-based representation, we consider two representative methods, namely word2vec-based and SLM-based.

- For word2vec-based method, we utilize the pre-trained GloVe dictionary [27]. We first tokenize each value into several tokens $\{tok_i\}_{i=0}^n$, and look up in the dictionary to obtain the word embedding emb_i for each token tok_i. Then the value is represented by averaging the word embeddings $\{emb_i\}_{i=0}^n$.
- For SLM-based method, we utilize Sentence Bert (SBERT) [34] to encode each value. Unlike word2vec-based method, SBERT uses a bidirectional attention mechanism to model the overall semantic interaction to obtain the encoded embeddings $\{emb_i'\}_{i=0}^n$, and averages them to obtain the final representation of the value.

5.2 SQL Query Selection

Based on each SQL sketch candidate obtained in Section 4, LLM completes the SQL query to select the optimal query as the final output. We trigger query revision if a query returns null results, inspired by the observation that humans usually think twice when encountering such queries. To achieve this, we design a selection strategy based on execution results. That is, if the current SQL query does not meet the requirements, we will use a new sketch candidate to guide the LLM to generate a new SQL query. Our primary insight is that the execution result of an SQL query indicates the quality of the query itself. For instance, when a query yields null results, humans typically reconsider the query.

Algorithm 2: SQL Query Selection

```
Input: Question $Q$, Database schema $D_{\text{schema}}$, Database $D$,
    ordered candidate set $\mathcal{D}_{\text{sketch}}$, Execution patience $p$
Output: The final SQL query $S''$
for each $t_i \in \mathcal{D}_{\text{sketch}}$ do
    LLM($Q, D_{\text{schema}}, t_i$) $\rightarrow S^0$;
    ExecutionCheck($S^0, D, p$) $\rightarrow S'$;
    if $S'$ then
        MultiLevelMatching($S', D$) $\rightarrow S''$;
        GetExecutionResult($S'', D$) $\rightarrow result$;
        if $result \neq NULL$ then
            break;
        else
            continue;
        end
    else
        continue;
    end
end
return $S''$;
```

Algorithm 2 illustrates the overall process of our selection strategy. Given an SQL sketch candidate $t_i \in \mathcal{D}_{\text{sketch}}$, together with the question Q and the database schema D_{schema}, they are fed into the LLM to generate an SQL query S^0 (Note that S^0 is already a complete query, while subsequent execution checks and multi-level matching aim to refine it). Then S^0 is subjected to executable check, and an executable query S' is obtained by continuously feeding back error information to the LLM. Please note that we set the number of feedback times not to exceed p; otherwise we discard t_i and repeat the above steps with the next candidate t_{i+1}. S' is passed to the multi-level matching step to obtain an SQL query S'' grounded on the database. If the execution result of S'' is not $NULL$, we return S'' as the final SQL query. Otherwise, we discard t_i and repeat the above steps for the next candidate t_{i+1}.

6 EXPERIMENTS

6.1 Experimental Setup

6.1.1 Datasets. We train ZeroNL2SQL using the Spider dataset and then use Dr.Spider, GeoQuery and KaggleDBQA to evaluate the effectiveness and robustness of the model. The databases and user questions in the train and test sets are completely different.

Spider is a well-known NL2SQL benchmark [48], which contains 200 databases that cover 138 domains, such as colleges, government, etc. There are 10,181 question instances with 5,693 unique SQL queries. Among them, Spider randomly selects 7000 annotated instances as the training set.

Dr.Spider is a comprehensive benchmark [4] based on Spider, which is designed for evaluating the performance of NL2SQL methods in new test environments, *i.e.,* the zero-shot setting studied in our paper. The basic idea is to make perturbations on the Spider dataset to simulate new test sets. Specifically, there are 3 test sets with *database perturbation*, 9 test sets with *question perturbation*, and 5 test sets with *SQL perturbation*. Database perturbation simulates the situations where data is represented in various ways in databases. Question perturbation simulates various task-specific linguistic phenomenon. SQL perturbation simulates the changes of SQL structures. Please refer to the original paper [4] for details.

KaggleDBQA is a test set [16] for NL2SQL tasks that is designed to closely mimic the data and questions that an NL2SQL model might encounter in real-world scenarios. The databases used in KaggleDBQA are sourced from Kaggle (https://www.kaggle.com/). The questions in KaggleDBQA are annotated to ensure that they reflect the interests of real users.

GeoQuery is a benchmark for NL2SQL tasks, focusing solely on US geography. It comprises a single database and encompasses 877 instances, each consisting of a natural language question paired with an SQL query.

Table 1 displays the various SQL query types and corresponding examples contained within these datasets.

6.1.2 Evaluation Metrics. As SQL expression styles used in LLMs may differ from the ground truth in the NL2SQL benchmarks [39], traditional string-based evaluation metrics such as Exact Match Accuracy [48] are not appropriate for evaluation in our paper. Therefore, following previous works [22, 33, 52], we use the Execution Accuracy (EX) metric, which compares the execution results of a generated and the corresponding ground-truth SQL queries retrieved from the database.

6.1.3 Baselines. We consider the following two types of baselines. First, we use SOTA SLM-based models fine-tuned on the Spider training set, including SMBOP [35] and RESDSQL [18]. We also report the fine-tuning results of LLaMA2 [40] model for comparison. Second, we use LLM-based methods, including vanilla LLMs, LLM + In-Context Learning, and DIN-SQL [29]. We briefly describe these methods as follows.

SMBOP introduces a semi-autoregressive bottom-up NL2SQL model as an alternative approach to top-down autoregressive model.

RESDSQL proposes a ranking-enhanced encoding method that selects the most relevant schema items to inject into the encoder.

LLaMA2 is an open-source LLM (https://ai.meta.com/llama/) that we fine-tune directly on the Spider dataset and report the fine-tuning results of two different model sizes, 7B and 13B.

Vanilla LLM refers to directly applying an LLM to the NL2SQL task without any other task-specific designs. Specifically, we input the user question, the database schema, and the task instruction "Translate the user question into an SQL query." to the LLM.

LLM + In-Context-Learning refers to adding some (user question, SQL) examples to the input of the LLM to facilitate its understanding and reasoning [1]. Considering that different example selection methods may produce different results [23], we consider two typical ways to select examples: one is to randomly select from an example pool, and the other is to select examples that are similar to the user question. Besides, we also evaluate a method (named as "Incremental ICL") to approximate a scenario where we start with zero examples in the train set. Specifically, we pick a test set, and do zero-shot with an LLM. Then, if the generation is correct based on the ground truth, it can be added into the train set and used for in-context learning in subsequent LLM invocations.

Table 1: Types of SQL queries contained in the dataset. The type is determined by the quantity of SQL components, selections, and conditions, meaning that queries with a higher number of SQL keywords (such as GROUP BY, ORDER BY, and aggregators) are considered to be more challenging.

Type	Question	SQL query
Easy	What is the number of cars with more than 4 cylinders?	SELECT COUNT(*) FROM cars_data WHERE cylinders > 4
Medium	For each stadium, how many concerts are there?	SELECT T2.name, COUNT(*) FROM concert AS T1 JOIN stadium AS T2 ON T1.stadium_id = T2.stadium_id GROUP BY T1.stadium_id
Hard	Which countries in Europe have at least 3 car manufacturers?	SELECT T1.country_name FROM countries AS T1 JOIN continents AS T2 ON T1.continent = T2.cont_id JOIN car_makers AS T3 ON T1.country_id = T3.country WHERE T2.continent = 'Europe' GROUP BY T1.country_name HAVING COUNT(*) >= 3
Extra Hard	What is the average life expectancy in the countries where English is not the official language?	SELECT AVG(life_expectancy) FROM country WHERE name NOT IN (SELECT T1.name FROM country AS T1 JOIN country_language AS T2 ON T1.code = T2.country_code WHERE T2.language = 'English' AND T2.is_official = 'T')

DIN-SQL focuses on breaking down complex text-to-SQL tasks into sub-tasks to enhance the performance of LLMs in reasoning.

6.1.4 Implementation Details. In this work, we use OpenAI's APIs with versions gpt-3.5-turbo-0613 and gpt-4-0613, as the backbone LLM. Specifically, we set the generation temperature to 0.0, frequency penalty to 0.0, and top-p to 1.0. For SQL sketch learning, we adopt T5-3B (https://huggingface.co/t5-3b) as the backbone Encoder-Decoder SLM. We set the batch size to 32 and the learning rate to 0.00005. In question-aware aligner, we adopt DeBERTaV3-Large (https://huggingface.co/microsoft/deberta-v3-large) as the backbone Encoder SLM. We set the batch size to 4 and the learning rate to 0.000005. In addition, for the number of hypotheses retained for different parts, we set K_1, K_2, and K_3 to 4, 2, and 2 respectively, and set p to 1 for the execution check part. For SQL query completion, we use Sentence Bert (https://huggingface.co/sentence-transformers/bert-base-nli-stsb-mean-tokens) and pre-trained GloVe dictionary (https://nlp.stanford.edu/projects/glove/) containing 200-dimensional representations of 400K English words to obtain the embedding of the value. We set the similarity threshold r in Algorithm 1 to 0.65 which is determined by a hyper-parameter search in our evaluation. For database components, we use SQLite as the database and the sqlite3 package in Python to parse and execute SQL queries over the databases.

All the experiments are implemented using PyTorch [26], and evaluated on NVIDIA RTX A6000 48G.

6.2 Comparison with open-source LMs.

First, we compare ZeroNL2SQL and baseline models on different zero-shot test environments (**Exp-1** and **Exp-2**).

Exp-1: What is the zero-shot reasoning ability on Dr.Spider (*i.e.,* single perturbation per test set)? We conduct experiments on the Dr.Spider benchmark [4] containing 17 test sets, and Table 2 reports the experimental results. Overall, ZeroNL2SQL based on gpt-3.5-turbo-0613 outperforms all the baselines, including the SLM-based SOTA models (74.9% VS. 71.7%), fine-tuned LLaMA2 (74.9% VS. 61.6%) and the vanilla gpt-3.5-turbo-0613 (74.9% VS. 63.5%). Furthermore, when combined with a more powerful backbone LLM, GPT-4, our method can demonstrate even better performance (77.2% VS. 74.9%). Specifically, we have the following findings: (1) Comparing the previous SLM-based SOTA model and vanilla gpt-3.5-turbo-0613, we can see that there is a gap on the average performance between the two methods, which is also reported in the existing work [22]. However, gpt-3.5-turbo-0613 exhibits relatively stable performance in different zero-shot tests and does not show a particularly significant performance declines due to certain types of perturbations. For example, on the *DBcontent-equivalence* test set, the EX accuracy of SMBOP [38] is only 37.2%, which is much worse than its average performance. In contrast, the EX accuracy of gpt-3.5-turbo-0613 is 54.5%, which is close to its average performance. (2) Combining with ZeroNL2SQL, gpt-3.5-turbo-0613 shows better and more stable NL2SQL performance than the SLM-based SOTA models. For example, on the difficult *value-synonym* test set, the SLM-based SOTA models have an EX accuracy between 29.1% and 53.2%, while ZeroNL2SQL achieves an EX accuracy of 70.6%, which shows a significant improvement of 17.4%. (3) The LLaMA2 model fine-tuned directly on the NL2SQL dataset does not perform as well as task-customized SLMs (*i.e.,* SMBOP, RESDSQL), which is consistent with the findings of DAIL-SQL [10]. As for the reason, our main analysis is that training LLMs on a specific task can easily lead to overfitting on a single task (*i.e.,* NL2SQL), which may impact their generalization capabilities.

Conclusion: This experiment shows that our method unifies the advantages of SLMs and LLMs for supporting zero-shot NL2SQL.

Exp-2: What is the zero-shot reasoning ability on unseen domains (*i.e.,* the distribution of databases, user question and SQL all changes)? We conduct experiments on the KaggleDBQA [16] and GeoQuery [49] datasets, which has a completely different style of databases, user questions, and SQL queries compared to the Spider dataset [48] used for training. Table 3 reports the experimental results. Comparing with SLM-based SOTA model and vanilla gpt-4-0613, the EX accuracy of "ZeroNL2SQL + gpt-4-0613" achieves the best NL2SQL performance. This indicates that when using the same training set, ZeroNL2SQL can exhibit better zero-shot reasoning ability on completely different test sets.

Conclusion: This experiment demonstrates that LLMs are more generalizable to new domains. Furthermore, ZeroNL2SQL evokes the superior zero-shot reasoning capabilities off LLMs.

6.3 Comparison with proprietary LLMs.

Exp-3: How does our model compare with proprietary LLM-based methods in terms of performance and cost? A typical method of applying LLMs to a specific task is *in-context learning*.

Table 2: Comparison of the execution accuracy (%) between ZeroNL2SQL and the open-source language models fine-tuned on the Spider dataset. "DB" represents perturbations on the database, "NLQ" represents perturbations on user issues, and "SQL" represents perturbations on SQL queries. We report macro-average scores over multiple perturbations.

Type	Perturbation	# Test	SMBOP [35]	RESDSQL [18]	LLaMA-2-7B	LLaMA-2-13B	gpt-3.5-turbo-0613	ZeroNL2SQL (gpt-3.5-turbo-0613)	ZeroNL2SQL (gpt-4-0613)
	Spider-dev	1034	78.0	**84.1**	72.7	74.6	72.4	82.0	84.0
DB	schema-synonym	2619	53.9	68.3	45.1	47.9	58.7	69.8	**71.4**
	schema-abbreviation	2853	59.0	70.0	56.9	60.8	64.7	74.8	**75.9**
	DBcontent-equivalence	382	37.2	40.1	48.4	49.8	54.5	56.8	**72.0**
NLQ	keyword-synonym	953	64.3	72.4	61.2	64.3	57.1	74.0	**74.5**
	keyword-carrier	399	79.2	83.5	78.2	80.6	85.7	88.2	**89.0**
	column-synonym	563	48.7	63.1	43.5	45.1	51.2	62.7	**64.5**
	column-carrier	579	64.6	63.9	62.0	64.6	57.9	71.7	**73.2**
	column-attribute	119	58.0	71.4	58.8	60.8	58.0	70.6	**73.9**
	column-value	304	58.9	**76.6**	52.3	55.0	64.5	76.0	74.3
	value-synonym	506	29.1	53.2	41.1	44.2	54.9	**70.6**	68.9
	multitype	1351	46.1	60.7	49.7	50.2	52.2	66.4	**67.6**
	others	2819	73.7	79.0	68.8	71.2	68.6	79.4	**81.3**
SQL	comparison	178	65.2	**82.0**	61.2	63.9	61.2	73.6	79.2
	sort-order	192	76.6	**85.4**	69.8	72.5	52.1	80.2	83.3
	nonDB-number	131	71.8	85.5	71.8	74.4	88.5	92.4	**92.4**
	DB-text	911	63.1	74.3	57.5	61.8	72.3	80.7	**82.1**
	DB-number	410	84.4	**88.8**	77.3	80.2	78.0	86.1	86.3
All	Average	-	60.8	71.7	59.0	61.6	63.5	74.9	**77.2**

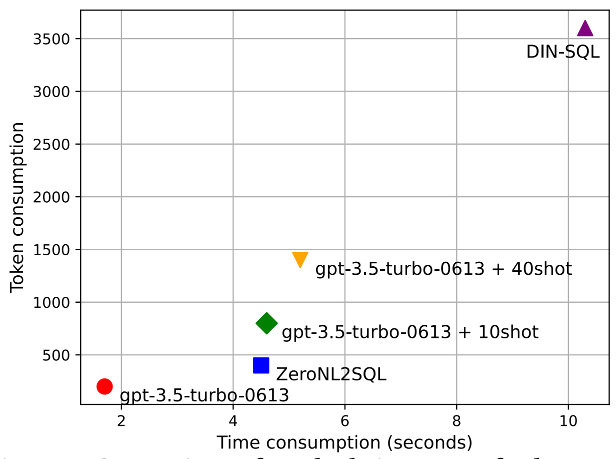


Figure 7: Comparison of methods in terms of token consumption and end-to-end time consumption (seconds). Both ZeroNL2SQL and DIN-SQL are based on gpt-3.5-turbo-0613.

Table 3: Comparison of the execution accuracy (%) between ZeroNL2SQL and the other methods on unseen domains.

Dataset	Previous SOTA	gpt-4-0613	ZeroNL2SQL (gpt-4-0613)
GeoQuery	74.7 [46]	65.4	**84.6**
KaggleDBQA	31.9 [18]	25.2	**42.4**

For In-context learning, we use the Spider training set as the example pool. Table 4 presents the experimental results. Comparing with in-context learning, our method can bring more significant improvements to LLM. For example, on the *DBcontent-equivalence* test set, in-context learning results in poor gpt-3.5-turbo-0613 performance (54.5% VS. 54.2%), while our method does not (54.5% VS. 56.8%). Furthermore, when combined with a more powerful backbone LLM, GPT-4, our method can demonstrate even better performance (67.7% VS. 63.9%). In addition, we compare OpenAI's text-embedding-ada-002 against SBERT in obtaining in-context learning examples. Each example is encoded into a high dimensional embedding using different methods (*i.e.,* text-embedding-ada-002 and SBERT), and then cosine similarity is used to select 20 annotated examples to be added to the prompt to assist LLM's inference. According to the experimental results in Table 5, the text-embedding-ada-002 performs slightly better than SBERT, as it can better encode the semantic information of sentences. However, considering that the encoding cost of SBERT is much lower than that of text-embedding-ada-002, we believe that using SBERT for retrieval is a reasonable choice.

We compare the methods on costs from two perspectives. As shown in Figure 7: (1) number of tokens consumed by our method is much lower than various in-context learning methods. (2) regarding the run time of end-to-end inference, ZeroNL2SQL is only second to directly using gpt-3.5-turbo-0613 for inference, and significantly better than other methods which are based on chain-of-thought reasoning and in-context learning. For detailed training and inference costs, training takes substantial time, but ZeroNL2SQL only needs to be trained once for inference in new domains, without the need for retraining. In addition, multi-level matching does not cause much delay, as we can cache the embeddings of values in the database, and cosine similarity calculations can fully utilize matrix multiplication, which can be further accelerated through cuda support.

Conclusion: This experiment demonstrates that ZeroNL2SQL + LLM outperforms direct usage of LLM alone in terms of both accuracy and efficiency in NL2SQL reasoning.

6.4 Ablation Study

Exp-4: What is the effect of each component in the SQL sketch generation module? Table 6 shows the impact of different parts on the string match accuracy of generating SQL sketch. Overall, database-aware serialization strategy has the greatest impact on generating SQL sketches, avoiding the model from over-focusing on user questions and enabling the model to generate valid SQL

Table 4: Comparison of the execution accuracy (%) between ZeroNL2SQL and different in-context learning methods based on proprietary LLMs. We report macro-average scores over datasets. † represents that the method does not use database content.

Dataset	gpt-3.5-turbo-0613	gpt-3.5-turbo-0613 + random 10-shot	gpt-3.5-turbo-0613 + random 20-shot	gpt-3.5-turbo-0613 + similarity 10-shot	gpt-3.5-turbo-0613 + similarity 20-shot	gpt-3.5-turbo-0613 + Incremental ICL	DIN-SQL † [29]	ZeroNL2SQL (gpt-3.5-turbo-0613)	ZeroNL2SQL (gpt-4-0613)
DBcontent-equivalence	54.5	51.3	54.2	51.3	53.9	64.9	48.7	56.8	**72.0**
column-value	64.5	64.5	67.4	64.5	66.1	70.5	51.6	**76.0**	74.3
DB-text	72.3	75.7	75.6	75.3	76.9	78.9	61.7	80.7	**82.1**
KaggleDBQA	24.3	26.5	29.7	27.6	28.1	34.6	27.0	**44.9**	42.4
Average	51.6	54.5	56.7	54.7	56.3	62.2	47.3	63.9	**67.7**

Table 5: Comparison between OpenAI's text-embedding-ada-002 and SBERT.

Approach	column-value	DB-text	KaggleDBQA
text-embedding-ada-002	67.2	76.9	28.9
SBERT	66.1	76.9	28.1

Table 6: Ablation on different components in the SQL sketch generation module (on the KaggleDBQA dataset).

Approach	select	keywords	from
T5-3B	54.2	65.7	76.8
ZeroNL2SQL (gpt-3.5-turbo-0613)	**61.1**	**68.6**	**81.6**
w/o database-aware serialization	55.1	66.5	76.8
w/o question-aware aligner	58.3	68.1	81.6
embedding-based aligner	54.2	63.1	75.2

sketches based on the database schema. Secondly, *question-aware aligner* has a gain on the SELECT part and the KEYWORDS part, indicating that further aligning these two parts with the user question can help SQL sketch generation. Additionally, we compare with an embedding-based aligner, which calculates the cosine similarity between the question and candidates. Experimental results indicate that "embedding-based aligner" performs weaker than our method used in ZeroNL2SQL. The main reason is that during the alignment step, semantically each candidate and question are very similar (with differences in only one or two tokens). Compared to "embedding-based aligner", our method can encode both the question and candidate, capturing more fine-grained alignment relationships through self-attention mechanisms. Table 7 shows two examples of what the aligner does specifically. That is, it reorders candidates according to the semantics of the natural language question. This assigns higher scores to the best matching candidates.

Exp-5: Evaluating the effect of each component in the SQL query completion and correction module? Firstly, to evaluate the effectiveness of our matching method, we conduct experiments in the following two aspects:

(1) We demonstrate the importance of similarity-based matching by comparing with other design choices, including providing random values from the predicted column to the LLM (denoted as "random match") and providing data formats, such as lowercase and abbreviations (denoted as "format match"). We report the experimental results in Table 9. We can see that both "random match" and "format match" perform worse than similarity-based match methods. This is because "random match" introduces too much noise, while "format match" is too limited. We also analyze the correlation between the predicted values and the actual values. We identify 300 error samples where the predicted values differed from the ground truth, categorized as "lexical errors" and "semantic errors". "Lexical errors" involve similar semantics but different spellings (*e.g.*,`True -> T`), amendable through similarity retrieval. "Semantic errors" involve distinct semantics (*e.g.*,`Alice -> Alan`), not amendable through similarity retrieval. Among the 300 errors, 268 were lexical errors and 32 were semantic errors. Hence, correcting LLM errors through similarity retrieval is reasonable.

(2) We evaluate the effectiveness of multi-level matching by comparing with the question-based method used in previous work [30, 37] (*i.e., baseline*), *in-column matching* and *in-database matching*. The latter two represent only matching in the predicted column and directly matching throughout the entire database, respectively. According to the results, *multi-level matching* achieves the best performance. The main reason is that *in-column matching* does not consider LLM's mis-prediction of columns, and thus matching in the wrong columns and feeding noisy information back to the LLM causes further errors. In addition, *in-database matching* completely disregards the predicted column, resulting in feedback to be significantly deviated from the original reasoning.

Secondly, to explore the best similarity calculation method for this scenario, we compare different representative methods in Table 10. Fuzzy matching method [15] can better capture character-level similarity, and thus it performs well in abbreviation/full name matching, *e.g.,* "HI" and "Hawaii". The Glove method [27], as a representative semantic level matching method, can better handle synonym situations, such as "putty" and "dog". However, GloVe only considers the co-occurrence of words in a text corpus, while may not capture more complex semantic relationships between words (*e.g.,* order). Moreover, due to the use of a bidirectional attention mechanism, SBERT [34] can better capture the overall meaning of the value. Therefore, SBERT exhibits more stable and excellent performance than both Glove and fuzzy matching methods.

Exp-6: Evaluating the relevance and usefulness of combining SLMs with LLMs. To validate the design of SLMs and LLMs in ZeroNL2SQL, we compare four different design choices: (1) Replacing the SQL Sketch generation module with LLM (prompted by in-context learning), with the subsequent modules remaining unchanged, denoted as "LLM +LLM"; (2) Replacing the SQL query completion module with SLMs (since SLMs does not have interactive capabilities, we removed the correction part), with the previous modules remaining unchanged, denoted as "SLM +SLM"; (3) Replacing SQL Sketch generation module with LLM and the SQL query completion with SLM, denoted as "LLM +SLM"; (4) The original design of ZeroNL2SQL, which is denoted as "SLM +LLM". The experimental results are shown in Table 8, and it can be seen that the design of ZeroNL2SQL outperforms the other three design methods. This indicates that SLM is suitable for controllable generation of SQL sketches, while LLM is suitable for generating parts that require complex reasoning, such as the conditions in SQL queries.

7 RELATED WORK

SLM-based NL2SQL. Many studies have shown that pre-trained language models can perform well in structured data (*e.g.,* table,

Table 7: **Two alignment examples from the aligner. Note that the order of the candidates in the table is determined by the generation probability of the SQL sketch generator, and the role of the aligner is to reorder these candidates based on the semantics of natural language question, with the goal of assigning the maximum score to the correct candidate (✓).**

Question	SELECT Candidates	Structure Candidates
For cars with four cylinders, show me the car models that have the most horsepower.	1. SELECT car_names.model, car_names.make 2. SELECT car_names.model (✓)	1. SELECT-FROM-WHERE-ORDER BY-LIMIT (✓) 2. SELECT-FROM-WHERE-GROUP BY
Please find the name and ranking points of the player who has won the most times.	1. select players.first_name, players.last_name, rankings.ranking_points 2. select matches.winner_name, matches.winner_rank_points (✓)	1. SELECT-FROM-WHERE 2. SELECT-FROM-GROUP BY-ORDER BY-LIMIT (✓)

Table 8: Comparison of different design choices, which reports execution accuracy (%).

Approach	KaggleDBQA	column-value	DB-text
SLM + LLM (ZeroNL2SQL)	**44.9**	**76.0**	**80.7**
LLM + SLM	21.4	71.4	75.6
LLM + LLM	25.6	67.5	75.7
SLM + SLM	29.7	72.6	73.8

Table 9: Ablation study on different matching methods, which reports execution accuracy (%).

Approach	column-value	DB-text	Average
Similarity-based Methods			
baseline	69.8	68.8	69.3
in-column matching	71.7	81.0	76.4
in-database matching	62.8	64.7	63.8
multi-level matching	76.0	80.9	**78.5**
Other Methods			
random match	70.1	67.2	68.7
format match	72.4	73.6	73.0

Table 10: Ablation study on different similarity calculation methods, which reports execution accuracy (%).

Approach	column-value	DB-text	Average
Fuzzy [15]	73.4	81.0	77.2
GloVe [27]	74.6	78.0	76.3
SBERT [34]	76.0	80.9	**78.5**

database) inference [12, 13, 41, 42]. Therefore, many works have applied pre-trained language models to the NL2SQL task, as pre-training on large amounts of text corpus enables SLMs to better model the semantic relationship between user question and database schema [11]. Overall, The SLMs used in these works is mainly divided into two types: Encoder-only SLMs and Encoder-Decoder SLMs. For Encoder-only SLMs, RATSQL [44] and LGESQL [3] leverage BERT [9] to encode the user question and database schema, and further adopt graph neural network to model the foreign keys and schema links. Then the encoded representation is fed into a grammar-based syntactic neural decoder to generate a SQL query. For Encoder-Decoder SLMs, PICARD [37], RASAT [30] and RESDSQL [18] formulate the NL2SQL task as an end-to-end translation problem and leverage the T5 model [32] to directly translate user question into SQL query. Additionally, CodeS [19] trains on the Star-Coder using SQL data and achieves good NL2SQL performance. Different from previous methods, the goal of ZeroNL2SQL is to enable SLMs to generate accurate SQL sketches on new test environments. Therefore, we focus on the impact of test environment changes on SLMs and propose adaptive methods to address this challenge.

LLM-based NL2SQL. With the excellent performance of LLMs in many natural language processing tasks [6], recent work attempts to apply LLMs to the NL2SQL task. Rajkumar et al. [33] evaluate the zero-shot NL2SQL capabilities of CodeX model [5]. Zhuo et al. [52] further validate the robustness of the Codex model on NL2SQL task and proposes effective example sampling method to enhance robustness. Recently, with the popularity of ChatGPT, Liu et al. [22] conduct experiments to explore its zero-shot NL2SQL inference ability and point out that it still has a certain gap with existing fine-tuned PLM-based methods, but it exhibits strong robustness on new datasets. To improve the NL2SQL effectiveness of LLMs, DIN-SQL [29] enables LLMs to generate SQL queries step by step by adding examples of different sub-tasks. Different from the above methods, ZeroNL2SQL provides grained guidance (*i.e.,* SQL sketch and value recommendation) for an LLM, significantly improving NL2SQL accuracy while ensuring high efficiency.

8 CONCLUSION AND FUTURE WORK

In this paper, we have proposed a ZeroNL2SQL framework, which combines tunable SLMs and LLMs to achieve zero-shot NL2SQL generation. ZeroNL2SQL mainly consists of two modules: *SQL sketch generation* by SLMs and *SQL query completion and correction* by LLMs. Our extensive experiments demonstrate that ZeroNL2SQL can achieve the best zero-shot NL2SQL performance, compared with the SOTA SLM-based methods and LLM-based methods.

For future work, we believe that the ZeroNL2SQL framework can be extended to different NL2SQL scenarios. First, considering the excellent interaction ability of LLMs, ZeroNL2SQL can be applied to conversational NL2SQL tasks [47]. Second, considering the effective value alignment method in ZeroNL2SQL, it can also be extended to extra large databases [20].

ACKNOWLEDGMENTS

This work was partly supported by the NSF of China (62122090, 62072461, 61925205, 62232009, 62102215 and 62072458), National Key R&D Program of China (2023YFB4503600), the Fund for Building World-Class Universities (Disciplines) of Renmin University of China, the Beijing Natural Science Foundation (L222006), the Research Funds of Renmin University of China, and the Beijing National Research Center for Information Science and Technology (BNRist). Lei Cao is supported by the NSF (DBI-2327954) and Amazon Research Award.

REFERENCES

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[2] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang. 2023. Sparks of Artificial General Intelligence: Early experiments with GPT-4. arXiv:2303.12712 [cs.CL]

[3] Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: Line Graph Enhanced Text-to-SQL Model with Mixed Local and Non-Local Relations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, 2541–2555. https://doi.org/10.18653/v1/2021.acl-long.198

[4] Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, Steve Ash, William Yang Wang, Zhiguo Wang, Vittorio Castelli, Patrick Ng, and Bing Xiang. 2023. Dr.Spider: A Diagnostic Evaluation Benchmark towards Text-to-SQL Robustness. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=Wc5bmZZU9cy

[5] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *CoRR* abs/2107.03374 (2021). arXiv:2107.03374 https://arxiv.org/abs/2107.03374

[6] Zui Chen, Zihui Gu, Lei Cao, Ju Fan, Samuel Madden, and Nan Tang. 2023. Symphony: Towards Natural Language Query Answering over Multi-modal Data Lakes. In *13th Conference on Innovative Data Systems Research, CIDR 2023, Amsterdam, The Netherlands, January 8-11, 2023*. www.cidrdb.org. https://www.cidrdb.org/cidr2023/papers/p51-chen.pdf

[7] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling Language Modeling with Pathways. *CoRR* abs/2204.02311 (2022). https://doi.org/10.48550/arXiv.2204.02311 arXiv:2204.02311

[8] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-Grounded Pretraining for Text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 1337–1350. https://doi.org/10.18653/v1/2021.naacl-main.105

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[10] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *arXiv preprint arXiv:2308.15363* (2023).

[11] Zihui Gu, Ju Fan, Nan Tang, Lei Cao, Bowen Jia, Sam Madden, and Xiaoyong Du. 2023. Few-shot Text-to-SQL Translation using Structure and Content Prompt Learning. *Proc. ACM Manag. Data* 1, 2 (2023), 147:1–147:28. https://doi.org/10.1145/3589292

[12] Zihui Gu, Ju Fan, Nan Tang, Preslav Nakov, Xiaoman Zhao, and Xiaoyong Du. 2022. PASTA: Table-Operations Aware Fact Verification via Sentence-Table Cloze Pre-training. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, 4971–4983. https://doi.org/10.18653/V1/2022.EMNLP-MAIN.331

[13] Zihui Gu, Ruixue Fan, Xiaoman Zhao, Meihui Zhang, Ju Fan, and Xiaoyong Du. 2022. OpenTFV: An Open Domain Table-Based Fact Verification System. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 2405–2408. https://doi.org/10.1145/3514221.3520163

[14] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. DeBERTa: Decoding-enhanced BERT with Disentangled Attention. *CoRR* abs/2006.03654 (2020). arXiv:2006.03654 https://arxiv.org/abs/2006.03654

[15] Heikki Hyyrö. 2004. Bit-parallel LCS-length computation revisited. In *Proc. 15th Australasian Workshop on Combinatorial Algorithms (AWOCA 2004)*. Citeseer, 16–27.

[16] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic Evaluation of Text-to-SQL Parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Online, 2261–2273. https://doi.org/10.18653/v1/2021.acl-long.176

[17] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 7871–7880. https://doi.org/10.18653/v1/2020.acl-main.703

[18] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. *CoRR* abs/2302.05965 (2023). https://doi.org/10.48550/arXiv.2302.05965 arXiv:2302.05965

[19] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL. *Proc. ACM Manag. Data* 2, 3 (2024), 127. https://doi.org/10.1145/3654930

[20] Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Chenhao Ma, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. *CoRR* abs/2305.03111 (2023). https://doi.org/10.48550/arXiv.2305.03111 arXiv:2305.03111

[21] Zekun Li, Baolin Peng, Pengcheng He, Michel Galley, Jianfeng Gao, and Xifeng Yan. 2023. Guiding Large Language Models via Directional Stimulus Prompting. *arXiv preprint arXiv:2302.11520* (2023).

[22] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. 2023. A comprehensive evaluation of ChatGPT's zero-shot Text-to-SQL capability. *arXiv preprint arXiv:2303.13547* (2023).

[23] Jiachang Liu, Dinghan Shen, Yizhe Zhang, Bill Dolan, Lawrence Carin, and Weizhu Chen. 2022. What Makes Good In-Context Examples for GPT-3?. In *Proceedings of Deep Learning Inside Out: The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures, DeeLIO@ACL 2022, Dublin, Ireland and Online, May 27, 2022*, Eneko Agirre, Marianna Apidianaki, and Ivan Vulic (Eds.). Association for Computational Linguistics, 100–114. https://doi.org/10.18653/v1/2022.deelio-1.10

[24] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019). arXiv:1907.11692 http://arxiv.org/abs/1907.11692

[25] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023). https://doi.org/10.48550/arXiv.2303.08774 arXiv:2303.08774

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai,

and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (Eds.). 8024–8035. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[27] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[28] Octavian Popescu, Irene Manotas, Ngoc Phuoc An Vo, Hangu Yeo, Elahe Khorashani, and Vadim Sheinin. 2022. Addressing Limitations of Encoder-Decoder Based Approach to Text-to-SQL. In *Proceedings of the 29th International Conference on Computational Linguistics, COLING 2022, Gyeongju, Republic of Korea, October 12-17, 2022*, Nicoletta Calzolari, Chu-Ren Huang, Hansaem Kim, James Pustejovsky, Leo Wanner, Key-Sun Choi, Pum-Mo Ryu, Hsin-Hsi Chen, Lucia Donatelli, Heng Ji, Sadao Kurohashi, Patrizia Paggio, Nianwen Xue, Seokhwan Kim, Younggyun Hahm, Zhong He, Tony Kyungil Lee, Enrico Santus, Francis Bond, and Seung-Hoon Na (Eds.). International Committee on Computational Linguistics, 1593–1603. https://aclanthology.org/2022.coling-1.137

[29] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. arXiv:2304.11015 [cs.CL]

[30] Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. RASAT: Integrating Relational Structures into Pretrained Seq2Seq Model for Text-to-SQL. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, 3215–3229. https://aclanthology.org/2022.emnlp-main.211

[31] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[32] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67. http://jmlr.org/papers/v21/20-074.html

[33] Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498* (2022).

[34] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. http://arxiv.org/abs/1908.10084

[35] Ohad Rubin and Jonathan Berant. 2021. SmBoP: Semi-autoregressive Bottom-up Semantic Parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 311–324. https://doi.org/10.18653/v1/2021.naacl-main.29

[36] Victor Sanh, Albert Webson, Colin Raffel, Stephen H. Bach, Lintang Sutawika, Zaid Alyafeai, Antoine Chaffin, Arnaud Stiegler, Arun Raja, Manan Dey, M Saiful Bari, Canwen Xu, Urmish Thakker, Shanya Sharma Sharma, Eliza Szczechla, Taewoon Kim, Gunjan Chhablani, Nihal V. Nayak, Debajyoti Datta, Jonathan Chang, Mike Tian-Jian Jiang, Han Wang, Matteo Manica, Sheng Shen, Zheng Xin Yong, Harshit Pandey, Rachel Bawden, Thomas Wang, Trishala Neeraj, Jos Rozen, Abheesht Sharma, Andrea Santilli, Thibault Févry, Jason Alan Fries, Ryan Teehan, Teven Le Scao, Stella Biderman, Leo Gao, Thomas Wolf, and Alexander M. Rush. 2022. Multitask Prompted Training Enables Zero-Shot Task Generalization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net. https://openreview.net/forum?id=9Vrb9D0WI4

[37] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 9895–9901. https://doi.org/10.18653/v1/2021.emnlp-main.779

[38] Peter Shaw, Ming-Wei Chang, Panupong Pasupat, and Kristina Toutanova. 2021. Compositional Generalization and Natural Language Variation: Can a Semantic Parsing Approach Handle Both?. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, 922–938. https://doi.org/10.18653/v1/2021.acl-long.75

[39] Richard Shin, Christopher Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained Language Models Yield Few-Shot Semantic Parsers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 7699–7715. https://doi.org/10.18653/v1/2021.emnlp-main.608

[40] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

[41] Jianhong Tu, Ju Fan, Nan Tang, Peng Wang, Chengliang Chai, Guoliang Li, Ruixue Fan, and Xiaoyong Du. 2022. Domain Adaptation for Deep Entity Resolution. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 443–457. https://doi.org/10.1145/3514221.3517870

[42] Jianhong Tu, Ju Fan, Nan Tang, Peng Wang, Guoliang Li, Xiaoyong Du, Xiaofeng Jia, and Song Gao. 2023. Unicorn: A Unified Multi-tasking Model for Supporting Matching Tasks in Data Integration. *Proc. ACM Manag. Data* 1, 1 (2023), 84:1–84:26. https://doi.org/10.1145/3588938

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[44] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 7567–7578. https://doi.org/10.18653/v1/2020.acl-main.677

[45] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned Language Models are Zero-Shot Learners. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net. https://openreview.net/forum?id=gEZrGCozdqR

[46] Jingfeng Yang, Haoming Jiang, Qingyu Yin, Danqing Zhang, Bing Yin, and Diyi Yang. 2022. SEQZERO: Few-shot compositional semantic parsing with sequential prompts and zero-shot models. *arXiv preprint arXiv:2205.07381* (2022).

[47] Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, Youxuan Jiang, Michihiro Yasunaga, Sungrok Shim, Tao Chen, Alexander Fabbri, Zifan Li, Luyao Chen, Yuwen Zhang, Shreya Dixit, Vincent Zhang, Caiming Xiong, Richard Socher, Walter Lasecki, and Dragomir Radev. 2019. CoSQL: A Conversational Text-to-SQL Challenge Towards Cross-Domain Natural Language Interfaces to Databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Association for Computational Linguistics, Hong Kong, China, 1962–1979. https://doi.org/10.18653/v1/D19-1204

[48] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, 3911–3921. https://doi.org/10.18653/v1/d18-1425

[49] John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*. 1050–1055.

[50] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL]

[51] Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded Adaptation for Zero-shot Executable Semantic Parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 6869–6882. https://doi.org/10.18653/v1/2020.emnlp-main.558

[52] Terry Yue Zhuo, Zhuang Li, Yujin Huang, Yuan-Fang Li, Weiqing Wang, Gholamreza Haffari, and Fatemeh Shiri. 2023. On Robustness of Prompt-based Semantic Parsing with Large Pre-trained Language Model: An Empirical Study on Codex. *arXiv preprint arXiv:2301.12868* (2023).