

XiYan-SQL: A Novel Multi-Generator Framework For Text-to-SQL

Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, Jingren Zhou, *Fellow, IEEE*

Abstract—To leverage the advantages of LLM in addressing challenges in the Text-to-SQL task, we present XiYan-SQL, an innovative framework effectively generating and utilizing multiple SQL candidates. It consists of three components: 1) a Schema Filter module filtering and obtaining multiple relevant schemas; 2) a multi-generator ensemble approach generating multiple high-quality and diverse SQL queries; 3) a selection model with a candidate reorganization strategy implemented to obtain the optimal SQL query. Specifically, for the multi-generator ensemble, we employ a multi-task fine-tuning strategy to enhance the capabilities of SQL generation models for the intrinsic alignment between SQL and text, and construct multiple generation models with distinct generation styles by fine-tuning across different SQL formats. The experimental results and comprehensive analysis demonstrate the effectiveness and robustness of our framework. Overall, XiYan-SQL achieves a new SOTA performance of 75.63% on the notable BIRD benchmark, surpassing all previous methods. It also attains SOTA performance on the Spider test set with an accuracy of 89.65%.

Index Terms—Generation, Text-to-SQL, Multiple model ensemble, LLM fine-tuning strategy.

I. INTRODUCTION

THE task of translating natural language (NL) queries into structured query language (SQL) queries, known as Text-to-SQL, or NL2SQL, is a long-standing task for the development of NL interfaces to relational database management systems. This capability significantly reduces the interaction cost of users accessing database systems. Recent research is empowered by advances in large language models (LLMs), which have significantly improved the performance of NL2SQL solutions [1]. A series of approaches based on in-context learning (ICL) have been developed to enhance the SQL generation performance of these models by designing different prompts, such as chain-of-thought (CoT) [2], question decomposition [3], and choice of demonstration examples [4]. Nonetheless, these methods are highly constrained by the sensitivity of LLMs to the structure and content of prompts.

Consequently, an established effective strategy is to generate multiple SQL candidates by designing various forms of prompts, followed by an SQL selection process. Compared to a single

SQL query, this approach [4], [5] of utilizing multiple forms of prompts enhances the diversity of the candidate samples and demonstrates promising performance on the challenging BIRD benchmark [6]. However, relying solely on the design of multiple prompts does not adequately ensure the quality and robustness of SQL generation while improving candidate diversity. And utilizing self-consistency as a selection criterion does not always yield the optimal SQL query.

To address these challenges, we propose XiYan-SQL, an end-to-end SQL generation framework, which focuses on generating SQL candidates with high accuracy and diversity, as well as effectively selecting the best candidate. We first introduce a Schema Filter Module that provides SQL generators with diverse and high-quality database schemas. This module performs multi-path retrieval of tables, columns, and values based on the inputs and subsequently employs a model for iterative column selection.

For the generation of SQL candidates, we propose a novel method that integrates multiple generators, leveraging the distinct advantages of each to generate high-quality and diverse SQL queries. Instead of designing different forms of prompts, we capitalize on the high controllability of supervised fine-tuning (SFT) to construct multiple SQL generation models. Specifically, inspired by the multi-tasking approach in natural language processing, we explore a multi-task joint fine-tuning strategy tailored for SQL generation. This strategy involves the incorporation of various SQL and natural language transformation tasks to enhance the model's syntactic alignment capabilities. Additionally, we investigate the multi-format SQL enhancement training approach aimed at developing generators with distinct generation advantages to improve the diversity of candidates. Ultimately, these strategies significantly improve the performance of fine-tuned models.

For the SQL candidate selection phase, we employ a selection model, fine-tuned using constructed comparative samples, to identify the best answer among all candidates. Furthermore, we explore a candidate reorganization strategy based on the consistency results, to improve the selection model's attention to potentially correct results, thereby improving selection performance.

In the experiments, we evaluate XiYan-SQL on two benchmarks, BIRD [6] and Spider [7], to demonstrate the superiority of the framework. For the well-known and challenging BIRD benchmark, we achieve an execution accuracy of 75.63%, surpassing all other methods and establishing a new SOTA performance. For the Spider test set, we achieve SOTA performance with an execution accuracy of 89.65%. The

Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, and Jingren Zhou are affiliated with Alibaba Cloud Computing Co., Ltd., Hangzhou, Zhejiang, China (emails: zhencang.lyf@alibaba-inc.com; sherlin.zy@alibaba-inc.com; gaoyingqi.gyq@alibaba-inc.com; godot.lzl@alibaba-inc.com; suian.lxx@alibaba-inc.com; shixiaorong.sxr@alibaba-inc.com; jiaxu.hyt@alibaba-inc.com; jinyang.gjy@alibaba-inc.com; lojze.ly@alibaba-inc.com; bolin.ding@alibaba-inc.com; jingren.zhou@alibaba-inc.com).

extensive experimental results and comprehensive analysis further confirm the effectiveness and robustness of our method. The related code and models have been gradually released to support community research. *

Our main contributions are as follows:

- We propose XiYan-SQL, a novel Text-to-SQL framework that effectively integrates multiple SQL generators to generate high-quality and diverse SQL candidates, achieving new SOTA performance on both the BIRD and Spider benchmarks.
- We introduce a Schema Filter Module that generates multiple high-quality schemas with different precision and recall through multi-path retrieval and iterative column selection.
- We implement effective fine-tuning methods to construct multiple SQL generators, including enhancing the model's adaptability to task complexity through multi-task fine-tuning strategies, thus developing high-accuracy generation models, and constructing diverse models with different generation advantages through multi-format SQL enhancement training.
- We propose a candidate reorganization strategy that utilizes an SQL selection model to identify the optimal SQL from multiple candidates.
- Comprehensive experimental results demonstrate the effectiveness of the XiYan-SQL framework and its key components.

This paper is organized as follows. In Section II, we review the development of Text-to-SQL and the current work related to LLMs. In Section III, we introduce the complete XiYan-SQL framework, along with detailed explanations of the methods for Schema Filtering, Multiple SQL Generation, and SQL Selection. Then, in Section IV, we present a thorough and comprehensive experimental evaluation. Finally, we provide important empirical insights related to the XiYan-SQL technology in Section V, and conclude the paper in Section VI.

II. RELATED WORK

In this section, we discuss the development of Text-to-SQL related work. Text-to-SQL systems have garnered significant attention in recent years due to their potential to bridge the gap between natural language processing and SQL execution. Early work in this field relied mainly on hand-crafted templates [8]. With the advancement of transformers [9] in deep neural networks, typical models have adopted an encoder-decoder architecture to generate SQL directly from natural language inputs. Several pre-trained language models (PLMs), such as BERT [10], TaBERT [11], and GraPPa [12], have been developed to encode structured data representations effectively. However, these early methods demonstrate limitations when applied to more complex and diverse scenarios. Currently, the rapid development of large language model technologies has demonstrated unique emerging capabilities in developing Text-to-SQL solutions. This includes advancements in prompt engineering techniques [13] [14] [5], as well as pre-training or

fine-tuning methods [15] [16] [17] for large language models. In this paper, we focus mainly on the technology related to LLMs.

Prompt Engineering Methods. Prompt engineering methods leverage the inherent capabilities of models to guide LLMs in generating diverse SQL queries by optimizing prompts [5], [13], [14], [18]. These methods are typically employed out-of-the-box on closed-source models with a vast number of parameters, such as GPT-4 [19] and Gemini [20]. Some approaches based on in-context learning address various stages of the workflow, including schema linking [21], [22], SQL generation [1], and SQL refinement [3], [14], [23], by designing question/task decomposition [3], demonstration selection [4], chain-of-thought reasoning [2], and multi-agent processing within the prompts. To enhance performance, an effective strategy is to generate multiple candidate SQL queries through different prompt designs and subsequently select the best one [4], [5]. However, these methods rely solely on prompt design, which presents challenges related to the sensitivity of LLMs to prompts, potentially compromising the quality and robustness of SQL generation, along with incurring significant inference overhead. In contrast, our approach effectively mitigates these problems by constructing different high-accuracy fine-tuned SQL generation models.

Fine-tuning Methods. The fine-tuning methods can align objectives in models with fewer parameters, allowing the fine-tuned models to demonstrate improved controllability in generating SQL queries [15]–[17], [24], [25]. Additionally, this method offers lower inference overhead and enhanced data privacy protection. Recently, CodeS [16] has been trained on a large corpus related to SQL, aimed at improving the general Text-to-SQL performance of smaller models. DTS-SQL [15] introduces a two-stage supervised fine-tuning method that decomposes the task into two simpler components: schema linking and SQL generation. Yang et al. [17] constructed the SENSE model by fine-tuning it on synthesized robust data and applying Direct Preference Optimization [26] on weak data. Despite these advancements, these methods exhibit limited performance and often struggle in more challenging scenarios or benchmarks.

This paper focuses on the development of high-accuracy SQL generation models. By utilizing the proposed multi-task fine-tuning strategy and multi-format data manner, the fine-tuned models demonstrate superior performance compared to the ICL-based closed-source model. Furthermore, to the best of our knowledge, this work represents the first exploration in this field to generate multiple candidate SQL queries through multi-generators. Through the comprehensive XiYan-SQL framework, we can effectively generate multiple candidates and select the optimal one.

III. METHODOLOGY

A. Overall Framework

We present the XiYan-SQL framework in Figure 1, where the inputs include a question, evidence (i.e., external knowledge), and the database schema (i.e., the overall organization of tables, columns, and values within the database). The framework

*<https://github.com/XGenerationLab/XiYan-SQL>

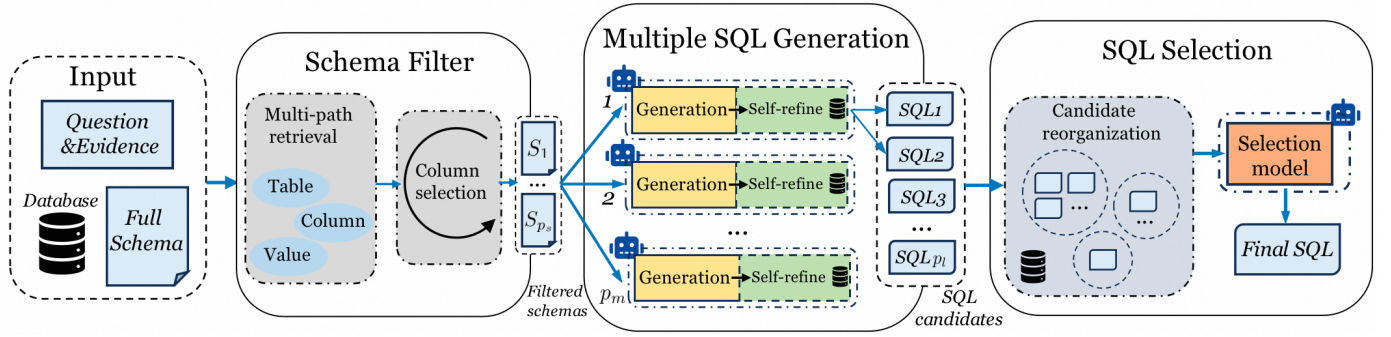


Fig. 1. Overview of the proposed XiYan-SQL framework, including three steps: Schema Filter, Multiple SQL Generation, and SQL Selection.

consists of three steps: Schema Filter, Multiple SQL Generation, and SQL Selection. The Schema Filter module initially generates multiple filtered schemas. Subsequently, multiple candidate SQL queries are generated through various SQL generators. Finally, the most accurate SQL query is selected by the selection model based on the candidate reorganization. The details of each step are described below.

B. Schema Filter

Our Schema Filter Module consists of multi-path retrieval and column selection, designed to effectively implement schema linking for large databases and generate multiple filtered schemas.

Multi-path Retrieval. Multi-path Retrieval is a pruning mechanism for large databases that employ cosine similarity for the embeddings of all tables, columns, and values. Initially, we used an out-of-the-box LLM \mathcal{M}_s to extract keywords from the question Q and evidence E , represented as $K = f_{\mathcal{M}_s}(Q, E)$. These keywords, denoted as $K = \{k_1, k_2, \dots\}$, are instrumental in identifying relevant columns and values within the database. In the table and column retrieval stage, we compute the scores between the keywords K and the original schema S , which can be represented as a set of columns, $\{c_1, c_2, \dots\}$. This process is calculated as the product of two distinct scores as follows.

$$\text{Score}(k_i, c_j) = \langle \mathbf{V}_{Q||E}, \mathbf{V}_{Tab(c_j)} \rangle \cdot \langle \mathbf{V}_{k_i}, \mathbf{V}_{c_j} \rangle \quad (1)$$

One part is the cosine similarity between the embedding of the keyword \mathbf{V}_{k_i} and that of the column metadata \mathbf{V}_{c_j} . The other part is the calculation between the input text and the table metadata. In detail, we concatenate the query and evidence to obtain the embedding representation $\mathbf{V}_{Q||E}$. The table to which c_j belongs can be determined by projection Tab , and the embedding representation of the table is $\mathbf{V}_{Tab(c_j)}$. Through this process, we identify the top- k columns that are most relevant to each keyword.

For the value retrieval, we first use edit distance to locate the top- k values of each column that are similar to each keyword. Subsequently, we employ the RoBERTa tokenizer to uniquely tokenize the value text along with column metadata. This step enhances the efficiency of Locality Sensitive Hashing

[27] in filtering text related to values. An embedding cosine similarity is then applied, using a threshold to refine the retrieved values. Through multi-path retriever, we filter out a schema $S^{triv} = \{c_{r_1}, c_{r_2}, \dots\}$ from the original schema S .

Column Selection. Column Selection introduces diversity into the retrieved schemas, facilitating subsequent multiple SQL generation. This process is detailed in Algorithm 1, which outputs p_s different schemas through multiple iterations. In each iteration, we prompt an LLM \mathcal{M}_s to select columns related to the question and evidence, saved as S_i^{slect} . The function PFKeyIdentifier then identifies primary and foreign keys according to the columns in S_i^{slect} . A new schema is generated by unifying the previous schemas with the selected columns. The selected columns in this iteration are removed from S^{triv} , while retaining essential columns such as primary keys. As iterations increase, the precision of the selected schemas tends to decrease while recall improves, resulting in a diversity of filtered schemas.

Algorithm 1 Column Selection Algorithm

Input: Set of columns $S^{triv} = \{c_{r_1}, c_{r_2}, \dots\}$, Question Q , Evidence E , The maximum iteration p_s

Output: Schema set $\mathcal{S} = \{S_1, \dots, S_{p_s}\}$

- 1: Initialize a list $\mathcal{S} \leftarrow []$
 - 2: **for** $i = 1$ to p_s **do**
 - 3: $S_i^{slect} \leftarrow f_{\mathcal{M}_s}(S^{triv}, Q, E)$ {Select from S^{triv} }
 - 4: $P_i \leftarrow \text{PFKeyIdentifier}(S_i^{slect})$
 - 5: $S_i \leftarrow \left(\bigcup_{k=1}^{i-1} S_k \right) \cup S_i^{slect} \cup P_i$ {Unify and generate S_i }
 - 6: Append S_i to \mathcal{S}
 - 7: $S^{triv} \leftarrow S^{triv} \setminus (S_i^{slect} \setminus P_i)$ {Remove the selected}
 - 8: **return** \mathcal{S}
-

C. Multiple SQL Generation

Generating multiple candidate SQL queries to explore a broader search space to improve performance has been demonstrated as a reasonable approach in previous studies [4], [5]. However, prompt-based methods are constrained by the model's sensitivity to prompt formats, presenting a challenge in ensuring SQL quality while enhancing diversity. Considering that supervised fine-tuning can better align the model with preferred behaviors [28], [29], we explore a dedicated training

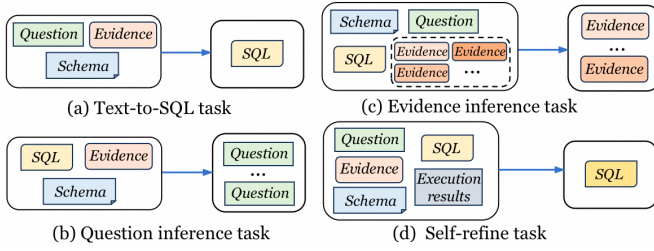


Fig. 2. The illustration of the process for multiple related tasks.

strategy to develop SQL generation models that generate high-quality and diverse candidates.

As a translation task, Text-to-SQL aims to achieve semantic alignment between structured representations and natural language [30]. Drawing inspiration from previous multi-task approaches [31], [32] that leverage auxiliary tasks for learning, we explore a multi-task fine-tuning approach that utilizes related tasks to enhance information alignment. The standard task process for converting text to SQL, given the inputs of the question, database schema, and evidence, is illustrated in Figure 2(a).

Due to the inherent flexibility of natural language forms, this unidirectional conversion method is insufficient to fully exploit the model’s semantic linking capabilities. To address this issue and enhance the semantic alignment capability of LLMs for this task, we further design auxiliary alignment tasks and explicitly fine-tune the SQL generation model for execution. In response to natural language questions, we design a reverse inference task, as illustrated in Figure 2(b), which aims to encourage the model to infer a set of potential questions based on SQL and relevant information. For natural language evidence knowledge, we establish a task focused on the reverse inference of evidence, shown in Figure 2(c). This task involves the model identifying the most relevant evidence from a set of candidate evidence based on the SQL generation process. Additionally, for SQL queries, we develop a self-refine task for the model, as depicted in Figure 2(d), which entails regenerating SQL after refinement based on previous SQL generation processes and execution results. By leveraging a series of specialized tasks to jointly fine-tune models, our developed single generator surpasses ICL-based closed-source models, such as GPT-4o and Gemini, resulting in the generation of higher-quality SQL queries.

We further develop from a single generator to multiple generators in order to enhance the diversity of generated candidates. For a single user question, there can be multiple SQL queries that correspond to fulfill the intent. Therefore, we aim to develop additional models with different ways of generation to increase the likelihood of arriving at the correct answers through diverse outputs. To achieve this, we restructure the format variations with SQL to introduce greater diversity during model building. A key aspect is the variation of SQL writing patterns, encouraging the model to attempt using more complex SQL structures, thereby enhancing its ability to cover a wider range of challenging scenarios (as illustrated by SQL₁ on the left side of Figure 3, which demonstrates a preference for utilizing advanced queries). Furthermore, we

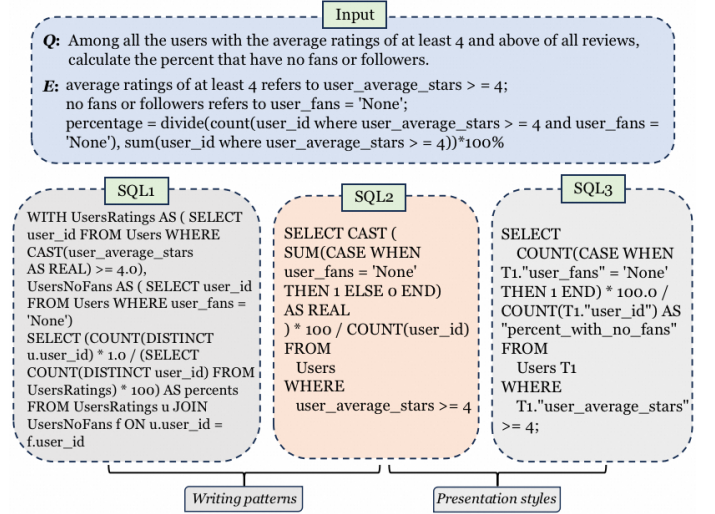


Fig. 3. Examples of multiple SQL queries with different format corresponding to the same input. The middle SQL query is a typical response. In contrast, the left SQL employs a chunked writing pattern, which exhibits greater complexity in structure compared to the middle; the SQL on the right differs from that in the middle in terms of presentation style.

also implement certain modifications to the aspect of SQL writing style, allowing the model to adopt variations in its writing conventions to increase diversity in the generated outputs (as demonstrated on the right side of Figure 3, SQL₃ presents a more standardized output representation). In practice, we conduct various aspects of reformulation on a subset of challenging queries to supplement the diverse training data. By leveraging these formatted and multi-task data for joint fine-tuning, we obtain high-accuracy models with diverse SQL generation preferences.

As shown in Figure 1, we provide the generation model with p_s different schemas generated during the Schema Filter phase to expand the candidate pool. To further enhance the diversity of candidates, our framework not only integrates multiple fine-tuned models but also incorporates an ICL-based model to collectively generate p_l candidate SQL queries, where $p_l = p_s * p_m$, and p_m represents the number of generators. Moreover, for each generation, if the execution of the generated SQL encounters syntax errors or anomalous values, the corresponding generator will utilize execution feedback to regenerate a SQL query through its inherent self-refine capabilities. The complete process for multiple SQL generation is illustrated in Algorithm 2. Please refer to Appendix I for details about the ICL-based generator.

D. SQL Selection

After obtaining multiple SQL candidates, the challenge remains to select the most accurate and reasonable SQL query from these candidates. Most candidate selection methods [4], [21] employ self-consistency [33] to select the SQL query that appears most consistently across multiple candidate samples. However, there are limitations present: it cannot adequately handle situations where there is a lack of majority consensus, and even the most consistent result is not always the correct

Algorithm 2 Multiple SQL Generation Algorithm

Input: Question Q , Evidence E , Filtered Schema set $\mathcal{S} = \{S_1, \dots, S_{p_s}\}$, List of SQL Generators \mathcal{M} , DataBase D
Output: Candidate SQL list $L = \{l_1, \dots, l_{p_l}\}$

- 1: Initialize Candidate SQL $L \leftarrow \emptyset$
- 2: **for** $i = 1$ to p_s **do**
- 3: **for** $j = 1$ to p_m **do**
- 4: Predict SQL $l_{ij} \leftarrow f_{\mathcal{M}_j}(Q, E, S_i)$
- 5: Execute Result $r \leftarrow \text{execute}(l, D)$
- 6: **if** No exception in r **then**
- 7: Append l_{ij} to L
- 8: **else**
- 9: **for** retry 1 times **do**
- 10: $\hat{l}_{ij} \leftarrow f_{\mathcal{M}_j}(Q, E, S_i, l, r)$ {Self-refine}
- 11: Append \hat{l}_{ij} to L
- 12: **return** L

case. To this end, we propose an enhanced selection strategy that employs a selection model to make decisions based on consistent results.

To develop a selection model, we focus on constructing enhanced synthetic training data. On the one hand, we employ various generators with diverse inputs to generate multiple candidate SQL queries. By grouping and clustering samples based on execution results, we are able to construct positive and negative contrastive samples. Additionally, to account for the complexity of the candidate results, we utilize an LLM to make a few controlled modifications for the correct queries to supplement the dataset. On the other hand, when organizing the candidate results into training prompts, we first perform a de-formalization process on the samples to reduce the interference caused by the stylistic aspects of SQL. For the constructed dataset, we ensure that the distribution of the combinations of positive and negative samples is balanced, that the generators from which the selected candidates originate are evenly distributed, and that the order of the candidates is uniformly distributed.

Due to the presence of numerous candidate information in the model selection process, we further propose a candidate reorganization strategy to enhance the model's attention. The complete selection process is illustrated in Algorithm 3. Initially, we obtain the execution results $R = \{r_1, \dots, r_n\}$ of the candidate SQL queries $L = \{l_1, \dots, l_n\}$ on the database, ensuring that any candidates with execution errors have already been excluded. We group the candidates based on the consistency of their execution results to obtain a collection of clusters $\mathcal{C} = \{C_1, \dots, C_m\}$. Subsequently, we perform inter-group and intra-group sorting. The inter-group sorting is performed in descending order based on the size of each group, resulting in \mathcal{C}' . The intra-group sorting is executed according to the order of the generators \mathcal{O} corresponding to the candidates, resulting in \mathcal{C}'' , where the sequence \mathcal{O} is arranged in descending order based on the performance evaluation of the generators.

If there exists a consensus result that predominates within the clusters, namely $|C''_1| \geq \lceil |L|/2 \rceil$, we organize all candidates sequentially according to order \mathcal{C}'' and present them to the selection model \mathcal{M}_c . Conversely, we select one candidate from

Algorithm 3 SQL Selection Algorithm

Input: Candidate SQL queries L , Question Q , Evidence E , Filtered Schema set $\mathcal{S} = \{S_1, \dots, S_{p_s}\}$, Selection Model \mathcal{M}_c , SQL Execution results R , Generators Order \mathcal{O}
Output: Selected SQL l^*

- 1: Clustering $\mathcal{C} = \{C_1, \dots, C_m\} \leftarrow \text{groupby}(L, R)$
- 2: Schema $\mathcal{S}^{un} \leftarrow \text{schema_union}(\mathcal{S})$
- 3: **if** $|\mathcal{C}| = 1$ **then**
- 4: **return** $l^* \leftarrow \arg_{l \in L} \min |l|$ {All are consistent; the shortest is selected.}
- 5: **else**
- 6: Clusters $\mathcal{C}' \leftarrow \text{sort}(\mathcal{C})$ by size, descending {Inter-group sorting}
- 7: Initialize Clusters $\mathcal{C}'' \leftarrow \emptyset$
- 8: **for** each cluster $C'_i \in \mathcal{C}'$ **do**
- 9: Append $C''_i \leftarrow \text{sort}(C'_i)$ order by \mathcal{O} {Intra-group sorting}
- 10: Initialize Candidate $L' \leftarrow \emptyset$, indicator $j \leftarrow 0$
- 11: **if** $|C''_1| \geq \lceil |L|/2 \rceil$ **then**
- 12: **for** each Cluster $C''_i \in \mathcal{C}''$ **do**
- 13: **for** each SQL $l \in C''_i$ **do**
- 14: Append $L'_j \leftarrow l, j \leftarrow j + 1$
- 15: **else**
- 16: **for** each Cluster $C''_i \in \mathcal{C}''$ **do**
- 17: Append $L'_j \leftarrow \arg_{l \in C''_i} \min |l|, j \leftarrow j + 1$
- 18: **return** $l^* \leftarrow f_{\mathcal{M}_c}(Q, \mathcal{S}^{un}, E, L')$ {Selection model prediction}

each group within \mathcal{C}'' and assemble the results in accordance with the inter-group ordering to present to \mathcal{M}_c . The model \mathcal{M}_c takes into account the question, the union of all filtered schemas, the evidence, and the reorganized candidates to make its decision. Due to the selection tendency of LLMs towards options that appear earlier [34], employing this clustering and ranking approach to organize candidates allows us to effectively leverage strong consistency priors while mitigating the influence of interfering information. This allows the model's attention to be concentrated, thereby enhancing their comprehension ability.

IV. EXPERIMENTS

In this section, we aim to provide a detailed evaluation of XiYan-SQL. We first introduce the experimental setting, followed by the demonstration of XiYan-SQL's SOTA performance on important benchmarks. Subsequently, we conduct ablation experiments on the overall framework and provide a comprehensive analysis of each component. To ensure fairness in our experiments, the results of previous methods used for comparative analysis have been sourced from publicly available benchmark tests, which typically represent their most powerful implementations. Additionally, the results we have reproduced for comprehensive comparison are obtained using the same configuration as XiYan-SQL (e.g., base models, etc.).

A. Experimental Setup

Dataset. We conduct experiments to evaluate the performance of the XiYan-SQL framework on two of the most rec-

ognized and challenging benchmarks in Text-to-SQL research in recent years.

BIRD [6], focusing on complex real-world databases, is the most challenging larger-scale cross-domain Text-to-SQL benchmark. It contains 12,751 unique question-SQL pairs and 95 big databases with a total size of 33.4 GB. The training, development, and unpublished test set contain 9,428, 1,534, and 1,789 pairs, respectively. We use its development set for local evaluation and submit our performance results on the unreleased test set for a fair evaluation.

Spider [7], widely used for Text-to-SQL evaluation, is a large-scale, complex, cross-domain benchmark. It comprises 10,181 questions and 5,693 distinct SQL queries across 200 databases, covering 138 different domains. This dataset is divided into training, development, and test sets, comprising 8,659, 1,034, and 2,147 examples, respectively, with a primary focus on evaluating the results on the test set.

Metrics. Following the previous studies, we use Execution Accuracy (EX) to evaluate the performance of Text-to-SQL methods. Since an SQL query can be expressed in various forms, EX is used to assess the validity of the predicted SQL query and to determine whether the execution results are consistent with the ground-truth SQL query. It should be noted that there are slight differences in the calculation of the Execution Accuracy between the BIRD and Spider benchmarks; however, their overall objective remains consistent.

In addition to EX, the latest version of the BIRD benchmark introduces a Reward-based Valid Efficiency Score (R-VES) to evaluate the execution efficiency of correctly predicted SQL queries. R-VES is an adjusted version of the previous VES, which assesses the model by considering both the accuracy of and the runtime performance of SQL queries [6]. We also present an evaluation of this metric on the BIRD benchmark in the main results.

Implementations. In the proposed framework, all database schemas are represented in the form of the M-schema [35]. For the Schema Filter, we utilize GPT-4o[†] [36] as the \mathcal{M}_s model for keyword extraction and column selection. We set the maximum number of iterations p_s to 2 to obtain the filtered schema set $\mathcal{S} = \{S_1, S_2\}$. For the Multiple SQL Generation part, we first leverage GPT-4o and Qwen-Max[‡] [37] for assistance in constructing relevant multi-task and multi-format SQL data. We conduct experiments on multiple models of varying sizes to thoroughly demonstrate the superiority of our fine-tuning approach. To achieve top performance with XiYan-SQL, we select the widely used Qwen2.5-Coder-32B [38] as the foundation for constructing the final multiple fine-tuned SQL generators. We utilize a total of $p_m = 5$ generators, which includes one ICL-based generator, ultimately resulting in the generation of $p_l = 10$ SQL candidates. Due to limited cost and resources, we have not increased the number of models or candidates. For the SQL selection part, we also utilize GPT-4o to augment the training data. Given that the selection task is relatively simpler than the generation task, we choose Qwen2.5-Coder-7B for developing the SQL selection model. All base

TABLE I
PERFORMANCE COMPARISON OF DIFFERENT TEXT-TO-SQL METHODS ON BIRD BENCHMARK.

Methods	EX(Dev)	EX(Test)	R-VES(%)
GPT-4o	58.47	-	51.75
TA-SQL + GPT-4 [39]	56.19	59.14	-
DAIL-SQL [13]	54.76	57.41	54.02
SFT CodeS-15B [16]	58.47	60.37	61.37
SuperSQL [1]	58.50	62.66	-
MCS-SQL [4]	63.36	65.45	61.23
PURPLE + RED + GPT-4o	68.12	70.21	65.62
Insights AI	72.16	70.26	66.39
CHESS [21]	68.31	71.10	66.53
Distillery + GPT-4o [22]	67.21	71.83	67.41
OpenSearch-SQL, v2 + GPT-4o	69.30	72.28	69.36
ExSL + granite-34b-code	72.43	73.17	71.37
DSAIR + GPT-4o	74.32	74.12	70.13
CHASE-SQL+Gemini [5]	74.46	74.79	70.57
XiYan-SQL	73.34	75.63	71.41

models used for fine-tuning in the following experiments are their respective instruction versions. For more implementation details, please refer to Appendix II.

B. Main Results

BIRD Results. We present the end-to-end performance of the XiYan-SQL on the BIRD benchmark, based on the official benchmark page. In addition to the results obtained from GPT-4o and XiYan-SQL, all other results are derived from the respective original publications. As shown in Table I, our method achieves an EX score of 75.63%, outperforming all previous works and leading the second-best result by 0.84%, establishing a new SOTA performance. This provides strong evidence for the effectiveness of XiYan-SQL as an advanced technique for multiple candidates generation and selection.

We further demonstrate the comparison of XiYan-SQL against other methods based on the R-VES metric, with results sourced from the official benchmark webpage. As shown in Table I, we achieve an R-VES score of 71.41%, maintaining our position as SOTA performance on the BIRD leaderboard, comparable to EX.

Spider Results. To further evaluate the generalizability of XiYan-SQL, we conduct experiments on the Spider test set, with results shown in Table II. Similar to the results obtained on the BIRD benchmark, XiYan-SQL achieves an EX score of 89.65%, surpassing previous methods and establishing a new SOTA performance. It is noteworthy that the testing on Spider does not involve any specific adjustments; instead, we directly employ the model and configuration strategies evaluated on BIRD. This highlights the robust generalizability of our approach, demonstrating consistent performance across various Text-to-SQL scenarios.

C. Ablation Study

To investigate the significance of each component within the full pipeline, we conduct ablation studies to assess the incremental impact of each component on the EX. The results of the end-to-end ablation on the BIRD dev set are presented

[†]gpt-4o-0806 for all GPT-4o (<https://openai.com/index/gpt-4/>)

[‡]<https://qwenlm.github.io/zh/blog/qwen2.5-max/>

TABLE II
PERFORMANCE COMPARISON OF DIFFERENT TEXT-TO-SQL METHODS ON SPIDER TEST BENCHMARK.

Methods	EX(%)
MCS-SQL + GPT-4 [4]	89.60
CHASE-SQL + Gemini 1.5 [5]	87.60
PET-SQL [40]	87.60
SuperSQL [1]	87.00
DAIL-SQL + GPT-4 [13]	86.60
DPG-SQL + GPT-4	85.60
Tool-SQL + GPT-4 [41]	85.60
DIN-SQL + GPT-4 [14]	85.30
GPT-4o	83.54
C3 + ChatGPT + Zero-Shot [18]	82.30
XiYan-SQL	89.65

TABLE III
ABLATION STUDIES OF EACH COMPONENT ON THE PERFORMANCE OF XIYAN-SQL ON BIRD DEV.

Methods	EX(%)	Δ EX(%)
XiYan-SQL (Full pipeline)	73.34	-
w/o Schema Filter	72.10	-1.24
w Only one SFT generator	69.30	-4.04
w Only majority voting	70.21	-3.13

in Table III. To begin with, when the Schema Filter module is removed, leaving only the full schema, it significantly affects the quality of the generated SQL candidates, leading to an overall performance drop of approximately 1.24%. This result highlights that the proposed schema filter method is crucial for generating high-quality schemas, thereby enhancing the performance of the SQL generators. When we perform an ablation on Multiple SQL Generation by utilizing a single fine-tuned generation model to generate one SQL query, we observe a significant performance decline of approximately 4%. This result strongly underscores the outstanding performance of our multiple SQL generation technique. Additionally, if the single generation model is not derived from the advanced training strategies we propose, but rather from a conventional fine-tuned model or a model based on ICL, the performance degradation would be even more pronounced. We will further discuss this in the next section. Finally, when we remove the proposed SQL Selection approach, which relies solely on the consistency of the candidate SQL execution results for a majority voting, we observe a significant decrease in performance. This strongly corroborates the importance of this specific technique within the overall framework. Overall, our ablation experiments indicate that each component of XiYan-SQL plays a vital role in achieving high accuracy.

D. Study of Schema Filter

We first evaluate the performance of our Schema Filter module in filtering database schemas. Results of different schema filter methods on BIRD are shown in the Table IV. We employ the evaluation metrics presented in schema linking [22], where P represents the column's precision in the filtered

TABLE IV
SCHEMA FILTER COMPARISON OF DIFFERENT TEXT-TO-SQL METHODS ON BIRD DEV.

Methods	$P(\%)$	$R_c(\%)$	$R_v(\%)$
Full schema	10.14	100	-
CHESS [21]	85.90	74.77	63.71
TA-SQL [39]	82.50	76.83	-
Our Schema Filter (schema S_1)	74.89	83.64	91.31
Our Schema Filter (schema S_2)	54.90	89.77	93.63

TABLE V
THE COMPARISON OF THE END RESULTS OF DIFFERENT SCHEMA FILTER METHODS ON BIRD DEV.

Methods	EX(%)	Δ EX(%)
Full schema	61.57	-
CHESS [21]	63.30	+1.73
Our Schema Filter (schema S_1)	63.75	+2.18
Our Schema Filter (schema S_2)	62.97	+1.40

schema, while R_c and R_v denote the recall for columns and values, respectively. Compared to the advanced methods for schema linking that we reproduce, our Schema Filter achieves higher recall for both columns and values while maintaining precision. Furthermore, as iterations increase, it effectively adjusts the precision and recall of the schema, resulting in diverse schemas.

We further supplement the performance of the final generated SQL queries under different schema linking methods. Due to performance differences among the various generators, we average the results from three different models: the fine-tuned model based on Qwen2.5-Coder-32B, GPT-4o, and Gemini-1.5-pro[§]. Notably, we emphasize the Δ EX metrics, which indicate the improvements achieved compared to the full schema method. As shown in Table V, our methods demonstrate significant enhancements. Additionally, we find that selecting a single iteration (i.e., $p_s = 1$) yields better results than the second iteration. Throughout the entire pipeline, we introduce greater schema diversity through multiple iterations. For additional experiments about the Schema Filter, please refer to Appendix III.

E. Study of Single SQL Generation

We first investigate the performance of a single fine-tuned generator in generating a single SQL query based on the proposed fine-tuning strategy. Table VI demonstrates the consistency improvements achieved through the multi-task fine-tuning strategy across different model sizes, and we also compare these results with those of directly using GPT-4o as an SQL generator. For the ICL-based methods, by incorporating effective demonstration examples, we observe an approximate 4% improvement compared to directly using the powerful GPT-4o. Furthermore, regardless of the model type or size, the SQL generation models based on our multi-task training strategy exhibit significant enhancements, with improvements ranging

[§]<https://deepmind.google/technologies/gemini/>

TABLE VI

COMPARISON OF EX OF DIFFERENT SINGLE GENERATORS ON BIRD DEV. ALL EVALUATION METHODS EXCLUDE ADDITIONAL STEPS SUCH AS SCHEMA LINKING AND SELF-REFINE. NOTE THAT THE BASE MODELS WE USED FOR FINE-TUNING ARE ALL THEIR INSTRUCT VERSIONS.

Methods	EX (%)
GPT-4o + zero-shot	58.47
GPT-4o + few-shot	62.65
Base model (<i>DeepSeek-Coder-V2-Lite</i>)	40.61
SFT generator	51.49
SFT generator + multi-task	55.99
Base model (<i>Qwen2.5-Coder-7B</i>)	50.39
SFT generator	56.06
SFT generator + multi-task	60.10
Base model (<i>Qwen2.5-Coder-14B</i>)	55.28
SFT generator	61.02
SFT generator + multi-task	63.95
Base model (<i>Codestral-22B</i>)	50.52
SFT generator	62.32
SFT generator + multi-task	65.25
Base model (<i>Qwen2.5-Coder-32B</i>)	58.60
SFT generator	64.67
SFT generator + multi-task	66.88

from 8.3% to 15.3%. Compared to conventional fine-tuning, our joint multi-task training approach significantly enhances the model’s understanding of the translation from natural language to SQL, thereby improving the model’s adaptability to task complexity. This is reflected in the performance improvements across five widely selected basic fine-tuned models, with increases of 4.5%, 4.04%, 2.9%, 2.9%, and 2.2%, respectively. The fine-tuned performance of our models with sizes of 14B and above surpasses that of the few-shot approach using GPT-4o. When we employ Qwen2.5-Coder-32B as the base model, the fine-tuned SQL generation performance can reach 66.88%, which is also the SOTA performance for single SQL models.

We further present a detailed study of the various auxiliary tasks during the training phase. Table VII reports the ablation results on two models to explore the impact of each auxiliary task, where No.1~No.5 represents the fine-tuning results based on the Codestral-22B model, and No.6~No.10 represents the fine-tuning results based on the Qwen2.5-Code-32B model. We observe that the Question Inference task yields the greatest relative improvement among all auxiliary tasks. However, it cannot achieve optimal performance without the complete multi-task fine-tuning strategy, which further illustrates the complementarity among multiple tasks. Additionally, during the inference phase, aside from the main Text-to-SQL generation task, we also selectively perform a self-refine process, which enhances performance by approximately 1% and contributes to improving the quality of the final candidate pool.

F. Study of Multiple SQL Generation

We continue to evaluate the performance of utilizing multiple SQL generators to obtain multiple candidate SQL queries. Figure 4 (a) presents the comparison of various candidate generation methods to highlight the superiority of our multiple fine-tuned models and multi-generator ensemble approach.

TABLE VII

THE ABLATION RESULTS ON THE EX PERFORMANCE OF MULTIPLE AUXILIARY TASKS ON BIRD DEV.

No.	Question inference task	Evidence inference task	Self-refine task	EX(%)
1	✓	✓	✓	65.25
2	✓			64.21
3		✓		63.10
4			✓	63.56
5				62.23
6	✓	✓	✓	66.88
7	✓			65.84
8		✓		64.99
9			✓	65.97
10				64.67

Here, we follow the calculations of upper-bound and lower-bound performance as described in [5]. The ICL-Temp indicates the results obtained using GPT-4o with different sampling temperatures, which, as anticipated, exhibit lower candidate diversity and accuracy in the selected outputs. The FT-Temp refers to the candidates generated by our fine-tuned generator with different sampling temperatures, which demonstrate higher accuracy than ICL-Temp but similarly lack good diversity. The ICL-Prompt represents candidates generated by GPT-4o using different example selection prompts. Compared to the first two methods, ICL-Prompt shows improvements in both upper-bound and selection accuracy; however, the model’s sensitivity to the prompts results in a lower-bound performance. The Multi-FT refers to the candidates generated from our multiple fine-tuned models with distinct generation advantages. The fine-tuned generation models employed in this method achieve high accuracy, and the generated SQL queries also demonstrate considerable diversity, effectively balancing the relationship between lower and upper bounds. This results in an approximate 2% accuracy improvement over ICL-Prompt. Furthermore, the best method we explored (i.e., the last one) incorporates an ICL-based generation method alongside our multiple fine-tuned models. This integration allows us to maintain high-quality generation while further enhancing diversity.

Figure 4 (b) illustrates the performance of our multi-generator ensemble approach varying with the number of candidates. The results for different numbers of candidates are obtained through multiple random samplings from the candidate pool of five SQL generators. As the number of candidates increases, the candidate space exhibits greater diversity, leading to a higher upper-bound; at $p_l = 10$, this upper-bound reaches 82.2%, showing significant potential. However, the lower-bound also decreases considerably, which indicates that the selection process becomes crucial. Therefore, we explore an enhanced SQL selection method tailored for scenarios involving multiple candidates.

G. Study of SQL Selection

After obtaining multiple candidate SQL queries, we introduce a strategy that combines candidate reorganization with a SQL selection model to choose the optimal candidate SQL. Next, we explore the selection strategy with a detailed comparative analysis. Table VIII presents the analysis of different SQL

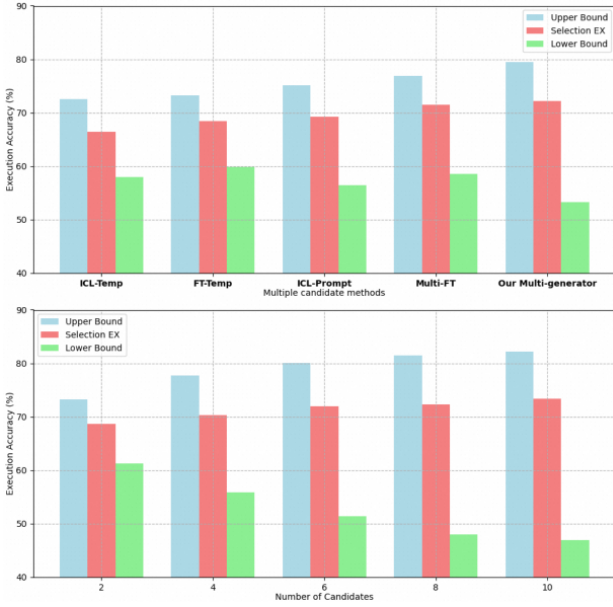


Fig. 4. (a) Comparison of EX among different multiple candidate methods with five candidates (Figure Above). (b) Performance of multi-generator method under different candidate numbers (Figure Below).

TABLE VIII
COMPARISON OF EX OF DIFFERENT SQL SELECTION METHODS ON BIRD DEV.

Methods	$p_l = 5(\%)$	$p_l = 10(\%)$
Majority voting	69.94	70.21
Our selection model (+ s_1)	69.69	68.19
Our selection model (+ s_2)	71.45	71.84
Our selection model (+ s_3)	72.29	73.34

selection methods. We observe that the majority voting strategy, based on consistency, achieves the EX of 69.82%. Furthermore, leveraging our fine-tuned selection model, we compare several distinct selection strategies. Here, s_1 denotes presenting the candidate samples to the selection model in a random order, s_2 indicates the organization of candidate SQL samples in descending order based on their corresponding model performance, and s_3 employs the proposed candidate reorganization to enhance the model’s attention. Our selection strategy (*i.e.*, s_3) outperforms the majority voting method by approximately 3% and demonstrated significantly better performance than the other selection strategies. As the number of candidates increases, the selection task becomes more complex, and the advantages of our current method become even more pronounced.

In the aforementioned experiments, we have demonstrated that our selection model, based on the proposed candidate reorganization strategy, achieves performance that significantly surpasses that of the Majority Voting method. This improvement can be attributed not only to our selection strategy, but also to the strong discriminative capability of our selection model. We further evaluate the performance of the selection model. To minimize the impact of candidate quantity and order on the model, we randomly sample three different candidate quantities (2, 6, and 10) from the BIRD dev dataset and

TABLE IX
EVALUATION OF THE INDEPENDENT SQL SELECTION JUDGMENT ABILITY OF DIFFERENT SELECTION MODELS.

Methods	EX(%)
Qwen2.5-Coder-7B	62.84
Gemini-1.5-pro	66.30
GPT-4o	67.47
Our fine-tuned selection model	69.56

organize the candidates in random order for presentation to the model. As shown in Table IX, our fine-tuned selection model outperforms the baseline model and also exceeds the performance of advanced out-of-the-box models (such as GPT-4o and Gemini). Additionally, due to the smaller size of our fine-tuned model, it exhibits faster response times and improved instruction-following capabilities.

H. Analysis of Results from Different SQL Generators

In this section, we present the results of different generators that contribute to the final performance of XiYan-SQL. As previously mentioned, we ultimately utilize five distinct generators, with each generator generating two candidates corresponding to two schemas. We provide the individual evaluation results of all the different generators in Table X, along with their contributions to overall performance. Specifically, the generator $SQLG_1$ is a fine-tuned model based on multi-task training without any special formatting; the generator $SQLG_2$ is a fine-tuned model that incorporates multi-task data with complex writing patterns formatting; the generator $SQLG_3$ is a fine-tuned model utilizing multi-task data with standardized presentation styles. The variations in writing patterns and presentation styles are illustrated in Figure 3. The generator $SQLG_4$ is a fine-tuned model that integrates all multi-task data along with various formatted data; the generator $SQLG_5$ is based on the multiple examples ICL method using the GPT-4o model.

For clarity, “Avg. EX” here represents the average of the two generations produced by a single generator based on the two schemas. The “CR” (contribution ratio) refers to the proportion of candidates from each generator after removing all consistent candidates (which account for approximately 45%, where the shortest candidate query is selected). It is essential to note that this ratio ultimately derives from the output of the subsequent SQL selection process.

As observed, the generator $SQLG_1$ exhibits the best performance and contributes the most significantly, representing the optimal performance of a single fine-tuned generation model within our multi-task strategy. The generator $SQLG_4$, serving as a comprehensive mixed data model, also exhibits commendable performance. The generators $SQLG_2$ and $SQLG_3$ focus on diversifying generation pathways by introducing new format data. Meanwhile, the generator $SQLG_5$ leverages the ICL capabilities of GPT-4o to further enhance the upper bound of the candidate pool.

We further present the distribution of the results of each SQL generator within the overall candidate pool, as shown in Table XI. Here, “ R_n ” refers to the ratio of the number of

TABLE X

RESULTS OF DIFFERENT INDIVIDUAL GENERATORS IN ACHIEVING THE FINAL PERFORMANCE OF XiYan-SQL ON THE BIRD DEV. THESE RESULTS ARE DIRECTLY DERIVED FROM THE FINAL PIPELINE AND MAINTAIN ALL CONFIGURATIONS.

SQL Generators	Avg. EX(%)	CR(%)
SQLG ₁	69.34	51.90
SQLG ₂	66.05	5.27
SQLG ₃	68.15	6.13
SQLG ₄	68.50	30.55
SQLG ₅	64.51	6.13

TABLE XI

THE DISTRIBUTION OF CORRECT RESULTS FROM DIFFERENT SQL GENERATORS AT VARIOUS LEVELS ON BIRD DEV. THE R_n VALUES, RANGING FROM R_1 TO R_5 , REPRESENT THE SITUATION WHERE ONLY n GENERATORS HAVE GENERATED CORRECT OUTPUTS.

SQL Generators	R_1 (%)	R_2 (%)	R_3 (%)	R_4 (%)	R_5 (%)
SQLG ₁	1.42	2.31	3.73	13.24	76.44
SQLG ₂	1.96	2.13	3.29	10.76	76.44
SQLG ₃	1.33	3.47	3.64	12.27	76.44
SQLG ₄	1.60	3.47	3.11	12.62	76.44
SQLG ₅	2.22	3.20	2.58	9.78	76.44

correct queries generated by a specific generator to the total number of correct samples when only n generators generate correct queries across all samples. For example, the R_1 value of SQLG₁ is 1.42%, meaning that 1.42% of all correct results are uniquely generated by this generator. As the model with the highest accuracy, SQLG₁ exhibits relatively high values of R_3 and R_4 , suggesting that its generation is more robust and overall quality is higher. In contrast, SQLG₃ and SQLG₄ show a more concentrated distribution in the middle range, allowing them to explore diverse generations while maintaining quality. On the other hand, SQLG₂ and SQLG₅ demonstrate higher R_1 values, indicating greater diversity, and are also capable of generating some unique SQL queries that yield the correct answers.

Furthermore, these two tables also illustrate that, compared to the generator SQLG₁ with the highest accuracy, the other fine-tuned generators sacrifice a certain level of precision while introducing diversity. This represents the trade-off between quality and diversity inherent in multiple candidate methods. In summary, each generator plays an important role, and their performances along with their contributions to the candidate pool align with our design objectives for the multi-generator framework of XiYan-SQL. Due to limited resources, further investigation into additional training tasks and methods remains to be explored in future research.

I. Analysis of Cost Efficiency

The main advantage of the XiYan-SQL framework lies in its unique multi-generator ensemble approach, which enables the generation and utilization of multiple candidate SQL queries, thereby achieving optimal performance. However, this may raise concerns about the need to access the models multiple times and the associated inference latency when processing a single user query. To address this issue, we conduct a

TABLE XII

THE AVERAGE INFERENCE COSTS OF THE XiYan-SQL FRAMEWORK ON BIRD DEV.

Stage	Base model	Input/Output (tokens)	Inference latency(s)
Schema Filter (Retrieval)	GPT-4o	413.9/37	11.8
Schema Filter (Selection)	GPT-4o	3791.5/63.1	6.8
Multiple SQL Generation	Qwen2.5-Coder-32B	582.5/47.8	2.3
Multiple SQL Generation	GPT-4o	763.2/54.7	2.8
SQL Selection	Qwen2.5-Coder-7B	1052.3/1	0.8
All pipeline	Multiple models	6700/205	40.5

comprehensive discussion on the operational costs of XiYan-SQL.

During the training phase, fine-tuning the Qwen2.5-Coder-32B to obtain a SQL generation model requires approximately 45 GPU hours (using NVIDIA A100 80G GPUs), consuming a total of about 180 million tokens. Fine-tuning the Qwen2.5-Coder-7B to obtain a selection model takes about 15 GPU hours, with a total of approximately 50 million tokens consumed. Detailed information about the training phase costs is reported in Appendix II.

During the inference stage, Table XII summarizes the inference costs of XiYan-SQL. It is important to note that the above results reflect the average per-sample statistics of XiYan-SQL on the BIRD dev set. The SQL generation process may include certain self-refine operations, which have been accounted for. We utilize the official API to call GPT-4o, while all other fine-tuned models are deployed using minimal resources. Additionally, due to factors such as network conditions and deployment environments, the results may exhibit some normal fluctuations. In the Schema Filter module, most of the time cost during the retrieval phase arises from the model calls for keyword extraction and the multi-path similarity retrieval, while the time costs during the selection phase result from two iterations of column selection. During the multiple SQL generation phase, the main time consumption involves generating multiple candidate SQL queries, whereas the SQL selection phase only requires the selection of the final result. Ultimately, the end-to-end inference latency for generating 10 candidate SQL queries is approximately 40.5 seconds.

J. Study of Generalization

XiYan-SQL is a multi-step Text-to-SQL framework, and it is imperative to discuss its generalization performance across a broader range of out-of-domain datasets, as this is an important indicator of its ability to address SQL generation challenges. In this section, we elaborate on this aspect from two perspectives.

First, from the perspective of the BIRD benchmark, as presented in Table I, XiYan-SQL demonstrates an improvement of 2.3% on the black-box test set compared to the publicly available development set. In particular, this enhancement exceeds that of all previous SOTA methods, which have not surpassed a 2% improvement, with many falling below 1%.

Second, to further illustrate the generalization performance of our fine-tuned model and framework, we present two challenging multi-dialect evaluation sets constructed by our internal data team based on real-world scenarios. These two

TABLE XIII
COMPARISON OF EXECUTION ACCURACY PERFORMANCE ON THE TWO
OUT-OF-DOMAIN EVALUATION SETS.

Methods	PG-Set(%)	MYSQL-Set(%)
GPT-4o	45.93	55.87
Gemini-1.5-pro	45.07	51.33
Qwen2.5-Coder-32B	40.38	46.59
SFT generator (Qwen2.5-Coder-32B)	45.31	52.18
XiYan-SQL (Full pipeline)	50.04	58.05

evaluation sets are utilized to rigorously test the performance of XiYan-SQL on out-of-domain data before the product launch. We present the evaluation results as shown in Table XIII. The PG-Set is a dynamically maintained test set for the PostgreSQL dialect, currently comprising 1,051 question-SQL samples and four large databases. In contrast, the MYSQL-Set test set consists of 1,068 question-SQL samples in the MySQL dialect, along with four databases. These samples, used for evaluation, closely resemble real user business scenarios and encompass multiple dimensions where SQL writing is prone to errors, including date and time calculations (e.g., year-on-year, month-on-month, and trend-related questions), advanced functions, metric statistics, and so on.

Our single fine-tuned model surpasses the baseline (Qwen2.5-Coder-32B) by 5% and 5.6% on two different test sets, respectively, while achieving performance comparable to that of closed-source LLMs, such as GPT and Gemini. The complete XiYan-SQL framework achieves the performance of 50.04% on PG-Set and 58.05% on MYSQL-Set, representing the highest performance, thereby demonstrating robust generalization capabilities. We will also release de-identified versions of these evaluation datasets to contribute to future research in the field. Overall, XiYan-SQL not only focuses on academic research in the domain of SQL generation, but also provides product services to numerous clients in the industry.

V. DISCUSSION

As a comprehensive Text-to-SQL solution, XiYan-SQL offers several empirical insights based on our experiments, presented as follows:

- For schema linking, it is crucial to balance precision and recall. By ensuring that recall reaches a certain threshold (e.g., 80%), improving precision becomes significantly more beneficial for the overall result. This is evidenced by the various iterations of our Schema Filter.
- For SQL generators, incorporating relevant multiple tasks can significantly improve the performance of fine-tuned models. These tasks may be adjusted based on thematic considerations. Furthermore, regarding multiple candidate approaches, whether through multiple generators or a single generator with multi-path generation, it is crucial to prioritize the quality of individual generation before increasing diversity in outputs.
- For SQL selection, similar to previous studies [34], [42], we have also found that LLMs, whether fine-tuned or not, exhibit sensitivity to the order and organization of

candidates. Therefore, supplementing with additional prior knowledge can help enhance performance.

VI. CONCLUSION

This study presents a novel Text-to-SQL framework, XiYan-SQL, which focuses on generating high-quality and diverse candidate SQL queries, effectively selecting the best among them. Specifically, we introduce a Schema Filter module that employs multi-path retrieval and iterative selection to obtain multiple high-quality schemas. Then, we leverage the proposed multi-task and multi-format fine-tuning strategy to develop a variety of SQL generators, each with distinct advantages, enabling the generation of high-quality and diverse SQL candidates. Finally, we employ a selection model enhanced by a strategy based on model attention to identify the optimal SQL. XiYan-SQL framework achieves SOTA performance on the well-known Text-to-SQL benchmarks, including BIRD and Spider.

There are numerous opportunities for future research on the XiYan-SQL framework. Although the multiple fine-tuned models constructed by this method effectively alleviates the issue of candidate diversity present in traditional fine-tuning approaches, the inherent constraints mean that the diversity it can achieve is still limited. Therefore, we also introduce a generation method based on in-context learning. In the future, we plan to enhance the cognitive capabilities of the fine-tuned models by incorporating additional fine-tuning strategies, thereby further addressing this issue. Moreover, we are currently investigating the integration of a broader range of tasks into an all-in-one SQL model, with the aim of expanding the XiYan-SQL model to encompass more relevant scenarios beyond SQL generation.

REFERENCES

- [1] B. Li, Y. Luo, C. Chai, G. Li, and N. Tang, "The dawn of natural language to sql: Are we fully ready?" *arXiv preprint arXiv:2406.01265*, 2024.
- [2] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2024.
- [3] B. Wang, C. Ren, J. Yang, X. Liang, J. Bai, L. Chai, Z. Yan, Q.-W. Zhang, D. Yin, X. Sun *et al.*, "Mac-sql: A multi-agent collaborative framework for text-to-sql," *arXiv preprint arXiv:2312.11242*, 2024.
- [4] D. Lee, C. Park, J. Kim, and H. Park, "Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation," *arXiv preprint arXiv:2405.07467*, 2024.
- [5] M. Pourreza, H. Li, R. Sun, Y. Chung, S. Taleai, G. T. Kakkar, Y. Gan, A. Saberi, F. Ozcan, and S. O. Arik, "Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql," *arXiv preprint arXiv:2410.01943*, 2024.
- [6] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo *et al.*, "Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls," *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [7] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman *et al.*, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," *arXiv preprint arXiv:1809.08887*, 2018.
- [8] J. M. Zelle and R. J. Mooney, "Learning to parse database queries using inductive logic programming," in *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, ser. AAAI'96. AAAI Press, 1996, p. 1050–1055.

- [9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.
- [10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.
- [11] P. Yin, G. Neubig, W.-t. Yih, and S. Riedel, "TaBERT: Pretraining for Joint Understanding of Textual and Tabular Data," *arXiv e-prints*, p. arXiv:2005.08314, May 2020.
- [12] T. Yu, C.-S. Wu, X. V. Lin, B. Wang, Y. C. Tan, X. Yang, D. Radev, R. Socher, and C. Xiong, "Grappa: Grammar-augmented pre-training for table semantic parsing," in *International Conference on Learning Representations*, 2021. [Online]. Available: <https://arxiv.org/abs/2009.13845>
- [13] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-sql empowered by large language models: A benchmark evaluation," *arXiv preprint arXiv:2308.15363*, 2023.
- [14] M. Pourreza and D. Rafiei, "Din-sql: Decomposed in-context learning of text-to-sql with self-correction," *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [15] —, "Dts-sql: Decomposed text-to-sql with small large language models," *arXiv preprint arXiv:2402.01117*, 2024.
- [16] H. Li, J. Zhang, H. Liu, J. Fan, X. Zhang, J. Zhu, R. Wei, H. Pan, C. Li, and H. Chen, "Codes: Towards building open-source language models for text-to-sql," *Proceedings of the ACM on Management of Data*, vol. 2, no. 3, pp. 1–28, 2024.
- [17] J. Yang, B. Hui, M. Yang, J. Yang, J. Lin, and C. Zhou, "Synthesizing text-to-sql data from weak and strong llms," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024, pp. 7864–7875.
- [18] X. Dong, C. Zhang, Y. Ge, Y. Mao, Y. Gao, J. Lin, D. Lou *et al.*, "C3: Zero-shot text-to-sql with chatgpt," *arXiv preprint arXiv:2307.07306*, 2023.
- [19] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.
- [20] G. Team, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.
- [21] S. Talaie, M. Pourreza, Y.-C. Chang, A. Mirhoseini, and A. Saberi, "Chess: Contextual harnessing for efficient sql synthesis," *arXiv preprint arXiv:2405.16755*, 2024.
- [22] K. Maamari, F. Abubaker, D. Jaroslawicz, and A. Mhedhbi, "The death of schema linking? text-to-sql in the age of well-reasoned language models," *arXiv preprint arXiv:2408.07702*, 2024.
- [23] Y. Xie, X. Jin, T. Xie, M. Lin, L. Chen, C. Yu, L. Cheng, C. Zhuo, B. Hu, and Z. Li, "Decomposition for enhancing attention: Improving llm-based text-to-sql through workflow paradigm," *arXiv preprint arXiv:2402.10671*, 2024.
- [24] Z. Lin, Y. Liu, Z. Luo, J. Gao, and Y. Li, "Momq: Mixture-of-experts enhances multi-dialect query generation across relational and non-relational databases," 2024. [Online]. Available: <https://arxiv.org/abs/2410.18406>
- [25] J. Li, J. Ye, Y. Mao, Y. Gao, and L. Chen, "Loftune: A low-overhead and flexible approach for spark sql configuration tuning," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–14, 2025.
- [26] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, "Direct preference optimization: your language model is secretly a reward model," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS '23. Red Hook, NY, USA: Curran Associates Inc., 2023.
- [27] M. Datar, N. Immerlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual symposium on Computational geometry*, 2004, pp. 253–262.
- [28] C. Zhou, P. Liu, P. Xu, S. Iyer, J. Sun, Y. Mao, X. Ma, A. Efrat, P. Yu, L. Yu *et al.*, "Lima: Less is more for alignment," *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [29] L. Tunstall, E. E. Beeching, N. Lambert, N. Rajani, K. Rasul, Y. Belkada, S. Huang, L. V. Werra, C. Fourier, N. Habib, N. Sarrazin, O. Sansevero, A. M. Rush, and T. Wolf, "Zephyr: Direct distillation of LM alignment," in *First Conference on Language Modeling*, 2024. [Online]. Available: <https://openreview.net/forum?id=aKkAwZB6JV>
- [30] X. Liu, S. Shen, B. Li, P. Ma, R. Jiang, Y. Luo, Y. Zhang, J. Fan, G. Li, and N. Tang, "A survey of nl2sql with large language models: Where are we, and where are we going?" *arXiv preprint arXiv:2408.05109*, 2024.
- [31] Y. Zhang and Q. Yang, "A survey on multi-task learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 12, pp. 5586–5609, 2022.
- [32] T. Standley, A. Zamir, D. Chen, L. Guibas, J. Malik, and S. Savarese, "Which tasks should be learned together in multi-task learning?" in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 9120–9132. [Online]. Available: <https://proceedings.mlr.press/v119/standley20a.html>
- [33] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: <https://openreview.net/forum?id=IPL1NIMMrw>
- [34] C. Zheng, H. Zhou, F. Meng, J. Zhou, and M. Huang, "Large language models are not robust multiple choice selectors," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=shr9PXz7T0>
- [35] Y. Gao, Y. Liu, X. Li, X. Shi, Y. Zhu, Y. Wang, S. Li, W. Li, Y. Hong, Z. Luo, J. Gao, L. Mou, and Y. Li, "A preview of xiyao-sql: A multi-generator ensemble framework for text-to-sql," 2024. [Online]. Available: <https://arxiv.org/abs/2411.08599>
- [36] OpenAI, "Gpt-4 technical report," *CoRR*, vol. abs/2303.08774, 2023. [Online]. Available: <https://arxiv.org/abs/2303.08774>
- [37] Q. Team, "Qwen2.5 technical report," *arXiv preprint arXiv:2412.15115*, 2024.
- [38] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Dang *et al.*, "Qwen2.5-coder technical report," *arXiv preprint arXiv:2409.12186*, 2024.
- [39] G. Qu, J. Li, B. Li, B. Qin, N. Huo, C. Ma, and R. Cheng, "Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-sql generation," *arXiv preprint arXiv:2405.15307*, 2024.
- [40] Z. Li, X. Wang, J. Zhao, S. Yang, G. Du, X. Hu, B. Zhang, Y. Ye, Z. Li, R. Zhao *et al.*, "Pet-sql: A prompt-enhanced two-stage text-to-sql framework with cross-consistency," *arXiv preprint arXiv:2403.09732*, 2024.
- [41] Z. Wang, R. Zhang, Z. Nie, and J. Kim, "Tool-assisted agent on sql inspection and refinement in real-world scenarios," *arXiv preprint arXiv:2408.16991*, 2024.
- [42] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.