UCS-SQL: Uniting Content and Structure for Enhanced Semantic Bridging In Text-to-SQL

Zhenhe Wu[1,2†], Zhongqiu Li[2†], Jie Zhang[2], Zhongjiang He[2*], Jian Yang[1*], Yu Zhao[2], Ruiyu Fang[2], Bing Wang [1], Hongyan Xie[1], Shuangyong Song[2], Zhoujun Li[1*],
[1]Beihang University, [2]TeleAI, China Telecom Corp Ltd,
{wuzhenhe,jiaya,bingwang,by2406233,lizj}@buaa.edu.cn
{lizq48,zhangj157,hezj,zhaoy11,fangry,songshy}@chinatelecom.cn

Abstract

With the rapid advancement of large language models (LLMs), recent researchers have increasingly focused on the superior capabilities of LLMs in text/code understanding and generation to tackle text-to-SQL tasks. Traditional approaches adopt schema linking to first eliminate redundant tables and columns and prompt LLMs for SQL generation. However, they often struggle with accurately identifying corresponding tables and columns, due to discrepancies in naming conventions between natural language questions (NL) and database schemas. Besides, existing methods overlook the challenge of effectively transforming structure information from NL into SQL. To address these limitations, we introduce **UCS-SQL**, a novel text-to-SQL framework, uniting both content and structure pipes to bridge the gap between NL and SQL. Specifically, the content pipe focuses on identifying key content within the original content, while the structure pipe is dedicated to transforming the linguistic structure from NL to SQL. Additionally, we strategically selects few-shot examples by considering both the **S**QL **S**keleton and **Q**uestion **E**xpression (SS-QE selection method), thus providing targeted examples for SQL generation. Experimental results on BIRD and Spider demonstrate the effectiveness of our UCS-SQL framework.

1 Introduction

Text-to-SQL is the task of translating natural language queries into SQL statements, which has garnered extensive research attention and practical application in database querying (Qin et al., 2022; Sun et al., 2023). Based on the representations encoded by BERT-style pre-trained models (Liu et al., 2023), abstract syntax trees (Wu et al., 2023; Guo et al., 2019; Wang et al., 2020) and predefined query sketches (He et al., 2019) are implemented for query decoding. Further, some works extract generalized question-to-SQL patterns by training encoder-decoder models on text-to-SQL corpora (Hui et al., 2022; Li et al., 2023a,b; Zheng et al., 2022; Gao et al., 2024a). Inspired by the powerful capabilities of large language model (LLM) in handling complex reasoning (Wei et al., 2022; Yao et al., 2023), the recent studies have achieved promising results by developing prompt engineering for LLMs (Dong et al., 2023; Pourreza and Rafiei, 2023; Gao et al., 2024a). For example, implementing schema linking to eliminate redundant information, followed by guiding LLMs to generate SQL queries through in-context learning processes.

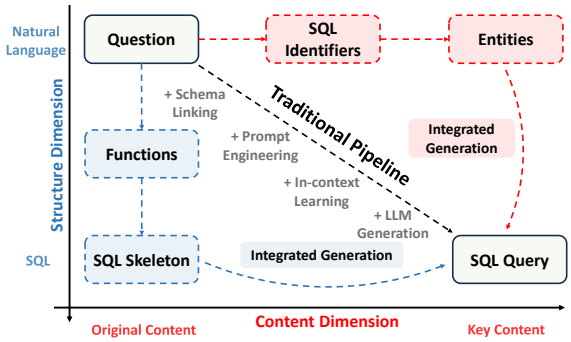

Figure 1: Traditional pipelines directly generate SQL query from NL question. We decompose text-to-SQL task into content and structure dimensions, enabling key content extraction and linguistic structure transformation to collaboratively enhance SQL generation.

The forefront researches adhere to the retrieve-then-generate paradigm. MCS-SQL (Lee et al., 2024) exploits LLMs' extensive sample libraries for schema linking and employs multiple prompts to elicit diverse LLM responses. RSL-SQL (Cao et al., 2024) integrates bidirectional schema linking, contextual information augmentation, binary selection strategy, and multi-turn self-correction to achieve state-of-the-art performance. However, the traditional retrieve-then-generate pipeline in text-to-SQL tasks still confronts two challenges. **(1)** Ex-

*Corresponding author.
†These authors contributed equally to this work.

Findings of the Association for Computational Linguistics: ACL 2025, pages 8156–8168
July 27 - August 1, 2025

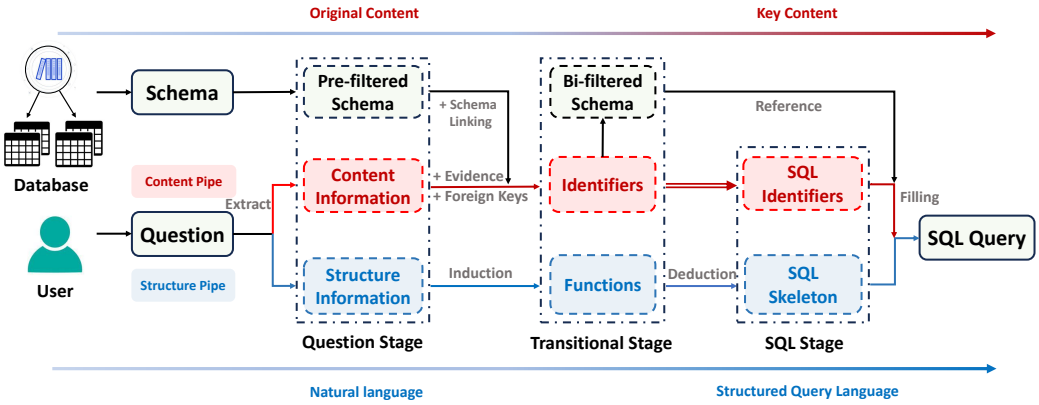

Figure 2: The framework of UCS-SQL, which consists of three stages uniting content and structure pipe. In question stage, LLM extracts content and structure information from the original question. Transitional stage facilitates the deduction from information to derive the candidate identifiers and functions. In SQL stage, they are combined to form the final query.

isting methods struggle with precise schema linking and identifier selection (e.g., column and table names) due to terminological mismatches between colloquial questions and schema identifiers. **(2)** Moreover, they seldom address the extraction and transformation of structural information from the natural language (NL) question to the SQL query, while bridging the gap of different linguistic structures is crucial for effective query generation. To address the aforementioned challenges, we plan to propose a more comprehensive deductive approach for in-context learning, emphasizing both content and structural dimensions. Specifically, in the content dimension, we deduce SQL identifiers step by step, thereby transforming the original content into key content. In the structure dimension, we derive the SQL skeleton from the question, facilitating the linguistic structure from NL to SQL. Figure 1 demonstrates the integration of question comprehension and SQL query generation from content and structure pipes.

In this paper, we introduce UCS-SQL , a three-stage text-to-SQL framework that integrates content and structure pipes to enhance semantic bridging. The content pipe extracts key information from the raw question, while the structure pipe converts NL into SQL syntax. Figure 2 illustrates the UCS-SQL framework. In the question stage, a LLM extracts content and structure information from the raw question. Then in the transitional stage, we prompt the LLM to transform them into SQL identifiers and functions. The content pipe integrates evidence and foreign keys to refine content information and performs bi-directional schema linking to identify relevant schema identifiers, while the structure pipe infers SQL functions based on the extracted structure information. Finally, in the SQL stage, we construct a SQL skeleton using the inferred functions and fill it with the identified identifiers to generate the final query. Additionally, we develop an SS-QE algorithm to select few-shot examples for the LLM, considering similarities in both SQL skeletons and question expressions. Comprehensive evaluations on the BIRD and Spider datasets demonstrate that UCS-SQL outperforms several baselines in terms of Valid Efficiency Score (VES) and Execution Accuracy (EX) across different difficulty levels. It achieves top-2 performance on all evaluation metrics on BIRD and top-3 performance on Spider.

To summarize, our contributions are as follows:

- We propose UCS-SQL, a three-stage text-to-SQL framework that unites content and structure pipes to respectively extract key content and transform linguistic structure, thereby collaboratively enhancing SQL query generation.
- We introduce SS-QE algorithm to select few-shot examples, which utilizes the structure information from UCS-SQL and takes into account the similarities in both SQL structure and question expression.
- Empirical evaluations indicate that UCS-SQL framework attains 3^{nd} place in all evaluation metrics on BIRD and Spider. Ablation experiments demonstrate that all three stages and both pipes in the UCS-SQL are crucial for performance enhancement.

2 Problem Definition

Text-to-SQL is the task of converting a natural language question Q into a correct SQL query Y. The database can be represented as $D = \{T_1, T_2...T_m\}$, m is the number of tables in the database. For $T = \{C_1, C_2...C_n\}$, C_i refers to columns in table T, n is the number of columns in the table. When dealing with complex database values, we can use external knowledge evidence K to support our model understand the inner relationship between question and database. The process could be formulated as follows:

$$Y = f\left(Q, D, K \mid \theta\right) \tag{1}$$

where $f(\cdot|\theta)$ represent a model with parameters θ.

3 UCS-SQL

3.1 Overview

We introduce UCS-SQL, a parallel text-to-SQL framework that integrates content and structure pipes to enhance semantic bridging. As illustrated in Figure 2, the framework comprises three stages: question, transitional, and SQL stages. (1) In the question stage, we prompt an LLM to extract content information and structure information from the original question. (2) The transitional stage acts as a bridge between the question and SQL stages. We combine the content information with the database schema to obtain candidate identifiers in the content pipeline and deduce SQL functions from the structure information in the structure pipeline. (3) In the SQL stage, we first generate an SQL skeleton based on the inferred SQL functions and the original question. Then, we fill the skeleton with the candidate identifiers to produce the final SQL query. Besides, a novel approach is proposed for selecting few-shot examples, considering similarities in both SQL structure and question expression. The following section provides a detailed introduction to UCS-SQL and the example selection method.

3.2 Stage 1: Question Stage

First, we perform pre-processing on the database schema prior by decomposing the original database into smaller sub-databases using the selector (Wang et al., 2023) to minimize interference from irrelevant tables and columns. Additionally, we maintain column descriptions for each column in the pre-filtered schema.

The question stage is designed to extract key content and structure from the original question.

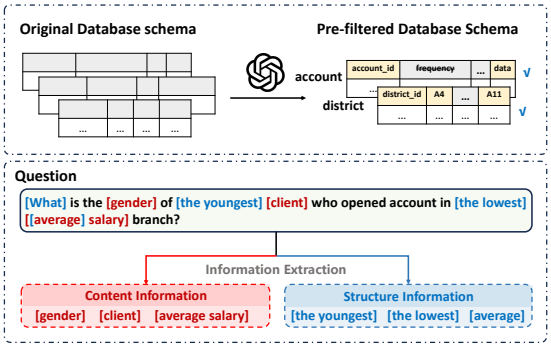

Figure 3: The question stage of UCS-SQL. The original database is pre-processed to filter out irrelevant tables and columns. Terms representing content and structure information are extracted from the raw question.

Specifically, we employ LLMs to identify significant content information (e.g., table names and column names) and structure information from the target question to facilitate SQL generation. In the subsequent operations, UCS-SQL proceeds via two parallel pipes, respectively handling SQL generation at the content and structure levels. Figure 3 illustrates the process of the question stage. We prompt the LLM to identify content information and structure information based on the original phrase of the question, which we highlight in red and blue colors. These identified elements are extracted and integrated to advance both parallel pipelines to the next stage. It is important to note that the LLM may identify the same word as both content and structure information, such as the word "average" shown in Figure 3. This dual identification aligns with our objective at the question stage, retaining sufficient relevant information. Further reasoning and selection will be conducted in subsequent stages.

3.3 Stage 2: Transitional Stage

The transitional stage is designed to transform the original information extracted from the question into SQL language elements. Figure 4 illustrates the process of this stage.

In the content pipe, we first integrate the content information with evidence to convert colloquial phrases into schema-compatible identifiers. Subsequently, we perform foreign key expansion to augment the candidate set of potential identifiers for SQL generation. Specifically, we prompt the LLM to identify relevant foreign keys and incorporate the corresponding identifiers into the candidate set. Next, we conduct bi-directional schema linking by combining the schema with the candidate set,

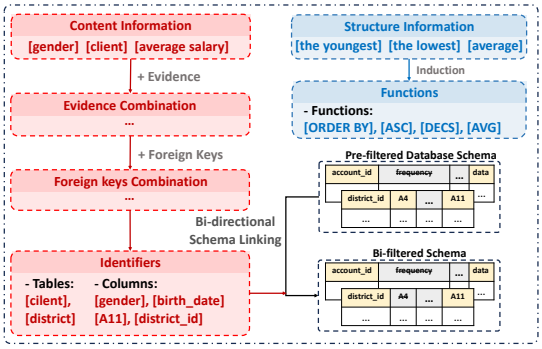

Figure 4: The transitional stage of UCS-SQL. Candidate identifiers and functions are obtained in the content and structure pipes. Bi-directional schema linking is performed with reference to the identifiers, yielding the bi-filtered schema.

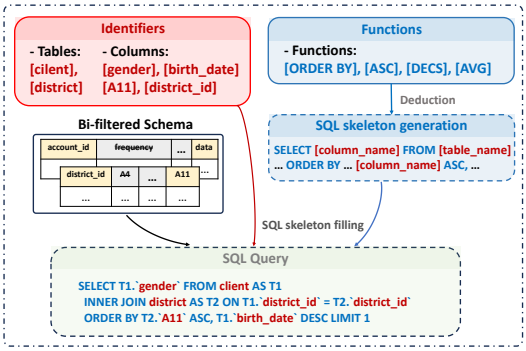

Figure 5: The SQL stage of UCS-SQL. The SQL skeleton is inferred from functions. The final SQL query is constructed by incorporating the identifiers into the SQL skeleton. The bi-filtered schema serves as a secondary reference to address cases where the identifiers are not correctly extracted.

which both deduces potential identifiers and filters out irrelevant columns from the schema. We match the content information with schema identifiers: if a close match with a table or column name is found, the corresponding identifier is retained. In cases where no direct match exists, the LLM searches for the most similar expression among column descriptions and retains the associated identifiers. Through careful analysis and deduction, we obtain a refined set of identifiers. For instance, the final identifiers reveal that column "A4" is irrelevant to the question, so we remove "A4" from the schema to arrive at the bi-filtered database schema.

In the structure pipe, we use the LLM to induce SQL functions based on the structure information extracted during the question stage. As shown in Figure 4, phrases such as "the youngest" and "the lowest" suggest the potential use of SQL functions like "ORDER BY", "ASC", and "DESC" while "average" implies the "AVG" function. In summary, we distill these insights into a final set of functions.

3.4 Stage 3: SQL Stage

In the SQL stage, we first create the SQL skeleton by referring to the candidate functions derived from the structure pipe. Subsequently, we integrate the identifiers from the content pipe with the bi-filtered schema to fill the generated skeleton. As a result, by unifying the structure and content pipes, we ultimately produce the final SQL query.

In Figure 5, we prompt the LLM to generate an SQL skeleton based on the constraints, question context, and all available structure information, with particular emphasis on the candidate functions from the structure pipeline. The LLM selects functions such as "ORDER BY", "ASC", and "DESC" to form the SQL skeleton. Next, we fill the skeleton using the identifiers from the content pipeline, focusing on the recommended identifiers. The LLM identifies "client" and "district" as table names and "gender", "birth_date", "A11", and "district_id" as column names to complete the final SQL query.

3.5 SS-QE Example Selection

We propose a novel SS-QE few-shot example selection approach, which consider the similarities in both SQL structure and question expression. We present the detailed algorithm of our selection method in Algorithm 1. We first extract final identifiers and final functions from all the question q_i in our training set Q following the process of question stage and transitional stage. F_i is the set of final functions, I_i is the set of final identifiers, while c_i is the complexity of q_i, which we define as the sum of final identifiers and final functions:

$$c_i = |I_i| + |F_i| \tag{2}$$

First, we evaluate the similarity between q_i and q in terms of SQL structure, using F_i and c_i as our basis. Specifically, samples with a larger intersection in their final functions and closer complexity values are considered to have a higher degree of SQL structure similarity. To filter these samples, we establish thresholds θ_1 and θ_2. Subsequently, we add each sample that satisfies these criteria to the set S_1, continuing this process until S_1 contains $2k$ samples. Next, we further utilize LLM to pick top-k examples from S_1 based on their similarity with q in question expression, Figure 6 shows the selection prompt in detail. As the result, we obtain

Algorithm 1: SS-QE Example selection

Input : All the questions Q in training set, target question q, the number of few-shot k, threshold θ_1 & θ_2

Output : Top-k questions S

$S_1 \leftarrow \emptyset$;
$I, F \leftarrow LLM_{InformationExtraction}(q)$;
$c \leftarrow |I| + |F|$;
foreach q_i *in* Q **do**
  $I_i, F_i \leftarrow LLM_{InformationExtraction}(q_i)$;
  $c_i \leftarrow |I_i| + |F_i|$;
  if $(F \cap F_i \geq \theta_1) \wedge (|c - c_i| < \theta_2)$ **then**
    Add q_i in S_1;
  end
  if $|S_1| = 2k$ **then**
    break;
  end
end
$S \leftarrow LLM_{SimilarQuestionExpression}(q, k, S_1)$;
return S;

k-shot examples that have similar SQL structure and question expression.

4 Experimental Settings

4.1 Datasets

BIRD (Li et al., 2023c) represents a cross-domain dataset that examines the impact of extensive database contents on text-to-SQL parsing. **Spider** (Yu et al., 2018) is a large-scale complex and cross-domain semantic parsing and text-to-SQL dataset. We report the statistics of datasets in appendix A.

4.2 Evaluation Metrics

Following BIRD (Li et al., 2023c), we utilize **execution accuracy (EX)** and **valid efficiency score (VES)** to evaluate text-to-SQL models. EX (Li et al., 2023c) is defined as the proportion of questions in the evaluation set for which the execution results of both the predicted and ground-truth inquiries are identical. VES (Li et al., 2023c) is designed to measure the efficiency of valid SQLs generated by models.

4.3 Baselines

DIN-SQL(Pourreza and Rafiei, 2023) and MAC-SQL(Wang et al., 2023) breaks down intricate queries into manageable sub-tasks.

Similar Question Expression Selection Prompt

Select three questions that are closest in question phrasing to the target question from the source questions and return the list which is contain the three questions ids. Format of the returned list is as follows:

[3, 672, 19]

target question: *{question}*

source questions: *{[source_questions]}*

returned list: *{result_ids}*

Figure 6: The prompt for selecting examples having similar question expression with target questions.

E-SQL(Caferoglu and Ulusoy, 2024) and CHESS(Talaei et al., 2024) seek to bridge the divide between natural language queries and database architectures. SQL-PaLM(Sun et al., 2023) and SuperSQL(Li et al., 2024a) utilize distinct prompting and fine-tuning methods for adapting large language models (LLMs) in SQL generation. TA-SQL(Qu et al., 2024) and CodeS(Li et al., 2024b) introduce strategies to reduce hallucinations in LLM-based SQL generation. DAIL-SQL(Gao et al., 2024a) is designed to tackle complex database environments. Additionally, methods based on multi-stage strategies are included, such as DTS-SQL(Pourreza and Rafiei, 2024) that employs two-stage fine-tuning, MAG-SQL(Xie et al., 2024) that adopts a multi-agent generative approach, and MCS-SQL(Lee et al., 2024) that utilizes multiple prompts and multiple-choice selection. MSc-SQL(Gorti et al., 2024) narrows the performance gap of smaller open-source models by sampling and comparing multiple SQL query results. RSL-SQL(Cao et al., 2024) achieve robust schema linking that maximize the benefits. CHASE-SQL (Pourreza et al., 2024) propose multiple chain-of-thought prompting methods and an online synthetic example generation technique. XiYan-SQL (Gao et al., 2024b) integrates the ICL approach to maximize the generation of high-quality and diverse SQL candidates.

5 Implementation Details

We use a single Tesla V100 GPU with 32 GiBs of memory on a server to restore intermediate data and execute SS-QE example selection algorithm, the retrieving time is about 3∼6 seconds for each sample. All the experiments utilize GPT-4-Turbo, the context window is 128000, the temperature is set to 0.1. We enable five threads to run UCS-SQL

Method	Model	EX				VES
		simple	moderate	challenging	total	
DIN-SQL	GPT-4	-	-	-	50.72	58.79
DAIL-SQL	GPT-4	-	-	-	54.76	56.08
DTS-SQL	DeepSeek-7B	-	-	-	55.80	-
TA-SQL	GPT-4	63.14	48.60	36.11	56.19	-
Codes	Codes-15B	-	-	-	58.47	59.87
SuperSQL	GPT-4	66.90	46.50	43.80	58.50	61.99
MAC-SQL	GPT-4	65.73	52.69	40.28	59.39	66.39
MAG-SQL	GPT-4	-	-	-	61.08	-
SQL-Palm	PaLM2	68.92	52.07	47.89	61.93	-
MCS-SQL	GPT-4	70.40	53.10	51.40	63.36	64.80
CHESS	Proprietary	-	-	-	65.00	-
E-SQL	GPT-4o	-	-	-	65.58	-
MSc-SQL	GPT-4o	72.00	**58.00**	49.00	65.60	-
RSL-SQL	GPT-4o	**74.38**	57.11	53.79	67.21	**70.32**
CHASE-SQL	Gemini 1.5	-	-	-	**73.01**	-
XiYan-SQL	-	-	-	-	**73.34**	-
UCS-SQL	DeepSeek	71.67	55.59	**54.11**	65.23	**68.49**
UCS-SQL	GPT-4o	**75.42**	**59.52**	**56.93**	**68.95**	**71.10**

Table 1: EX and VES on dev set of BIRD. UCS-SQL achieves top two performances in both VES and EX across datasetes of varying difficulty levels.

Method	Model	EX (dev)	EX (test)
DIN-SQL	GPT-4	82.8	85.3
DAIL-SQL	GPT-4	84.4	86.6
DTS-SQL	DeepSeek-7B	85.5	84.4
TA-SQL	GPT-4	85.0	-
MAC-SQL	GPT-4	**86.8**	82.8
MAG-SQL	GPT-4	85.3	85.6
MCS-SQL	GPT-4	**89.5**	**89.6**
CHESS	Openllms	-	87.2
MSc-SQL	GPT-4o	-	84.7
RSL-SQL	GPT-4o	-	87.9
CHASE-SQL	Gemini 1.5	-	87.6
XiYan-SQL	-	-	**89.6**
UCS-SQL	DeepSeek	86.5	87.7
UCS-SQL	GPT-4o	**87.3**	**88.0**

Table 2: UCS-SQL achieves top three performances in EX on dev and test set of Spider.

(approximately 200∼500 samples for each according to the size of dataset), it costs about 4∼6 hours to generate all results. The token consumption of 2-shot UCS-SQL on dev set for BIRD and Spider are approximately 5.29 million and 3.47 million in input, 0.9 and 0.5 million in output. We use all-MiniLM-L6-v2 model to get sentence embedding and calculate cosine similarity in the discussion. In SS-QE selection method, we set $\{\theta_1, \theta_2\}$ to {2,3}.

6 Results and Analysis

6.1 Overall Results

The overall results of all the baselines and our proposed UCS-SQL on BIRD and Spider are shown in Table 1 and Table 2. The experiment results indicate that our proposed UCS-SQL achieves better performance than several competitive baselines on the two datasets.

In Table 1, we report the performance of UCS-SQL and other competitive baselines on development set of BIRD. These recent methods have their own distinct characteristics and utilize different LLM models, such as GPT-4, GPT-4o, and DeepSeek. Experiments show that our proposed UCS-SQL method almost outperforms the first thirteen baselines by at least 3.35% on EX and by at least 4.71% on VES, only slightly trailing the concurrent work RSL-SQL. In addition, UCS-SQL performs better than RSL-SQL on samples of moderate and challenging difficulty, which indicates that UCS-SQL can address problems more specifically when dealing with samples of higher complexity. On the other hand, Table 2 shows the execution accuracy of UCS-SQL and other baselines on Spider. Our proposed UCS-SQL achieves excellent results of 87.3% and 88.0% on dev and test sets, surpassing the vast majority of baselines. In conclusion, UCS-SQL respectively achieves top-3 results in VES and EX across different difficulty on BIRD and Spider datasets, and ranked first on high-difficulty data, demonstrating the method's high efficiency in handling complex examples and its generalizability across different scenarios.

Method	EX			
	simple	moderate	challenging	total
Content & Structure Pipes Dimension (fixed 2-shot)				
w/o content pipe	69.81	54.98	**50.48**	63.58(↓ 1.24)
w/o structure pipe	**70.45**	54.54	49.77	63.78(↓ 1.04)
w/ both pipes	70.24	**58.25**	**50.48**	**64.82**
Three Stages Dimension (fixed 2-shot)				
w/o question & transitional stage	70.13	53.24	47.66	63.00(↓ 1.82)
w/o transitional stage	69.59	56.07	49.07	63.65(↓ 1.17)
w/o SQL stage	69.70	55.42	48.66	63.39(↓ 1.43)
w/ three stages	**70.24**	**58.25**	**50.48**	**64.82**
Few-shot Example Selection				
0-shot	67.45	53.89	44.14	61.24(↓ 7.71)
2-shot (fixed)	70.24	58.25	50.48	64.82(↓ 4.13)
2-shot (SS-QE)	71.99	58.81	52.89	66.28(↓ 2.67)
5-shot (SS-QE)	**75.42**	**59.52**	**56.93**	**68.95**

Table 3: Ablation study on dev set of BIRD. The three groups of experiments respectively compare the impact of two pipes, three stages, and different few-shot example selections on UCS-SQL.

6.2 Ablation Study

To evaluate the contributions of different components in UCS-SQL, we conducted a series of ablation studies using the BIRD dataset, which is a large-scale dataset with complex database structures. The experiments were divided into three groups: (1) assessing the roles of the content pipe and structure pipe, (2) analyzing the contributions of the three stages, and (3) comparing the effectiveness of the SS-QE example selection method. The results are summarized in Table 3.

Group 1: Content Pipe vs. Structure Pipe Removing either the content pipe or the structure pipe from the framework led to a decrease in execution accuracy across datasets of varying difficulty, with respective drops of 1.24% and 1.04%. The results indicate that integrating both content and structure information is crucial for guiding LLMs to generate SQL queries effectively.

Group 2: Contributions of the Three Stages The ablation studies also examined the necessity of the three stages in UCS-SQL. Removing the transitional stage or the SQL stage resulted in a decrease in execution accuracy by 1.17% and 1.43%, respectively. Additionally, removing the question stage further reduced performance by 0.65%. These findings demonstrate that each stage plays a vital role in the overall performance of the framework.

Group 3: SS-QE Example Selection Method The effectiveness of the SS-QE example selection method was evaluated by comparing the execution accuracy of 0-shot, fixed 2-shot, SS-QE-selected 2-shot, and SS-QE-selected 5-shot scenarios. The results showed a progressive improvement in performance, with a 1.46% increase from fixed 2-shot to SS-QE-selected 2-shot and an additional 2.67% improvement when increasing the number of shots from 2 to 5. This indicates that the SS-QE method effectively selects guiding examples that enhance the overall performance of the framework.

The ablation studies demonstrate that both pipes and all three stages in the UCS-SQL framework are essential for performance enhancement. This validates the rationality of integrating content and structure dimensions in a multi-stage framework for SQL generation. Additionally, the SS-QE example selection method further leverages the framework's capabilities by selecting few-shot examples that match the target question's function and complexity, thereby improving the overall performance.

7 Discussion

7.1 Effect of bi-filtered schema

This section examines the impact of bi-filtered schemas generated through the bi-directional schema linking process, using the BIRD dataset due to its large scale and complexity. We define the schema column count as the total number of columns across all tables in a schema. Figure 7 illustrates the schema column counts for each sample in the BIRD dev set. The "Overall column counts" represent raw database schemas, which consist of 11 distinct databases. The "Gold column counts" reflect columns used as identifiers in gold SQL queries, setting the performance limits for schema

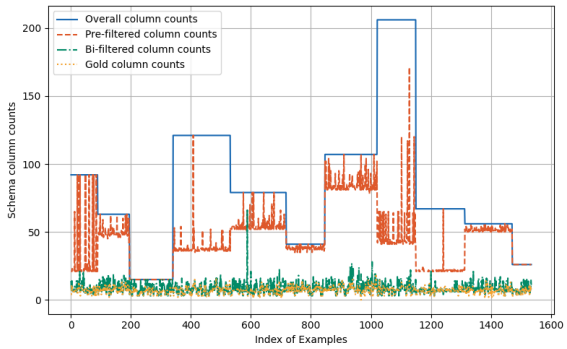

Figure 7: The schema column counts on dev set of BIRD. The bi-filtered column counts have significantly decreased compared to the pre-filtered column counts, and are slightly higher than the gold column counts.

Metric (%)	UCS-SQL			
	simple	moderate	challenging	total
Pre-filtered Schema				
Precision	11.23	15.78	20.56	13.26
Recall	97.33	95.11	96.19	96.42
F1 score	20.14	27.07	33.87	23.31
Bi-filtered Schema				
Precision	59.71	73.81	78.90	65.83
Recall	89.14	84.09	85.44	86.93
F1 score	71.51	78.62	82.04	74.92

Table 4: Evaluation of pre-filtered schema and bi-filtered schema on dev of BIRD.

linking. The "Pre-filtered column counts" denote schemas filtered using the method described earlier, while the "Bi-filtered column counts" represent our bi-filtered schemas. The pre-filtered schemas show inconsistent filtering effectiveness, with a significant gap compared to gold schemas. In contrast, bi-filtered schemas achieve substantial filtering, closely approximating gold schemas in terms of column counts.

To further evaluate the effectiveness of bi-filtered schemas, we calculated precision, recall, and F1 score using gold schemas as the standard. Table 4 presents the evaluation results. Pre-filtered schemas have high recall across datasets but poor precision and F1 score (13.26% and 23.31%, respectively). In contrast, bi-filtered schemas achieve a recall of 86.93%, slightly lower than pre-filtered schemas, but with significantly higher precision (65.83%) and F1 score (74.92%).

Bi-filtered schemas effectively eliminate irrelevant columns, closely matching gold schemas in column counts. The high precision, recall, and F1 score demonstrate the superior performance of bi-directional schema linking.

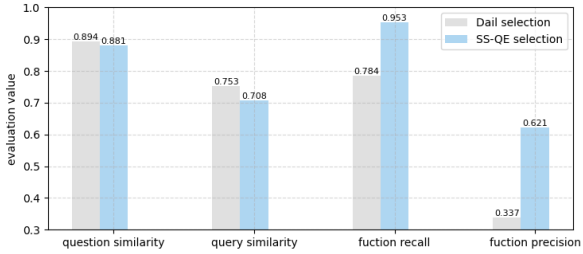

Figure 8: The evaluation statistics of UCS-SQL using DIAL or SS-QE selection method on dev of BIRD.

UCS-SQL (5-shot)	EX			
	simple	moderate	challenging	total
$w/DAIL_S$	73.74	58.94	51.07	67.21
$w/SS-QE$	75.42	59.52	56.93	68.95

Table 5: EX results of UCS-SQL with two example selection method on dev of BIRD.

7.2 Effect of SS-QE method

This study further examines the impact of the SS-QE selection method using the BIRD dataset. The DAIL Selection method, as reported in (Gao et al., 2024a), is a leading selection technique that pre-generates SQL queries and selects examples based on high similarity between masked questions and masked predicted SQL. We conducted experiments on UCS-SQL using both DAIL and SS-QE methods in a 5-shot setting. Table 5 shows that the SS-QE method outperforms DAIL.

To explore the underlying reasons, we analyzed the selected examples. As shown in Figure 8, the DAIL method slightly outperforms SS-QE in average masked question cosine similarity and masked query cosine similarity. However, the SS-QE method exhibits significantly higher function precision and recall. These results indicate that SS-QE more accurately identifies examples with analogous functions to the target sample, making it more suitable for guiding SQL query generation. Moreover, examples with similar complexity and consistent functions provide a broader selection pool compared to those based solely on high SQL skeleton similarity. Even when the sequence or placement of functions varies, they still offer guiding significance, which would likely be overlooked by similarity-based selection alone.

8 Related Work

Large language models (LLMs) have demonstrated significant advancements in various natural language processing (NLP) tasks (Wang et al., 2024a;

Li et al., 2024c; Touvron et al., 2023; OpenAI, 2023; Yang et al., 2024; Shao and Li, 2024; Wang et al., 2024b). Some researchers have leveraged LLMs in text-to-SQL tasks to further enhance performance. A critical aspect of this approach is the design and utilization of prompts, which directly influence the accuracy of SQL generation by guiding LLMs effectively. For example, some methods (Tai et al., 2023) try to improve the inference capabilities of LLMs using chain-of-thought prompting, including both the original chain-of-thought prompt and the least-to-most prompt. Further, a comprehensive analysis (Chang and Fosler-Lussier, 2023) of the impact of prompt construction across various settings in text-to-SQL tasks is studied. DAIL-SQL (Gao et al., 2024a) considers both questions and SQL queries to select few-shot examples, adopts an example organization strategy to balance quality and quantity, and utilizes code representation prompting for question representation. Besides, C3-SQL (Dong et al., 2023) and DIN-SQL (Pourreza and Rafiei, 2023) have introduced innovative frameworks for database simplification, query decomposition, and prompt engineering.

Researchers introduce schema linking to identify the database tables and columns associated with natural language queries and have proposed complex and integrated prompting engineering methods (Li et al., 2024d; Wu et al., 2025). MAC-SQL (Wang et al., 2023; Lee et al., 2024) centered on multi-agent collaboration can be utilized for more intricate data scenarios and a broader spectrum of error types for detection and correction. TA-SQL (Qu et al., 2024) and CodeS (Li et al., 2024b) introduce strategies to mitigate hallucinations in LLM-based SQL generation. CHESS (Talaei et al., 2024) enables more accurate schema linking by retrieving relevant information from database catalogs and database values. Another approach, MAG-SQL (Xie et al., 2024) features a multi-agent generative approach with soft schema linking and iterative Sub-SQL refinement. MSc-SQL (Gorti et al., 2024) mitigates the performance gap of smaller open-source models by sampling and comparing multiple SQL query results. RSL-SQL combines bidirectional schema linking, contextual information augmentation, binary selection strategy, and multiturn self-correction to achieve an efficient framework. But previous works are difficult to accurately retrieve the correct identifiers content from the schema and often overlook the understanding and reasoning of structure hints during SQL generation.

9 Conclusion

In this paper, we propose UCS-SQL, a three-stage text-to-SQL framework integrating a content pipe for key content extraction and a structure pipe for linguistic structure transformation. Through the three stages of UCL-SQL, the content and structure pipes jointly accomplish the generation from NL question to SQL query. Then, we introduce the SS-QE example selection method, which selects appropriate few-shot examples for target questions by taking into account the similarities in both SQL structure and question expression. The experimental results on the BIRD and Spider datasets demonstrate that UCS-SQL achieves superior performance in EX and VES, and substantiate the effectiveness of both the content and structure pipes. The integration of content and structure pipes will guide future research to jointly consider the text-to-SQL generation process from both dimensions.

Limitations

UCS-SQL is a pipeline based approach, it will be subjected to cascading error effect. An intelligent bakctracking approach to reflect and correct the intermediate stage outputs can make the system more robust, which we will design in the future work.

Ethics Statement

In this work, all of the datasets, models, code and related documents are not associated with any ethical concerns.

Acknowledgments

This work was partially supported by the National Natural Science Foundation of China (Grant Nos. 62276017, 62406033, U1636211, 61672081), and the State Key Laboratory of Complex& Critical Software Environment (Grant No. SKLCCSE-2024ZX-18).

References

Hasan Alp Caferoglu and Özgür Ulusoy. 2024. E-SQL: direct schema linking via question enrichment in text-to-sql. *CoRR*, abs/2409.16751.

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. RSL-SQL: robust schema linking in text-to-sql generation. *CoRR*, abs/2411.00073.

Shuaichen Chang and Eric Fosler-Lussier. 2023. How to prompt llms for text-to-sql: A study in zero-shot, single-domain, and cross-domain settings. *CoRR*, abs/2305.11853.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: zero-shot text-to-sql with chatgpt. *CoRR*, abs/2307.07306.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024a. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145.

Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. 2024b. Xiyan-sql: A multi-generator ensemble framework for text-to-sql. *CoRR*, abs/2411.08599.

Satya Krishna Gorti, Ilan Gofman, Zhaoyan Liu, Jiapeng Wu, Noël Vouitsis, Guangwei Yu, Jesse C. Cresswell, and Rasa Hosseinzadeh. 2024. Msc-sql: Multi-sample critiquing small language models for text-to-sql translation. *CoRR*, abs/2410.12916.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4524–4535. Association for Computational Linguistics.

Pengcheng He, Yi Mao, Kaushik Chakrabarti, and Weizhu Chen. 2019. X-SQL: reinforce schema representation with context. *CoRR*, abs/1908.08113.

Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Yanyang Li, Bowen Li, Jian Sun, and Yongbin Li. 2022. S^2sql: Injecting syntax to question-schema interaction graph encoder for text-to-sql parsers. In *Findings of the Association for Computational Linguistics: ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 1254–1262. Association for Computational Linguistics.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2024. MCS-SQL: leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *CoRR*, abs/2405.07467.

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024a. The dawn of natural language to SQL: are we fully ready? [experiment, analysis & benchmark]. *Proc. VLDB Endow.*, 17(11):3318–3331.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. RESDSQL: decoupling schema linking and skeleton parsing for text-to-sql. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023,*, pages 13067–13075. AAAI Press.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024b. Codes: Towards building open-source language models for text-to-sql. *Proc. ACM Manag. Data*, 2(3):127.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023,*, pages 13076–13084. AAAI Press.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023c. Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Xiang Li, Yiqun Yao, Xin Jiang, Xuezhi Fang, Chao Wang, Xinzhang Liu, Zihan Wang, Yu Zhao, Xin Wang, Yuyao Huang, Shuangyong Song, Yongxiang Li, Zheng Zhang, Bo Zhao, Aixin Sun, Yequan Wang, Zhongjiang He, Zhongyuan Wang, Xuelong Li, and Tiejun Huang. 2024c. Tele-flm technical report. *CoRR*, abs/2404.16645.

Zhongqiu Li, Zhenhe Wu, Mengxiang Li, Zhongjiang He, Ruiyu Fang, Jie Zhang, Yu Zhao, Yongxiang Li, Zhoujun Li, and Shuangyong Song. 2024d. Scalable database-driven kgs can help text-to-sql. In *Proceedings of the ISWC 2024 Posters, Demos and Industry Tracks:*, volume 3828 of *CEUR Workshop Proceedings*.

Shixuan Liu, Chen Peng, Chao Wang, Xiangyan Chen, and Shuangyong Song. 2023. icsberts: Optimizing pre-trained language models in intelligent customer service. In *INNS DLIA@IJCNN 2023, Gold Coast, Australia, 23 June 2023*, volume 222 of *Procedia Computer Science*, pages 127–136. Elsevier.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan Ö. Arik. 2024. CHASE-SQL: multi-path reasoning and preference optimized candidate selection in text-to-sql. *CoRR*, abs/2410.01943.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: decomposed text-to-sql with small large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 8212–8220. Association for Computational Linguistics.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022. A survey on text-to-sql parsing: Concepts, methods, and future directions. *CoRR*, abs/2208.13629.

Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before generation, align it! A novel and effective strategy for mitigating hallucinations in text-to-sql generation. In *Findings of the Association for Computational Linguistics, ACL 2024,*, pages 5456–5471. Association for Computational Linguistics.

Jiawei Shao and Xuelong Li. 2024. AI flow at the network edge. *CoRR*, abs/2411.12469.

Ruoxi Sun, Sercan Ö. Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2023. Sql-palm: Improved large language model adaptation for text-to-sql. *CoRR*, abs/2306.00739.

Chang-Yu Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. Exploring chain of thought style prompting for text-to-sql. In *EMNLP 2023, Singapore, December 6-10, 2023*, pages 5376–5393. Association for Computational Linguistics.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: contextual harnessing for efficient SQL synthesis. *CoRR*, abs/2405.16755.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

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*, pages 7567–7578. Association for Computational Linguistics.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. MAC-SQL: A multi-agent collaborative framework for text-to-sql. *CoRR*, abs/2312.11242.

Zihan Wang, Xinzhang Liu, Shixuan Liu, Yitong Yao, Yuyao Huang, Zhongjiang He, Xuelong Li, Yongxiang Li, Zhonghao Che, Zhaoxi Zhang, Yan Wang, Xin Wang, Luwen Pu, Huihan Xu, Ruiyu Fang, Yu Zhao, Jie Zhang, Xiaomeng Huang, Zhilong Lu, Jiaxin Peng, Wenjun Zheng, Shiquan Wang, Bingkai Yang, Xuewei He, Zhuoru Jiang, Qiyi Xie, Yanhan Zhang, Zhongqiu Li, Lingling Shi, Weiwei Fu, Yin Zhang, Zilu Huang, Sishi Xiong, Yuxiang Zhang, Chao Wang, and Shuangyong Song. 2024a. Telechat technical report. *CoRR*, abs/2401.03804.

Zihan Wang, Yitong Yao, Li Mengxiang, Zhongjiang He, Chao Wang, Shuangyong Song, et al. 2024b. Telechat: An open-source billingual large language model. In *Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing (SIGHAN-10)*, pages 10–20.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Hefeng Wu, Yandong Chen, Lingbo Liu, Tianshui Chen, Keze Wang, and Liang Lin. 2023. Sqlnet: Scale-modulated query and localization network for few-shot class-agnostic counting. *CoRR*, abs/2311.10011.

Zhenhe Wu, Zhongqiu Li, Mengxiang Li, Jie Zhang, Zhongjiang He, Jian Yang, Yu Zhao, Ruiyu Fang, Yongxiang Li, Zhoujun Li, and Shuangyong Song. 2025. MR-SQL: multi-level retrieval enhances inference for llm in text-to-sql. *DASFAA*.

Wenxuan Xie, Gaochen Wu, and Bowen Zhou. 2024. MAG-SQL: multi-agent generative approach with soft schema linking and iterative sub-sql refinement for text-to-sql. *CoRR*, abs/2408.07930.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *ICLR 2023, Kigali, Rwanda, May 1-5, 2023*.

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*, pages 3911–3921. Association for Computational Linguistics.

Yanzhao Zheng, Haibin Wang, Baohua Dong, Xingjun Wang, and Changshan Li. 2022. HIE-SQL: history information enhanced network for context-dependent text-to-sql semantic parsing. In *Findings of the Association for Computational Linguistics: ACL 2022,*, pages 2997–3007. Association for Computational Linguistics.

Datasets	Train	Dev	Test	DB	Table/DB	Row/DB
BIRD	9,428	1,534	1,789	95	7.3	549k
Spider	8,659	1,034	2,147	200	5.1	2k

Table 6: The statistics of BIRD and Spider datasets.

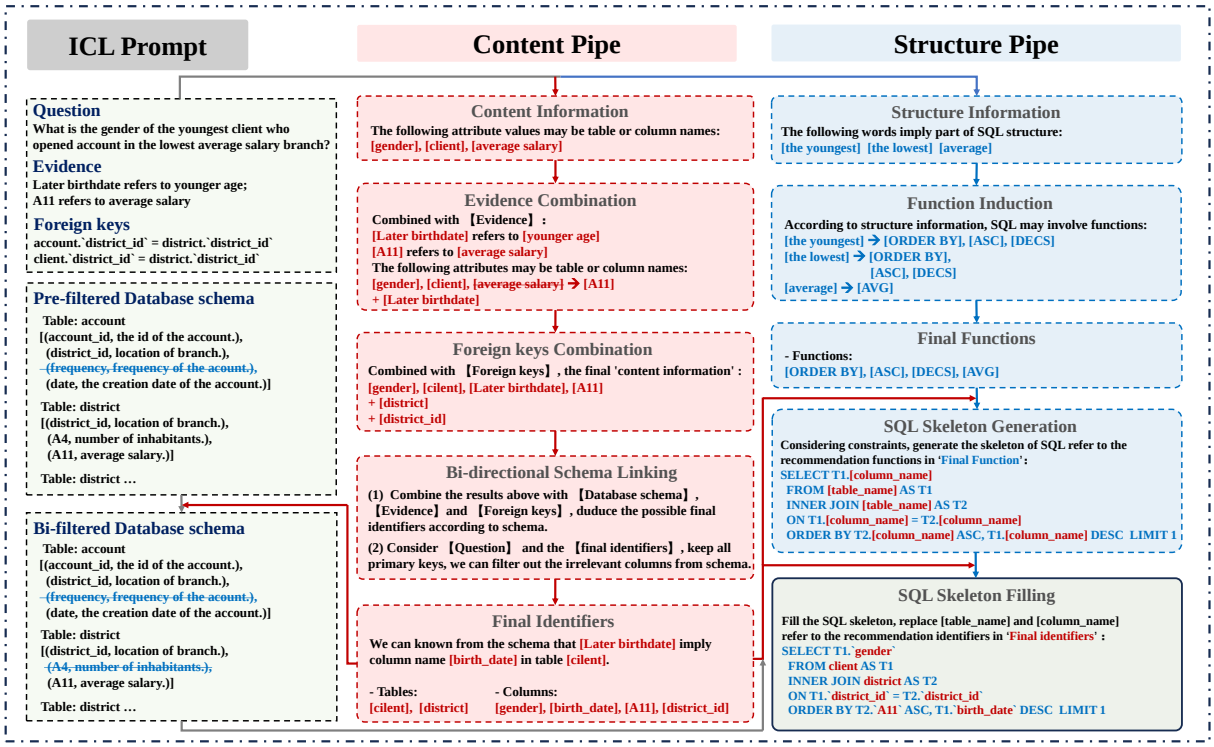

Figure 9: Detailed prompt design for UCL-SQL. The left column displays the original question and database information. Content and structure information are extracted from the question to activate the content and structure pipes. Identifiers and SQL skeletons are generated at the end of each pipe and combined to form the final SQL query.

A Statistics of datasets.

Table 6 shows the statistics of two datasets.

B Case Study

In this section, we elucidate the UCS-SQL framework through a detailed examination of an in-context learning prompt, as depicted in Figure 9. The process commences with the extraction of both content and structural information from the original question.

Within the content pipe, the LLM discerns that the natural language phrase "average salary" corresponds to the schema identifier "A11". Consequently, "average salary" is mapped to "A11". Additionally, the mention of "Later birthdate" prompts its provisional inclusion in the candidate set. Subsequently, we engage in the Foreign Keys Combination phase. Since "A11" is associated with the "district" table and "gender" with the "client" table, a foreign key linking these two tables is essential for SQL query generation. This necessitates the inclusion of "district" and "district_id" from the equivalence "client.'district_id' = district.'district_id'" into our candidate set. Advancing to the Bi-directional schema linking phase, we identify "gender", "A11", and "client" within the schema. The schema's description of "birth_data" (birth date) is recognized as the closest match to "Later birthdate", leading the LLM to select "birth_data" as a substitute. Ultimately, the final identifiers consist of the tables "client" and "district", and the columns "gender", "birth_date", "A11", and "district_id". Thereafter, the LLM integrates these final identifiers, refining the pre-filtered schema by eliminating the redundant column "A4", obtaining the bi-filtered schema.

In the structure pipe, the LLM initially infers candidate functions “ORDER BY”, “ASC”, “DESC”, and “AVG” from the structure information “the youngest”, “the lowest” and “average”. Integrating these with the query and constraints, the LLM constructs the SQL skeleton. Ultimately, with reference to the bi-filtered schema, the LLM populates the SQL skeleton with the candidate table names and column names from the content pipe, thereby completing the final SQL query.