DSQG-Syn: Synthesizing High-quality Data for Text-to-SQL Parsing by Domain Specific Question Generation

Anonymous ACL submission

Abstract

Synthetic data has recently proven effective in enhancing the accuracy of Text-to-SQL parsers. However, existing methods generate SQL queries first by randomly sampling tables and columns based on probability and then synthesize natural language questions (NLQs). This approach often produces a large number of NLQ-SQL pairs that are irrelevant to the target domain and inconsistent in query intent, significantly diminishing the fine-tuning effectiveness of LLMs. In this paper, we introduce DSQG-Syn, a novel text-to-SQL data synthesis framework that based on domain-specific question generation. Specifically, we design a question generation method that creates domain-relevant questions based on predefined question types, ensuring coverage of major SQL operations. Guided by these questions, we synthesize NLQ-SQL pairs that are both domain-relevant and intent-consistent. To further enhance data quality, we filter out noisy samples from the generated pairs. When popular open-source LLMs are fine-tuned on our high-quality synthesized dataset, they achieve significant accuracy improvements, surpassing the performance of closed-source LLM-based approaches. Moreover, we demonstrate that our method outperforms existing state-of-the-art (SOTA) data synthesis techniques.

1 Introduction

Text-to-SQL parsing (Li et al., 2024; Katsogiannis-Meimarakis and Koutrika, 2023) involves converting natural language questions (NLQs) into SQL queries, enabling non-experts to interact with databases through natural language. The growing availability of open-source LLMs (Bi et al., 2024; Yang et al., 2024a; Roziere et al., 2023) has garnered significant attention, as these models exhibit capabilities comparable to their closed-source counterparts across various natural language processing (NLP) tasks. Building on these advancements, we assess the performance of various open-source LLMs on the Text-to-SQL task to determine their feasibility as practical alternatives. However, our results reveal a considerable performance gap between open-source and closed-source models. Specifically, the widely adopted open-source model DeepSeek-Coder 33B achieves an accuracy rate that is 20% lower than that of GPT-4o on the BIRD benchmark (Li et al., 2024), as shown in Table 4 and 5.

Synthesis method in (Hu et al.,2023) Domain irrelevant and meaningless

Question: *What is the sum of the difference between the height ID in the height information and the Elite ID in the player information multiplied by the height of the player for players from Sweden?*

SQL: SELECT SUM((T1.height_id - T2.ELITEID) * T2.height) FROM height_info AS T1 JOIN PlayerInfo AS T2 ON T2.height = T1.height_id WHERE T2.nation = 'Sweden'

Our DSQG-Syn

Question: *Show the player with the most points in Playoffs season 2000-2001* √

SQL: SELECT PlayerName FROM PlayerInfo p JOIN SeasonStatus s ON p.ELITEID = s.ELITEID WHERE s.SEASON = '2000-2001' AND s.GAMETYPE = 'Playoffs' ORDER BY s.P DESC LIMIT 1

Figure 1: Synthetic examples from (Hu et al., 2023) and DSQG-Syn.

Enhancing the text-to-SQL capabilities of open-source LLMs through supervised fine-tuning (SFT) remains challenging due to the high cost of acquiring text-to-SQL data, which relies heavily on manual expert annotations. To address this issue, there is growing interest in leveraging synthetic data to improve downstream performance. Recent approaches (Hu et al., 2023; Zhang et al., 2024; Yang et al., 2024b; Awasthi et al., 2022; Wang et al., 2021) typically follow a two-stage process: first, they randomly generate SQL queries containing various operations based on probabilistic sampling; next, a SQL-to-text generator is used to compose corresponding NLQs. However, we find that these methods often produce many domain-irrelevant and nonsensical NLQs, as illustrated in Figure 1. While

the SQL queries themselves are executable, the use of randomly synthesized SQL—characterized by independent probabilistic column sampling and arbitrary table joins—results in illogical outputs that diminish the quality of NLQs. An alternative reversed pipeline, proposed in (Yang et al., 2021), employs an entity-to-question model to generate NLQs first, followed by a text-to-SQL parser to generate the corresponding SQL queries. While this method ensures that the generated NLQs remain relevant to the domain, it struggles to maintain intent consistency between the SQL queries and their NLQ counterparts due to the inherent limitations of the text-to-SQL parser's accuracy. The performance of LLMs fine-tuned on datasets containing large numbers of NLQ-SQL pairs with domain irrelevance and intent inconsistency can degrade significantly.

To address these challenges, we propose a novel NLQ-SQL pair synthesis framework, DSQG-syn, guided by domain-specific question generation. To ensure comprehensive NLQ coverage, we define nine question types encompassing major SQL operations. Domain-relevant NLQs are generated by prompting with the database schema, domain-specific keywords, and predefined question types. To maintain domain relevance and query intent consistency between NLQs and their corresponding SQL queries, we introduce a question-guided NLQ-SQL synthesis method begins by retrieving the relevant tables and columns through schema linking. For each NLQ, we generate multiple SQL skeletons, synthesize SQL queries using the identified schema and skeleton, and regenerate the NLQ from the SQL query to ensure alignment between NLQ intent and SQL logic. To further improve data quality, we introduce a semantic similarity-based optimization method to filter out irrelevant NLQ-SQL pairs. Extensive experiments on three real-world datasets evaluate the effectiveness of our framework, with LLMs fine-tuned on DSQG-Syn data achieving superior performance compared to SOTA methods.

The main contributions of this paper are as follows:

1. We propose a novel data synthesis framework for Text-to-SQL parsing that generates data directly from database schemas without relying on pre-existing NLQ-SQL pairs. In contrast to previous methods, our framework first synthesizes NLQs and subsequently constructs corresponding SQL-NLQ pairs. We believe this approach offers a new reference point for the practical application of Text-to-SQL data synthesis in real-world scenarios.

2. We propose a question generation method that creates domain-specific questions from structured databases based on nine predefined question types, covering all SQL operations.

3. We introduce a question-guided NLQ-SQL pair synthesis method that ensures the domain relevance of NLQs in the synthesized data and maintains intent consistency between NLQs and their corresponding SQL queries.

4. We conduct extensive experiments on four real-world datasets to validate the effectiveness of our approach. The results show that LLMs fine-tuned on data synthesized by our method outperform those fine-tuned with existing state-of-the-art (SOTA) methods.

2 Related Work

Early efforts in NLQ-SQL synthesis for text-to-SQL parsing (Yu et al., 2018, 2019; Wang et al., 2020b,a) relied on hand-crafted templates that often depended on predefined rules or grammars. These approaches required significant human effort and were difficult to scale. To automate the synthesis process, recent studies (Hu et al., 2023; Zhang et al., 2024; Yang et al., 2024b; Awasthi et al., 2022; Wang et al., 2021; Wu et al., 2021; Zhong et al., 2020) introduced SQL-to-NL pipelines. These methods first extract SQL templates from existing data, generate SQL queries using the templates, and then employ SQL-to-NL models to generate natural language questions. For instance, (Hu et al., 2023) employs template synthesis strategies with strong typing constraints, key relationship preservation, and schema-distance-weighted column sampling to enhance the quality of synthetic data. ScienceBenchmark (Zhang et al., 2024) proposed a four-step pipeline for complex data synthesis in specific domains: SQL queries, questions, and SQL abstract syntax trees are sampled from seed data, SQL queries are then generated, followed by the generation of natural language questions through a SQL-to-NL system, and finally, the questions are verified. While these methods automate the synthesis process and reduce manual effort, the generated data often contains a substantial number of domain-irrelevant and nonsensical NLQ-SQL pairs.

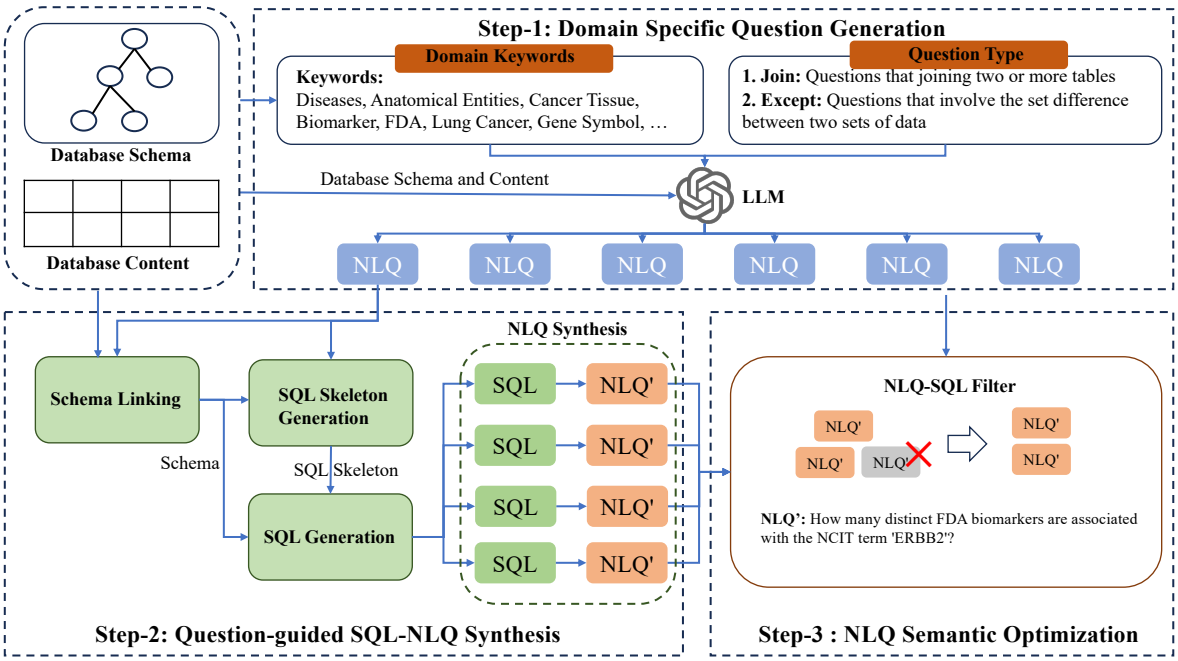

Figure 2: The overview of our DSQG-Syn framework, which comprises three steps: (i) Domain specific question generation, responsible for generating domain-relevant questions from database schema and content, (ii) Question-guided SQL-NLQ synthesis, which synthesis SQL-NLQ pairs according to the generated questions, and (iii) NLQ semantic optimization, which filter out samples that do not match the original question.

To improve domain relevance, (Yang et al., 2021) proposed an NL-to-SQL parser-based pipeline. This approach first generates natural language questions based on the database schema, followed by generating corresponding SQL queries. However, the effectiveness of this method is limited by the accuracy of the NL-to-SQL parser, which may result in inconsistencies between the intent of the NLQ and the corresponding SQL query.

In contrast to the above approaches, our DSQG-Syn framework eliminates the need for pre-existing NLQ-SQL pairs as seed data. It directly generates domain-specific questions from the database schema, retrieves relevant schema components based on the generated questions to construct potential SQL queries, subsequently generates natural language questions from the SQL queries, and finally optimizes the questions. This approach ensures both the domain relevance of the NLQs and the query intent consistency between the NLQ and SQL pairs.

3 Methodology

DSQG-Syn is composed of three steps, as illustrated in Figure 2. The output of the first step is a set of domain specific questions $Q = \{q_1, q_2, .., q_N\}$, where N is the number of nature language questions (NLQ). These questions are not only relevant to the domain but also aim to encompass as many potential query variations as possible within the field. In the second step, SQL-NLQ pairs are synthesized based on the generated questions. Unlike previous approaches, we avoid directly synthesizing SQL from the question, which would be constrained by the accuracy of the Text-to-SQL model. Instead, we first select the relevant schema and generate an SQL skeleton for each question. Then, using the selected schema and skeleton, we generate the corresponding SQL queries and synthesize the natural language questions for each SQL query. This results in a new set of questions, $Q^{'} = \{q_1^{'}, q_2^{'}, ..., q_M^{'}\}$, where M represents the number of questions, with $M > N$. The third step focuses on optimizing the NLQs. Questions with significant semantic differences between Q and $Q^{'}$ are filtered out using a semantic similarity comparison method.

3.1 Domain Specific Question Generation

The aim of this section is to generate domain-specific, comprehensive natural language questions (NLQs). A common approach is to prompt LLMs using the database schema and content directly. However, this approach presents challenges in con-

trolling the scope and granularity of the generated questions, often leading to a high number of random or irrelevant outputs.

To address this issue, we first extract domain-specific keywords from the database schema and content. These keywords serve as constraints, ensuring that the synthesized questions remain relevant to the domain. Additionally, to guarantee comprehensive coverage of SQL operations, we define nine distinct question types, aligned with the SQL operations specified in (Eyal et al., 2023), as outlined in Table 1. It is important to note that a single question can belong to multiple question types. For instance, the question "*What are the gene symbols and expression scores for genes associated with anatomical entities described as 'gray matter'?*" may involve both the *Join* and *Filter* types.

To ensure that the synthesized questions encompass all tables within the database, we propose a question generation method that systematically traverses the entire database schema. This approach constructs a connected subgraph based on the primary and foreign key relationships, where each edge represents a primary-foreign key pair, and each node represents a table. Starting from a randomly selected leaf node, we iteratively select k connected tables, with k determined by the number of allowed joins in the SQL query. The detailed question generation process is outlined in Algorithm 1. The prompt is constructed using domain-specific keywords, question types, and the selected database schema, leading to the synthesis of the natural language question Q. In this algorithm, the combination of nine predefined question types and the schema traversal method ensures that all relevant questions involving the database are synthesized.

3.2 Question-guided NLQ-SQL Synthesis

The objective of this section is to synthesize NLQ-SQL pairs that maintain consistent intent based on Q. A straightforward approach to generating NLQ-SQL pairs is to use the NLQ Q as input for Text-to-SQL models (Yang et al., 2021) to produce the corresponding SQL query. However, this approach does not guarantee the accuracy of the generated SQL query. An alternative method involves generating SQL queries based on probabilistic schema selection, followed by generating NLQs from the SQL. While this ensures consistency between NLQ-SQL pairs, it often results in a large number of domain-independent SQL queries and overly mechanical NLQs.

Algorithm 1 Question Generation Algorithm

Input: Question type T, database schema S, database content C
Output: Question set Q

```
Constrcut a set of connected subgraph G = {G_1, G_2, ..., G_n} from S based on primary and foreign keys
Q ← ∅
for each G_i in G do
    for each node g in G_i do
        g_k ← Select k connected tables with g from G_i
        D_g ← Prompt_KG(g_k)
        Q ← Prompt_QG(g_k, D_g, T, C)
    end for
end for
return Q
```

To address these limitations, we propose a question-guided NLQ-SQL synthesis method. This method neither generates SQL queries directly from NLQs nor selects database schema randomly to produce arbitrary SQL queries followed by NLQs. Instead, it constrains SQL query generation based on the NLQ. Given a generated question, we first utilize schema linking techniques to select the relevant tables and columns. For each question $q_i \in Q$, we generate several potential SQL skeletons. Finally, for each SQL skeleton, we synthesize corresponding SQL queries based on the selected schema, and subsequently generate a refined NLQ $Q^{'}$ from the SQL queries.

Schema Linking. Unlike the probabilistic schema selection approach used for SQL synthesis in (Hu et al., 2023), we employ schema linking to select the minimal schema relevant to the question. For each question $q_i \in Q$, and following the approach of MAC-SQL (Wang et al., 2023), the selected schema s_i is determined as $s_i \leftarrow Prompt_{SL}(q_i, S)$, where $Prompt_{SL}$ denotes the schema linking prompt, and S represents the database schema.

SQL Skeleton Generation. As illustrated in Figure 3, we observe that the accuracy of generating SQL skeletons is significantly higher than that of generating SQL queries. In this work, we define a SQL skeleton as a SQL query without the associated schema, as shown in Table 2. For each question, we synthesize multiple distinct SQL skeletons by constructing prompts based on the question and

Question Types	SQL Operations	Description
BrowseType	**Scan**	Questions that involve scanning all rows in a table with optional filtering (e.g., identifying all relevant biomarker tests or clinical trials).
SummarizeType	**Aggregate**	Questions that require grouping data and performing aggregation (e.g., counting the number of tests or trials for each group).
RefineType	**Filter**	Questions that require filtering out rows that don't match a specific criterion (e.g., selecting only approved tests or trials related to a particular cancer type).
ArrangeType	**Sort**	Questions that involve sorting results based on one or more attributes (e.g., sorting tests by approval status or sorting trials by enrollment size).
SelectTopType	**TopSort**	Questions that select the top-K rows based on certain criteria (e.g., selecting the top clinical trials with the highest success rate).
LinkType	**Join**	Questions that require joining two or more tables (e.g., linking clinical trial data with biomarker information to analyze correlations between test results and trial outcomes).
ExcludeType	**Except**	Questions that involve computing the difference between two sets of data (e.g., identifying tests or trials that meet one set of criteria but not another).
OverlapType	**Intersect**	Questions that involve computing the intersection of two sets of data (e.g., identifying common biomarkers or trials that appear in multiple datasets).
CombineType	**Union**	Questions that require computing the union of two sets of data (e.g., merging datasets from different sources to provide a comprehensive overview).

Table 1: Description of question types

Question	How many articles are associated with biomarkers related to risk?
SQL	SELECT COUNT(ba.pmid) FROM biomarker_article ba JOIN biomarker b ON ba.biomarker_internal_id = b.id WHERE b.biomarker_description LIKE '%risk%'
SQL Skeleton	SELECT COUNT(col_1) FROM table_1 JOIN table_2 WHERE col_2 LIKE value_1

Table 2: A example of question, SQL, and SQL skeleton

Dataset		#DBs	#Tabs	#Cols	#NLQs	#Syn NLQs
OncoMX	Train	1	25	106	100	1065
	Dev	1	25	106	99	-
CORDIS	Train	1	19	82	100	1306
	Dev	1	19	82	100	-
Bird	Train	69	522	3608	9428	-
	Dev	11	75	798	1534	-

Table 3: Statistics of Datasets

the selected database schemas. This process is formalized as $ss_i \leftarrow Prompt_{SSG}(g_i, s_i)$, where $Prompt_{SSG}$ represents the prompt used for SQL skeleton generation.

SQL Generation. For each NLQ, after obtaining the relevant schema and SQL skeleton, we proceed to generate SQL queries, represented as $sql_i \leftarrow Prompt_{SG}(q_i, s_i, ss_i)$, where $Prompt_{SG}$ denotes the prompt for SQL generation. It is important to note that each SQL skeleton is used to generate multiple SQL queries. In contrast to randomly synthesized SQL queries, our approach ensures that the synthesized SQL queries are closely aligned with the domain-specific NLQ, as they incorporate the necessary database schema and SQL skeleton. This ensures that the resulting SQL queries capture all the knowledge required for establishing the correct matching relationship between the NLQ and the database schema.

SQL-NLQ Synthesis. For each generated SQL query, we synthesize the corresponding NLQ, denoted as $q^{'} \leftarrow Prompt_{SQL2NLQ}(sql_i)$, where $Prompt_{SQL2NLQ}$ represents the prompt for NLQ generation. At this stage, the natural language question and SQL query are consistent in intent, recorded as $Q^{'} = \{q_1^{'}, q_2^{'}, q_3^{'}, ..., q_m^{'}\}$, where m represents the number of NLQ-SQL pairs, with $m > n$.

3.3 NLQ Semantic Optimization

After NLQ-SQL synthesis, some domain-irrelevant NLQs may still remain. To further improve the quality of the NLQ-SQL pairs, it is essential to filter out pairs that are semantically inconsistent with domain-specific questions.

NLQ-SQL Filter. To mitigate the impact of domain-irrelevant NLQ-SQL pairs on LLM fine-tuning, we propose an NLQ filtering method based on semantic similarity. Given that multiple NLQs can convey the same meaning through different expressions, conventional similarity evaluations may introduce errors. To address this, we employ a multi-vector retrieval method based on the M3-Embedding model (Chen et al., 2024) to calculate the semantic similarity between $q_i^{'}$ (the generated NLQ) and q_i (the domain-relevant question). We rank the candidate NLQs by their retrieval scores and retain the top K NLQ-SQL pairs in $Q^{'}$.

dataset	Methods	DeepSeek-Coder-v1		DeepSeek-Coder-v2	CodeLLaMA			StarCoder	
		6.7B	**33B**	**16B**	**7B**	**13B**	**34B**	**7B**	**15B**
OncoMX	No-finetuning	37.37	46.46	40.4	24.24	27.27	32.32	28.28	26.26
	(Hu et al., 2023) -Syn	33.33	34.34	34.34	9.09	19.19	19.19	8.08	32.32
	(Hu et al., 2023) -Syn+Real	29.29	36.36	35.35	27.27	30.34	20.20	6.06	36.36
	(Zhang et al., 2024) -Syn	7.07	15.15	27.27	11.11	4.04	7.07	3.03	8.08
	(Zhang et al., 2024) -Syn+Real	39.39	36.36	39.39	25.25	21.21	21.21	6.06	**44.44**
	Ours	50.51	42.42	41.41	30.3	31.31	**38.38**	**31.31**	38.38
	Ours+Real	**51.52**	**44.44**	**42.42**	**30.30**	30.30	36.36	**34.34**	39.39
CORDIS	No-finetuning	44.00	**51.00**	**48.00**	26.00	25.00	36.00	24.00	29.00
	(Hu et al., 2023) -Syn	14.00	22.00	31.00	20.00	22.00	23.00	20.00	27.00
	(Hu et al., 2023) -Syn+Real	21.00	19.00	22.00	19.00	17.00	21.00	21.00	24.00
	(Zhang et al., 2024) -Syn	7.00	3.00	26.00	6.00	5.00	7.00	9.00	8.00
	(Zhang et al., 2024) -Syn+Real	22.00	19.00	14.00	14.00	15.00	12.00	14.00	20.00
	Ours	46.00	44.00	40.00	29.00	29.00	**36.00**	**39.00**	39.00
	Ours+Real	**50.00**	41.00	37.00	**32.00**	**29.00**	34.00	38.00	**39.00**
BIRD	No-finetuning	22.10	26.40	25.55	9.91	9.26	15.71	15.71	22.88
	(Hu et al., 2023) -Syn	16.56	23.01	17.67	10.63	13.30	15.19	13.82	19.04
	(Hu et al., 2023) -Syn+Real	22.16	28.49	21.77	15.91	19.43	**26.66**	19.36	25.88
	(Zhang et al., 2024) -Syn	29.99	23.73	20.47	16.75	18.19	19.04	18.32	22.23
	(Zhang et al., 2024) -Syn+Real	**48.63**	**31.36**	23.34	21.32	25.62	26.34	22.08	**27.57**
	Ours	32.59	19.69	23.27	18.32	18.06	20.27	18.32	21.77
	Ours+Real	45.11	29.66	**24.38**	**22.75**	**25.62**	23.08	**22.62**	27.05

Table 4: Main results (%) of various open-source LLMs on methods.

4 Experiments

4.1 Experiment Setup

Datasets. As shown in Table 3, we evaluate our proposed approach using three real-world datasets. OncoMX (Zhang et al., 2024) provides information on cancer biomarkers, gene expression in healthy anatomical entities, differential gene expression between healthy and cancerous samples, and cancer-related mutations. CORDIS (Zhang et al., 2024) originates from the Community Research and Development Information Service, the European Commission's primary repository of results from EU-funded research and innovation projects. This dataset contains detailed hierarchical information on funding frameworks and the network of industrial and academic institutions, encoded in specialized EU terminology. BIRD (Li et al., 2024) is a large-scale cross-domain dataset encompassing 37 professional fields, including areas such as blockchain, healthcare, education, and hockey.The detailed statistics of these datasets are provided in Table 3.

Baselines. To evaluate the performance of our data synthesis method, we use two SOTA SQL-to-NL based methods ScienceBenchmark (Zhang et al., 2024) and (Hu et al., 2023) as baselines. For a fair comparison, all methods, including our DSQG-Syn, utilize GPT-3.5 as the underlying LLM.

Models. We selected eight open-source LLMs for SQL generation experiments: DeepSeek-Coder 6.7B/33B (Guo et al., 2024), DeepSeek-Coder-V2 16B (Zhu et al., 2024), Code LLaMA 7B/13B/34B (Grattafiori et al., 2023), and StarCoder 7B/15B (Li et al., 2023). These models were first fine-tuned on synthetic data and then evaluated on the validation set.

Metrics. Following (Hu et al., 2023), we use execution accuracy (EX) as the metric to evaluate the quality of the synthetic data by measuring the performance of LLMs fine-tuned on it.

Method/Model	OncoMX	CORDIS	BIRD
GPT-3.5-turbo	43.43	44.00	37.22
GPT-4o	48.48	45.00	46.09
(Hu et al., 2023) -Syn	34.34	27.00	23.01
(Hu et al., 2023) -Syn+Real	36.36	24.00	28.49
(Zhang et al., 2024) -Syn	27.27	26.00	29.99
(Zhang et al., 2024) -Syn+Real	39.39	22.00	**48.63**
Ours	50.51	46.00	32.59
Ours+Real	**51.52**	**50.00**	45.11

Table 5: Comparison results (%) between open-source LLMs and closed-source LLMs

# of NLQs	OncoMX	CORDIS
3	0.4747	0.44
5	0.4040	0.45
9	0.5151	0.46
15	0.4242	0.44

Table 6: The impact of varying the number of NLQs for each target table on data quality.

4.2 Main Results

Our DSQG-Syn consistently outperforms the baseline methods across almost all scenarios. In contrast, the data synthesized by the baseline methods not only fails to improve the performance of LLMs but also significantly reduces their accuracy in some cases. For example, on the OncoMX dataset, compared to the original LLMs without fine-tuning, LLMs fine-tuned on the synthetic data generated by ScienceBenchmark (Zhang et al., 2024) exhibit an accuracy drop of 13% to 31%. Similarly, the synthesis method proposed in (Hu et al., 2023) leads to a reduction in accuracy by 4% to 20%. This degradation occurs because the baseline methods rely on random sampling of schemas, resulting in numerous SQL-NLQ pairs that are irrelevant to the domain and misaligned with the intended query semantics, which severely impacts model performance. In contrast, our method enhances model performance, achieving up to a 13% improvement on the OncoMX dataset. These results demonstrate the effectiveness of our approach, which constrains SQL-NLQ generation by first synthesizing domain-relevant questions. This not only ensures consistency with query intent but also highlights the great potential of our framework.

We also evaluated the effectiveness of synthetic data for data augmentation. As shown in Table 4, combining synthetic data with real data yields better model performance compared to using synthetic data alone, for both our method and the baseline approach. However, a key difference is that while our method further enhances model performance, the baseline approach shows minimal improvement, with results falling between no fine-tuning and fine-tuning using only synthetic data. These findings demonstrate that the data synthesized by our method is robust and well-suited for data augmentation, while the data generated by the baseline approach negates the beneficial effects of real data on LLMs.

Table 5 presents a comparison between the best-performing models of different methods and closed-source models (GPT-3.5 and GPT-4o). The experimental results demonstrate that the models fine-tuned with the synthetic data generated by our method outperform the closed-source models on the OncoMX and CORDIS datasets. On the BIRD dataset, our method achieves results comparable to those of the closed-source models, while the baseline methods consistently yield lower performance than the closed-source models.

4.3 Question Generation Study

We investigated the impact of synthesizing varying numbers of NLQs for each target table on data quality during the domain-specific question generation phase, as shown in Table 6. Specifically, we generated between 3 and 15 questions per table. The experimental results indicate that the best performance is achieved when nine questions are synthesized for each table—a result that aligns closely with the number of question types we defined. This finding suggests that generating one domain-specific question for each predefined question type can effectively enhance the quality of the synthesized data.

4.4 NLQ-SQL Pairs Generation Study

SQL Generation VS SQL Skeleton Generation. We evaluated the accuracy of generating SQL directly from domain-specific questions versus generating SQL skeletons during the NLQ-SQL synthesis stage. To eliminate the influence of randomness in the domain question generation phase, we used the original 100 questions from the OncoMX and CORDIS datasets as the initial domain questions.

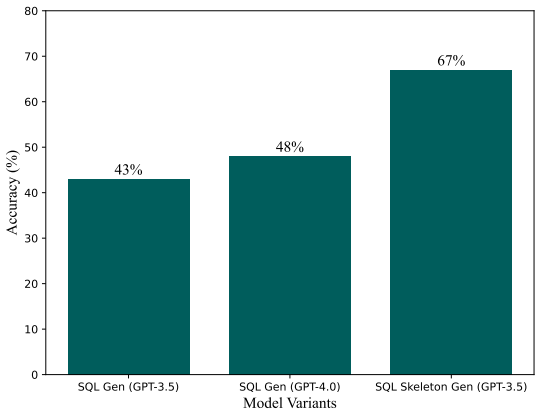

Figure 3: SQL generation VS SQL skeleton Generation

As shown in Figure 3, the accuracy of generating SQL skeletons significantly surpasses that of GPT-3.5 and GPT-4o in domain-specific SQL generation, demonstrating that our method effectively enhances the domain relevance of SQL through SQL skeleton synthesis.

# of SQL Skeleton	OncoMX	CORDIS
1	0.3939	0.45
3	0.4545	0.39
5	0.404	0.49
10	0.3939	0.43

Table 7: The impact of varying the number of SQL skeleton on data quality

Number of SQL Skeleton. We examined the impact of generating varying numbers of SQL skeletons for each domain-specific question on the quality of synthetic data during the NLQ-SQL synthesis phase. For each domain question, we generated between 1 and 10 SQL skeletons. As shown in Table 7, the best performance was achieved when three SQL skeletons were generated per question. Beyond this number, data quality declined, suggesting that a limited number of SQL skeletons is sufficient to fully capture the relationships between the schemas associated with each domain question.

4.5 NLQ Semantic Optimization

We investigated the impact of retaining different numbers of NLQ-SQL pairs for each domain-specific NLQ during semantic optimization on the experimental results. To eliminate the influence of question generation, we used the original 100 questions on the OncoMX and CORDIS datasets as the initial domain-specific questions and resynthesized the data starting from Step 2 in Figure 2. As shown in Table 8, the model's performance improves steadily with an increasing number of retained samples, achieving the best results when five samples are retained for each original question.

# of SQL-NLQ	OncoMX	CORDIS
1	0.3636	0.41
3	0.3939	0.45
5	0.4040	0.48
10	0.3636	0.46

Table 8: The impact of varying the number of NLQ-SQL pairs retained for each domain-specific NLQ during NLQ Semantic Optimization

4.6 Abalation Study

Method	OncoMX	CORDIS
Overall pipeline	**0.5051**	**0.46**
w/o Keywords Generation	0.3939	0.44
w/o Question Types	0.3737	0.45
w/o NLQ Semantic Optimization	0.3434	0.41

Table 9: Ablation study on the three proposed synthesis techniques.

We conducted an ablation study to evaluate the contribution of each technique in our proposed framework. Specifically, we removed each technique individually and regenerated the synthetic data using the framework on the OncoMX and CORDIS datasets. As shown in Table 9, removing any single technique results in a decline in performance, indicating that each of the three proposed synthesis strategies plays a crucial role in generating high-quality synthetic data.

5 Conclusion

In this paper, we propose a novel data synthesis framework, DSQG-Syn, to explore the use of synthetic data in text-to-SQL parsing. By combining domain-specific question generation with a question-guided NLQ-SQL synthesis method, DSQG-Syn produces NLQ-SQL pairs that are both domain-relevant and intent-consistent. Extensive experiments demonstrate that DSQG-Syn outperforms existing SOTA methods, significantly bridging the performance gap between open-source and closed-source models.

6 Limitations

Although DSQG-Syn has demonstrated promising results and significant progress in various aspects, there are limitations and areas for further improvement. First, due to constraints in computational resources and time, we fine-tuned the model using CodeLLaMA 34B as the largest available option, leaving the effectiveness of our data synthesis approach on even larger models uncertain. Second, our evaluation primarily focused on the text-to-SQL task. However, the broader potential of our domain question generation-based synthesis framework—especially for tasks like Python code generation—remains unexplored and warrants further investigation.

References

Abhijeet Awasthi, Ashutosh Sathe, and Sunita Sarawagi. 2022. Diverse parallel data synthesis for cross-database adaptation of text-to-sql parsers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11548–11562.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. 2024. Deepseek llm: Scaling open-source language models with longtermism. *CoRR*.

Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 2318–2335, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Ben Eyal, Moran Mahabi, Ophir Haroche, Amir Bachar, and Michael Elhadad. 2023. Semantic decomposition of question and sql for text-to-sql parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13629–13645.

Wenhan Xiong Grattafiori, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv e-prints*, pages arXiv–2401.

Yiqun Hu, Yiyun Zhao, Jiarong Jiang, Wuwei Lan, Henghui Zhu, Anuj Chauhan, Alexander Hanbo Li, Lin Pan, Jun Wang, Chung-Wei Hang, et al. 2023. Importance of synthesizing high-quality data for text-to-sql parsing. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1327–1343.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. 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*, 36.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caiming Xiong. 2021. Learning to synthesize data for semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2760–2766.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Lijie Wang, Ao Zhang, Kun Wu, Ke Sun, Zhenghua Li, Hua Wu, Min Zhang, and Haifeng Wang. 2020a. Dusql: A large-scale and pragmatic chinese text-to-sql dataset. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6923–6935.

Ping Wang, Tian Shi, and Chandan K Reddy. 2020b. Text-to-sql generation for question answering on electronic medical records. In *Proceedings of The Web Conference 2020*, pages 350–361.

Kun Wu, Lijie Wang, Zhenghua Li, Ao Zhang, Xinyan Xiao, Hua Wu, Min Zhang, and Haifeng Wang. 2021. Data augmentation with hierarchical sql-to-question generation for cross-domain text-to-sql parsing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8974–8983.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024b. 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)*, pages 7864–7875.

Wei Yang, Peng Xu, and Yanshuai Cao. 2021. Hierarchical neural data synthesis for semantic parsing. *arXiv preprint arXiv:2112.02212*.

Tao Yu, Rui Zhang, Heyang Er, Suyi Li, Eric Xue, Bo Pang, Xi Victoria Lin, Yi Chern Tan, Tianze Shi, Zihan Li, et al. 2019. Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1962–1979.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 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*. Association for Computational Linguistics.

Yi Zhang, Jan Milan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. 2024. Sciencebenchmark: a complex real-world benchmark for evaluating natural language to sql systems. *Proceedings of the VLDB Endowment*, 17(4):685–698.

Victor Zhong, Mike Lewis, Sida I Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6869–6882.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, et al. 2024. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*.

A Prompts

These are the prompts mentioned in the main body.

A.1 Question Generation

You are an expert in a specific domain. I will provide you with a database schema and a set of keywords. Based on these, identify the domain of expertise and generate a set of professional-level questions related to the given table that can be solved through SQL queries on the database.

[Instructions]

1. First, identify the domain of expertise based on the given keywords and schema.
2. Generate 15 questions related to the {table_name} table, ensuring that the questions reflect different types of SQL operations and address problems relevant to the identified domain.

Question Generation Guidelines: Your task is to generate questions related to the {table_name} table. These questions may involve multiple table joins with other tables in the schema. The questions should be designed to solve problems relevant to the identified domain.

- **Scan:** Generate questions that involve scanning all rows in a table with optional filtering.
- **Aggregate:** Generate questions that require grouping data (e.g., by object type or survey results) and performing aggregation.
- **Filter:** Generate questions that require filtering out rows that don't match a specific criterion.
- **Sort:** Generate questions that involve sorting results based on one or more attributes (e.g., distance, magnitude).
- **TopSort:** Generate questions that select the top-K rows based on certain criteria (e.g., closest objects).
- **Join:** Generate questions that require joining two or more tables using relevant foreign key relationships.
- **Except:** Generate questions that involve computing the set difference between two sets of astrophysical data (e.g., objects detected in two different surveys).
- **Intersect:** Generate questions that involve computing the intersection of two sets of astrophysical data (e.g., common objects across two surveys).
- **Union:** Generate questions that require computing the union of two sets of astrophysical data (e.g., merging results from two observations).

[Output format]

```
{
    "Domain": "<Identified Domain of Expertise>",
    "NLQ": "<Natural Language Question>",
    "Keyword": "<Keywords for the question>",
    "Related Schema": "<Relevant tables from theschema>",
    "Related SQL operator": "<SQL operations: Scan, Aggregate, Filter, Sort,\\
    TopSort, Join, Except, Intersect, Union>"
}
```

For example:

```
[
    {
        "Domain": "Astrophysics",
        "NLQ": "What is the distance between each object and its neighbor?",
        "Keyword": "distance, objid, neighborobjid",
        "Related Schema": "neighbors",
        "Related SQL operator": "Scan"
    },
    {
        "Domain": "Astrophysics",
        "NLQ": "Which objects were detected in the 'boss' survey but not \
        in the 'eboss' survey, and what is their average redshift?",
        "Keyword": "objects, boss survey, eboss survey, average redshift,
        set difference",
        "Related Schema": "specobj",
        "Related SQL operator": "Except, Aggregate"
    },
    {
        "Domain": "Astrophysics",
        "NLQ": "Which objects were observed in both the 'sdss' and 'boss' \
        surveys, and sort them by their velocity dispersion (veldisp)?",
        "Keyword": "objects, sdss survey, boss survey, velocity dispersion",
        "Related Schema": "specobj",
        "Related SQL operator": "Intersect, Sort"
    }
]
```

Database Schema: {Schema}
Keywords: {Keywords}

A.2 SQL Skeleton Generation

Please generate an SQL template based on the given question and schema. Ensure that all SQL clauses are included, such as `SELECT`, `FROM`, `JOIN`, `WHERE`, `GROUP BY`, `ORDER BY`, and `HAVING`. Use placeholders for specific table and column names as follows:

1. Use `col_#` for column names.
2. Use `table_#` for table names.
3. Use `value_#` for constant values.

Example 1:

```
Input:
{"question": "Show me the redshift of spectroscopic object with subclass \
of STARFORMING"}
Schema:
CREATE TABLE specobj (
```

```
    specobjid number Example Values[(Decimal('299489952322840576'),), ...],
    subclass text Example Values[(None,), ('BROADLINE',), ('STARFORMING',)],
    z number Example Values[(7.01124,), (0.00415325,),
    (0.00415325,)],
    . . . . . .
    primary key (specobjid)
)
Output:
{"template": "SELECT col_0 FROM table_1 WHERE col_0 = value_0"}
```

Example 2:

```
Input:
{"question": "Find the photometric objects with object ids,\
spectroscopic object id whose spectroscopic class is 'GALAXY' ..."}
Schema:
CREATE TABLE photoobj (
    objid number Example Values[(1237645879551066262,), ...],
    u number Example Values[(24.6346,), ...],
    r number Example Values[(24.802,), ...]
);
CREATE TABLE specobj (
    specobjid number Example Values[(Decimal('299489952322840576'),), ...],
    class text Example Values[('GALAXY',), ...],
    primary key (specobjid)
)
Output:
{"template": "SELECT col_1, col_2 FROM table_0
JOIN table_1 WHERE col_3 = value_0 AND col_4 - col_5 < value_1
AND col_4 - col_4 > value_2"}
```

Now, apply the same transformation to the question below. Do not let specific table names, column names, or constant values (like "description", "name", "GALAXY", or "BROADLINE") appear in the template.

Input:

```
{"question": "{Question}"}
Schema:
{Schema}
```

Output:

A.3 SQL Generation

You are an expert in a specific domain. You are provided with:

1. An SQL query template.
2. A question that the query needs to answer.
3. The schema of the relevant database.

Your task is to:

1. Strictly use the information from the provided schema to complete the PostgreSQL query. Ensure that all necessary table names, column names, and clauses (such as `FROM` and `JOIN`) come from the schema only.
2. Avoid introducing any table names, column names, or other elements that are not explicitly defined in the schema.
3. The generated 10 SQLs need to be directly related to the given question and fit the SQL query template.
4. Keep the output in JSON format.

Example:

```
Input:
SQL Query Template:
SELECT col_1, col_2 FROM table_1 JOIN table_0 WHERE col_3 = value_0;
Question:
What are the names and descriptions of the different types of photos associated
with objects in the astrophysical classifications from the specobj table?

Database Schema:
CREATE TABLE photo_type (
    value number Example Values[(6,), (2,), (4,)],
    name text Example Values[('GHOST',), ('STAR',), ('NOTATYPE',)],
    description text Example Values[
        ('Sky: Blank sky spectrogram (no objects in this arcsecond area).',),
        ('Trail: A satellite or asteroid or meteor trail. (not yet used)',),
        ('Unknown: Object type is not known.',)
    ],
    primary key (value),
    foreign key (value) references photoobj(type),
    foreign key (value) references neighbors(neighbortype)
);

CREATE TABLE specobj (
    specobjid number Example Values[(Decimal('299489952322840576'),), ...],
    bestobjid number Example Values[(1237649920046661771,), ...],
    survey text Example Values[('boss',), ('sdss',), ('eboss',)],
    class text Example Values[('GALAXY',), ('STAR',), ('QSO',)],
    subclass text Example Values[(None,), ('BROADLINE',), ('STARFORMING',)],
    ra number Example Values[(128.95375,), (131.37494,), (138.87159,)],
    dec number Example Values[(0.120958,), (11.281341,), (14.749983,)],
    zerr number Example Values[(-1.0,), (-4.0,), (-6.0,)],
    veldisp number Example Values[(0.0,), (850.0,), (262.5,)],
    veldisperr number Example Values[(0.0,), (-3.0,), (-4.0,)],
    primary key (specobjid),
    foreign key (bestobjid) references photoobj(objid)
);
```

```
Output:
{
    "sql": [
        "SELECT p.name, p.description FROM photo_type p JOIN specobj s
        ON p.value = s.bestobjid WHERE s.class = 'STAR';",
        "SELECT p.name, s.programname FROM photo_type p JOIN specobj s
        ON p.value = s.bestobjid WHERE s.subclass = 'BROADLINE';",
        "SELECT p.name, s.programname FROM photo_type p JOIN specobj s
        ON p.value = s.bestobjid WHERE s.class = 'QSO';",
        "SELECT p.name, s.programname FROM photo_type p JOIN specobj s
        ON p.value = s.bestobjid WHERE s.ra > 130;",
        "SELECT p.name, s.programname FROM photo_type p JOIN specobj s
        ON p.value = s.bestobjid WHERE s.zerr < -2;",
        "SELECT p.name, s.programname FROM photo_type p JOIN specobj s
        ON p.value = s.bestobjid WHERE s.veldisp > 500;",
        "SELECT p.name, s.programname FROM photo_type p JOIN specobj s
        ON p.value = s.bestobjid WHERE s.survey = 'boss';",
        "SELECT p.name, s.programname FROM photo_type p JOIN specobj s
        ON p.value = s.bestobjid WHERE s.programname = 'eboss';",
        "SELECT p.name, s.programname FROM photo_type p JOIN specobj s
        ON p.value = s.bestobjid WHERE s.dec > 10;",
        "SELECT p.name, s.programname FROM photo_type p JOIN specobj s
        ON p.value = s.bestobjid WHERE s.veldisperr < -3;"
    ]
}
```

Now, it's your turn.

Input:

```
SQL Query Template: {Template}
Question: {Question}
Database Schema:
{Database Schema}
```

Output:

A.4 NLQ Generation

You need to generate the corresponding user query according to a given SQL query. Here are the examples:

Example 1:

```
Input:
"sql": "SELECT count(*) FROM biomarker_fda_test_trial
JOIN biomarker_fda_test
ON
biomarker_fda_test_trial.test_submission = biomarker_fda_test.test_submission
AND
biomarker_fda_test_trial.test_trade_name = biomarker_fda_test.test_trade_name
```

```
WHERE biomarker_fda_test.test_manufacturer = '23andMe'",

Output:
{
    "question" "Show number of test trials of 23andMe"
}
```

Example 2:

```
Input:
"sql": "SELECT DISTINCT T3.name, T2.name
FROM healthy_expression AS T1
JOIN anatomical_entity AS T3 ON T1.uberon_anatomical_id = T3.id
JOIN cancer_tissue AS T4 ON T3.id = T4.uberon_anatomical_id
JOIN disease AS T2 ON T4.doid = T2.id;",

Output:
{
    "question": "which diseases have related anatomical entities?"
}
```

Now, it's your turn. Just return 1 user query in JSON format.
Input:

```
"sql": "{SQL query}"
```

Output:

A.5 Keyword Generation

You are a domain expert. Please carefully review the following database schema and interpret the field names, data types, and example values to identify relevant domain-specific terms. For instance, recognize that certain variables (e.g., z) in a schema might represent key concepts in a specific field, such as z representing redshift in astrophysics.

Below is the definition of the database schema:

```
{Schema}
```

Please analyze the schema and provide the following output in JSON format:

```
{
    "domain": "your inferred domain",
    "keywords": ["keyword1", "keyword2", . . . . . ,"keywordN"]
}
```

In this case, the output should not simply repeat the column names. Instead, you should map them to real-world domain-specific concepts (e.g., z could be recognized as "redshift" in astrophysics) and provide relevant keywords based on your understanding of the schema and its context.