BookSQL: A Large Scale Text-to-SQL Dataset for Accounting Domain

Rahul Kumar[†] **Amar Raja Dibbu**[†] **Shrutendra Harsola**[*]
Vignesh Subrahmaniam[*] **Ashutosh Modi**[†]
[†] Indian Institute of Technology Kanpur (IIT Kanpur) [*] Intuit
{rahulkumar21,amard21}@iitk.ac.in
{shrutendra_harsola,vignesh_subrahmaniam}@intuit.com
{ashutoshm}@cse.iitk.ac.in

Abstract

Several large-scale datasets (e.g., WikiSQL, Spider) for developing natural language interfaces to databases have recently been proposed. These datasets cover a wide breadth of domains but fall short on some essential domains, such as finance and accounting. Given that accounting databases are used worldwide, particularly by non-technical people, there is an imminent need to develop models that could help extract information from accounting databases via natural language queries. In this resource paper, we aim to fill this gap by proposing a new large-scale Text-to-SQL dataset for the accounting and financial domain: BookSQL. The dataset consists of 100k natural language queries-SQL pairs, and accounting databases of 1 million records. We experiment with and analyze existing state-of-the-art models (including GPT-4) for the Text-to-SQL task on BookSQL. We find significant performance gaps, thus pointing towards developing more focused models for this domain.

1 Introduction

Relational databases are pervasive in all modern-day organizations, from financial establishments to educational institutes. Typically, query languages such as SQL are used to extract the required data from relational databases. However, formulating queries in SQL needs mastery of the language itself; consequently, this excludes people (particularly those without technical background, e.g., financial accountants) who do not know SQL from using databases. It is imperative to develop techniques to address the research question, can relational databases be queried using natural language? In this paper, we take a step toward this goal; in particular, we explore if one could develop a natural language interface for accounting databases. In recent years, several large-scale general-purpose datasets (Deng et al., 2022) have been proposed for developing Text-to-SQL systems[1], such as Spider (Yu et al., 2018) and WikiSQL (Zhong et al., 2017). Such datasets,[2] though cross-domain, are still not suitable for developing systems that could address real-world business use cases, such as accessing accounting databases via natural language interfaces. The primary reason is that these large-scale datasets have a considerable breadth regarding types of domains. However, they either lack certain domains (such as accounting) or have limited data and query types for specific domains (e.g., financial, sales, and marketing). In this paper, we try to address this gap by proposing a large-scale Text-to-SQL dataset (called **BookSQL**) for the accounting and business domain. We collaborate with financial experts to create a dataset that reflects actual accounting databases used in the industry.

Model	Spider	BookSQL
UniSAr	70%	3.8%
SEDE	63.2%	0.0%
RESDSQL	80.5%	10.8%

Table 1: Performance (Exact Match Accuracy (c.f. §5)) of pre-trained SOTA Text-to-SQL models on Spider and the proposed **BookSQL** dataset. As can be observed existing models have very poor performance on **BookSQL** indicative of poor domain generalization.

To the best of our knowledge, there is no large-scale dataset in the accounting domain that contains granular records of accounting books used in businesses. To give an idea about the scale of usage of accounting databases: there are around 33

[1]By Text-to-SQL system we refer to a system that, given a natural language query, automatically retrieves the desired information from a database or multiple databases by converting a natural language query to SQL query as an intermediate representation.

[2]By Text-to-SQL dataset we refer to a dataset having both the natural language queries with corresponding SQL formulation and correct answers along with the corresponding database against which queries are fired

Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), pages 497–516

Dataset	#Size	#DB	#D	#T/DB	Domain	ORDER BY	GROUP BY	NESTED
Spider	10,181	200	138	5.1	Cross	1335	1491	844
WikiSQL	80,654	26,521	-	1	Cross	0	0	0
Advising	3,898	208	1	10	Single	15	9	22
BIRD	12,751	95	37	7.3	Cross	2576	881	0
IMDB	131	1	1	16	Single	10	6	1
Yelp	128	1	1	7	Single	18	21	0
BookSQL	100k	1	1	7	Single	17,529	11,508	4,456

Table 2: Comparison of benchmark datasets with **BookSQL**. #Size, #DB, #D, and #T/DB represent the numbers of query-SQL pairs, databases, domains, and the averaged number of tables per domain, respectively. The "-" in the #D column indicates an unknown number of domains. Last 3 columns indicate the query types. Yelp dataset is based on Yelp website, IMDB is based on movie domain and Advising dataset is based on the University Course domain

million small businesses[3] in the US alone. Most of these businesses use accounting software to maintain their books to keep track of their finances, i.e., money-in transactions (e.g., invoice and sales receipt) and money-out transactions (e.g., expense, purchase order, and bill payment). Additionally, for tax purposes, these books need to follow standard accounting principles like double-entry accounting,[4] hierarchical chart of account structure,[5] and accrual accounting.[6] Transactions in the accounting database span across multiple tables. The corresponding SQL queries can involve complex operations such as aggregations, computing distinct counts, and nested queries to extract information from these. For a novice user, this is not an easy task. Moreover, as observed in our initial experiments (Table 1), existing state-of-the-art (SOTA) Text-to-SQL models trained on Spider have very poor performance on domain-specific **BookSQL** dataset, pointing towards the need for a accounting domain specific dataset which will further lead to the development of SOTA models. In a nutshell, in this resource paper, we make the following contributions:

1. We create a new and large-scale Text-to-SQL financial dataset referred to as **BookSQL**. The dataset consists of a financial-accounts database of 1 million records. The corresponding natural language queries are designed to address various practical intricacies of the accounting domain. BookSQL has 100k Query-SQL pairs which is about 1.25 times the existing largest Text-2-SQL dataset: WikiSQL. In particular, for designing the queries, we consulted financial experts to understand various practical use cases.

2. We run existing state-of-the-art models (including GPT-4) for the Text-to-SQL task on BookSQL to see the performance and analyze the shortcomings of the models trained on existing large-scale datasets such as Spider, pointing towards developing specialized models for this domain. We release the dataset and model code via GitHub: https://github.com/Exploration-Lab/BookSQL.

2 Related Work

Due to its importance in practical applications, developing natural language interfaces to databases has been an active area of research. Due to space constraints, we cannot cover all the research, and we refer the reader to the survey by Deng et al. (2022). We outline some of the main works in this area in this section. Several datasets have been proposed for Text-to-SQL task in recent years. For example, *Spider* (Yu et al., 2018) dataset has been proposed; it covers 138 different domains. A large-scale dataset, WikiSQL (Zhong et al., 2017), consisting of 24241 Wikipedia tables, has been created. Similarly, Squall (Shi et al., 2020b), KaggleDBQA (Lee et al., 2021), and BIRD-SQL (Li et al., 2023b) datasets have been generated to evaluate the generalization property of models on unseen domains. Domain-specific datasets have also been proposed, such as those based on Yelp and IMDB (Yaghmazadeh et al., 2017), Advising domain (Finegan-Dollak et al., 2018), MIMICSQL (Wang et al., 2020), SEDE (Hazoom et al., 2021a),

[3] https://tinyurl.com/mr3vrptj

[4] https://en.wikipedia.org/wiki/Double-entry_bookkeeping

[5] https://en.wikipedia.org/wiki/Chart_of_accounts

[6] https://en.wikipedia.org/wiki/Basis_of_accounting

BookSQL	Stats
Size of the database	1 million
Total Businesses	27
Size of Question-SQL Pair	100k
Number of Easy SQL	10,000
Number of Medium SQL	45,000
Number of Hard SQL	45,000

Table 3: Statistics of BookSQL

Restaurants domain (Tang and Mooney, 2001), and Academic domain (Li and Jagadish, 2014). The purpose of these datasets is to evaluate the performance of models with a high degree of precision while disregarding the generalization characteristic of the models.

Comparison. We compare BookSQLwith other popular datasets in Table 2. As can be observed, BookSQL has a much large number of Query-SQL pairs, has a more diverse number of queries in terms of the SQL clauses (e.g., `ORDER BY`), and involves more complex (and nested) queries. Benchmark dataset such as Spider have a very wide coverage over various domains (138) but very few queries per domain (e.g., average number of queries per domain is 74 in the case of Spider), limiting its performance in a specific domain (see also Table 1). Moreover, BookSQL can be merged with the existing Spider dataset to increase its coverage in the business domain.

Models. Various models have been proposed for the Text-to-SQL task (Deng et al., 2022). Some state-of-the-art models include the non-invasive UniSAr model (Dou et al., 2022) based on Seq2Seq architecture. The model has shown high accuracy on the multi-domain, multi-table Spider dataset. RESDSQL (Li et al., 2023a) decouples the schema linking and the skeleton parsing for Text-to-SQL generation. Schema linking identifies the table and columns required for a given question. Skeleton parsing first generates the SQL skeleton and then the final SQL. It achieves SOTA performance on the Spider benchmark.

3 BookSQL Dataset

Given the importance and wide prevalence of business databases across the world, the proposed dataset, BookSQL focuses on the finance and accounting domain. Accounting databases are used across a wide spectrum of industries like construction, healthcare, retail, educational services, insurance, restaurant, real estate, etc. Business in these industries arranges their financial transactions into their own different set of categories (called a chart of accounts[7] in accounting terminology). For example, a restaurant business could have categories like advertising, license fees, etc., a real estate brokerage business could have categories like commissions, office supplies, etc. Keeping generalization in mind BookSQL dataset includes a variety of businesses from different industries. Hence, a Text-to-SQL system developed on BookSQL will be robust at handling various types of accounting databases. The total size of the dataset is 1 million. The dataset is prepared under financial experts' supervision, and the dataset's statistics are provided in Table 3. The dataset consists of 27 businesses, and each business has around 35k - 40k transactions. The distributions of all businesses and their products are shown in Appendix Figure 3 and Figure 4.

3.1 BookSQL Tables

Figure 1 shows the detailed database schema. The schema is reflective of real-life databases used in the finance and accounting domain. There are seven tables in the BookSQL, namely, Master Transactions, Customer, Employees, Product Service, Vendor, Chart of Account, and Payment Method tables. We arrived at the list of seven tables after examining (and corresponding discussions with finance experts) the databases of several businesses. Given the nature of accounting domain, majority of databases used by businesses across the globe are restricted mainly to these seven tables only. The main table is the "Master Transaction" table (e.g., Appendix Table 8), which records money-in transactions (invoice, sales receipt, etc.) and money-out transactions (expense, purchase order, bill payment, etc.) This table also records additional corresponding transaction details, like the customer, vendor, product/service, credit account, debit account, and amount. The "Chart of accounts" table (e.g., Appendix Table 9) contains information on all account names and types. The "Customer" table (e.g., Appendix Table 10) contains all the customer's details, i.e., name, billing, and shipping address. The "Vendors" table (e.g., Appendix Table 11) contains all the vendor details of all the businesses, i.e., vendor names and billing addresses. The "Employees" table (e.g., Appendix

[7] https://www.investopedia.com/terms/c/chart-accounts.asp

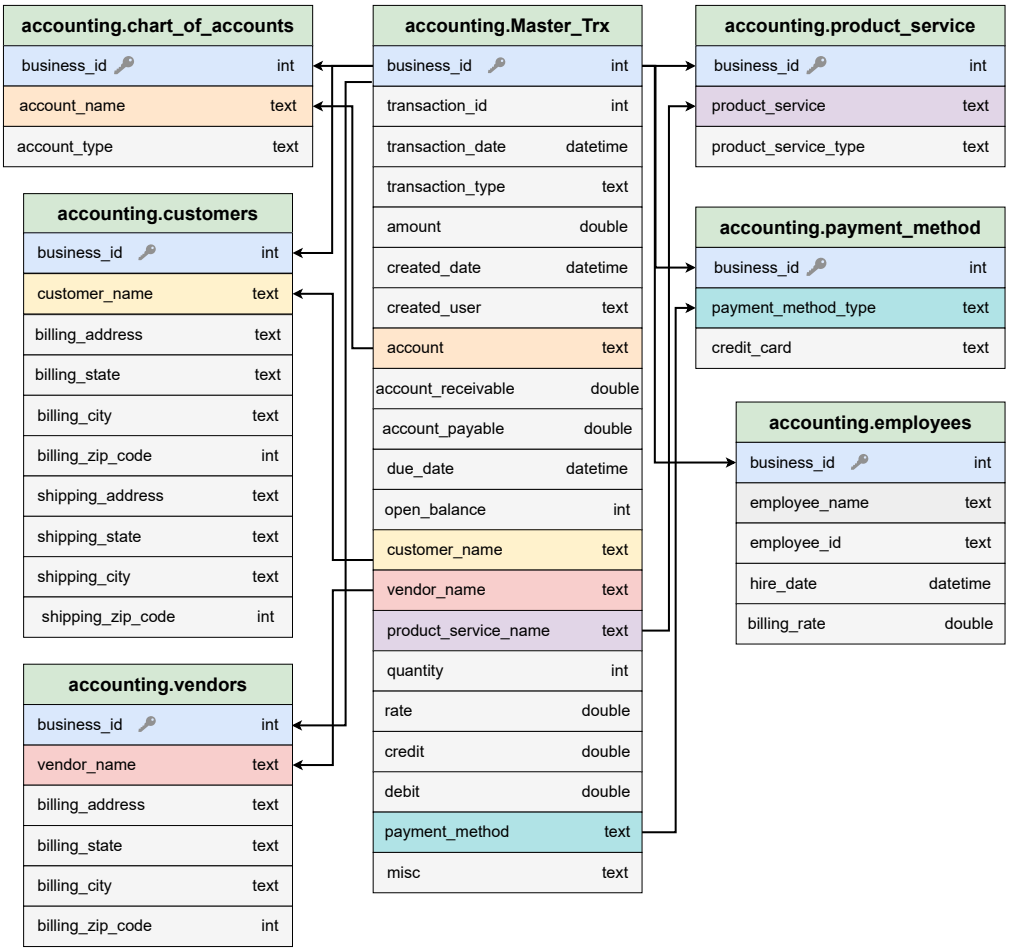

Figure 1: BookSQL Database schema

Table 12) contains information about all the business employees. The "Product service" table (e.g., Appendix Table 13) contains the details of all the products and services. The "Payment method" table (e.g., Appendix Table 14) contains different payment methods the business uses.

3.2 Financial Constraints

For creating the dataset, we took existing accounting databases based on the schema described above and anonymized the names and entries in the tables, i.e., actual names, businesses, and numbers were replaced with fictional ones while adhering to the financial constraints (described next). This is done to maintain the privacy of individuals and businesses. The resulting database is a true reflection of a real-world accounting setting. Accounting databases follow certain accounting rules and financial constraints, which were followed when anonymizing the database. In particular, standard double-entry accounting was followed, which means every entry to an account needs a corresponding and opposite entry to a different account, i.e., debit and credit. So, the sum of debit should always be equal to the sum of credit for every transaction. All seven tables were partitioned by business_id. For a given transaction_id, the sum of the credits column should equal the sum of the debits column, and both should equal the amount column in the Master Transactions table. Credit (in the Master Transaction table) should be equal to the product of Quantity and Rate. The chart of accounts was anonymized using the industry-wise list published by a popular CPA.[8] Business-specific custom fields were anonymized using the examples provided in the help articles of various accounting software. The created database was cross-checked with financial experts to make sure that the created database looked like a real-world accounts database.

3.3 Dataset Creation and Annotation

BookSQL dataset consists of 100k questions in natural language and their corresponding SQL on

[8] https://hectorgarcia.com/resources/

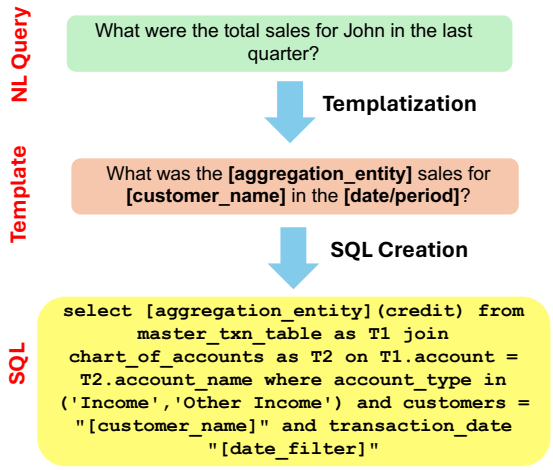

Figure 2: An example showing the pipeline for creating BookSQL dataset. Note, here we can replace *aggregation_entity* by max, min, total, and average, and *customer_name* can be replaced with any possible name to get the Question-SQL pair. Similarly, *date/period* can be replaced with *last quarter, this quarter, last month*.

multiple tables, covering 27 different businesses. We involved financial experts in the query creation process. We collaborated with two financial experts who have previously been involved in the creation of accounting software. Moreover, these experts have the knowledge and experience in dealing with customer interactions involving account books. The financial experts helped us on a pro bono basis since the creation of Text-to-SQL system for the accounting domain would help them and their customers.

The question-SQL pair formulation process is as follows. With the help of financial experts, we first created a list of typical questions (based on the account book) that customers (or business people) usually ask or questions about the information that customers are interested in knowing. We tried to keep the questions (queries) as natural as possible to capture real-world scenarios. We relied on the experience of financial experts to keep the list as exhaustive as possible. We also created the corresponding SQL query for each of the natural language queries in the list. The queries in the list were then used to create more queries via the process of templatization. Figure 2 explains the process with help of an example.

In order to be as exhaustive as possible, with the help of experts, we arrived at a list of 183 unique natural language questions that customers typically ask when interacting with accounting databases. These natural language questions were used to create query templates, and this was further used to generate diverse range of Question-SQL pairs in BookSQL. Additionally, we performed a second round of verification of the BookSQL corpus and query templates with financial experts to verify the consistency, veracity, and ensure that the dataset reflected the real-world scenario. Note that existing general Text-to-SQL datasets (e.g., Spider and WikiSQL) consist of databases from multiple domains and BookSQL is focused on the financial domain, hence the number of templates may appear to be less. However, the number of templates is still large when compared across a single domain, for example, to give a rough estimate, Spider dataset uses 5693 templates and spans 138 domains, so a rough estimate of number of templates per domain is about 41 ($\sim 5693/138$). Note that Spider doesn't provide details about templates for each domain, hence a rough estimate. Moreover, questions in existing Text-to-SQL datasets (like Spider) are created by students (Yu et al., 2018), whereas questions in BookSQL are created by financial experts who use accounting systems on a regular basis and are well-versed. Although our dataset is small (in terms of total number of templates), it is of high quality and more complex; hence it helps in learning models that would generalize well. Moreover, while experimenting with models, the queries in the test set are based on templates that are not used during training (see section 5).

To the best of our knowledge, BookSQLis the first Text-to-SQLdataset to have multi-step questions, which requires nested SQL queries to get the answer. For example - *"What products are selling less than last month/week?"* It would first require computing monthly/weekly product level sales and then comparing each product's current and last month's/week's sales. BookSQLdatabase schema also contains complex column types. Additionally, BookSQLis the first Text-to-SQLdataset to have extensive time-based filters like last month, this quarter to date, last financial year, between July to August, this week, yesterday, etc.

3.4 Complexity of SQL in BookSQL

SQL queries in BookSQL are diverse and cover various levels of complexity, i.e., it covers the following operations: SELECT with multiple columns and aggregations, WHERE, GROUP BY, HAVING, ORDER BY, LIMIT, JOIN, INTERSECT, UNION, NOT IN, OR, AND, EXISTS, CONTAINS as well as nested queries. Table 2 shows the comparisons

Complexity	Question	SQL
Easy	What is the balance owned by John?	SELECT balance from Customers where customer_name = 'John'
Medium	What is the maximum sales for John in the last month?	SELECT MAX(credit) FROM master_txn_table where account_type in ('Income', 'Other Income') AND customer = 'John' AND month(transaction_date) = month(current_timestamp) - 1
Hard	What products are selling less than last month?	SELECT A.product_service, revenue_this_month, revenue_last_month FROM (SELECT product_service, SUM(credit) as revenue_this_month FROM master_txn_table WHERE account_type in ('Income', 'Other Income') AND month(transaction_date) = month(current_timestamp) GROUP BY 1) AS A INNER JOIN (SELECT product_service, SUM(credit) as revenue_last_month FROM master_txn_table WHERE account_type in ('Income', 'Other Income') AND month(transaction_date) = month(current_timestamp) - 1 GROUP BY 1) AS B ON A.product_service = B.product_service WHERE revenue_this_month < revenue_last_month

Table 4: Examples of Question-SQL pairs from BookSQLbased on complexity of the query.

of all Text-to-SQL datasets. In terms of complexity, BookSQL consists of complex SQL queries containing 17,529 ORDER BY, 11,508 GROUP BY, and 4,456 NESTED queries. We further divided all Query-SQL pairs into three categories: Easy, Medium, and Hard, based on the complexity of SQL. Table 4 shows examples for each category. Table 3 shows the main statistics of the BookSQL. BookSQL consists of 7,193 `Hard` SQL queries, making it a more complex, large, and challenging dataset. We used the following criteria to decide on the complexity of a query.

- EASY: simple queries with single WHERE condition
- MEDIUM: multiple conditions in WHERE clause and multiple columns in SELECT clause
- HARD: Join, Group by, Inner queries, Union, Except as these are hard to predict from Natural Language question.

4 Baseline Models

We benchmark existing state-of-the-art (SOTA) Text-to-SQL models on BookSQL dataset.

SEDE: We fine-tuned the SEDE model (Hazoom et al., 2021b) on the BookSQL dataset. SEDE is a T5-based sequence-to-sequence model (Raffel et al., 2020). It takes unordered schema items (tables and column names) along with questions as input and generates the corresponding SQL query as output.

UniSAr: We fine-tuned the UniSAr model (Dou et al., 2022) on the BookSQL train dataset, with T5-large as the base language model. UniSAr converts any seq-to-seq language model into a text2sql model by three non-invasive extensions: (1) Structure Mark to encode database schema in the model input, (2) Constrained Decoding to generate well-structured SQL. For the BookSQL dataset, we removed the constrained decoding module of UniSAr, since it did not support the SQL queries with complex grammar present in the BookSQL dataset. (3) SQL Completion for completing potential missing JOIN relationships.

RESDSQL: RESDSQL (Li et al., 2023a) decouples the schema linking and the skeleton aware decoding for SQL generation. A cross-encoder is trained to rank the tables and columns required for a given query for schema linking. For SQL generation, a seq-to-seq model with skeleton-aware decoding is used, which first generates an SQL skeleton, and then the model predicts the actual SQL query from it. The masked self-attention method in the decoder allows the first created skeleton to direct the future SQL parsing implicitly.

DIN-SQL + GPT4: We use prompt chaining tech-

Model	Spider		BookSQL				
	EMA	EA	EMA	PCM-F1	EA	BLEU-4	ROUGE-L
SEDE	63.2%	-	43.4%	0.82	44.3%	0.69	0.83
UniSAr	70%	-	43.0%	0.78	47.6%	0.72	0.80
RESDSQL	80.5%	84.1%	51.5%	0.81	54.4%	0.74	0.81
DIN-SQL+GPT4	60%	85.3%	9.3%	0.63	7.6%	0.43	0.68
DFew+GPT4	-	-	47.5%	0.89	67.2%	0.86	0.90

Table 5: Results on Spider and BookSQL datasets. EMA refers to Exact Match Accuracy, EA refers to Execution Accuracy, and PCM-F1 refers to Partial Component Match F1. DFew+GPT4 refers to Dynamic few-shot prompt+GPT4

nique as proposed in Pourreza and Rafiei (2023). It decomposes Text-to-SQL task into multiple sub-tasks and then solves each sub-task one by one by prompting GPT4 (Achiam et al., 2023) with sub-task-specific prompts. It uses the following sub-tasks:

1. **Schema Linking:** This module identifies references to database tables and columns required to answer the natural language question.
2. **Classification and Decomposition:** This module classifies each question into easy, non-nested complex, and nested complex. This signifies the type of SQL query required for the given question.
3. **SQL Generation:** This module generates the SQL using the output of previous modules.
4. **Self Correction module:** This module is responsible for correcting any minor mistakes in the SQL generated by the previous module.

Sample prompts for each of these sub-tasks are provided in the Appendix §C.

Dynamic few-shot prompt + GPT4 (DFew+GPT4): We follow a dynamic few-shot prompting technique similar to Sun et al. (2023). Firstly, a vector database is created by an embedding train set questions using SentenceTransfomers *all-MiniLM-L6-v2* model.[9] This model is trained on the 1 billion sentence pairs dataset[10] and is best suited for generating sentence embeddings. This created embedding database is called trainDB. Then, at inference time, embedding for the test question is created using the same SentenceTransfomers model, and this embedding is used to do ANN (Approximate Nearest Neighbor) search in trainDB to get ten examples from the train set. These ten examples and database schema is used to create the few-shot SQL generation prompt for GPT4. Pseudo-code and sample prompts are provided in Appendix §B. We use ChromaDB[11] as the underlying vector database and for ANN search.

5 Experiments, Results and Analysis

5.1 Evaluation Metrics

We use the standard evaluation metrics (details in Appendix D) of Exact Match Accuracy (EMA) (Yu et al., 2018), Execution Accuracy (EA) (Yu et al., 2018), Partial Component Match F1 (PCM-F1) (Hazoom et al., 2021a), BLEU-4 (Papineni et al., 2002), and ROUGE-L (Lin, 2004).

5.2 Experimental Setup

We divide the dataset into 70% train, 10% validation, and 20% test sets based on query templates. The test set contains 14.37% easy, 78.43% medium, and 7.2% hard SQL queries. In order to check the generalization performance, queries in the test set are based on templates that are not used during training. Given limitations on the number of calls to OpenAI GPT4 API, we used a random 10% of BookSQL test set for GPT4-based approaches. We provide details about training and hyper-parameters in Appendix E.

5.3 Results

Table 5 and Table 6 shows the performance of baseline models. Table 5 shows the performance of

[9] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[10] https://huggingface.co/blog/1b-sentence-embeddings

[11] https://github.com/chroma-core/chroma

Query	SEDE	UniSAr	RESDSQL	GPT4
E	100	100	100	100
M	43.08	46.49	62.12	71.35
H	15.00	12.34	15.00	22.08

Table 6: Execution Accuracy (in %) of various models on SQL queries of varying complexity. E refers to **Easy** query, M refers to **Medium** query and H refers to query with **Hard** complexity.

SOTA Text-to-SQL models fine-tuned on the BookSQL dataset. RESDSQL performs best as can be observed with regard to exact match accuracy and execution accuracy scores. SEDE and UniSAr have poor exact match and execution accuracy scores. Though BookSQL and Spider are not directly comparable, we also include results of the models on the Spider dataset to provide a reference for comparison purposes. As can be observed, the models that perform well on Spider do not have a good performance on BookSQL, indicating the complexity of the dataset. Table 5 shows the in-context learning performance of GPT4 on the BookSQL test set. DIN-SQL+GPT4 could only get 9.3% exact match accuracy, while Dynamic few-shot prompt+GPT4 comes close to the best-fine-tuned model, with exact match accuracy of 47.5% and execution accuracy of 67.2%. Table 6 shows the performance on easy, medium and hard queries. All models have a perfect performance (100% execution accuracy) on easy queries but struggle with medium and hard queries.

5.4 Error Analysis

We observed that SOTA models fail on queries with date filters, nested queries, distinct aggregations, and domain-specific filters. Table 7 shows the outputs of models on some examples from the test set. **DIN-SQL + GPT4** performs very poorly with execution accuracy of 7.6%. Perhaps, the reason for the bad performance is that it uses the same static chain-of-thought prompt, irrespective of the test question. BookSQL questions are very diverse and require domain knowledge. It is impossible to capture this diversity and domain knowledge in only a few examples in the prompt. Due to this, DIN-SQL fails whenever the test question is completely different from the examples provided in the prompt. **Dynamic few shot prompt + GPT4** model addresses the limitations of DIN-SQL by dynamically selecting a few shot examples for the prompt based on the test question. It significantly improves execution accuracy to 67.2%. Possibly, the reasons for poor performance are: 1) Getting confused between different columns (like `WHERE` clause on product_service vs. account column - see table 7), 2) Mixing up credit, debit, and amount columns and using incorrect columns in aggregations, 3) Not able to generate Nested SQL, even when required to answer the test question correctly, 4) failing when domain-specific information is required to generate SQL correctly. For example, transaction_type filters of the invoice, sales receipt, and purchase order, or account_type filters of expense, income, account receivable, and account payable are incorrectly applied.

SEDE fails to generate correct SQL, possibly due to a lack of question and schema linking in the input to the T5 model. Due to this, it mixes up different columns like customer, vendor, product_service, and account. **UniSAr** performs poorly, possibly due to complex queries introduced in BookSQL like date filters, nested queries, distinct aggregations, etc. UniSAr introduces constrained grammar-based decoding, which works well for simple queries but fails with such complex queries.

RESDSQL is the best-performing model. Poor performance is possibly due to: 1) Failure at complex time-based questions like *"What is average revenue for customer X in last 6 years"* (see table 7); 2) mixing up of credit and debit columns; 3) failure when distinct aggregations are required like `COUNT(DISTINCT transaction_id)`; 4) failure in case of many nested queries.

6 Future Directions

Results show the poor performance of the SOTA models on BookSQL. We outline some of the possible directions for the future to improve performance.

Multi-task learning: One could employ a multi-task learning setup, i.e., in addition to optimizing for SQL generation objective, adding other multi-task objectives could help improve the performance on hard SQL queries. These objectives could include (1) nested vs. non-nested SQL classification, (2) distinct keyword classification, and (3) date format classification.

Pre-training: For large databases, it is difficult for any model to relate the question tokens with column names when the question might refer to some table cell value. Before the Text-to-SQL task,

1	Question:	**What was the average invoice value for Biogenic municipal waste-fueled power generation?**
	Gold SQL:	**SELECT** avg(credit) **FROM** master_txn_table **WHERE** transaction_type = 'invoice' **AND** instr(account, 'Biogenic municipal waste-fueled power generation')
	Few-shot GPT4:	**SELECT** avg(amount) FROM master_txn_table **WHERE** transaction_type = 'invoice' **AND** product_service = 'Biogenic municipal waste-fueled power generation' ✗
	SEDE:	**SELECT** avg(credit) **FROM** master_txn_table **WHERE** transaction_type = 'invoice' AND instr(account, 'biogenic municipal waste-fueled power generation') ✓
	UniSAr:	**SELECT** avg(credit) **FROM** master_txn_table **WHERE** transaction_type = 'invoice' **AND** product_service = 'Biogenic municipal waste-fueled power generation' ✗
	RESDSQL:	**SELECT** avg(credit) **FROM** master_txn_table **WHERE** transaction_type = 'invoice' **AND** instr(account, 'biogenic municipal waste-fueled power generation') ✓
2	Question:	**What was the last invoice value for Drain cleaning in This week to date?**
	Gold SQL:	**SELECT** max(credit) **FROM** master_txn_table **WHERE** transaction_type = 'invoice' **AND** instr(account,'Drain cleaning') **AND** transaction_date **BETWEEN** date(current_date, 'weekday 0', '-7 days') AND date(current_date)
	Few-shot GPT4:	**SELECT** credit **FROM** master_txn_table **WHERE** transaction_type = 'invoice' **AND** product_service = 'Drain cleaning' **AND** transaction_date BETWEEN date(current_date, 'weekday 0', '-7 days') **AND** date(current_date) **ORDER BY** transaction_date DESC LIMIT 1 ✗
	SEDE:	**SELECT** max(credit) **FROM** master_txn_table **WHERE** transaction_type = 'invoice' **AND** customers = 'drain cleaning' **AND** transaction_date **BETWEEN** date(current_date, 'weekday 0', '-7 days') **AND** date(current_date) ✗
	UniSAr:	**SELECT** max(credit) **FROM** master_txn_table **WHERE** transaction_date **BETWEEN** date (current_date , 'weekday 0' , '-7 days') **AND** date (current_date) ✗
	RESDSQL:	**SELECT** max (credit) **FROM** master_txn_table **WHERE** transaction_type = 'invoice' **AND** instr (account , 'Drain cleaning') **AND** transaction_date **BETWEEN** date (current_date , 'weekday 0' , '-7 days') **AND** date (current_date) ✓
3	Question:	**What is my average revenue for Customer Nathan Hernandez in the last 6 years?**
	Gold SQL:	**SELECT** sum(credit)/6 **FROM** master_txn_table **WHERE** customers = 'Nathan Hernandez' **AND** strftime('%Y', transaction_date) >= strftime('%Y', current_date)-6-1
	Few-shot GPT4:	**SELECT** sum(credit)/6 **FROM** master_txn_table **WHERE** customers = 'Nathan Hernandez' **AND** strftime('%Y', transaction_date) >= strftime('%Y', current_date)-6-1 ✓
	SEDE:	**SELECT** avg(credit) **FROM** master_txn_table AS t1 JOIN chart_of_accounts AS t2 ON t1.account = t2.account_name **WHERE** account_type IN ('income','other income') **AND** customers = 'nathan hernandez' **AND** transaction_date **BETWEEN** date(current_date,'start of year','-6 year') **AND** date(current_date,'start of year','-1 day') ✗
	UniSAr:	**SELECT** avg(credit) **FROM** master_txn_table as T1 **JOIN** chart_of_accounts as T2 on T1.account = T2.account_name **WHERE** account_type in ('Income' , 'Other Income') **AND** customers = 'Nathan Hernandez' **AND** transaction_date **BETWEEN** date (current_date ,'start of year' , '-6 year') **AND** date (current_date ,'start of year' , '-1 day') ✗
	RESDSQL:	**SELECT** sum (credit) / 6 FROM master_txn_table **WHERE** customers = 'Nathan Hernandez' **AND** strftime ('%Y' , transaction_date) >= strftime ('%Y' , current_date) - 1 **GROUP BY** strftime ('%Y' , transaction_date) ✗

Table 7: Error analysis for different models on BookSQLtest set

one could do pre-training to better understand the question and table relationships. This can be done using mask modeling by defining tasks such as column recovery and column predictions where few tokens could be masked, and the model tries to recover and predict the masked tokens; a similar approach is proposed by Shi et al. (2020a) via the GAP model.

Multi-step few-shot prompting: One could also generate SQL in multiple steps using dynamic few-shot prompting instead of generating in a single step.

Value Encoding: In-context learning models (GPT4) mixes up different columns due to a lack of knowledge about table contents. Adding related table rows in the prompt could alleviate this issue.

7 Conclusion

In this paper, we propose BookSQL, a Text-to-SQL dataset that will have broad applications in the finance and accounting domain. The experimental outcomes of several Text-to-SQL models indicate considerable room for improvement. In the future, we aim to build a more robust model that can handle hard queries and improve performance.

Limitations

Since this is a resource paper, we release a large dataset and consequently focus less on modeling the Text-to-SQL system. We tested existing Text-to-SQL systems to see how well these fare on the new dataset. The results are indicative of considerable scope for improvement. In the future, we will focus on developing new models with better performance on BookSQL. Moreover, we hope that once the dataset is released, it will foster more research in this domain, resulting in more interesting models.

Ethics Statement

Considering the privacy aspect, we create anonymized entries in the dataset. Moreover, the dataset was verified by financial experts to make sure that the entries adhere to accounting principles and are reflective of real-life scenarios. We will be releasing the dataset publicly for research uses. To the best of our knowledge, we are not aware of any other possible ethical consequences of the proposed dataset.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Naihao Deng, Yulong Chen, and Yue Zhang. 2022. Recent advances in text-to-SQL: A survey of what we have and what we expect. In *COLING*, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Longxu Dou, Yan Gao, Mingyang Pan, Dingzirui Wang, Wanxiang Che, Dechen Zhan, and Jian-Guang Lou. 2022. Unisar: A unified structure-aware autoregressive language model for text-to-sql.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. 2018. Improving text-to-SQL evaluation methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360, Melbourne, Australia. Association for Computational Linguistics.

Moshe Hazoom, Vibhor Malik, and Ben Bogin. 2021a. Text-to-SQL in the wild: A naturally-occurring dataset based on stack exchange data. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*, pages 77–87, Online. Association for Computational Linguistics.

Moshe Hazoom, Vibhor Malik, and Ben Bogin. 2021b. Text-to-sql in the wild: a naturally-occurring dataset based on stack exchange data. *arXiv preprint arXiv:2106.05006*.

Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2261–2273, Online. Association for Computational Linguistics.

Fei Li and H. V. Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 8(1):73–84.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI)*.

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Chenhao Ma, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023b. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. pages 311–318.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. 2020a. Learning contextual representations for semantic parsing with generation-augmented pre-training.

Tianze Shi, Chen Zhao, Jordan Boyd-Graber, Hal Daumé III, and Lillian Lee. 2020b. On the potential of lexico-logical alignments for semantic parsing to SQL queries. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1849–1864, Online. Association for Computational Linguistics.

Ruoxi Sun, Sercan O Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2023. Sql-palm: Improved large language modeladaptation for text-to-sql. *arXiv preprint arXiv:2306.00739*.

Lappoon R. Tang and Raymond J. Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Proceedings of the 12th European Conference on Machine Learning*, pages 466–477, Freiburg, Germany.

Ping Wang, Tian Shi, and Chandan K. Reddy. 2020. Text-to-sql generation for question answering on electronic medical records. In *Proceedings of The Web Conference 2020*, WWW '20, page 350–361, New York, NY, USA. Association for Computing Machinery.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Sqlizer: Query synthesis from natural language. *Proc. ACM Program. Lang.*, 1(OOPSLA).

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

Appendix

A Dataset Details

- Table 8 shows master transaction table.
- Table 9 shows Chart of account table.
- Table 10 shows Customer table
- Table 11 shows Vendor table.
- Table 12 shows Employee table.
- Table 13 shows Product Service table.
- Table 14 shows Payment Methods table.

Distribution of Businesses in BookSQL The distributions of all businesses and their products are shown in Figure 4. Each industry is represented in the inner circle layer, which is followed by its businesses (in the middle circle) and its products in the outer circle. Each industry comprises multiple businesses, and each business consists of multiple products and services.

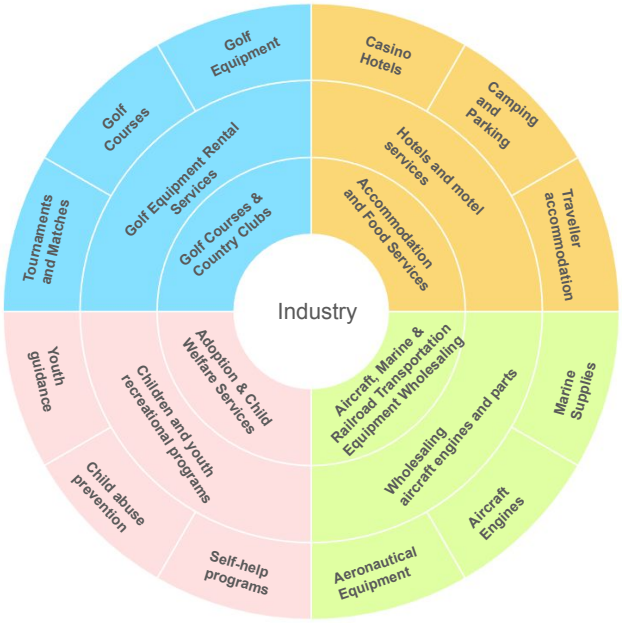

Figure 3: Sample BookSQL Business Distribution. The middle section shows the sample set of businesses, inner section shows the industries associated with the corresponding business and outer most section shows the corresponding product of the business. This chart is made with the information available at: https://www.ibisworld.com/united-states/list-of-industries/.

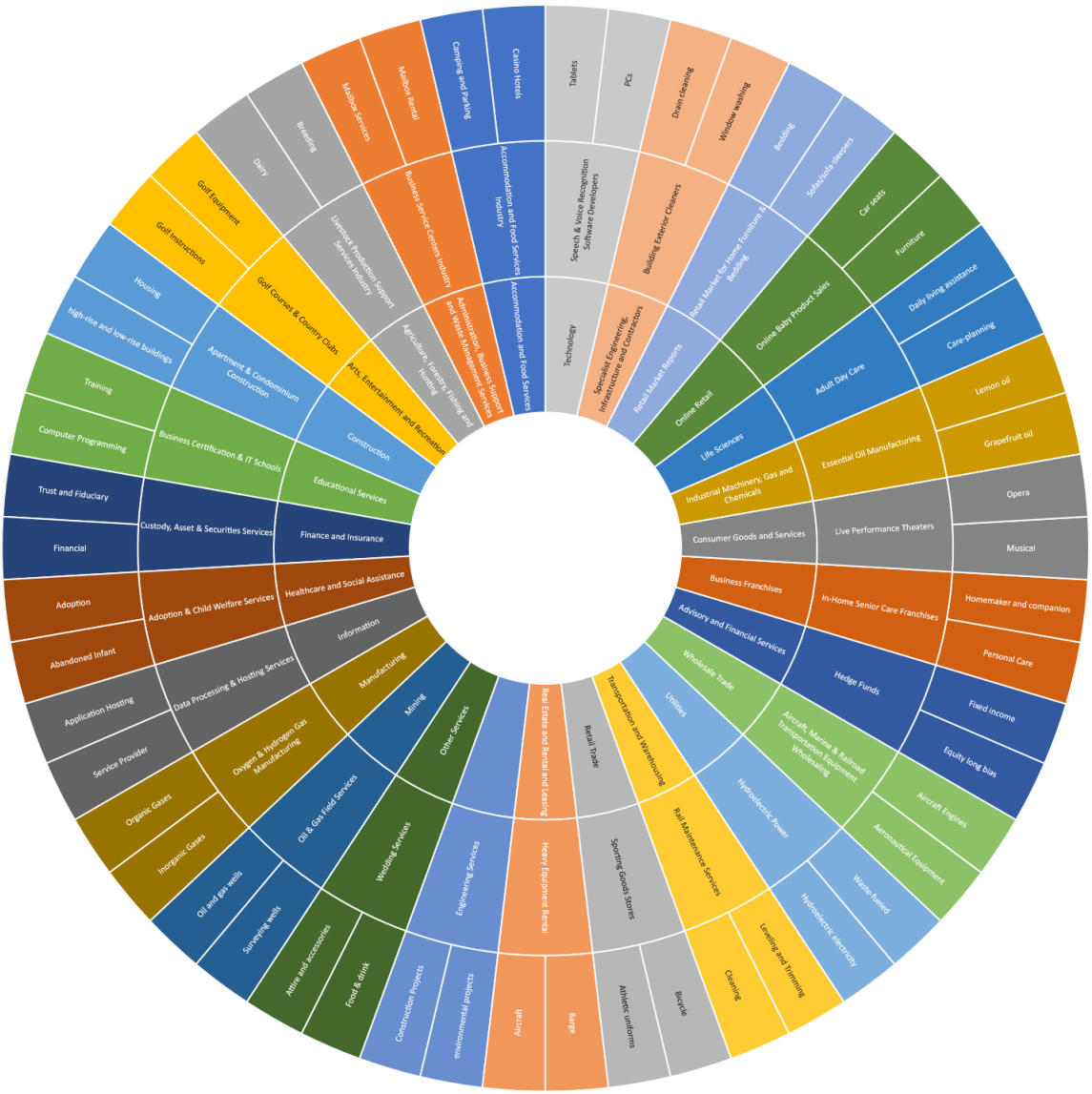

Figure 4: BookSQLBusiness Distribution. Here, inner circle indicates the industries , middle circle shows the sets of businesses associated to respective industry , and the outer most circle indicate corresponding product of the business. This chart is made with the information available at: https://www.ibisworld.com/united-states/list-of-industries/.

B Dynamic Few-shot Prompt + GPT4

Pseudo-code for dynamic few-shot train example selection for a given test question:

```
from langchain.embeddings.huggingface import HuggingFaceEmbeddings

from langchain.prompts.example_selector import MaxMarginalRelevanceExampleSelector

from langchain.vectorstores import Chroma

example_selector =
    MaxMarginalRelevanceExampleSelector.from_examples(
        examples,
        HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2"),
        Chroma,
        k=10,
        input_keys=["input"]
    )
```

B.1 Example Prompt

Database schema:
Table master_txn_table, columns = [, Transaction_ID, Transaction_DATE, Transaction_TYPE, Amount, CreatedDATE, CreatedUSER, Account, AR_paid, AP_paid, Due_DATE, Open_balance, Customers, Vendor, Product_Service, Quantity, Rate, Credit, Debit, payment_method, Misc]*
Table chart_of_accounts, columns = [, Account_name, Account_type] Table customers, columns = [*, customer_name, customer_full_name, Billing_address, Billing_city, Billing_state, Billing_ZIP_code, Shipping_address, Shipping_city, Shipping_state, Shipping_ZIP_code, Balance]*
Table employees, columns = [, Employee_name, Employee_ID, Hire_date, Billing_rate, Deleted]*
Table products, columns = [, Product_Service, Product_Service_type]*
Table vendors, columns = [, Vendor_name, Billing_address, Billing_city, Billing_state, Billing_ZIP_code, Balance]*
Table payment_method, columns = [, Payment_method, Credit_card]*

Foreign_keys = [master_txn_table.Account = chart_of_accounts.Account_name, master_txn_table.Customers = customers.customer_name, master_txn_table.Vendor = vendors.Vendor_name, master_txn_table.Product_Service = products.Product_Service, master_txn_table.payment_method = payment_method.payment_method]

Following are the example of questions and corresponding SQL queries.
«10 Few shot examples from train set»

Translate following question to SQL query.
Input: How much open credit does customer Ronald Bailey have?
Output: SELECT

C DIN-SQL+GPT4 Prompts

Following section shows the sample prompts used in different DIN-SQL modules. For brevity, we have added only 1 few shot example in these sample prompts. Though in practice, 5-10 few shot examples are used and is mentioned at the end of prompt in « ».

C.1 Schema Linking Prompt

Table master_txn_table, columns = [*, Transaction_ID, Transaction_DATE, Transaction_TYPE, Amount, CreatedDATE, CreatedUSER, Account, AR_paid, AP_paid, Due_DATE, Open_balance, Customers, Vendor, Product_Service, Quantity, Rate, Credit, Debit, payment_method, Misc]
Table chart_of_accounts, columns = [*, Account_name, Account_type]
Table customers, columns = [*, customer_name, customer_full_name, Billing_address, Billing_city,

Billing_state, Billing_ZIP_code, Shipping_address, Shipping_city, Shipping_state, Shipping_ZIP_code, Balance]
Table employees, columns = [*, Employee_name, Employee_ID, Hire_date, Billing_rate, Deleted]
Table products, columns = [*, Product_Service, Product_Service_type]
Table vendors, columns = [*, Vendor_name, Billing_address, Billing_city, Billing_state, Billing_ZIP_code, Balance]
Table payment_method, columns = [*, Payment_method, Credit_card]
Foreign_keys = [master_txn_table.Account = chart_of_accounts.Account_name, master_txn_table.Customers = customers.customer_name, master_txn_table.Vendor = vendors.Vendor_name, master_txn_table.Product_Service = products.Product_Service, master_txn_table.payment_method = payment_method.payment_method]

Q: How much open credit does customer Ronald Bailey have?
S: select sum(open_balance) from (select distinct transaction_id, open_balance from master_txn_table where customers = 'Ronald Bailey')
A: Let's think step by step. In the question "How much open credit does customer Ronald Bailey?", we are asked: "How much open credit", so we need column = [master_txn_table.open_balance] "open credit does customer Ronald Bailey", so we need column = [master_txn_table.transaction_id, master_txn_table.customers] Based on the columns and tables, we need these Foreign_keys = []. Based on the tables, columns, and Foreign_keys, The set of possible cell values are = [Ronald Bailey]. So the Schema_links are: Schema_links: [master_txn_table.open_balance, master_txn_table.customers, master_txn_table.transaction_id, Ronald Bailey]
«9 MORE FEW-SHOT EXAMPLES»

C.2 Classification prompt

Q: What are my transactions MTD?
schema_links: [master_txn_table.transaction_id, master_txn_table.amount, master_txn_table.transaction_date]
A: Let's think step by step. The SQL query for the question "What are my transactions MTD?" needs these tables = [master_txn_table], so we don't need JOIN. Plus, it doesn't require nested queries with (INTERSECT, UNION, EXCEPT, IN, NOT IN), and we need the answer to the questions = [""]. So, we don't need JOIN and don't need nested queries, then the the SQL query can be classified as "EASY".
Label: "EASY"

Q: How many products are never sold with total value higher than 5?
schema_links: [Product_Service.transaction_id, master_txn_table.transaction_type]
A: Let's think step by step. The SQL query for the question "How many products are never sold with total value higher than 5?" needs these tables = [Product_Service,master_txn_table], so we need JOIN. Plus, it requires nested queries with (INTERSECT, UNION, EXCEPT, IN, NOT IN) or inner query inside from clause, and we need the answer to the questions = ["products that are sold with total value higher than 5"]. So, we need JOIN and need nested queries, then the the SQL query can be classified as "NESTED".
Label: "NESTED"

Q: YTD, what was our smallest expense?
schema_links = [master_txn_table.account = chart_of_accounts.account_name, master_txn_table.credit, master_txn_table.transaction_date, master_txn_table.account_type, master_txn_table.debit]
A: Let's think step by step. The SQL query for the question "YTD, what was our smallest expense?" needs these tables = [master_txn_table,chart_of_accounts], so we need JOIN. Plus, it doesn't need nested queries with (INTERSECT, UNION, EXCEPT, IN, NOT IN), and we need the answer to the questions = [""]. So, we need JOIN and don't need nested queries, then the the SQL query can be classified as "NON-NESTED".
Label: "NON-NESTED"

«7 MORE FEW-SHOT EXAMPLES»

C.3 SQL Generation

C.3.1 Easy Prompt

Q: "How much open credit does customer Ronald Bailey?"
Schema_links: [master_txn_table.open_balance, master_txn_table.transaction_id, master_txn_table.customers,Ronald Bailey]
SQL: select sum(open_balance) from (select distinct transaction_id, open_balance from master_txn_table where customers = 'Ronald Bailey')

«4 MORE FEW-SHOT EXAMPLES»

C.3.2 Non-Nested Complex Prompt

Q: "How many Traveller accomodation did we sell to Ethan Walker today?"
Schema_links: [master_txn_table.quantity ,master_txn_table.customers, master_txn_table.product_service, master_txn_table.transaction_type, master_txn_table.transaction_date]
A: Let's think step by step. For creating the SQL for the given question, we need to join these tables = []. First, create an intermediate representation, then use it to construct the SQL query. Intermediate_representation: select sum(master_txn_table.quantity) from master_txn_table where master_txn_table.customers = 'Ethan Walker' and master_txn_table.product_service = 'Traveller accomodation' and master_txn_table.trasaction_type in ('invoice','sales receipt') and master_txn_table.transaction_date BETWEEN date(current_date) AND date(current_date)
SQL: select sum(quantity) from master_txn_table where customers = Ëthan Walkeränd product_service = T̈raveller accomodationänd trasaction_type in ('invoice','sales receipt') and transaction_date BETWEEN date(current_date) AND date(current_date)

«9 MORE FEW-SHOT EXAMPLES»

C.3.3 Nested Complex Prompt

Q: "How many products are never sold with total value higher than 5?" Schema_links: [master_txn_table.product_service, master_txn_table.transaction_type, master_txn_table.credit, product_service.*]
A: Let's think step by step. "How many products are never sold with total value higher than 5?" can be solved by knowing the answer to the following sub-question "Show me all the products which are never sold with total credit value higher than 5?". The SQL query for the sub-question "Show me all the products which are never sold with total credit value higher than 5?" is SELECT count(*) FROM Product_Service WHERE product_service NOT IN (SELECT product_service FROM master_txn_table WHERE transaction_type in ('invoice','sales receipt') group by product_service having sum(credit)>5)
So, the answer to the question "How many products are never sold with total value higher than 5?" is = Intermediate_representation: SELECT count(Product_Service.*) FROM Product_Service WHERE Product_Service.product_service NOT IN (SELECT master_txn_table.product_service FROM master_txn_table WHERE master_txn_table.transaction_type in ('invoice','sales receipt') group by master_txn_table.product_service having sum(master_txn_table.credit) > 5)
SQL: SELECT count(*) FROM Product_Service WHERE product_service NOT IN (SELECT product_service FROM master_txn_table WHERE transaction_type in ('invoice','sales receipt') group by product_service having sum(credit) > 5)

«9 MORE FEW-SHOT EXAMPLES»

C.4 Self Correction Prompt

For the given question, use the provided tables, columns, foreign keys, and primary keys to fix the given SQLite SQL QUERY for any issues. If there are any problems, fix them. If there are no issues, return the SQLite SQL QUERY as is.
Use the following instructions for fixing the SQL QUERY:
1) Use the database values that are explicitly mentioned in the question.

2) Pay attention to the columns that are used for the JOIN by using the Foreign_keys.
3) Use DESC and DISTINCT when needed.
4) Pay attention to the columns that are used for the GROUP BY statement.
5) Pay attention to the columns that are used for the SELECT statement.
6) Only change the GROUP BY clause when necessary (Avoid redundant columns in GROUP BY).
7) Use GROUP BY on one column only.

D Evaluation Metrics

The following standard metrics are used:

- **Exact Match Accuracy (Yu et al., 2018):** Both predicted and the Gold SQL are decomposed into different SQL components like SELECT, WHERE, GROUP BY, etc. Predicted SQL is marked as correct if all SQL components exactly match with the Gold SQL.
- **Execution Accuracy (Yu et al., 2018):** Output of predicted SQL is the same as Gold SQL's output on execution against the database.
- **Partial Component Match F1 (Hazoom et al., 2021a):** Both the predicted query and the gold query are parsed into tress using JSqlParser[12]. These two parsed trees are compared, and an aggregated score is calculated based on the number of matching sub-trees.
- **BLEU-4 (Papineni et al., 2002):** It measures the number of matching n-grams between the predicted and the Gold SQL.
- **ROUGE-L (Lin, 2004):** It is based on the longest common sub-sequence (LCS) between the predicted and the Gold SQL. A longer shared sequence indicates more similarity between the predicted and the Gold SQL.

E Training Details and Hyper-parameters

All experiments were done on a single NVIDIA A10G Tensor Core GPU.

For SEDE, we used T5-Large as the base seq-to-seq model, with a learning rate of $5e-5$ with 15 epochs and batch size of 6. For decoding, a beam size of 6 was used, with max decoding steps of 250.

For UniSAr, we use T5-Large as a base language model with a learning rate of 1e-5 and max tokens is 1024. We adopt the polynomial_decay with 5,000 warmup updates. The dropout rate is 0.1. Optimizer is Adam with the default parameters. The max-update is set to 10,000. Empirically, the model obtained the best performance about $10 \sim 15$ epochs in BookSQL. The Fairseq dynamically tunes the batch size to realize higher GPU utilization.

For RESDSQL, we used settings recommended by the original paper and code. The Schema Item Classifier module used a RoBERTa-large model with a learning rate of $1e-5$ and an effective batch size of 32 (using gradient accumulation). topk_table_num value of 4 and topk_column_num value of 8 were used. For the text2sql module, a T5-large model was used with a learning rate of $5e-5$ and an effective batch size of 32 (using gradient accumulation). Beam search decoding was used with num_beams set to 8 and num_return_sequences set to 8.

For DIN-SQL+GPT4 and Dynamic few shot prompt + GPT4, we used OpenAI GPT4 API with following settings: *n = 1, temperature=0.0, max_tokens=600, top_p = 1.0, frequency_penalty=0.0, presence_penalty=0.0*. Given limitations on the number of calls to OpenAI GPT4 API, we used a random 10% of BookSQL test set for GPT4-based approaches.

[12]https://github.com/JSQLParser/JSqlParser

business Id	Transaction ID	Transaction date	Transaction type	Amount	Created date	Created user	Account	A/R paid	A/P paid	Due date	Open balance	Customer name	Vendor name	Product Service	Quantity	Rate	Credit	Debit
4	1867	2022-08-31	credit card credit	999.58	2023-05-11	Joshua Hudson	Visa	-	-	2023-09-17	232.85	-	-	-	-	-	-	999.58
4	1867	2022-08-31	credit card credit	999.58	2023-05-11	Joshua Hudson	Savings	-	-	2023-09-17	232.85	-	-	-	-	-	999.58	-
4	1716	2022-08-15	Bill	784.19	2023-05-14	Joshua Hudson	Accounts Payable (A/P)	-	unpaid	2023-07-12	539.03	-	Jade Barnett	-	-	-	784.19	-
4	1716	2022-08-15	Bill	784.19	2023-05-14	Joshua Hudson	Lawyer	-	unpaid	2023-07-12	539.03	-	Jade Barnett	-	-	-	-	784.19
4	1818	2022-08-17	Payment	2204	2022-11-22	Joshua Hudson	prepaid expenses	paid	-	2023-06-25	1841.82	Andrew Rose	-	-	-	-	-	2204
4	1818	2022-08-17	Payment	2204	2022-11-22	Joshua Hudson	Accounts Receivable (A/R)	paid	-	2023-06-25	1841.82	Andrew Rose	-	-	-	-	2204	-

Table 8: Master Transaction Table

Business Id	Account name	Account Full Name	Account type
2	Accumulated Depreciation	Accumulated Depreciation	Fixed Asset
2	Furniture and Equipment	Furniture and Equipment	Fixed Asset
2	Payroll Liabilities	Payroll Liabilities	Other Current Liability
2	Opening Balance Equity	Opening Balance Equity	Equity
2	Owners Draw	Owners Draw	Equity
2	Owners Equity	Owners Equity	Equity
2	Accounting Service Income	Accounting Service Income	Income
2	Consulting Income	Consulting Income	Income
2	Tax Preparation Services Income	Tax Preparation Services Income	Income

Table 9: Chart of Account

Business Id	Customer name	Customer full name	Billing address	Billing city	Billing state	Billing ZIP code	Shipping address	Shipping city	Shipping state	Shipping ZIP code	Balance
2	Valerie Kline	Valerie Kline	5120 Shelia Valleys Suite 824	New Cynthiaburgh	AL	18662	40109 Pamela Extension	West Patrickville	TN	21599	67.32
2	Greg Cardenas	Greg Cardenas	59614 Margaret Roads	Transide	OK	72668	3758 Savage Garden Suite 126	Seanbury	VT	7317	167.00
2	Mr. Zachary Levy	Mr. Zachary Levy	30939 Brandon Ford Suite 571	South Joanntown	NV	71097	2752 Austin Brooks Suite 864	Stoutville	MI	51839	69.34
2	Taylor Hughes	Taylor Hughes	7945 Soto Point	Monicamouth	ND	46573	04225 Edwards Valley Suite 176	Taylorton	DE	516	169.34
2	Jodi Bishop	Jodi Bishop	850 Brent Parks	Shieldsberg	AL	7549	1558 Brown Hills	South Robert	ID	94105	799.37
2	Andrew Flores	Andrew Flores	3141 Jamie Isle Apt. 494	South Nicholasmouth	OH	71180	4662 Peters Parkways Suite 775	Desireebury	AR	34741	79.37
2	Earl Lee	Earl Lee	017 Lisa Skyway	Lake Kristinebburgh	AL	79642	8285 Thornton Motorway Suite 926	Lawsonville	ID	12448	84.75
2	Thomas Jackson	Thomas Jackson	09653 Christian Stravenue	North Johntown	MS	43479	1205 Shawna Fork Suite 756	Tracymouth	TX	77146	684.7
2	Jason Johnson	Jason Johnson	46474 Alan Cove Suite 685	Michaelside	VT	50177	Unit 0702 Box 5832	DPO	AA	15548	747.44
2	Craig Greer	Craig Greer	496 Moreno Brooks	Lake Katrinamouth	NM	14367	USNV Gutierrez	FPO	AP	71930	47.34
2	Jeffrey Fisher	Jeffrey Fisher	632 Robert Plains Apt. 260	Woodardville	LA	86396	46671 Joseph Flat Apt. 818	Sweeneyshire	DC	70691	5829.13

Table 10: Customer Table

Business Id	Vendor name	Billing address	Billing city	Billing state	Billing ZIP code	Balance
2	Shelly Ramos	82768 Dawn Crescent	West Cynthia	WY	39877	4042.15
2	Jade Barnett	782 Mitchell Camp Suite 676	Grahambury	KS	80370	12949.89
2	Nicole Jordan	14959 Mccullough Green Suite 029	East Kevinfurt	WI	42930	5294.89
2	Adam Pena	192 Brenda Gardens	Erinmouth	IA	93008	6949.89
2	Jeffrey Roman	784 Cameron Parks Apt. 902	North Gloriafurt	AR	48141	7299.89
2	Zachary Butler	61717 Christopher Cliffs Apt. 122	Port Joshua	MT	44164	465.09
2	Taylor Moses	19368 Jenny Courts Apt. 094	Kerristad	OR	25430	65.09
2	John Russo	Unit 6387 Box 0856	DPO	AA	73133	1538.8
2	Robert Phillips	USCGC Steele	FPO	AA	91533	55388.8

Table 11: Vendor Table

Business Id	Employee name	Employee ID	Hire date	Billing rate	Deleted
2	Stephanie Baker	STE123	07/17/2022	–	No
2	Julia Rivera	JUL456	07/31/2002	–	No
2	Valerie Kline	VAL232	04/15/2012	–	Yes
2	Greg Cardenas	GRE443	08/27/2013	–	No
2	Mr. Zachary Levy	ZAC998	01/28/2000	–	Yes
2	Taylor Hughes	TAY009	07/17/2022	–	Yes
2	Jodi Bishop	JOD778	12/27/2016	–	Yes
2	Andrew Flores	AND667	05/20/2018	–	No
2	Earl Lee	EAR221	08/19/2002	–	No

Table 12: Employee Tables

Business Id	Product_service	Product_Service_type
2	Hours	Service
2	Services	Service
2	Design	Service
2	Installation	Service
2	Lighting	Service
2	Maintenance & Repair	Service
2	Refunds & Allowances	Service

Table 13: Product Service Table

Business Id	Payment method	Credit card
1	Cash	No
1	Check	No
1	Visa	Yes
1	MasterCard	Yes
1	Discover	Yes
1	American Express	Yes
1	Diners Club	Yes

Table 14: Payment Methods