BEAVER: An Enterprise Benchmark for Text-to-SQL

Anonymous ACL submission

Abstract

Existing text-to-SQL benchmarks have largely been constructed from web tables with human-generated question-SQL pairs. LLMs typically show strong results on these benchmarks, leading to a belief that LLMs are effective at text-to-SQL tasks. However, how these results transfer to enterprise settings is unclear because tables in enterprise databases might differ substantially from web tables in structure and content. To contend with this problem, we introduce a new dataset BEAVER, the first enterprise text-to-SQL benchmark sourced from real private enterprise data warehouses. This dataset includes natural language queries and their correct SQL statements, which we collected from actual query logs. We then benchmark *off-the-shelf* LLMs on this dataset. LLMs perform poorly, even when augmented with standard prompt engineering and RAG techniques. We identify three main reasons for the poor performance: (1) schemas of enterprise tables are more complex than the schemas in public data, resulting in SQL-generation tasks intrinsically harder; (2) business-oriented questions are often more complex, requiring joins over multiple tables, aggregations, and nested queries; (3) public LLMs cannot train on private enterprise data warehouses that are not publicly accessible, and therefore it is difficult for the model to learn to solve (1) and (2). We believe BEAVER will facilitate future research in building text-to-SQL systems that perform better in enterprise settings.

1 Introduction

LLMs have shown potential for solving text-to-SQL tasks on existing datasets, such as Spider, KaggleDBQA, and Bird (Li et al., 2024; Sen et al., 2019; Yu et al., 2018; Lee et al., 2021). For example, on Spider, GPT-4 can achieve an execution accuracy above 85% (Gao et al., 2024). However, these datasets focus on tables collected from public sources and question-SQL pairs written by crowd-sourced annotators. As such, they do not represent real-world enterprise settings for the following reasons.

First, enterprise databases, typically designed for internal business use, often utilize more intricate schemas than tables from public datasets. Hence, understanding them may require database or business-specific knowledge. Public LLMs are mainly trained on public data. In contrast, enterprise data is private, which makes public LLMs lack access to such knowledge. Recent work (Kandpal et al., 2023) has shown that LLMs do not perform well on data domains they have never seen before. Consequently, public LLMs may not perform well on enterprise text-to-SQL tasks. As we will show later in this paper, they often generate queries that contain either *incorrect or insufficient columns*, and *invalid values*, in particular, in WHERE clauses.

Second, questions posed to enterprise databases are generally more complex than questions from public datasets. Public datasets are usually small and typically general-purpose. Questions from these datasets are often collected from annotators who are not enterprise users, database admins, or business analysts from specific data domains. For instance, the Spider dataset (Yu et al., 2018) was annotated by 11 computer science undergraduates. Therefore, the questions posed tend to be simple and may only involve one or two tables. In contrast, queries on enterprise databases typically involve joins and aggregates over multiple tables.

Third, enterprise databases often contain a large number of tables, rows, and columns. The scale of enterprise tables makes selecting the relevant tables for text-to-SQL even more challenging (Chen et al., 2024). These size issues are often absent from the public databases used for benchmarking text-to-SQL.

To study the above issues, we have curated a dataset BEAVER derived by anonymizing a subset of two real-world data warehouses. SQL statements were gathered from actual user query logs and reports, and corresponding natural language questions were formulated in collaboration with experienced database administrators. Specifically, we benchmarked recent *off-the-shelf* LLMs (including GPT-4o and Llama3.1-70B-Instruct) on BEAVER. These models achieved close to 0 end-to-end execution accuracy, demonstrating the challenging nature of our dataset. This illustrates that *off-the-shelf* LLMs trained on public datasets are unable to generalize to the same text-to-SQL tasks when presented with real data warehouse data.

In summary, our contributions are as follows: (1) We introduce BEAVER, the first enterprise text-to-SQL benchmark, for benchmarking text-to-SQL models under enterprise settings. This dataset includes tables from private and real enterprise data warehouses, annotated question-SQL pairs, and column mapping annotation for each question. LLMs powering current text-to-SQL systems are not trained on them. (2) We evaluate LLM-based text-to-SQL approaches on BEAVER and show their dramatically degraded performance, demonstrating the value of our benchmark for evaluation. (3) We provide an extensive error analysis that reveals why enterprise data and queries are challenging for LLMs. We then propose steps to address these challenges, informing future text-to-SQL systems that can perform better on enterprise data and queries.

2 Dataset

As described in Section 1, existing public datasets do not reflect enterprise data warehouses with high schema and query complexity. To study this issue, we have gathered datasets from two enterprise data warehouses and annotated them with real-world question-SQL pairs. We describe the text-to-SQL task and then provide details on the datasets and annotation.

2.1 Task Formulation

Following the standard problem setup of text-to-SQL, the input to an LLM includes a natural language question and a database of tables, and the output is a SQL statement whose execution answers the user's question. A database includes a set of tables. Each table includes a schema (that describes the names of columns and data types of each column) and instances of each table column.

2.2 Sources

The first data warehouse, called DW, contains 99 tables and 1544 columns from an existing Oracle data warehouse. These tables contain information on the physical administration of plants in a major university, including buildings, rooms, and their use, as well as age information and maintenance records. We collected 103 pairs of natural language questions and real-user SQL queries from this warehouse. An example user question is "*What are the building names, department names, building street addresses, total number of rooms, and total area of all rooms for the electrical engineering and computer science department and the material science and engineering department?*".

The second data warehouse, called NW, includes 366 tables and 2708 columns from five separate MySQL databases. These tables hold information on virtual machines and networking in an enterprise computing infrastructure and describe networking policies, virtual machine status, IP addresses, and virtual machine migrations. We gathered 100 pairs of real-user natural language questions and SQL queries from this warehouse. A sample question is "*Provide information (including security groups, system metadata, info caches, and metadata) about the instance [instance id] under project [project id].*".

2.3 Annotation

Databases. We retrieved table information directly from each database, including column names, column types, and rows.

SQL statements. To reflect the true complexity of queries posed on enterprise databases, we first collected real user query logs and reports from source organizations. We then extracted SQL statements from these real logs and reports.

Natural language questions. Four graduate students and two professional database administrators from the data warehouse support group collectively constructed natural language questions for the collected SQL statements. The students first collaboratively generated the natural language question for the corresponding SQL statement. Then, they passed these questions to the two database administrators for review. If some questions lacked clarity, they were sent back to the students for editing after discussion with the administrators. The above process repeated until both database administrators approved all questions.

Column mapping. To generate a correct SQL statement from a natural language question, models need to identify information mentioned in the user question (e.g., "*building names*" and "*material science and engineering department*") and map them to either table columns (e.g., column `BUILDING_NAME` in table `BUILDINGS`) or table instances (`'Materials Science and Eng'` in column `DEPARTMENT_NAME` in table `ORGANIZATION`). The former is called column mapping and the latter is instance mapping. As hinted in Section 1, high schema complexity makes column mapping and instance mapping challenging. Therefore, the students and administrators collectively annotated the column mapping. For each topic phrase (e.g., *building names*) mentioned in a user question, mappings to some appropriate table columns were annotated as a pair of (topic phrase, columns names). This annotation serves two purposes: (1) it can benchmark the ability of models to perform the task of column mapping, which is crucial to the quality of text-to-SQL, and (2) it can be provided to models to help improve their chance of generating correct SQL, as we will show in Section 3.4. However, instance mappings were not annotated due to considerable complexity (details can be found in Appendix C).

2.4 Statistics

Table 1 summarizes the domain, dataset size, table statistics, and query complexity of our dataset (DW and NW combined) as well as two popular open-source datasets: Spider (Yu et al., 2018) and Bird (Li et al., 2024). Similar to (Lan et al., 2023; Li et al., 2024), we measure query complexity along three dimensions: the average number of joins per query, which indicates the number of tables that need to be joined to include sufficient information to answer the user question; the average number of aggregations per query, which indicates the number of aggregation keywords such as `max`, `count`, `group by` that appear in a SQL statement; and the nesting depth which indicates how deep subqueries appear (e.g., `SELECT ... FROM (SELECT ...)` has a nesting depth of two). Compared to all existing datasets, BEAVER has the largest number of tables per database and the highest query complexity. Figure 1 visualizes the query complexity across all datasets and complexity dimensions.

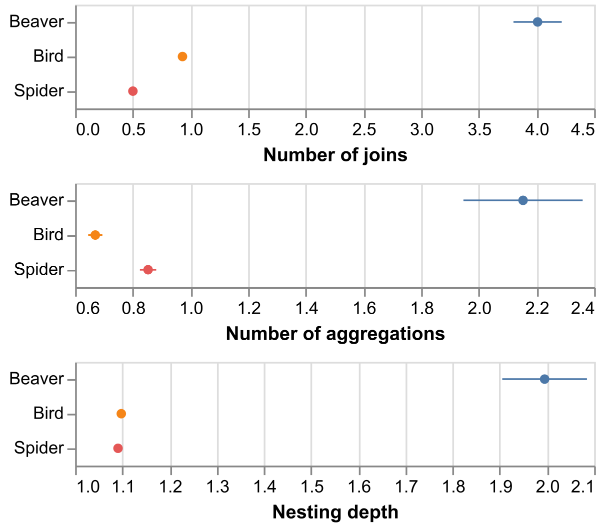

Figure 1: The mean values for the number of joins, aggregations, and nesting depth for Spider, Bird, and BEAVER.

3 Benchmark

In this section, we evaluate recent retrievers and LLMs on our dataset and existing public text-to-SQL datasets on table retrieval, column mapping, and SQL generation tasks. We then propose evaluation metrics for each task and analyze the results, linking them to the characteristics of an enterprise database mentioned in Section 1.

3.1 Experimental setup

Datasets. We evaluated our dataset separately on each database. However, doing so for Spider and Bird make them too simple compared with our dataset. As seen in Table 1, the average number of tables per database is significantly smaller on Spider and Bird compared to BEAVER. Therefore, we aggregated tables from all databases to construct a centralized database, resulting in 81 tables for Spider and 75 for Bird. This step ensures the table corpus sizes of Spider and Bird are comparable with our dataset (77.5 tables per database). For Spider and Bird, we still track the original databases of each table to evaluate SQL statements.

Retrieval-augmented Generation (RAG).
Table Retrieval. As seen in Table 1, the average number of tables and columns per database in previous datasets (Yu et al., 2018; Li et al., 2024) is small, averaging 4.05 tables per database and 5.44 columns per table in Spider and 6.82 tables per database and 10.6 columns per table in Bird. This makes it feasible to provide the schema information

Table 1: Domains, dataset size, table statistics, and query statistics of text-to-SQL datasets.

Dataset	Domain	#Queries	#DB	Avg. #Table/DB	Avg. #Cols/Table	Avg. #Joins/query	Avg. #Aggregation/query	Avg. Nesting depth/query
Spider (Dev)	Misc.	1034	20	4.05	5.44	0.506	0.854	1.09
Bird (Dev)	Misc.	1534	11	6.82	10.6	0.918	0.663	1.09
BEAVER	Facilities, computing infrastructure	203	6	77.5	9.14	4.01	2.15	2.0

for all tables in a database without exceeding the maximum context length of an input prompt. However, a key characteristic of enterprise databases is that they typically contain a large number of tables and a large number of columns per table, which makes it much more challenging to fit all this information into LLM's input prompt. Moreover, recent work shows that models might overlook some information in long prompts (Liu et al., 2024) and providing the schema information for fewer relevant tables can improve the performance in text-to-SQL tasks due to decreased noise (Chen et al., 2024).

A common method to enhance LLMs with knowledge from a large external data source is retrieval-augmented generation (Lewis et al., 2020). Following this approach, given the input of a user question and a database, instead of feeding the user question and the full schema information of the database directly to LLMs for SQL generation, an embedding-based retrieval system is first used to retrieve the top-k tables based on the semantic relatedness between the user question and the table schema. Relatedness is considered as the cosine similarity between the embedding of the user question and the table schema[1]. Embeddings are computed using recent retriever models, including UAE-Large-V1 (Li and Li, 2023), Stella_en_400M_v5 [2], and GTE-large-en-v1.5 (Li et al., 2023). Then, the schema of the top-k most relevant tables, along with the user question, are provided as input to the LLM to generate a SQL query answering the question.

SQL Generation. A SQL statement is generated given a user question and a set of tables. In particular, a table is represented as a string consisting of the table name, columns, and column types. As mentioned in Section 2.3, column mappings of each question-SQL pair were also annotated, which can be provided as input to models to test models' ability to generate SQL statements when provided with more hints. We adopted 1-shot prompting and performed this task on GPT-4o (Achiam et al., 2023) and Llama-3.1-Instruct (70B and 8B) (Touvron et al., 2023). Temperature (a random seed) was set to 0 to minimize randomness. Detailed prompts for SQL generation can be found in Appendix A.1.

Column mapping prediction. As discussed in Section 2.3, accurate column mappings are crucial for high-quality SQL generation, but they are difficult to achieve on enterprise databases. To quantify the difficulty, we benchmark the performance of models on the task of column mapping prediction. For a question-SQL pair, a column mapping is a list of (topic phrases, column names) pairs. We provided LLMs with the exact topic phrases from the gold column mappings to evaluate the models' ability to predict relevant columns based on topic phrases. Furthermore, we provided the tables used in the gold SQL statement and tasked the models with predicting a list of columns most relevant to each topic phrase. We adopted 1-shot prompting and evaluated this task on GPT-4o and Llama-3.1-Instruct (70B and 8B). Because Spider and Bird datasets do not provide column mappings, we randomly sampled 50 queries from each dataset and annotated the column mappings manually to serve as a comparison.

3.2 Evaluation metrics

Table retrieval. In a RAG setup, the quality of the retrieved tables significantly impacts the performance of SQL generation. To measure the retrieval performance, we compare the retrieved tables with the tables in the corresponding gold SQL statement (gold tables). The standard method for evaluating retrieval performance is computing precision, recall, and F1 @ top-k. However, these metrics may be insufficient. We notice that a SQL statement is unlikely to be generated correctly without all gold tables provided in the input. Therefore, in addition to the above metrics, we also measure the percentage of questions for which the top-k retrieved tables include all gold tables (denoted as PR).

[1] Table schema are serialized as space-separated strings of table names and column names.

[2] https://huggingface.co/dunzhang/stella_en_400M_v5

Table 2: Precision, Recall, F1 and Perfect-recall (PR) @ top-k across all datasets and recent embedding models.

	Top-5				Top-10			
	P	R	F1	PR	P	R	F1	PR
UAE-Large-V1								
Spider	29.1	96.4	43.5	94.6	14.9	98.6	25.4	97.9
Bird	34.8	91.3	49.1	82.6	18.9	97.5	31.1	94.5
BEAVER	28.1	36.0	30.3	7.9	19.7	48.3	26.9	12.3
Stella_en_400M_v5								
Spider	30.1	99.6	44.9	99.3	15.1	100	25.8	99.9
Bird	35.4	93.0	50.0	85.6	18.9	97.8	31.2	95.1
BEAVER	32.0	39.6	33.9	7.4	22.5	54.4	30.5	15.3
GTE-large-en-v1.5								
Spider	29.0	96.6	43.3	94.1	14.9	99.0	25.5	98.1
Bird	33.2	87.8	47.0	76.7	18.5	96.0	30.5	91.7
BEAVER	29.0	36.7	30.8	9.4	19.8	48.8	27.1	14.3

Table 3: 1-shot column mapping performance. Results are sampled on 50 queries from each dataset except BEAVER (full) which includes the performance on the entire dataset.

	Spider		Bird		BEAVER		BEAVER (full)	
	F1	Exact	F1	Exact	F1	Exact	F1	Exact
GPT-4o	80.8	64.0	75.9	50.0	59.6	6.0	55.4	6.8
Llama3.1-70B-It	80.7	66.0	74.0	48.0	61.0	6.0	60.7	5.8
Llama3.1-8B-It	72.1	56.0	63.8	34.0	42.8	4.0	42.6	2.9

SQL generation. Execution accuracy (Yu et al., 2018; Li et al., 2024) is used to evaluate the end-to-end performance of a predicted SQL statement. To calculate it, the predicted SQL statement s and the corresponding gold SQL statement s^* are executed, producing outputs o and o^*, respectively. The execution accuracy is 1 if o is the same as o^* and 0 otherwise.

Column mapping. We adopted two metrics to compare predicted column mappings and the gold mappings, exact score and F1 score. The exact score is 1 if the predicted column mapping is identical to the gold mapping and 0 otherwise. To give credit to partial matches, we further treat each pair of (topic phrase, column names) as a basic unit, which can then be used to compute F1 performance.

3.3 Overall performance

Table retrieval performance. From Table 2, we observe that precision, recall, F1 and PR @ top-k on BEAVER are the worst across all models and datasets. Average recall @ top-10 is 48.7 points lower on BEAVER compared to Spider and 46.6 points lower than Bird, across all retriever models. Average PR @ top-10 is 84.7 points lower on BEAVER compared to Spider and 79.8 points lower than Bird, across all retriever models. This indicates that accurately identifying the set of tables that contain the necessary information to answer a user question is significantly more challenging in the context of an enterprise database.

Table 4: 1-shot end-to-end execution accuracy across all datasets. Top-10 tables from the best-performing retriever model (*Stella_en_400M_v5*) were provided to the models.

	Spider	Bird	BEAVER
GPT-4o	69.5	30.9	0.0
Llama3.1-70B-It	60.3	25.8	0.0
Llama3.1-8B-It	51.1	13.8	0.0

Table 5: 1-shot execution accuracy on BEAVER when different hints are provided, across different models.

	Baseline	With gold tables	With gold tables and column mappings
GPT-4o	0.0	4.2	8.4
Llama3.1-70B-It	0.0	0.0	0.0
Llama3.1-8B-It	0.0	0.0	0.0

Column mapping performance. Table 3 shows the performance on column mapping. The first six columns show the performance of different models on 50 sampled queries from each of the three datasets. The last two columns show the performance of different models on the entire BEAVER dataset. Focusing on performance on the 50 sampled queries, average F1 on BEAVER is 23.4 points lower than Spider and 16.8 points lower than Bird and the average exact score on BEAVER is 56.7 points lower than Spider and 38.7 points lower than Bird. This quantitatively shows that identifying the set of correct columns is challenging on BEAVER, and much more difficult compared to both Spider and Bird. The low performance in terms of exact score (up to 6.0%) on BEAVER indicates that models, while capable of correctly mapping some keywords, struggle to accurately map all keywords in a question. The performance on the entire BEAVER dataset is also similar to the performance on the random sample, indicating the challenging nature of column mappings across the entire dataset.

End-to-end execution accuracy. As seen from Table 4, the end-to-end execution accuracy on BEAVER is the lowest across all datasets and models. None of the off-the-shelf LLMs can answer any question correctly, compared to an average performance of 60.3 on Spider and 23.5 on Bird, highlighting the challenging nature of BEAVER. The low accuracy can be due to poor table retrieval and column mapping performance as seen in Table 2 and 3. As we mentioned earlier, it is unlikely that a model can generate SQL correctly without all gold tables provided in the input. Not having the relevant table information in context also prevents the model from associating information in user questions with the correct columns needed to answer the question.

3.4 Analysis

As mentioned in Section 1, BEAVER differs from public text-to-SQL datasets in terms of (1) larger database size (2) higher schema complexity, and (3) higher query complexity. In this section, we want to show how each of these aspects affects LLM performance. Results after providing different gold information as hints are summarized in Table 5.

Providing gold tables increases performance. As seen in Table 5, providing models with the gold tables can significantly improve performance on the GPT-4o model compared to feeding it with tables from retriever models, which can include both noise (due to the presence of irrelevant tables) and insufficient information (due to lack of gold tables). This indicates that the large database size indeed makes the task more challenging.

Providing column mapping increases performance. We note that in Table 5, providing column mappings further improved performance, as seen in the 4.2% increase for GPT-4o (from column 2 to column 3). This indicates that a complex schema makes it challenging for models to perform column mapping. Therefore, providing gold information about such mapping can partially close this gap. However, we also note that providing column mapping cannot fully address the problem of schema mapping because instance mapping is not covered by column mapping.

Increased query complexity reduces performance. To understand how query complexity affects query performance, the number of correctly predicted SQL statements from GPT-4o (provided with both gold tables and column mappings) are shown against different buckets of query complexity (defined in Section 2.4), as shown in Table 6. We observe that as query complexity increases, the number of correctly predicted SQL statements decreases. This effectively shows that as complexity increases, fewer questions can be answered correctly, which means that high query complexity leads to a performance decrease in our dataset.

Table 6: Number of correctly answered questions over three buckets (0-4, 5-9, 9+) of each dimension of complexity.

Average number	0-4	5-9	9+
Join			
# total queries	84	17	1
# correct predictions	8	0	0
Aggregation			
# total queries	67	29	6
# correct predictions	4	3	1
Nesting depth			
# total queries	93	9	0
# correct predictions	7	1	0

4 Error analysis

In the above, we provided an overview of the performance of *off-the-shelf* LLMs on BEAVER, indicating their limited capabilities of performing text-to-SQL in a real-world enterprise setting. Here, we discuss in detail the error sources during both table retrieval and SQL generation phases by examining randomly sampled 50 questions from our dataset. For table retrieval, we examined the performance of the best-performing retriever model (*Stella_en_400M_v5*). For SQL generation, we inspected the performance of the best-performing LLM (*GPT-4o*).

Table 7: Common error types encountered in table retrieval and SQL generation tasks for retriever and LLM models, respectively.

Error types	% questions
Table retrieval (Stella_en_400M_v5)	
Not retrieving sufficient information	89.1
Misses connecting tables	6.52
Cannot handle domain-specific information	4.38
SQL generation (GPT-4o)	
Incorrect column mapping	59.1
Incorrect instance mapping	22.7
Unable to handle complex queries	27.3
Misses implicit assumptions	50.0

4.1 Table retrieval analysis

As seen in the top half of Table 7, the retriever model made three major mistakes during table retrieval. Firstly, *the retrieval model may not retrieve the set of tables with sufficient information to answer the user question*. For instance, given the user question "*What is the name of the building and fee of the shortest sessions?*" and a table corpus including the three tables shown in Figure 2, the retriever model retrieved table `SUBJECT_SESSION` to cover "shortest and longest sessions", and table `SUBJECT_DETAIL` to cover "fee". However, the model did not retrieve table `BUILDINGS` to cover "name of the building".

Secondly, *the retrieval model can miss connecting tables*. This occurs when models retrieved a set of tables that can cover information in the user question, but they might not be connected through join relationships, so other tables need to be used to connect these tables. For instance, given the user question "*What is the building name that accommodates the most students?*" and a table corpus including the three tables shown in Figure 3, the retriever model retrieved `FCLT_BUILDING` and `STUDENT_DIRECTORY` to cover "building name" and "students" respectively. However, these two tables can only be joined via `FCLT_ROOMS`, which was not retrieved. This shows that models are not necessarily aware of join relationships during retrieval, which leads to information not being connected.

Lastly, *retrieval models may not be able to retrieve correct tables if domain-specific information is involved.* For example, given the user question "*List the name of mailing lists, and name of the faculty who teaches in 2023 fall.*" that requires information from tables `MOIRA_LIST`, `MOIRA_LIST_DETAIL`, and `SUBJECT_OFFERED`, the retriever model only retrieved the table `SUBJECT_OFFERED`, but was unable to retrieve the other two tables that are related to "moira list", potentially because it does not know that "moira" is the name of the system used to manage mailing lists.

These behaviors suggest that existing retriever models struggle to retrieve relevant tables for a user question in the enterprise setting.

4.2 SQL generation analysis

As seen in the bottom half of Table 7, models made four major mistakes in SQL generation. Firstly, *models can map topics mentioned in user questions to incorrect columns (i.e., incorrect column mapping)*. For instance, given the user question "*What are the building names and building street addresses for the computer science department?*", GPT-4o mapped "*building street address*" to the column `BUILDING_ADDRESS`. However, GPT-4o is not aware that the same building can have multiple addresses for different purposes (e.g., street, mail, package), and thus failed to also map this topic to the column `ADDRESS_PURPOSE` and instance `'STREET'`. Column mapping also fails when user questions are vague. For instance, when network administrators pose questions like "*Provide information (including info on caches and security groups) for the virtual machine with ID [id].*", they would like to gather as much information as possible to perform diagnosis and monitoring. Therefore, the gold SQL statement is very comprehensive, whereas GPT-4o only predicted a few columns. The full example can be found in Appendix B.1.

Secondly, *models can map literals mentioned in user questions to incorrect instances (i.e., incorrect instance mapping)*. For example, given a user question "*What is the total fee for all virtual sessions?*", GPT-4o associated the literal "virtual" with column `SESSION_LOCATION` and instance `'Virtual'`. While the column mapping is correct, the instance mapping is incorrect because `SESSION_LOCATION` includes multiple instances that represent virtual locations (e.g., `'online'`, `'webinar'`, `'remote'`, `'online via zoom'`), so the model would need to associate "virtual" with all these different instances or explore a more efficient filter for virtual locations.

Thirdly, models can fail to derive the correct SQL syntax when queries are complex. For instance, given the user question "For each course, list the cumulative number of courses held in the same year or preceding years.", the correct approach is to `partition` courses by year, sort courses by year, and restrict courses to those that have the same year or before using the function `range between unbounded preceding and current row`. However, GPT-4o was not able to use the window function in its predicted SQL statement.

Finally, models cannot reflect implicit assumptions in SQL statements. For instance, when users pose questions like "*Provide information about virtual machines with ID [id].*", by default, they only want to know the information

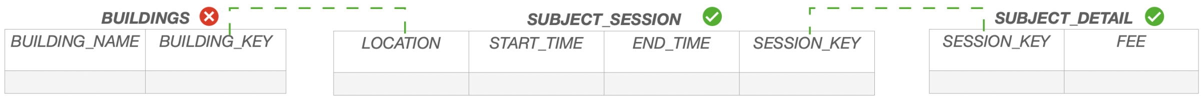

Figure 2: Schema of tables to illustrate retriever models did not retrieve sufficient information. A green tick means the table was retrieved, and a red cross means the table was not retrieved. Green dotted lines represent join relationships.

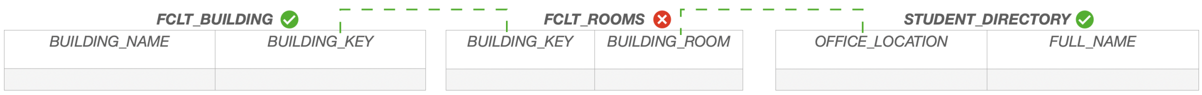

Figure 3: Schema of tables to illustrate retriever model did not retrieve connecting tables.

about *active* instances (i.e., not deleted). As such, the gold SQL statement includes the predicate `instances.deleted = 0`. However, GPT-4o was not able to recover this implicit assumption (and thus the predicate) in its SQL statement.

Overall, the error analysis highlights that retrieving relevant tables from a large corpus, performing schema mapping (both column mapping and instance mapping), and understanding complex queries are big challenges for models to solve enterprise-level text-to-SQL in an end-to-end fashion. Moreover, models might also need to deal with ambiguity and implicit assumptions in user questions.

5 Discussion and future directions

5.1 Column semantics

As seen in Section 4, LLMs perform poorly on tasks that require a holistic understanding of each column and its instances, such as schema mapping (including both column mapping and instance mapping). A straightforward approach involves feeding all table rows to LLMs to handle schema mapping. While feasible for small tables, handling large tables with billions of rows and terabytes of data presents significant challenges due to LLMs' input context size limit. Moreover, processing a large number of rows can introduce significant efficiency issues.

5.2 Verbosity level of user questions

Questions in public text-to-SQL datasets tend to be very verbose, containing information about every column that needs to be in the SQL statement. This makes an automatic and standardized evaluation based on the outputs of SQL statements possible. However, in the enterprise setting, different users have different standards of verbosity level, which encourages us to think about the next appropriate task formulation of text-to-SQL. As seen in Section 4, network administrators managing computing databases are highly knowledgeable about the underlying database systems. As a result, their queries often take the form of "*Show me information about an instance with ID [id],*" without explicitly defining the exact information required. These queries can also include implicit assumptions. For instance, users may assume that only active instances are of interest and thus exclude terminated or deleted cases, even if this is not explicitly stated in their user questions. This encourages human-in-the-loop iterative solutions that can propose clarifying questions and refine their outputs based on continuous human feedback.

6 Conclusion

Text-to-SQL is essential to bridging the gap between natural language question answering and table querying. The performance of off-the-shelf LLMs on existing text-to-SQL benchmarks seems to suggest strong performance. However, these benchmarks do not reflect real-world enterprise settings and thus do not reflect the performance of LLMs on enterprise queries over enterprise databases. The enterprise setting differs from existing public settings as it includes unseen domain-specific knowledge, a large number of tables that require an intermediate retrieval stage, and higher levels of query and schema complexity. Our results show that enterprise queries bring significant challenges to off-the-shelf models regarding table retrieval and SQL generation. We hope this paper serves as the foundation for future work that examines large-scale and complex text-to-SQL.

7 Ethics

As mentioned in Section 2.3, we recruited four graduate students and two professional database administrators to perform the annotations. We ensure fair compensation for each volunteer, considering the minimum salary of the region these volunteers are in. Because this dataset involves only factual annotations, no subjective opinions or personal information were collected, and thus, it should pose minimal risks to annotators and the general public. All database contents and questions will be anonymized according to rules set by the private organizations before releasing them to the public.

8 Limitations

Privacy and legal considerations restricted our access to private databases, limiting the diversity of domains represented in our dataset. Furthermore, in order to collect real SQL statements, we focused on query logs and reports. However, interpreting the intent of the SQL queries was difficult, making the generation of precise natural language questions a slow process. We plan to continue expand number of queries in our dataset in the future.

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*.

Peter Baile Chen, Yi Zhang, and Dan Roth. 2024. Is table retrieval a solved problem? exploring join-aware multi-table retrieval. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2687–2699, Bangkok, Thailand. Association for Computational Linguistics.

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

Nikhil Kandpal, Haikang Deng, Adam Roberts, Eric Wallace, and Colin Raffel. 2023. Large language models struggle to learn long-tail knowledge. In *International Conference on Machine Learning*, pages 15696–15707. PMLR.

Wuwei Lan, Zhiguo Wang, Anuj Chauhan, Henghui Zhu, Alexander Li, Jiang Guo, Sheng Zhang, Chung-Wei Hang, Joseph Lilien, Yiqun Hu, Lin Pan, Mingwen Dong, Jun Wang, Jiarong Jiang, Stephen Ash, Vittorio Castelli, Patrick Ng, and Bing Xiang. 2023. Unite: A unified benchmark for text-to-sql evaluation. *Preprint*, arXiv:2305.16265.

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.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

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.

Xianming Li and Jing Li. 2023. Angle-optimized text embeddings. *arXiv preprint arXiv:2309.12871*.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Jaydeep Sen, Fatma Ozcan, Abdul Quamar, Greg Stager, Ashish Mittal, Manasa Jammi, Chuan Lei, Diptikalyan Saha, and Karthik Sankaranarayanan. 2019. Natural language querying of complex business intelligence queries. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1997–2000.

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

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. *arXiv preprint arXiv:1809.08887*.

A Prompts

A.1 1-shot prompt for SQL generation

We use the following 1-shot prompt for end-to-end SQL generation.

B Examples for error analysis

B.1 Column mapping for vague questions

User question: "*Provide information (including info caches, and security groups) for these VMs f5a08397-5aac-44b4-b359-f03ff6ce228a, e7c1acd1-6a47-4a08-8601-5022d4d50aa7.*"

Gold SQL statement:

```sql
SELECT instances.created_at AS
instances_created_at,
instances.updated_at AS
instances_updated_at,
instances.deleted_at AS
instances_deleted_at, instances.deleted
AS instances_deleted, instances.id AS
instances_id, instances.user_id AS
instances_user_id, instances.project_id
AS instances_project_id,
instances.image_ref AS
instances_image_ref, instances.kernel_id
AS instances_kernel_id,
...
security_groups_1.project_id AS
security_groups_1_project_id
FROM instances LEFT OUTER JOIN
instance_info_caches AS
instance_info_caches_1 ON
instance_info_caches_1.instance_uuid =
instances.uuid LEFT OUTER JOIN
(security_group_instance_association AS
security_group_instance_association_1
INNER JOIN security_groups AS
security_groups_1 ON

security_groups_1.id =
security_group_instance_association_1.security_group_id
AND
security_group_instance_association_1.deleted
= 0 AND security_groups_1.deleted = 0)
ON
security_group_instance_association_1.instance_uuid
= instances.uuid AND instances.deleted =
0
WHERE instances.uuid IN
('f5a08397-5aac-44b4-b359-f03ff6ce228a',
'e7c1acd1-6a47-4a08-8601-5022d4d50aa7')
ORDER BY instances.created_at DESC,
instances.id DESC
```

Predicted SQL statement by GPT-4o

```sql
SELECT
    instances.uuid AS instance_uuid,
    instances.display_name,
    instances.vm_state,
    instances.host,
    instance_info_caches.network_info,
    security_groups.name AS
    security_group_name,
    security_groups.description AS
    security_group_description
FROM
    instances
LEFT JOIN
    instance_info_caches ON
    instances.uuid =
    instance_info_caches.instance_uuid
LEFT JOIN
    security_group_instance_association
    ON instances.uuid =
    security_group_instance_association.instance_uuid
LEFT JOIN
    security_groups ON
    security_group_instance_association.security_grou
    = security_groups.id
WHERE
    instances.uuid IN (

        'f5a08397-5aac-44b4-b359-f03ff6ce228a',

        'e7c1acd1-6a47-4a08-8601-5022d4d50aa7'
    )
```

C Complexity of instance mapping

Consider the user question "*List the long building names constructed before 1950 that have more than 100 employees and the built year and number of employees.*" which has a gold SQL statement of

```sql
SELECT * FROM (SELECT DISTINCT
    a.BUILDING_NAME_LONG, a.year_built,
    COUNT(distinct
    employee_directory.ID) OVER
    (PARTITION BY a.BUILDING_NAME_LONG,
    a.year_built) as num_employees
FROM (SELECT * FROM (SELECT DISTINCT
    FCLT_BUILDING_KEY,
    BUILDING_NAME_LONG, extract(year
    FROM TO_DATE(date_built,
    'MM/DD/YYYY')) as year_built FROM
    wareuser.fclt_building_hist) WHERE
    year_built < 1950) a JOIN fclt_rooms
    ON fclt_rooms.FCLT_BUILDING_KEY =
    a.FCLT_BUILDING_KEY JOIN
    employee_directory ON
    employee_directory.OFFICE_LOCATION =
    fclt_rooms.BUILDING_ROOM ) WHERE
    num_employees > 100;
```

In this case, the literal "*100 employees*" should be mapped to

```sql
COUNT(distinct employee_directory.ID)
    OVER (PARTITION BY
    a.BUILDING_NAME_LONG, a.year_built)
    > 100
```

which involves one grouping and aggregation.

The literal "before 1950" should be mapped to

```sql
extract(year FROM TO_DATE(date_built,
    'MM/DD/YYYY'))  < 1950
```

which involves one custom function call.

As seen above, compared to column mappings, instance mapping is considerably more complex and much harder to evaluate. Therefore, instance mappings were not annotated.

Table 8: 1-shot prompt for generating SQL. <tables> and <question> refer to the new set of tables and the new user question given at inference time. LLMs should leverage both the 1-shot example and the new input to complete the SQL statement after “SQL:”.

```
Given the question and tables, output the SQL statement that can answer the question correctly. You should only output the SQL statement.

CREATE TABLE SUBJECT_OFFERED(
  SUBJECT_KEY VARCHAR2,
  SUBJECT_OFFERED_SUMMARY_KEY VARCHAR2,
  MASTER_SUBJECT_KEY VARCHAR2,
  COMPOSITE_SUBJECT_KEY VARCHAR2,
  TERM_CODE VARCHAR2,
  MASTER_COURSE_NUMBER VARCHAR2,
  MASTER_COURSE_NUMBER_SORT VARCHAR2,
  MASTER_COURSE_NUMBER_DESC VARCHAR2,
  MASTER_SUBJECT_ID VARCHAR2,
  MASTER_SUBJECT_ID_SORT VARCHAR2,
  COURSE_NUMBER VARCHAR2,
  COURSE_NUMBER_SORT VARCHAR2,
  COURSE_NUMBER_DESC VARCHAR2,
  SUBJECT_ID VARCHAR2,
  SUBJECT_ID_SORT VARCHAR2,
  SUBJECT_TITLE VARCHAR2,
  SECTION_ID VARCHAR2,
  IS_MASTER_SECTION VARCHAR2,
  IS_LECTURE_SECTION VARCHAR2,
  IS_LAB_SECTION VARCHAR2,
  IS_RECITATION_SECTION VARCHAR2,
  IS_DESIGN_SECTION VARCHAR2,
  OFFER_DEPT_CODE VARCHAR2,
  OFFER_DEPT_NAME VARCHAR2,
  OFFER_SCHOOL_NAME VARCHAR2,
  RESPONSIBLE_FACULTY_NAME VARCHAR2,
  RESPONSIBLE_FACULTY_ID VARCHAR2,
  MEET_TIME VARCHAR2,
  MEET_PLACE VARCHAR2,
  CLUSTER_TYPE VARCHAR2,
  CLUSTER_TYPE_DESC VARCHAR2,
  CLUSTER_LIST VARCHAR2,
  HGN_CODE VARCHAR2,
  HGN_CODE_DESC VARCHAR2,
  FORM_TYPE VARCHAR2,
  FORM_TYPE_DESC VARCHAR2,
  SUBJECT_ENROLLMENT_NUMBER NUMBER,
  SECTION_ENROLLMENT_NUMBER VARCHAR2,
  CLUSTER_ENROLLMENT_NUMBER NUMBER,
  EVALUATE_THIS_SUBJECT VARCHAR2,
  IS_OSE_SUBJECT VARCHAR2,
  IS_CREATED_BY_DATA_WAREHOUSE VARCHAR2,
  SUBJECT_GROUPING_KEY VARCHAR2,
  WAREHOUSE_LOAD_DATE DATE,
  NUM_ENROLLED_STUDENTS NUMBER,
  SUBJECT_SUMMARY_KEY VARCHAR2,
  IS_REPEATABLE_SUBJECT VARCHAR2,
  PRIMARY KEY (SUBJECT_KEY),
  FOREIGN KEY (SUBJECT_OFFERED_SUMMARY_KEY) REFERENCES SUBJECT_OFFERED_SUMMARY (SUBJECT_OFFERED_SUMMARY_KEY),
  FOREIGN KEY (SUBJECT_ID) REFERENCES COURSE_CATALOG_SUBJECT_OFFERED (SUBJECT_ID)
)
Question: How many distinct subjects are being offered?
SQL: SELECT COUNT(DISTINCT SUBJECT_KEY) FROM SUBJECT_OFFERED;


<tables>
Question: <question>
SQL:
```