ASSESSING SEVERAL LOCAL LLMS FOR A TEXT TO SQL SOLUTION

OLTI QIRICI

Department of Informatics, Faculty of Natural Sciences,

University of Tirana, Albania

e-mail: olti.qirici@fshn.edu.al

Abstract

This paper evaluates the feasibility of using local large language models (LLMs) to perform Text-to-SQL generation under limited computational resources. The study focuses on a small star-schema data mart and investigates whether locally deployed, pre-trained LLMs can correctly translate natural-language queries into executable SQL statements without relying on cloud-based services. Six local LLMs from different providers and model families were evaluated, including two Microsoft Phi models, three DeepSeek models, and one Mistral model, covering a range of model sizes and parameter counts. All models were assessed using prompt-based inference only, without additional training or fine-tuning, to reflect realistic constraints on hardware, time, and resources. The evaluation compares the models in terms of query correctness and response time. The results highlight the strengths and limitations of current local LLMs for Text-to-SQL tasks and provide practical insights into their suitability for on-premises and privacy-sensitive analytical environments.

Key words**:** *Generative AI, local LLM, text to SQL, models comparing.*

Përmbledhje

Ky punim vlerëson mundësinë e përdorimit të modeleve lokale të gjuhëve të mëdha (LLM) për të kryer gjenerimin e SQL-së nga teksti në presence të burimeve të kufizuara llogaritëse. Studimi fokushet në një datamart të të dhënave me skemë yll dhe vlerëson nëse LLM-të e para-trajnuar dhe të vendosur në nivel lokal mund të përkthejnë saktë kërkesat prej gjuhëve natyrore në SQL të ekzekutueshme pa u mbështetur në shërbime të bazuara në cloud. U vlerësuan gjashtë LLM lokale nga ofrues të ndryshëm dhe familje modelesh, duke përfshirë dy modele Microsoft Phi, tre modele DeepSeek dhe një model Mistral, që mbulojnë një gamë të madhësive të modelit dhe

numërimit të parametrave. Të gjitha modelet u vlerësuan duke përdorur vetëm inxhenierim me prompt, pa trajnim shtesë ose rregullime të hollësishme, për të reflektuar kufizime realiste në harduer, kohë dhe burime. Vlerësimi krahason modelet për sa i përket saktësisë së SQL-së së gjeneruar dhe kohës së përgjigjes. Rezultatet nxjerrin në pah pikat e forta dhe kufizimet e LLM-ve lokale aktuale për detyrat e konvertimit të tekstit në SQL dhe ofrojnë njohuri praktike mbi përshtatshmërinë e tyre për mjedise analitike lokale dhe të ndjeshme ndaj privatësisë.

Fjalë kyçe: *AI Gjeneruese, LLM Lokal, tekst në SQL, krahasim modelesh.*

Introduction

The rapid technological disruption driven by generative artificial intelligence (GenAI) has led to a growing number of services that rely on large language models (LLMs). Most of these services depend on cloud-based APIs provided by major LLM vendors. GenAI has fundamentally changed problem-solving approaches, enabling solutions to challenges that previously appeared difficult or impractical.

Many organizations have enhanced their software systems beyond traditional rule-based or programmatic functionality by integrating remote AI services. These systems typically transmit user inputs to external LLM providers, which process the requests and return results in machine- or human-readable form. While this paradigm has expanded the capabilities of numerous applications, it also introduces dependencies on external infrastructure and third-party service providers.

Although generative AI is often described as a democratizing technology, this democratization largely applies to the use of human-generated knowledge, rather than to control over the underlying models. Access to high-performance LLMs remains concentrated among a limited number of organizations, and the internal mechanisms governing model training, data retention, and inference behavior are generally opaque. This lack of transparency has contributed to ongoing legal disputes concerning intellectual property and data ownership.

From a data privacy and security perspective, the continuous transmission of potentially sensitive information to remote LLM services raises significant concerns. In particular, it remains unclear whether submitted data may be retained, reused, or indirectly influence future model behavior. These concerns are amplified by geopolitical considerations, as most widely used LLMs are operated by companies based in a small number of jurisdictions. In contexts

involving sensitive or confidential data—especially analytical workloads—the associated risks become even more pronounced.

A practical and trusted alternative to cloud-based inference is the deployment of local LLMs within fully isolated environments, where data remains on-premise and is never transmitted to external services. This approach is particularly relevant for data analysis tasks, where confidentiality, compliance, and governance requirements are often stringent.

Prior research has demonstrated the feasibility of local LLMs across multiple domains. In the social sciences, local LLMs have achieved performance comparable to human experts for classification and information extraction tasks involving sensitive child welfare data (Perron et al., 2024). In medical applications, researchers have shown that while local LLMs may not yet match the capabilities of large commercial models, they nonetheless represent viable and actionable alternatives under technical and policy constraints (Bumgardner et al., May 2024). Additional studies have confirmed the effectiveness of local, open-source LLMs for querying and interpreting medical reports (Vaid et al., Sep 2024), as well as for scientific literature understanding through adaptable fine-tuning frameworks (Li et al., 2025).

Motivated by these findings, this study investigates the applicability of local LLMs to Text-to-SQL conversion. A simplified star-schema data mart was designed to represent a controlled analytical environment, and prompt engineering techniques were applied to translate natural-language queries into executable SQL statements.

The remainder of this paper presents the methodology, experimental results, and conclusions of this evaluation. As anticipated, the findings indicate that while local LLMs demonstrate promising capabilities for small-scale and well-structured problems, their current performance and resource requirements limit their applicability to more complex analytical scenarios.

Methodology: Data mart creation for the experiment

To address the proposed problem, the study began with the design of a star-schema data mart. The choice of a data warehouse–oriented structure, and specifically a star (or snowflake) schema, was deliberate and motivated by both practical and methodological considerations.

Data warehouses are typically characterized by well-structured, curated, and cleansed data, explicitly designed for analytical and business intelligence use cases. As a result, table and column names are generally more consistent,

meaningful, and semantically expressive, reducing the need for additional schema descriptions when performing automated query generation. This property is particularly advantageous for Text-to-SQL tasks.

More importantly, star schemas provide a clear and unambiguous structural organization, with a single central fact table enriched by multiple dimension tables connected through foreign-key–primary-key relationships. This design implicitly defines a unified analytical view of the data, similar to the semantic models employed by business intelligence platforms such as Oracle Analytics Server (Oracle, 2024). By enforcing this structure, the risk of ambiguous joins is significantly reduced.

In contrast, more complex schemas containing multiple fact tables connected through shared dimensions can introduce substantial ambiguity. In such cases, a model may incorrectly join fact tables via a common dimension, resulting in unintended many-to-many relationships and record multiplication, ultimately producing erroneous query results. The star-schema design adopted in this study avoids this pitfall by construction. This constraint reflects a common design principle in commercial business intelligence systems, many of which support only star or snowflake schemas within their logical models.

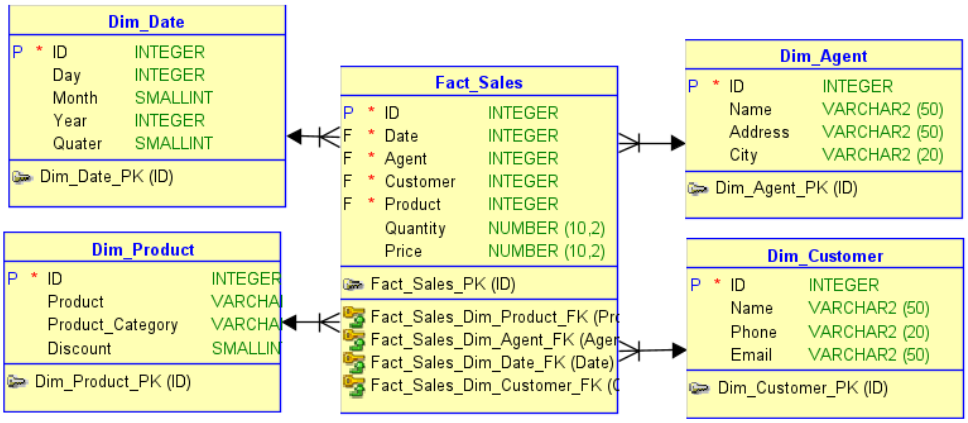
A small Sales Data Mart, illustrated in Figure 1, was designed specifically for this experimentation. While real data warehouse schemas could have been used, the schema was intentionally constructed with clear and semantically meaningful table and column names that accurately reflect the stored information. Although such naming conventions may not always be present in real-world systems, they represent the best practice in data warehouse design and significantly facilitate automated query generation.

When meaningful naming is not feasible, an alternative approach is to rely on column descriptions to convey semantic information. However, this strategy introduces additional challenges. Column descriptions are often lengthy, inconsistently populated, or unsupported across different relational database management systems. Moreover, incorporating detailed descriptions increases the prompt size, leading to higher processing costs and longer inference times.

Another possible approach is to preprocess schema metadata and derive simplified or normalized column names from descriptions. While potentially effective, this method introduces additional preprocessing complexity and overhead, particularly when schema metadata must be retrieved and

transformed prior to executing the Text-to-SQL conversion, as would be required in the context of this study.

Figure 1. Mini Data Mart for Sales Data



As illustrated in the figure above, a star-shaped data mart was designed for experiments, representing one of the simplest and most commonly used data warehouse structures. The scheme consists of a central fact table containing sales data and four associated dimension tables representing Agents, Customers, Dates, and Products.

To simplify the experimental setup, relational links between the tables—such as primary and foreign keys—were explicitly defined at the database level. However, these relationships were not explicitly provided to the models as part of the prompt. Instead, the models relied on consistent naming conventions, whereby foreign key attributes shared the same names as the corresponding primary keys in the dimension tables. This approach allowed the models to infer table relationships implicitly, without requiring explicit schema relationship descriptions.

Methodology: Choosing the models for the experimentation

Following the system design, the implementation phase focused on evaluating locally deployed large language models (LLMs) for the proposed Text-to-SQL task. A total of six models from three different organizations were selected for this experimentation: two Phi models developed by Microsoft, three DeepSeek models developed by DeepSeek (China), and one Mistral model developed by Mistral AI SAS (France).

These models were chosen based on their strong reputation within the technical community for code generation tasks. In particular, DeepSeek-Coder, Mistral, and Phi models are frequently cited in the literature and developer communities as effective solutions for programming-related use cases, with Microsoft actively promoting Phi models as lightweight yet capable coding-oriented LLMs. The inclusion of DeepSeek models allowed for a comparative assessment between code-specialized models and general-purpose LLMs, revealing notable performance differences, as discussed later in this paper.

Several prior studies have analyzed and compared local LLMs. One notable example evaluated more than 35 Japanese and multilingual LLMs (Saito et al., Dec 2025). However, that study did not include several of the models examined in this work. Given the rapid pace at which new LLMs are released, continued comparative evaluation is increasingly important. The models selected in this study represent some of the most widely adopted local LLMs for code generation, making them relevant candidates for investigating Text-to-SQL translation.

Figure 2. Local methods tested

NAME	SIZE	PRODUCER
deepseek-coder:1.3b	776 MB	DeepSeek
deepseek-coder:6.7b	3.8 GB	DeepSeek
deepseek-r1:1.5b	1.1 GB	DeepSeek
mistral:7b	4.1 GB	Mistral
Phi-3:latest	2.2 GB	Microsoft
Phi-4:latest	9.1 GB	Microsoft

All selected models were downloaded and deployed locally using the Ollama framework. Figure 2 summarizes the evaluated models along with their respective sizes, highlighting substantial variability even among models belonging to the same family.

The models were evaluated in ascending order of size, from the smallest to the largest. Special care was taken to account for caching effects, which could otherwise bias performance measurements. Given the size of the models and the associated inference times, caching could disproportionately benefit

smaller models, potentially skewing comparative results if not properly controlled.

Methodology: Prompts preparation

Before initiating the experimental evaluation, a common prompt was defined and applied consistently across all tested models. This prompt included explicit constraints intended to guide SQL generation and limit the scope of allowed operations. In particular, the prompt instructed the models to generate read-only SQL queries, explicitly disallowing data manipulation operations such as INSERT, UPDATE, DELETE, or schema-altering commands (e.g., CREATE or DROP).

These constraints were introduced to establish a controlled experimental baseline and to mitigate potential security risks associated with executing model-generated queries directly against a database. In scenarios where generated SQL is executed automatically, unintended or hallucinated instructions could otherwise result in destructive operations, such as table truncation or database deletion.

Despite these precautions, it cannot be guaranteed that a model will never hallucinate or violate constraints. For this reason, additional safeguards are recommended for practical deployments. These include the use of database views or synonyms instead of direct table access, as well as enforcing strict access control, such that the database user executing the generated queries is limited to SELECT privileges only.

The use of synonyms is particularly advantageous when combined with schema segregation, where a dedicated schema exposes only logical representations of the data while the physical tables reside in a separate schema. This approach is a common best practice in data warehouse and data mart environments and provides an additional layer of protection.

The following excerpt (Figure 3) illustrates a portion of the prompt used in this study. For experimental simplicity, the queries were executed directly against the underlying tables; however, the security considerations discussed above should be applied in real-world implementations.

Figure 3. Prompt engineering the result

```
You are a SQL expert. Your job is to generate correct SQL Server queries for human requests.

### Rules:
- **Use only the provided tables and columns**.
- **Do not use `LIMIT`**, instead use `TOP N` (SQL Server syntax).
- **Never use `"` (double quotes)** in queries.
- **Do NOT include any explanations**, return only the SQL query.
- **No `INSERT`, `UPDATE`, `DELETE`, `DROP`, `ALTER` queries**, only `SELECT` is allowed.
- If the request involves "today", use **`GETDATE()`** (SQL Server function).

### Database Schema:


In order to produce the tables and columns following is the metadata for a Data Mart.
The SQL queries should query these tables. Table and columns naming match the information they store. Follwoing are the metadata:
The schema name is DWHL and DIM in front of the names stands for Dimension and FAC for Facts. The rest is the name of the table.
{db_schema_metadata}
```

The placeholder {db_schema_metadata} represents the additional contextual information dynamically injected into the prompt during execution. This metadata augments the prompt with relevant details about the database schema, including table names, column names, data types, and relationships between tables. Such information is required to enable the model to accurately translate natural-language requests into valid SQL queries.

In practice, this metadata is incorporated into the prompt as structured textual instructions that explicitly define the available tables and their corresponding columns. An example of the resulting prompt structure is illustrated below.

Fact_Sales: ID (BIGINT), Agent (Integer), Customer (Integer), Date (Integer), Product (Integer), Price (MONEY), Quantity (NUMERIC)

Dim_Agent: ID (Integer), Address (VARCHAR), City (VARCHAR), Name (VARCHAR)

Dim_Customer: ID (Integer), Email (VARCHAR), Name (VARCHAR), Phone (VARCHAR)

Dim_Date: ID (Integer), Day (Integer), Month (SMALLINT), quarter (SMALLINT), Year (Integer)

Dim_Product: ID (Integer), Product (VARCHAR), Product_Category (VARCHAR), Discount

Before proceeding with the experimental evaluation, the size of the prompt was analyzed to ensure that the model context window was sufficient to accommodate all required information. The objective was not to optimize token consumption, but rather to verify that the full context would not be truncated, which would invalidate the use of schema-related metadata in the prompt.

Using the OpenAI tokenizer, the base prompt—excluding metadata descriptions—contained 872 characters, corresponding to 207 tokens. Considering that the smallest context window among the evaluated models supports 4096 tokens (Microsoft, 10 March 2025) for both input and output, the prompt size was well within the supported limits, exceeding the required capacity by more than an order of magnitude. This confirms that no context truncation occurs at this stage.

To further assess the impact of schema-related metadata on context size, an auxiliary script was developed to extract table names, column names, and data types from the database schema. This information was incorporated into the prompt to enrich the model's understanding of the underlying structure. In addition, providing one or two illustrative SQL examples served to guide the model's reasoning process and further reduce ambiguity in query generation.

An important requirement for the evaluated models is the ability to manage this expanded contextual information effectively. The extracted metadata contributes linearly to the prompt size and includes essential relational details such as primary and foreign keys, which are critical for accurate query construction.

Given that the evaluated schema follows a star-shaped structure, the relationships between tables are straightforward to infer from the provided metadata. All query paths originate from a single fact table and extend to the associated dimension tables, which are explicitly labeled in accordance with common data mart and data warehouse design conventions. This structural simplicity further reduces the complexity of context management and supports reliable Text-to-SQL generation.

Methodology: Experimentation last considerations

Two different software interfaces were used to evaluate the generated results: the OpenAI Python client library and the Ollama framework, with all experimental scripts implemented in Python. Specifically, the functions

openai.OpenAI.chat and ChatOllama (from the LangChain library, used to invoke Ollama.chat) were employed to interact with the evaluated models. Based on findings reported in previous studies (Oliveira et al., 2024), the evaluation process began with the Phi-3 model, followed by the more recent Phi-4 model. This progression strategy – starting from models with lower expected performance and moving toward newer and theoretically more capable versions – was applied consistently across all evaluated models.

Following the evaluation of the Phi models, additional experiments were conducted using DeepSeek models, including both the general-purpose SML variant and code-oriented DeepSeek-Coder models. The final model evaluated was Mistral. Due to hardware constraints, the experiments were limited to local, medium-sized models, although it should be noted that models such as Phi-4 and Mistral cannot be considered small and require substantial computational resources, resulting in non-trivial execution times.

Although several studies advocate the use of retrieval-augmented generation (RAG) for Text-to-SQL tasks (Ziletti & DAmbrosi, 2024), a few-shot prompting approach was adopted in this work. Given the limited number of tables in the evaluated schema, two to three illustrative examples were sufficient to guide the models toward correct query generation without introducing the additional complexity associated with retrieval-based methods.

The results of this evaluation are presented in the following section and reflect the performance of the tested models on the described problem using a modest hardware configuration consisting of 32 GB of RAM and a single CPU (with 4 cores – 8 logical processors). Such computational resources are representative of contemporary mid-range personal computers and are increasingly comparable to the capabilities of high-end mobile devices.

Results

Each of the evaluated models was tested using the same set of input queries, and performance was assessed based on two primary criteria: query correctness and response time. When the query generated was incorrect, response time was considered irrelevant. For correct outputs, shorter response times were regarded as indicative of better suitability for Text-to-SQL conversion and overall model performance.

Results with response times exceeding five minutes were excluded from further analysis, as such delays render the solution impractical for interactive use, regardless of query correctness.

To reduce the likelihood of coincidental correct outputs, response time measurements were considered only for queries that produced three consecutive correct results. Queries with fewer correct responses were treated as unreliable, as they are more likely to fail when applied to larger or more complex schemas than the one used in this study.

In addition, each query was executed three consecutive times per model to mitigate the potential impact of caching effects, which could otherwise bias performance measurements in favor of subsequent executions.

Table 1. Results of local LLMs

		TIME	RESULT	TIME	RESULT	TIME	RESULT	AVG. TIME
	Input	*Query: Use the above information to build SQL from text to get number of product categories*		*Query: Use the above information to build SQL from text to get the total sale made during March 2025*		*Query: Use the above information to build SQL from text to get the quantity of sales for each product during the first Quarter of 2024*		
phi3:latest	***Output***	82 s	Correct	89 s	Correct	110 s	Wrong	
phi4:latest	***Output***	289 s	Correct	96 s	Correct	295 s	Correct	**227**
deepseek-r1:1.5b	***Output***	50 s	Wrong	> 300 s	Wrong	65 s	Wrong	
deepseek-coder:1.3b	***Output***	48 s	Wrong	48 s	Correct	92 s	Wrong	
deepseek-coder:6.7b	***Output***	164 s	Correct	166 s	Correct	183 s	Correct	**171**
mistral:7b	***Output***	112 s	Correct	133 s	Correct	14 s	Correct	**86**

Table 1 summarizes the results obtained from executing the Text-to-SQL queries generated by the evaluated models. For each test case, the input query and the corresponding generated SQL were recorded. To efficiently assess correctness, the generated SQL statements were executed directly on the database, and the correctness of the results was verified without relying solely on visual inspection of the output. When queries were executed successfully, the returned results were additionally reviewed to ensure that they matched the expected semantics of the original request.

Response time, measured in seconds, was reported only for models that consistently produced correct outputs. For these models, the average response time across valid executions was also calculated.

The results presented in Table 1 allow several observations to be drawn regarding both correctness and performance.

First, not all evaluated models were able to consistently generate correct SQL queries. Smaller models such as Phi-3, DeepSeek-R1, and DeepSeek-Coder (1.3B) produced at least one incorrect result. The inclusion of DeepSeek-R1 in the evaluation was motivated by its relatively small model size, despite recommendations in the literature to favor DeepSeek-Coder variants for code generation tasks (Guo et al., May 2025).

In contrast, the remaining three models successfully generated correct SQL queries for all three test cases, producing results that were both syntactically valid and semantically correct.

With respect to response time, Mistral (7B) demonstrated the best overall performance, achieving nearly twice the speed of the evaluated DeepSeek model and approximately three times faster response times than Phi-4 (see Table 1). However, Phi-4 exhibited a more comprehensive problem-solving approach, often producing richer and more informative query outputs. The performance of DeepSeek-Coder (6.7B) was comparable to that of Mistral in terms of response time, though with slightly less expressive output in some cases.

Close up overview for the comparison between Mistral and Phi-4

Figure 4 presents a detailed comparison of two of the evaluated models, focusing on the SQL queries generated in response to the final test question. The comparison illustrates differences in how the models incorporate additional contextual information during query construction while preserving the overall semantic intent of the request. Although the inclusion of supplementary information may introduce ambiguity in certain scenarios, it is generally appropriate in the context of Text-to-SQL systems, which are designed to assist users who may not be familiar with SQL syntax or query formulation. In such cases, enriched output can improve both the interpretability and practical usability of the generated results.

To support this analysis, the SQL outputs produced by the Phi-4 and Mistral models are presented side by side to enable direct comparison.

Figure 4. Phi-4 vs Mistral - comparison of queries produced

Phi 4	Mistral
SELECT p.Product, SUM(f.Quantity) AS Total_Quantity FROM MINIDM.Fact_Sales f JOIN MINIDM.Dim_Date d ON f.Date = d.ID JOIN MINIDM.Dim_Product p ON f.Product = p.ID WHERE d.Quarter = 1 AND d.Year = 2024 GROUP BY p.Product;	SELECT Product, SUM(Quantity) as Total Quantity FROM MINIDM.Fact_Sales JOIN MINIDM.Dim_Date ON Fact_Sales.Date = Dim_Date.ID WHERE Dim_Date.Year = 2024 AND Dim_Date.Quarter = 1 GROUP BY Product;

Both generated queries are syntactically and semantically correct for the given request (Use the above information to build SQL from text to get the quantity of sales for each product during the first quarter of 2024)", and they are functionally equivalent. However, a closer examination reveals qualitative differences in the outputs. The Phi-4 model produced a more interpretable result by including the product name in the output (the column Product in Dim_Product is the product name and not just the product id so through the additional join additional information where introduced in the final output relevant to the user) rather than only the product identifier (in the Mistral query the Product is the identifier of the Products in the Fact_sales, therefore it is just a representation of the Foreign Key), as observed in the outputs of other models. From a result quality perspective, this behavior can be considered superior.

This improvement in output clarity, however, came at a significant cost in response time. The Phi-4 model required approximately five minutes to generate the result, which would render it impractical for interactive use on resource-constrained systems. Even the best-performing model in terms of latency, Mistral, exhibited response times that remain relatively high when compared to cloud-based solutions. For example, executing the same prompt using GPT-4o required only a few seconds. For moderately complex queries, such as those examined in this study, achieving response times below 10–15

seconds would likely be necessary for practical adoption, potentially requiring substantially greater computational resources.

An additional observation concerns the behavior of the Mistral model when answering the second query involving the year 2025. Instead of explicitly specifying the year, the model generated a query using the database function *YEAR(GETDATE())*. While this approach is logically valid, it introduces an implicit temporal assumption. It is unclear whether this choice reflects contextual reasoning by the model or incidental behavior, and the underlying cause cannot be conclusively determined.

Finally, it was observed that minor typographical errors in the input text, such as the misspelling of ‘quater’ instead of ‘quarter’, did not prevent the higher-performing models from correctly interpreting the intent of the query and generating the appropriate SQL statements. This robustness in input noise is a desirable characteristic for user-facing Text-to-SQL systems.

Conclusion

This study demonstrates that locally deployed large language models (LLMs) are capable of semantically interpreting the schema of a small star-schema data mart and generating correct SQL queries from natural-language input. However, the experimental results also indicate that, under typical personal computer–level hardware constraints, the response times of these models remain relatively high. Even the best-performing model required, on average, more than one minute to produce a correct query, which limits the practicality of such solutions for interactive use on resource-constrained devices.

It should also be noted that the evaluation was conducted on a small and well-structured data mart. The observed performance and correctness may not directly generalize to larger data warehouses with a higher number of objects or to queries of increased structural and semantic complexity.

One possible approach to reducing response time is the use of a dedicated local server to handle inference. Compared to cloud-based API solutions, such a setup can offer lower operational costs while preserving data locality and privacy. From an architectural perspective, this configuration resembles a client–server model, where a centralized server performs the computationally intensive processing on behalf of multiple client devices.

Despite these limitations, recent advances in generative AI are significant. The ability of local LLMs to perform Text-to-SQL translation represents a capability that was largely impractical only a few years ago. Nevertheless, the

cost of deploying and operating such models remains non-trivial, and concerns related to data security and privacy persist, particularly for cloud-based solutions. These considerations highlight the importance of cautious and informed adoption of generative AI technologies.

Among the evaluated models, Mistral demonstrated the best overall performance in terms of response time and correctness, followed by DeepSeek-Coder (6.7B parameters) and Microsoft Phi-4. While all three models were able to generate correct SQL queries, none consistently achieved response times below one minute on the tested hardware.

Finally, an interesting observation is that some local LLMs implicitly accounted for temporal aspects of queries, for example by directly using database time functions instead of deriving the current date from stored records. Such behavior may be preferable in user-oriented analytical scenarios, where the primary focus is on obtaining correct results rather than enforcing strict assumptions about data refresh cycles or system architecture.

Future work

An important issue identified in this study is that all schema-related information required for SQL generation must be provided to the model as part of the prompt for each request. Repeatedly supplying the same metadata represents a significant overhead, particularly in scenarios where the same data warehouse scheme is used consistently and follows fixed rules and logic. Improving this process, for example by reducing redundant context injections or enabling persistent schema awareness, could increase both the prediction accuracy and overall performance of local LLM-based solutions.

Another direction for future work is the extension of the proposed approach to larger data warehouses. Although this study primarily focuses on Text-to-SQL generation, it also highlights the challenges introduced by high data volumes and complex schemas. These challenges are not limited to local LLMs but also affect larger cloud-based models. A key open problem is the design of mechanisms that allow a model to identify and extract only the relevant tables and columns required for a given query before constructing the final SQL statement. While this approach appears feasible, it may prove to be more complex than the SQL generation task itself.

Finally, this work considers only data warehouse environments. An open question is whether the same conclusions would be held for operational

databases, where schema structures are typically more complex and lack star or snowflake organization.

Addressing these challenges is increasingly important as large language models continue to evolve into practical tools for reducing the complexity of traditional software development and data querying tasks.

Abbreviations

The following abbreviations have been utilized throughout this paper:

AI – Artificial Intelligence

API – Application programming interface

CPT – Continual Pre-Training

GenAI – Generative Artificial Intelligence

GPT – Generative Pre-training Transformer

LLM – Large Language Model

OAS – Oracle Analytics Services

RAG – Retrieval-Augmented Generation

RDBMS – Relational Database Management System

SFT – Supervised Fine-Tuning

SML – Service Modeling Language

SQL – Structured Query Language

References

Bumgardner VKC, Mullen A, Armstrong SE, Hickey C, Marek V, Talbert J. (May 2024). Local large language models for complex structured tasks. AMIA Jt Summits Transl Sci Proc., 105–114.

Guo D., Zhu Q., Yang D., Xie Z., Dong K., Zhang W., Chen G., Bi X., Wu Y., Li Y.K., Luo F., Xiong Y., Liang W. (May 2025). DeepSeek-coder: When the large language model meets programming - the rise of code intelligence.

Li S., Huang J., Zhuang J., Shi Y., Cai X., Xu M., Wang X., Zhang L., Ke G. & Cai H. (2025). SciLitLLM: How to adapt LLMs for scientific literature understanding.

Microsoft. (10 March 2025). Featured models of azure AI foundry.

Oliveira A., Nascimento E., Pinheiro J., Avila C. V. S., Coelho G., Feijó L., Izquierdo Y., García G., Leme L. A. P. P., Lemos M. & Casanova M. A. (2024). Small, medium, and large language models for text-to-SQL.

Oracle. (2024). Building semantic models in oracle analytics server. In Oracle.com.

Perron, B. E., Luan, H., Victor, B. G., Hiltz-Perron, O. & Ryan, J. (2024). Moving beyond ChatGPT: Local large language models (LLMs) and the secure analysis of confidential unstructured text data in social work research. Research on Social Work Practice.

Saito K., Mizuki S., Ohi M., Nakamura T., Shiotani T., Maeda K., Ma Y., Hattori K., Fujii K., Okamoto T., Ishida S., Takamura H., Yokota R. & Okazaki N. (Dec 2025). Why we build local large language models: An observational analysis from 35 japanese and multilingual LLMs. https://arxiv.org/abs/2412.14471

Vaid A, Duong SQ, Lampert J, Kovatch P, Freeman R, Argulian E, Croft L, Lerakis S, Goldman M, Khera R & Nadkarni GN. (Sep 2024). Local large language models for privacy-preserving accelerated review of historic echocardiogram reports. J Am Med Inform Assoc.

Ziletti, A. & DAmbrosi, L. (2024). Retrieval augmented text-to-SQL generation for epidemiological question answering using electronic health records.