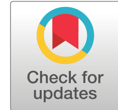
Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.078330



ARTICLE

Evaluating Open-Source LLM Agents for SQL Generation and Structured Analytics on Relational Databases

Karlo Borovčak[1], **Marina Bagić Babac**[1,*] and **Vedran Mornar**[2]

[1]Faculty of Electrical Engineering and Computing, University of Zagreb, Unska 3, Zagreb, Croatia
[2]VERN' University, Palmotićeva 82/1, Zagreb, Croatia

*Corresponding Author: Marina Bagić Babac. Email: marina.bagic@fer.hr

Received: 29 December 2025; Accepted: 24 April 2026; Published: 15 June 2026

ABSTRACT: This study examines the potential of open-source foundation models for structured data analytics, with particular emphasis on SQL generation and business-oriented interpretation in single-agent and multi-agent large language model (LLM) systems. The proposed framework addresses a practical problem in analytics-intensive environments, where natural-language requests must be translated into executable, semantically appropriate SQL queries and subsequently interpreted in a form useful for business decision-making. The system is evaluated in two complementary settings: a custom SQL test suite designed around realistic marketing and e-commerce analytics tasks, and the public Spider benchmark, which supports comparison with prior text-to-SQL research and enables assessment of cross-domain generalization. The analysis includes Mistral, Devstral, Qwen2.5-Coder, and Qwen3. On the custom SQL test suite, performance was assessed using exact match, safe SQL rate, and an independent semantic judge score. Qwen2.5-Coder achieved the strongest overall result, reaching an independent semantic score of 90.14% while maintaining a 98.59% safe SQL rate. Qwen3 followed with a semantic score of 77.46% and completely safe SQL generation. These results indicate that in domain-specific analytics settings, strict query-level matching alone is too conservative to capture practical model usefulness, since semantically appropriate SQL queries may differ substantially from the reference formulation. The Spider benchmark results provide complementary evidence regarding broader model behavior. Qwen2.5-Coder achieved the highest single-agent execution accuracy (72.44%), whereas Devstral obtained the strongest single-agent exact-match score (28.14%). Qwen3 remained competitive and delivered the lowest single-agent latency (0.41 s) among the evaluated models. At the architectural level, the effect of multi-agent decomposition was not uniform: it yielded modest gains in execution accuracy for some model families, but reduced performance for others, while consistently increasing latency and token consumption. Taken together, the findings show that open-source LLM agents can provide effective support for structured analytics, but that their performance depends strongly on model family, prompting strategy, and agent architecture. More broadly, the study demonstrates that the evaluation of text-to-SQL systems benefits from combining benchmark-based metrics, domain-oriented semantic assessment, and efficiency-aware analysis, thereby offering a more realistic basis for the deployment of open-source LLM systems in analytics-intensive environments.

KEYWORDS: Large language models (LLMs); multi-agent systems; SQL agents; text-to-SQL

1 Introduction

In the rapidly evolving digital economy, data has become one of the most valuable assets for businesses, particularly in the domains of e-commerce and online advertising. Platforms such as Facebook Ads, Google Ads, and various e-commerce systems generate vast amounts of data every day. This data contains critical

Copyright © 2026 The Authors. Published by Tech Science Press. This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

insights into customer behavior, marketing performance, sales trends, and operational efficiency and it is usually stored in databases or data warehouses [1]. However, extracting meaningful information from these systems typically requires technical expertise, especially in formulating SQL queries and interpreting raw datasets. As a result, access to insights often remains limited to data analysts and technical teams, creating bottlenecks in decision-making processes [2].

With recent advancements in large language models (LLMs), new possibilities have emerged for automating complex tasks traditionally performed by humans [3], including data querying, summarization, and interpretation. LLMs have demonstrated strong capabilities in understanding natural language and generating code [4], making them increasingly relevant in the context of business intelligence. Much of the current research and commercial applications in this area focus on retrieval-augmented generation (RAG) [5], where models retrieve external unstructured documents to enhance their responses [6]. While powerful in open-domain contexts [4], such approaches may not be ideal for scenarios involving structured, tabular business data where precision and schema awareness are crucial [7].

This paper explores an alternative and more grounded method by evaluating how open-source LLMs can function as SQL-based agents in the context of e-commerce analytics. Instead of relying on document retrieval, the focus here is on agents that understand natural language queries, translate them into executable SQL, fetch relevant data from a structured database, and produce interpretable insights. These agents serve as intermediaries between complex datasets and business users, with the goal of making data more accessible, insightful, and actionable without requiring technical expertise in SQL or database management.

The core contribution of this work lies in designing and evaluating a system where open-source language models are deployed as intelligent agents for querying and interpreting data from a structured SQL database. This research investigates how well different LLMs perform in terms of accuracy, reliability, and usefulness when applied to real-world business data. By comparing the capabilities of various models in this structured environment, the study aims to highlight their strengths, limitations, and potential as tools for automated business intelligence. The central objective of this study is to evaluate the effectiveness of large language models in structured data analytics by comparing the performance of single-agent and multi-agent system architectures. This includes assessing how different open-source models, such as Qwen3, Qwen2.5-Coder, Devstral, and Mistral, translate natural language queries into executable SQL and generate actionable insights. Particular attention is given to architectural choices such as agent specialization, prompt engineering, and feedback-based workflows, with the goal of identifying which factors most influence performance in terms of accuracy, clarity, and utility. In doing so, the study contributes both to the theoretical understanding and practical advancement of agentic LLM systems for real-world analytics tasks.

The remainder of this paper is dedicated to detailing the technical design, implementation, and evaluation of the proposed system. It begins with a review of related literature in text-to-SQL LLM-powered agents, evaluating text-to-SQL agents and data analytics automation. This is followed by a discussion of the system architecture, the complete implementation process, and the criteria used to select and compare different open-source models and agents. The paper concludes by analyzing experimental results and reflecting on broader implications, challenges encountered, and opportunities for future research in this emerging intersection of natural language processing and data analytics.

2 Related Work

Recent years have witnessed remarkable progress at the intersection of natural language processing, database interaction, and autonomous systems [8]. Advances in large language models and text-to-SQL techniques have opened new possibilities for intelligent data analysis, while novel agentic frameworks

are reshaping how these models interact with complex information sources [9]. The CHASE-SQL framework represents a significant advancement in text-to-SQL systems through its innovative multi-agent approach [10]. This trend is further exemplified by [8], which proposed QCMA-SQL, a framework that classifies questions based on difficulty and dynamically routes them through specialized agents, achieving 87.4% execution accuracy on the Spider dataset. The researchers from Google Cloud and Stanford University developed a detailed system that employs test-time compute in multi-agent modeling to improve candidate generation and selection. Their methodology consists of four core components: value retrieval using locality-sensitive hashing (LSH) for extracting relevant database values [11], candidate generation through multiple strategies, query fixing for error correction, and a sophisticated selection agent. The value retrieval component specifically utilizes keyword extraction combined with LSH techniques to retrieve syntactically similar words while maintaining robustness against typographical errors. For candidate generation, CHASE-SQL implements three distinct approaches: a divide-and-conquer method that decomposes complex queries into manageable sub-queries, chain-of-thought reasoning based on query execution plans that mirror database engine operations, and instance-aware synthetic example generation for tailored few-shot demonstrations [10]. The selection agent employs pairwise comparisons using a fine-tuned binary-candidates selection LLM, which has proven more robust than alternative selection methods. The framework achieved state-of-the-art execution accuracy of 73.0% and 73.01% on the test set and development set of the BIRD Text-to-SQL dataset benchmark [12], positioning it as the top submission on the leaderboard at the time of publication.

On the other hand, researchers at Alibaba Group developed the XiYan-SQL framework which introduces a multi-generator ensemble approach that combines the strengths of supervised fine-tuning with in-context learning [13]. This system addresses text-to-SQL challenges through several innovative components, with M-Schema serving as a cornerstone semi-structured schema representation method. M-Schema enhances database structure understanding by providing hierarchical relationships between databases, tables, and columns, including metadata such as data types, primary key identifiers, and example values. The representation employs specific tokens to clearly label different elements, significantly improving how LLMs interpret schema information compared to traditional DDL representations. The framework implements a two-part retrieval module consisting of column and value retrievers that emphasize efficiency through semantic similarity and locality-sensitive hashing for selecting the most relevant schema components [13]. The candidate generation process employs multiple generators using both supervised fine-tuning and in-context learning methodologies. The fine-tuned generator utilizes a two-stage multi-task training pipeline: basic-syntax training for SQL dialect-agnostic patterns and generation-enhance training leveraging multi-task learning to improve semantic relationships and stylistic variations. The ICL generator uses skeleton representation to mask named entities within user queries [14], focusing on semantic structure rather than specific entities while selecting top K examples from training data. XiYan-SQL achieved state-of-the-art execution accuracy results of 89.65% on the Spider test set [15], 69.86% on SQL-Eval, 41.20% on NL2GQL [16], and 72.23% on the Bird development benchmark [12]. Similarly, Guo et al. [6] proposed a retrieval-augmented text-to-SQL framework leveraging Poincaré-Skeleton retrieval and meta-instruction reasoning, further enhancing schema understanding and generalization across relational contexts. Their method emphasizes flexible prompt reasoning and advanced linking over complex schemas. The framework's custom fine-tuned selection model assesses generated SQL queries based on execution results, allowing for subtle distinctions among candidates instead of relying solely on surface-level consistency.

The MAC-SQL framework introduces a novel LLM-based multi-agent collaborative approach specifically designed to address challenges in extensive databases, intricate user queries, and erroneous SQL results [17]. Researchers from Beihang University and Tencent Cloud AI developed a three-agent system

comprising the Selector, Decomposer, and Refiner agents, each with specialized responsibilities. The Selector agent is responsible for condensing voluminous databases and preserving relevant table schemas for user questions, effectively addressing the challenge of large-scale database navigation. The Decomposer agent disassembles complex user questions into more straightforward sub-problems and resolves them progressively, enabling the system to handle intricate queries that would otherwise overwhelm single agent approaches. The Refiner agent validates and refines defective SQL queries, ensuring output quality and correctness through iterative improvement processes. This modular agent design aligns with recent work by Shen et al. [18], who demonstrated an open multi-agent orchestration platform for LLM-based SQL generation. Their system supports dynamic agent configuration and real-time debugging, offering greater transparency and adaptability in real-world settings. The researchers conducted detailed experiments on two major Text-to-SQL datasets, BIRD and Spider [15], achieving a state-of-the-art execution accuracy at the time of writing, of 59.59% on the BIRD test set [12]. Additionally, they open-sourced an instruction fine-tuning model called SQL-Llama, based on Code Llama 7B [12], along with an agent instruction dataset derived from training data based on BIRD and Spider. The SQL-Llama model demonstrated encouraging results on the development sets of both benchmarks, though the researchers noted significant potential for enhancement when compared to GPT-4.

While many systems compete to achieve higher benchmark scores, there's also growing research interest in defining what evaluation should truly measure. The FLEX (False-Less EXecution) evaluation approach addresses critical limitations in the widely used Execution Accuracy (EX) metric through LLM-based evaluation that emulates human expert-level assessment [15]. Researchers found that the EX metric is susceptible to false positives and negatives, obscuring accurate assessment of model performance due to strict result-matching criteria. The FLEX methodology leverages Large Language Models to evaluate generated SQL queries by analyzing their semantic consistency with original questions based on sophisticatedly designed criteria. This approach provides more detailed assessment of query correctness, even considering noisy ground truth data that traditional metrics struggle to handle. The validation process demonstrated significantly higher agreement with human expert judgments, improving Cohen's kappa from 61 to 78.17 compared to the existing EX metric [15]. Re-evaluation of top-performing models on Spider and BIRD benchmarks using FLEX revealed substantial shifts in performance rankings, with an average performance decrease of 3.15 due to false positive corrections and an increase of 6.07 from addressing false negatives [12,15]. This work contributes to more accurate and subtle evaluation of text-to-SQL systems, potentially reshaping understanding of state-of-the-art performance in the field. In addition, reference [13] provides a systematic survey of such LLM-driven systems, emphasizing the need for interpretability, more robust evaluation metrics, and improved schema-grounded reasoning in Text-to-SQL pipelines. Their analysis highlights FLEX-style evaluations as a key advancement.

While benchmarks like Spider and BIRD have significantly advanced text-to-SQL research [12,15], they largely focus on normalized, well-structured databases and fall short of reflecting the realities of production environments, where denormalized data is common. In real-world settings data is often stored in large, denormalized tables optimized for analytics, with flattened structures, ambiguous or redundant columns, and implicit relationships that are not easily captured by standard benchmarks [19]. This discrepancy means that models performing well on academic datasets often struggle when faced with the complexities and ambiguities of actual enterprise data. Pinterest's engineering team addressed these challenges by building a text-to-SQL system specifically designed to interpret and generate queries over denormalized schemas, emphasizing robust schema understanding, query optimization for wide tables, and adaptation to their unique data landscape [20]. Initially, Pinterest evaluated their system to ensure performance was on par with published results, finding comparable accuracy on Spider but quickly realizing that benchmark tasks

were much simpler than the real-world queries their users faced, as Spider involves only a small number of pre-specified tables with well-labeled columns [15]. In production, Pinterest observed user interactions and iteratively improved their system, leading to a notable increase in first-shot acceptance rates for generated SQL from 20% to over 40% as both the technology and user familiarity matured. Despite these improvements, most queries still required multiple iterations before finalization, reflecting the complexity of real-world data tasks.

To overcome these hurdles, Pinterest engineered a system that goes beyond standard schema linking: they developed advanced schema parsing techniques to interpret column names and group related fields, even when naming conventions were inconsistent or when relationships were only implicit. Additionally, Pinterest's solution integrates a semantic layer that maps user queries to business concepts, bridging the gap between natural language and the technical details of their denormalized schemas. This approach allows users to ask questions in everyday language, while the system intelligently translates these into efficient SQL queries tailored for their specific data layout [18]. The team also placed a strong emphasis on query optimization, ensuring that generated SQL was performant on massive datasets a critical requirement for production analytics [20]. Throughout development, they iterated closely with internal users, refining the system based on real-world feedback and adapting their models to handle the nuances of Pinterest's evolving data. This pragmatic, user-focused approach enabled Pinterest to deploy a robust text-to-SQL solution that meets the demands of large-scale, denormalized data environments, setting an example for how industry applications can extend beyond the limitations of academic benchmarks. In the broader context of practical deployment, Ojuri et al. (2025) explored how intelligent agents embedded within LLM pipelines improve SQL generation across complex data environments. Their results demonstrate that agent-enhanced models outperform prompt-only baselines by integrating error correction, schema understanding, and iterative refinement into the generation process.

3 System Design

3.1 Architecture

The proposed system architecture, as depicted in Fig. 1, illustrates the high-level design of a system in which a user interacts with an intelligent agent system that leverages a Large Language Model (LLM) to facilitate natural language understanding, data retrieval, and insight generation. This system is designed to abstract away the technical complexity of data querying, enabling users to access and interpret database content using natural language inputs.

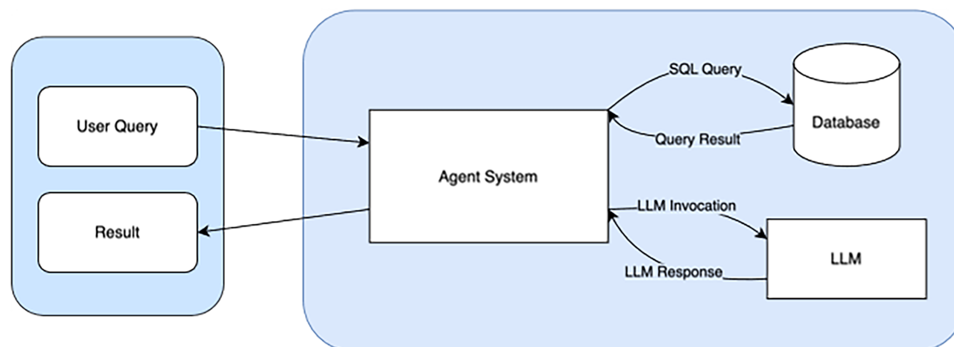


Figure 1: High-level overview of the system architecture.

The process begins with the user submitting a natural language query through a user interface. This **User Query** is then received by the **Agent System**, which acts as an orchestrator between different components. The agent interprets the user's intent and utilizes the LLM to generate a corresponding SQL query. This SQL query is constructed dynamically based on the context and schema of the target **Database**.

Once the SQL query is generated, it is executed against the database, and the resulting data referred to as the **Query Result** is returned to the agent system. Depending on the complexity and nature of the user's request, the agent may invoke the LLM again, this time to analyze or summarize the query result and produce human-readable insights. This process is depicted as an **LLM Invocation** that results in an **LLM Response**, containing contextual insights, summaries, or recommendations derived from the data. Finally, the processed output whether it is raw data, a summary, or an analytical insight is returned to the user as the **Result**. This architecture demonstrates a seamless integration of LLMs into data-intensive systems, enabling natural language interfaces over structured databases.

3.2 Agent Design

An agent, in the context of artificial intelligence and data systems, is an autonomous system or program capable of performing tasks on behalf of a user or another system [21]. It operates by interpreting user intent, planning actions, and executing those actions often utilizing external tools or data sources to achieve complex goals. Unlike static tools, agents can reason, adapt, and make decisions dynamically based on the context and outcomes of previous actions. For example, an LLM SQL agent translates natural language queries into SQL commands, executes them against a database, and returns business insights in a user-friendly format all without requiring the user to have SQL expertise. The distinction between an agent and a workflow lies in flexibility, autonomy, and decision-making:

- **Workflow:** A workflow is a predefined, structured sequence of steps or code paths designed to accomplish a specific task. Each step is explicitly programmed, and the workflow follows these steps in a predictable manner. Workflows excel at handling repeatable, well-defined tasks where consistency and reliability are paramount. For example, a workflow might always perform data extraction, transformation, and loading in the same order, regardless of context.
- **Agent:** An agent, by contrast, is designed to function independently and adaptively. It can dynamically decide which actions to take, which tools to use, and in what order, based on the current context and user intent. Agents employ reasoning and reflection, mimicking human decision-making, to determine the best approach for each unique situation. This enables agents to handle more complex, ambiguous, or variable tasks that require interpretation and adaptation, such as answering open-ended business questions by querying a database and synthesizing insights.

Modern AI agents are increasingly required to handle complex, multi-step tasks that involve both reasoning and interaction with external systems. Traditional approaches, where agents simply respond based on static knowledge, fall short when tasks require adaptive planning, tool use, or iterative problem solving. To address these challenges, the ReAct (Reasoning + Acting) pattern has emerged as a leading design paradigm [22].

The ReAct pattern has become foundational in agentic AI. In this framework, an agent alternates between two core capabilities:

- **Reasoning:** The agent engages in chain-of-thought (CoT) reasoning, breaking down tasks, planning steps, and interpreting intermediate results.
- **Acting:** The agent performs concrete actions, such as querying a database, calling external APIs, or invoking tools, based on its reasoning.

This cyclical process reason, act, observe, repeat enables agents to dynamically adapt their behavior, integrate new information, and solve tasks that would be infeasible for simple, static models [22].

The provided agent graph in Fig. 2 exemplifies these theoretical concepts in a practical, data-driven context:

- **start:** The entry node initializes the agent, preparing it to interact with the business database.
- **fetch_schema:** The agent first retrieves the database schema. This is a preparatory action, ensuring the agent understands the structure and available fields before formulating queries.
- **chat:** This node represents the agent's reasoning phase. Here, the agent interprets the user's request, plans the required analysis, and determines what information is needed. This may involve multiple cycles of deliberation, especially if intermediate results prompt further questions or require clarification.
- **execute_sql:** When the agent has determined what data is needed, it transitions to this action node, where it formulates and executes an SQL query against the business database. The results are then fed back into the reasoning process.
- **end:** Once the agent has gathered sufficient information and completed its analysis, it transitions to the end node, finalizing the workflow.

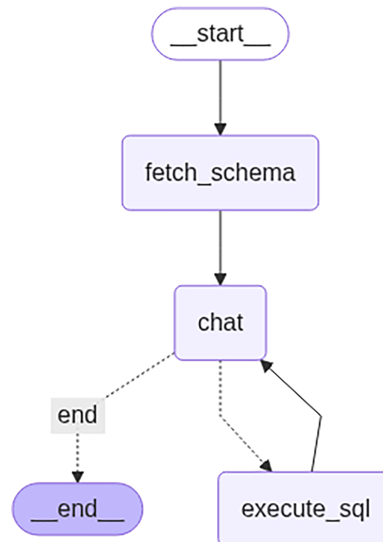


Figure 2: Agent workflow graph.

The graph structure allows for iterative reasoning and acting; after executing SQL, the agent can return to the chat node to further interpret results, refine its analysis or potentially fix the incorrect SQL that was generated in the previous iteration. This loop continues until the agent determines that it has reached a satisfactory answer and transitions to the end node.

Representing the agent as a graph, rather than a linear sequence, is key for managing complex, multi-step workflows. It makes the agent's logic transparent, supports modular development, and enables advanced features like parallel execution or dynamic rerouting capabilities that are increasingly important in modern agentic systems.

3.3 Multi-Agent Design

Multi-agent systems (MAS) are composed of multiple autonomous agents, each capable of independent decision-making and action. Each agent operates autonomously, often specializing in a particular task or

domain, and communicates with other agents to contribute to the overall objective [23]. This distributed approach allows for modularity, specialization, and scalability, making MAS particularly effective for handling tasks that are too complex for a single agent. By breaking down large problems into smaller, manageable parts and distributing them among specialized agents, MAS improves efficiency and adaptability in dynamic environments.

The use of MAS is driven by the need to overcome the limitations of single agent systems, such as bottlenecks in processing, lack of domain expertise, and challenges in scaling. In practical applications, MAS can be found in areas like document processing, market analysis, customer service, and autonomous robotics, where agents gather information, analyze data, generate responses, and coordinate actions to achieve a common goal. Within MAS, two prominent architectural styles stand out: the supervisor architecture and the swarm architecture [23].

The supervisor architecture is characterized by a central supervisor agent that coordinates the activities of specialized subordinate agents. The supervisor controls all communication and task delegation, making decisions about which agent to invoke based on the current context and requirements. Subordinate agents focus on their areas of expertise and report results back to the supervisor, who orchestrates the workflow and ensures efficient completion of tasks. This centralized approach allows for explicit control, clear task assignments, and optimized workflows, but it can also introduce a single point of failure and may limit scalability if the supervisor becomes a bottleneck. In contrast, the swarm architecture, where agents operate in a decentralized and self-organized manner. There is no central controller; instead, each agent follows simple local rules and interacts with its immediate neighbors or environment. The collective intelligence of the system emerges from these local interactions, enabling the MAS to adapt dynamically to changes and recover from partial failures. Swarm architectures are highly robust and scalable, as they distribute decision-making and coordination across all agents, allowing the system to handle large-scale, dynamic tasks efficiently.

Both architectures offer distinct advantages depending on the requirements of the problem domain. Supervisor architectures provide centralized oversight and are well-suited for tasks requiring explicit coordination and control, while swarm architectures excel in scenarios demanding flexibility, resilience, and large-scale collaboration. In this paper, the swarm architecture is implemented, with each agent specializing in a distinct aspect of the data analysis workflow. One agent is dedicated to SQL querying, focusing on extracting relevant data from relational databases, while another agent specializes in generating business insights from the queried data. This division of labor leverages the strengths of both agents: the SQL querying agent ensures efficient and accurate data retrieval, while the business insights agent interprets the results to provide meaningful recommendations and strategic information.

For this paper a swarm architecture was chosen to orchestrate the business data analysis workflow. The system consists of two specialized agents, each powered by a different large language model (LLM) tailored to its role:

- **analyst_assistant:** This agent is responsible for high-level reasoning, problem decomposition, and interpreting user queries. It is equipped with an LLM optimized for analytical reasoning and natural language understanding.
- **sql_assistant:** This agent specializes in generating, validating, and executing SQL queries. It uses an LLM fine-tuned for database interaction and code generation.

The agents collaborate in a feedback loop: the analyst_assistant interprets the user's request and determines what data is needed, passing tasks to the sql_assistant, which retrieves or manipulates the data as required. Results are then fed back to the analyst_assistant for further reasoning or final analysis. This

division of labor exemplifies the principle of specialization in swarm systems, where each agent contributes its unique expertise to the overall solution.

Fig. 3 above illustrates the implemented swarm architecture. The process begins at the **start node**. The **analyst_assistant** node represents the agent responsible for reasoning and analysis. The **sql_assistant** node represents the agent handling SQL-related tasks. Arrows indicate the flow of information and task delegation between agents. The bidirectional paths between **analyst_assistant** and **sql_assistant** demonstrate the iterative, collaborative nature of the swarm: analysis and data retrieval are performed in cycles until the task is complete.

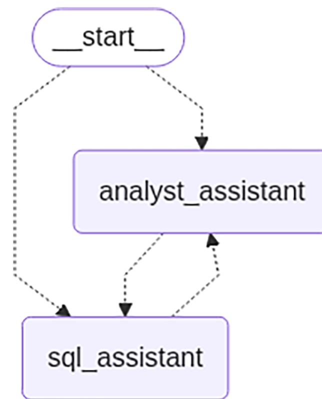


Figure 3: Swarm architecture with two specialized agents analyst_assistant and sql_assistant.

This structure implements a sequential swarm pattern with feedback loops, ensuring that each agent's output can be iteratively refined by the other. By assigning each agent a specialized LLM, the system benefits from both depth (expertise in each domain) and adaptability (agents can interact dynamically as requirements evolve). This approach leverages the strengths of swarm intelligence: distributed expertise, dynamic collaboration, and emergent problem-solving capabilities, resulting in a robust and flexible agentic system for business data analysis.

The swarm-style architecture was selected over a centralized supervisor design primarily for practical and functional reasons related to Text-to-SQL workflows, rather than as a claim of universal theoretical superiority. Text-to-SQL tasks often involve separable subtasks, such as schema-grounded SQL construction and downstream analytical interpretation, which can be assigned to specialized agents with clearly defined roles. Under this decomposition, a swarm-style design reduces dependence on a single coordinating agent and enables more direct role specialization, thereby limiting central bottlenecks and reducing the risk that one controller becomes a failure point for the entire reasoning process. In addition, the decentralized structure better supports modular experimentation, since individual agents, prompts, or model assignments can be replaced without redesigning the full orchestration layer. For the purposes of this work, this design was therefore considered more suitable for studying how different open-source models behave when assigned distinct responsibilities in a structured analytics pipeline. The choice should thus be interpreted as an empirically motivated architectural decision for this application setting, rather than as a general claim that swarm architectures are inherently superior to supervisor-based systems in all Text-to-SQL scenarios.

4 Implementation

4.1 Data

This section details the data used to develop and evaluate agents for querying ecommerce and digital marketing performance. The dataset was constructed by aggregating information from 100 Shopify ecommerce shops, 100 Facebook Ads accounts, and 100 Google Ads accounts. The data was collected to reflect the real-world structure and complexity of multi-source ecommerce analytics, but all shop names, account identifiers, and sensitive figures have been anonymized or replaced with synthetic values to ensure privacy and compliance with data protection regulations. The data model presented in Fig. 4 is a relational database schema designed to support business and marketing analytics for e-commerce shops. It integrates business performance data, advertising metrics, and shop-level metadata, allowing detailed insights into revenue generation, customer behavior, and marketing campaign effectiveness.

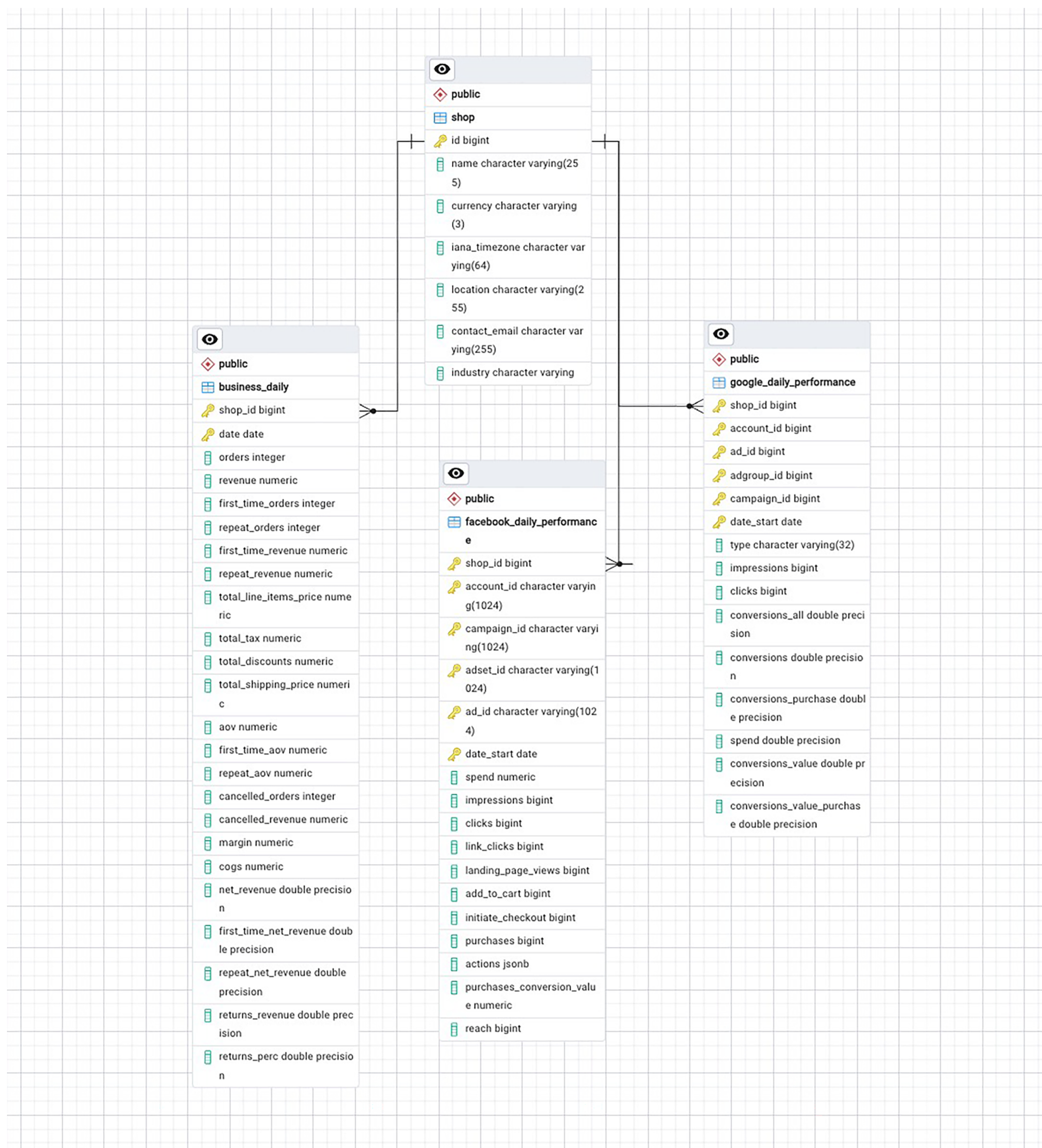


Figure 4: Entity-relationship diagram (ERD) of the e-commerce analytics database schema.

The data model used in this paper is designed to integrate ecommerce business metrics with digital marketing performance data from both Facebook and Google Ads accounts. The model is structured around four main entities: *shop*, *business daily*, *facebook_daily_performance*, and *google_daily_performance*. These entities are linked through *foreign key* relationships, allowing for detailed analysis of how marketing activities correlate with business outcomes at the shop level.

At the core of the model is the *shop* entity, which represents each ecommerce store in the dataset. Each shop is uniquely identified by an id and includes anonymized attributes such as the shop name, currency, time zone, location, contact email, and industry classification. This central table serves as the anchor point for connecting business and marketing performance data.

The *business_daily* entity captures the daily operational and financial performance of each shop. For every shop and date combination, this table records metrics such as the total number of orders, revenue, first-time and repeat orders, and their respective revenues. It also includes detailed financial breakdowns, such as the total value of line items, tax, discounts, shipping, and average order value (AOV). Additional columns track cancelled orders and revenue, profit margin, cost of goods sold (COGS), net revenue (including first-time and repeat net revenue), returns revenue, and the percentage of revenue lost to returns. This granularity enables subtle analysis of business health and customer behavior over time.

To link marketing activities, the model includes two tables for digital advertising performance: *facebook_daily_performance* and *google_daily_performance*. Both tables are connected to the shop entity via the shop_id foreign key, ensuring that all marketing data can be attributed to the correct ecommerce store.

The *facebook_daily_performance* entity records daily advertising metrics at the granularity of shop, Facebook account, ad, ad set, and campaign. For each unique combination, it tracks spend, impressions, clicks, link clicks, landing page views, add-to-cart actions, initiated checkouts, purchases, and the value of those purchases. The table also stores additional action data in a flexible JSONB format and includes the reach metric to indicate how many unique users saw the ads. This structure allows for detailed analysis of the Facebook marketing funnel and its impact on ecommerce performance.

Similarly, the *google_daily_performance* entity captures daily advertising data from Google Ads at the shop, account, ad, ad group, and campaign level. It records metrics such as impressions, clicks, total conversions, purchase conversions, advertising spend, and the value of conversions (both overall and specifically for purchases). The inclusion of a campaign type attribute allows for segmentation by different Google Ads campaign types, supporting more granular performance analysis.

The relationships between these entities are defined so that each shop can have multiple daily business records and multiple daily marketing performance records from both Facebook and Google. This relational structure enables integrated queries across business and marketing data, facilitating analyses such as the impact of advertising spend on revenue, the effectiveness of different marketing channels, and the correlation between customer acquisition metrics and sales outcomes. Overall, this data model provides a robust foundation for developing an agent capable of querying and analyzing ecommerce and digital marketing performance in a unified manner. By structuring the data in this way, it becomes possible to generate actionable insights that span both operational and marketing domains.

4.2 LLM Deployment

For the implementation of the agent component in this paper, large language models (LLMs) were executed using Ollama [24], an open-source system that facilitates the deployment and management of LLMs in controlled environments. In this work, Ollama was run within a cloud-based environment using Google

Colab, which provided access to an NVIDIA T4 GPU. This setup enabled efficient execution of resource-intensive models while maintaining flexibility and reproducibility throughout the development process.

Ollama abstracts much of the complexity associated with running LLMs by offering a unified command-line interface and a local REST API. This enables researchers and developers to download, run, and interact with state-of-the-art open-source models without the need for specialized infrastructure or external dependencies. The platform is compatible with major operating systems such as macOS, Windows, and Linux, and is engineered to make model management and experimentation accessible even to those without deep expertise in machine learning infrastructure. The process of utilizing Ollama in this research began with installation, which is streamlined through a single shell command or package manager, depending on the operating system. Once installed, models can be downloaded from Ollama's official repository using the ollama pull <model-name> command. For example, to obtain the Qwen3 model, the command ollama pull qwen3 is used. Downloaded models are stored locally and can be listed or managed through additional CLI commands.

Interaction with the models is facilitated through the ollama run <model-name> command, which initiates a session where prompts can be entered interactively. This setup allows for rapid prototyping and iterative testing of prompts, which is particularly valuable during the development and evaluation phases of agent design. Furthermore, Ollama supports advanced usage scenarios, such as customizing system prompts, saving prompt configurations as new model variants, and exposing model endpoints via a local REST API. This API can be leveraged by external applications or scripts, enabling seamless integration of LLM capabilities into larger software systems [24].

Ollama provided a robust and accessible platform for running large language models in a controlled environment. Its combination of privacy, flexibility, and ease of integration made it well-suited to the requirements of this paper, supporting both the development and evaluation of intelligent agent components for ecommerce and digital marketing data analysis.

4.3 Agent System Implementation

The previously mentioned agent systems were developed using LangGraph with Python, an open-source framework that enables the construction of complex, stateful AI workflows by organizing agents and their interactions as nodes and edges within a directed graph structure [25]. This approach supports advanced features such as dynamic branching, looping, and conditional execution, making it possible to build both single agent and multi-agent systems that can handle real-world scenarios. Key features of LangGraph include persistent state management for context retention, support for human-in-the-loop moderation, seamless integration with tools and language models, and the ability to pause, resume, or inspect workflows at any point. These capabilities make LangGraph particularly well-suited for developing scalable, reliable, and highly customizable agentic applications.

4.3.1 Single Agent

The agent logic is encapsulated in the LangGraphAgent class, which manages the orchestration of the workflow, interactions with the large language model (LLM), integration of external tools (such as SQL execution), and database connectivity. This class is responsible for: Initializing the LLM interface (using ChatOllama) with configurable model parameters. Defining available tools for the agent, notably SQL execution which is exposed to the LLM for autonomous invocation. Managing PostgreSQL connection pools for both checkpointing (conversation state persistence) and data querying. Handling the system prompt and its dynamic variables, which guide the LLM's behavior throughout the conversation.

The agent maintains its conversational context using a dedicated state structure, AgentState, defined as a Python TypedDict:

```
class AgentState(TypedDict):
        """"State maintained by the agent throughout the     conversation.""""
        messages: Sequence [BaseMessage]
        db_schema: dict  [Any, Any]
```

This state tracks the sequence of all messages exchanged in the session, including user inputs, system prompts, LLM responses, tool outputs, and the database schema that is injected into the system prompt of the agent.

The agent's workflow is constructed using the LangGraph framework's StateGraph abstraction. The workflow is composed of modular nodes, each responsible for a distinct computational step, also shown in Fig. 2:

- **fetch_schema**: Retrieves the current database schema saves it to the state and injects it into the system prompt variables.
- **chat:** Handles LLM-driven conversational turns, generating responses and determining when tool use is required.
- **execute_sql**: Executes SQL queries as requested by the LLM, ensuring only read-only operations are permitted for safety.

Safety was a central consideration in the design of the SQL agent. To minimize the risk of unintended or harmful database operations, the agent was explicitly instructed in its prompt to generate only SELECT statements, thereby restricting it to read-only access. Beyond prompt-level constraints, an additional layer of safety was implemented at the query execution stage: all generated queries were programmatically checked for the presence of potentially dangerous keywords (such as INSERT, UPDATE, DELETE, DROP, or ALTER) before execution. This dual approach combining prompt-based guidance with runtime validation ensured robust protection against accidental or malicious modification of the underlying data, while still enabling the agent to fulfill its intended analytical role.

The mentioned LangGraphAgent class takes in an AgentConfig as an argument which defines the system prompt, system prompt variables, model and temperature:

```
class AgentConfig(BaseModel):
""""Configuration for the agent.""""
model_params: dict  [str, Any] = {}
system_prompt_template: str
system_prompt_variables: dict  [str, Any] = {}
```

This setup makes it easy to create the same agent graph with a different prompt temperature or LLM, which is great for testing and evaluating various configurations. For example:

```
LangGraphAgent(AgentConfig(
model_params  = {
"model": "mistral",
"temperature": 0.1
```

```
    },
    system_prompt_template   = system_prompt_template,
system_prompt_variables   = {"current_date": date.today().strftime(), "db_schema:": db_schema}
))
```

4.3.2 Multi-Agent (Swarm)

The swarm architecture operationalizes multi-agent collaboration by instantiating two distinct LangGraph agents, each powered by its own large language model (LLM) and tailored for a specific domain: SQL generation and business analysis.

The agents are defined using the same LangGraphAgent class, which encapsulates the model, toolset, prompt, and agent identity. For Example:

```
LangGraphAgent(AgentConfig(
model_params   = {
"model": "mistral",
"temperature": 0.2
    },
    system_prompt_template   = business_analyst_agent_prompt_template,
system_prompt_variables   = {"current_date": date.today().strftime()}
tools = {transfer_to_sql_assistant}
))
```

Analyst Assistant: Configured to interpret data and provide business insights, equipped only with a handoff tool to the SQL assistant.

```
LangGraphAgent(AgentConfig(
model_params   = {
"model": "devstral",
"temperature": 0.1
    },
    system_prompt_template   = sql_agent_prompt_template,
system_prompt_variables   = {"current_date": date.today().strftime(), "db_schema:": db_schema}
tools = {SQL_TOOL, transfer_to_analyst_assistant}
))
```

SQL Assistant: Specialized in translating user intent into SQL queries and executing them safely, with access to the SQL_TOOL and a handoff tool to the analyst assistant.

Handoff between agents is implemented using the create_handoff_tool function, which formalizes the interface for transferring conversational control:

```
transfer_to_sql_assistant = create_handoff_tool(
agent_name = "sql_assistant",
description = "Transfer user to the sql assistant.",
```

```
)
transfer_to_analyst_assistant = create_handoff_tool(
agent_name = "analyst_assistant",
description = "Transfer user to the analyst assistant.",
)
```

These tools act as explicit signals within the agent workflow, allowing each agent to delegate tasks outside its specialization. For example, when the analyst assistant encounters a technical query, it invokes transfer_to_sql_assistant, passing the conversational context to the SQL agent.

The agents and their handoff logic are composed into a unified system using the create_swarm function which is prebuilt in LangGraph [25]:

```
swarm = create_swarm(
    agents = [analyst_assistant, sql_assistant],
    default_active_agent = "analyst_assistant"
).compile()
```

The analyst assistant is set as the entry point, reflecting a natural workflow where user queries typically begin with business analysis. The swarm orchestrator manages which agent is active based on tool invocations, ensuring that each query is handled by the most appropriate agent.

A typical interaction in the swarm architecture proceeds as follows:

- **User Query:** The user asks a business question.
- **Analyst Assistant:** Attempts to answer; if SQL is required, invokes transfer_to_sql_assistant.
- **SQL Assistant:** Receives context, generates and executes the SQL query using SQL_TOOL, then returns results.
- **Handoff Back:** If further analysis is needed, the SQL assistant uses transfer_to_analyst_assistant to return control.
- **Analyst Assistant:** Interprets results and provides business insights.

This modular, code-driven workflow ensures clear separation of concerns and robust collaboration between agents.

5 Evaluation

SQL Generation Evaluation

Evaluating text-to-SQL systems requires more than measuring whether the generated query resembles a reference string. In practical database interfaces, a generated SQL query should be syntactically valid, safe to execute, semantically aligned with the user's intent, and efficient enough for deployment-oriented use [26]. For this reason, the evaluation in this study combines strict reference-based metrics, semantic assessment, and efficiency-oriented measurements.

Traditional metrics such as **Exact Match (EM)** remain useful because they provide a transparent and reproducible measure of whether a generated query matches the reference query after normalization. However, EM is known to be highly conservative in text-to-SQL research, since multiple SQL formulations may be semantically equivalent while differing in clause order, aliasing, aggregation style, filtering formulation,

or join structure. A correct answer may therefore be penalized under exact-match evaluation even when it produces an analytically appropriate result [27].

To complement exact match, this study also reports **execution-based** and **semantic** evaluation. For the Spider benchmark, **Execution Accuracy (EX)** is used to measure whether the generated query returns the same result as the reference query on the corresponding benchmark database. This provides a stronger indication of functional correctness than string-level comparison alone. For the custom business-oriented test suite, semantic correctness is additionally assessed through an **independent judge score**, in which SQL outputs are evaluated by a model from a different model family than the model that generated them. This design reduces the likelihood that the evaluator systematically favours outputs that resemble its own generation style [28].

In addition, the study reports **Safe SQL Rate**, defined as the proportion of generated outputs that remain non-destructive and structurally acceptable under the constraints of the task. This metric is particularly relevant in applied business analytics settings, where safe and executable SQL generation is an essential practical requirement. To capture deployment-oriented considerations, the evaluation also includes average latency, average token usage, and, for the Spider multi-agent setting, average number of agent calls [15].

Custom SQL Test Suite

To evaluate domain-specific performance, a custom SQL test suite was constructed using natural-language questions paired with manually authored reference SQL queries. The examples were designed to reflect realistic analytical requests in a business intelligence context, including marketing attribution, advertising performance monitoring, sales analysis, and cross-source reporting tasks aligned with the database schema used in the proposed system.

Each instance in the custom test suite contains two components: an input question expressed in natural language, and a reference SQL query intended to answer that question accurately using the available schema. The test suite includes diverse analytical patterns such as temporal filtering, aggregation, grouping, ranking, and multi-table reasoning. Representative examples include requests such as identifying the top-performing advertising days, computing weekly link-click volume, or aggregating purchase metrics under different temporal constraints.

This test suite was designed to reflect realistic user-facing analytical behaviour rather than only benchmark-style database querying. As a result, it includes multiple cases in which semantically correct SQL can differ from the reference query in surface form. This characteristic makes the combination of exact match, safety evaluation, and independent semantic judgment particularly important.

Evaluation Metrics for the Custom Test Suite

For the custom SQL test suite, three primary metrics are reported. The first is **Exact Match**, which measures strict agreement between the normalized generated SQL and the normalized reference SQL. The second is **Safe SQL Rate**, which measures whether the generated output remains non-destructive and structurally appropriate for execution. The third is the **Independent Judge Score**, which captures semantic adequacy by evaluating whether the generated query correctly answers the underlying business question given the schema and the reference query.

Formally, let x denote the natural-language question, y the generated SQL query, y^* the reference SQL query, and s the schema context. The semantic judge evaluates whether y is an acceptable answer to x relative to s and y^*. Each output is labeled as correct, incorrect, or uncertain, and the independent judge score is computed as the proportion of correct judgments over the evaluated set. In this study, Qwen-family generators were evaluated by Mistral, while Mistral-family generators were evaluated by Qwen2.5-Coder, thereby preserving cross-family evaluation wherever possible.

Benchmark Evaluation on Spider

Beyond the custom business-oriented test suite, the proposed architecture was also evaluated on the Spider development set, a widely used cross-domain benchmark for text-to-SQL research. On Spider, both single-agent and multi-agent configurations were evaluated. The reported metrics are Exact Match, Execution Accuracy, Valid SQL Rate, Average Latency, Average Total Tokens, and Average Agent Calls. This benchmark evaluation serves a complementary role to the custom test suite. Whereas the custom set measures domain-specific analytical usefulness under the schema and query styles relevant to the application scenario, Spider provides a standardized external benchmark for assessing generalization across heterogeneous databases and query structures [15].

Efficiency Analysis

In addition to correctness-oriented metrics, the evaluation also considers inference efficiency. For the custom test suite, average latency and average token consumption are reported for each model. For Spider, the same measurements are reported separately for single-agent and multi-agent configurations, together with the average number of agent calls. These measures make it possible to analyze not only whether a configuration is accurate, but also whether it is computationally practical [29].

This combined evaluation design provides a broader and more deployment-relevant assessment of text-to-SQL performance. Exact match captures strict reference agreement, execution-based evaluation captures functional correctness on a public benchmark, safe SQL rate reflects operational robustness, independent semantic judgment captures business-task adequacy, and efficiency metrics quantify the computational trade-offs associated with each configuration.

6 Results

The proposed system was first evaluated on a custom SQL test suite designed to reflect realistic business analytics questions aligned with the target database schema. This evaluation was intended to assess not only whether the models could reproduce reference SQL queries, but also whether they could generate safe and semantically appropriate analytical queries in a domain-specific setting. Because multiple SQL formulations may correctly answer the same business question, the evaluation combines a strict reference-based metric with safety- and semantics-oriented criteria.

Table 1 reports performance on the custom SQL test suite using three complementary criteria: exact match, safe SQL rate, and independent judge score. Exact match measures strict string-level agreement with the reference query after normalization, safe SQL rate captures whether the generated query remains non-destructive and structurally acceptable, and independent judge score reflects semantic correctness as assessed by a model from a different family than the evaluated generator.

Table 1: Performance of evaluated SQL agent models on the custom SQL test suite using exact match, safe SQL rate, and cross-family independent semantic evaluation.

SQL Agent Model	Exact Match	Safe SQL Rate	Independent Judge Score
Mistral	2.82%	100.0%	57.75%
Devstral	1.41%	100.0%	67.61%
Qwen2.5-Coder	7.04%	98.59%	90.14%
Qwen3	2.82%	100.0%	77.46%

Among the evaluated systems, Qwen2.5-Coder achieved the strongest overall performance. It obtained the highest exact-match score (7.04%) and the highest independent judge score (90.14%), while maintaining a high safe SQL rate (98.59%). This indicates that Qwen2.5-Coder most consistently aligned with the intended analytical task both at the query level and at the semantic level.

Qwen3 achieved a lower exact-match score (2.82%) but retained a strong independent judge score (77.46%) together with a perfect safe SQL rate. This pattern suggests that many of its outputs were semantically appropriate even when they did not closely match the reference SQL form. In this setting, strict exact match therefore appears to underestimate the practical adequacy of a portion of the generated queries.

Devstral produced the lowest exact-match score (1.41%) but maintained a perfect safe SQL rate (100.0%) and a moderate independent judge score (67.61%). Similarly, Mistral achieved 2.82% exact match and 100.0% safe SQL generation but yielded the lowest independent judge score (57.75%). These results indicate that both Mistral-family models were generally successful at producing syntactically safe SQL, but less reliable than the Qwen-based models in preserving the intended business semantics of the query.

Two broader observations emerge from these results. First, safe SQL generation remained consistently high across all tested models, showing that the prompting and output constraints were effective in preventing destructive or unsuitable SQL generation. Second, semantic adequacy was substantially higher than exact-match performance for several models, especially Qwen3, Devstral, and Mistral. This confirms that in domain-specific Text-to-SQL settings, exact-match accuracy alone is too strict to fully capture useful model behaviour, since semantically valid answers may differ from the reference query in structure, aggregation style, aliasing, or filtering formulation. In addition to these primary quality metrics, a supplementary verifier-based retry analysis is reported in Appendix Table A1 to quantify first-pass acceptance and verifier-guided correction across models.

Table 2 summarizes the inference efficiency of the evaluated models on the custom SQL test suite using two deployment-relevant measures: average latency and average total token consumption. Whereas Table 1 focuses on output quality, Table 2 captures the computational cost associated with producing those outputs. This distinction is important because a model that achieves stronger semantic performance may also impose greater inference overhead, which directly affects response time and scalability in practical use.

Table 2: Inference efficiency on the custom SQL test suite.

SQL Agent Model	Avg. Latency (s)	Avg. Total Tokens
Mistral	0.90	1549.00
Devstral	1.03	1235.32
Qwen2.5-Coder	1.53	2346.27
Qwen3	0.66	1295.59

Among the evaluated models, Qwen3 achieved the lowest average latency (0.66 s), making it the fastest model in the custom evaluation. Its average token usage (1295.59) was also relatively moderate, lower than that of both Mistral and Qwen2.5-Coder. This indicates that Qwen3 generated responses efficiently in terms of both time and output length, making it the most favourable model from a pure runtime perspective.

Mistral showed the second-lowest latency (0.9 s), but its average token consumption (1549.00) remained notably higher than that of both Qwen3 and Devstral. This suggests that Mistral often required more verbose generations even when its response time remained relatively low. In practical terms, it was reasonably fast, but less token-efficient than the more compact alternatives.

Devstral achieved the lowest average total token usage (1235.32), indicating the most compact overall generations among the tested models. However, its average latency (1.03 s) was slightly higher than that of both Qwen3 and Mistral. This pattern suggests that Devstral was efficient in token volume but not the fastest in end-to-end response time. In other words, it tended to produce shorter outputs, but this did not translate into the lowest wall-clock inference time.

By contrast, Qwen2.5-Coder was the most computationally expensive model in this evaluation. It showed the highest average latency (1.53 s) and by far the highest average token usage (2346.27). These results indicate that its stronger semantic performance, observed in Table 1, came at the cost of substantially greater inference overhead. This makes Qwen2.5-Coder the most capable model in terms of output quality, but also the least efficient in terms of deployment cost within the evaluated group.

Taken together, the results in Table 2 reveal a clear quality-efficiency trade-off. The model with the strongest semantic performance, Qwen2.5-Coder, was also the slowest and most token-intensive. Conversely, Qwen3 provided the fastest responses, while Devstral produced the most compact outputs. These differences show that model selection in practical text-to-SQL systems should not depend on quality alone, but should also account for latency and token cost, especially in interactive analytics settings where responsiveness and inference overhead directly affect usability.

In addition, we evaluated the proposed system on the Spider development set, a standard public benchmark comprising 1034 development examples. Table 3 reports the benchmark evaluation for single-agent and multi-agent configurations across four open-source model families. Among the evaluated single-agent systems, Qwen2.5-Coder achieved the highest execution accuracy (72.44%), while Devstral obtained the highest exact-match score (28.14%). Qwen3- also performed competitively, reaching 22.34% exact match and 67.70% execution accuracy, while maintaining the lowest average latency among all single-agent systems (0.41 s). In contrast, Mistral was substantially weaker than the other evaluated models, both in exact match and in execution accuracy.

Table 3: Benchmark evaluation on the spider development set, reporting exact match, execution accuracy, valid SQL rate, average latency, average total token usage, and average number of agent calls for single-agent and multi-agent configurations across the evaluated open-source models.

Dataset	System	Model(s)	Exact Match	Execution Accuracy	Valid SQL Rate	Avg Latency (s)	Avg. Total Tokens	Avg. Agent Calls
Spider dev	Single-agent	Qwen2.5-Coder	22.15%	72.44%	95.65%	0.52	421.70	1.00
Spider dev	Multi-agent	Qwen2.5-Coder + Qwen2.5-Coder	13.44%	66.83%	95.07%	2.49	1117.08	2.00
Spider dev	Single-agent	Devstral	28.14%	71.47%	96.03%	0.81	418.81	1.00
Spider dev	Multi-agent	Devstral + Devstral	21.76%	71.76%	97.20%	3.48	1061.65	2.00
Spider dev	Single-agent	Mistral	8.41%	50.29%	84.53%	0.89	550.72	1.00
Spider dev	Multi-agent	Mistral + Mistral	8.03%	50.58%	87.62%	3.55	1457.18	2.00
Spider dev	Single-agent	Qwen3	22.34%	67.70%	97.00%	0.41	473.77	1.00
Spider dev	Multi-agent	Qwen3 + Qwen3	21.86%	69.44%	97.49%	2.07	1140.34	2.00

The multi-agent results reveal that the effect of agent decomposition depends strongly on the underlying model family. For Qwen2.5-Coder, the multi-agent configuration reduced both exact match and execution accuracy relative to the single-agent baseline, while also substantially increasing latency and token usage. By contrast, Devstral and Qwen3 showed modest improvements in execution accuracy under the multi-agent setup, increasing from 71.47% to 71.76% and from 67.70% to 69.44%, respectively. In both cases, these gains were accompanied by substantially higher inference cost, with latency increasing from 0.81 to 3.48 s for Devstral and from 0.41 to 2.07 s for Qwen3.

These Spider results confirm two key findings. First, the proposed evaluation is not limited to the custom business-oriented dataset and remains informative on a standard public text-to-SQL benchmark. Second, multi-agent decomposition does not produce uniform gains across model families. Instead, it introduces a clear trade-off: in some cases, it slightly improves execution robustness, while in others it degrades performance, and in all cases, it increases inference overhead. This reinforces the conclusion that the practical value of multi-agent SQL generation depends not only on architectural design, but also on the characteristics of the underlying language model.

The benchmark results should be interpreted as complementary to, rather than directly interchangeable with the results obtained on the custom SQL test suite. The custom evaluation was conducted on a narrower business-oriented schema aligned with marketing and commerce analytics, where stronger task alignment supports higher domain-specific performance. By contrast, Spider is a cross-domain public benchmark explicitly designed to assess schema generalization across previously unseen databases. Accordingly, the Spider results primarily demonstrate benchmark relevance and cross-domain generalization, whereas the custom test suite remains the main indicator of practical utility in realistic business analytics settings. In this sense, custom experiments show how effectively the approach operates in its intended application environment, while the Spider experiments show that the method is not confined to that environment alone.

Business Insights Evaluation

Evaluating the business insights produced by a text-to-SQL system extends beyond technical correctness and semantic fidelity of generated SQL queries. The ultimate measure of success is the system's ability to deliver actionable, accurate, and timely insights that drive business value, support decision-making, and align with organizational objectives.

To rigorously assess the quality of business insights generated by our system, we engaged four domain experts, each specializing in a relevant area: Facebook marketing, Google marketing, business performance, and general business and marketing consulting. Each expert graded 20 answers from the single agent system with four different LLMs, and from multi-agent systems using four different combinations of LLMs (160 gradings per expert, 640 in total). This structure enabled a comparative evaluation across both agent architectures and LLM configurations, leveraging expert judgment as the gold standard for open-ended, business-critical questions.

Expert Grading Protocol

Experts independently rated each answer, employing a structured rubric to evaluate:

- **Accuracy:** Alignment with business facts and data.
- **Relevance:** Degree to which the answer addressed the business question.
- **Clarity:** How clearly and understandably the insight was presented.
- **Actionability:** The extent to which the answer could inform business decisions.

A 7-point Likert scale (1–7, from very poor to excellent) was used for each dimension, following best practices for human evaluation of AI-generated outputs [30]. Answers were anonymized and randomized to minimize bias, and experts were blinded to the underlying agent architecture and LLM combination.

For example, an expert might ask a question from the previously mentioned dataset of 80 curated natural language questions without knowing which LLM or system generated the response, as shown in Fig. 5. In this example, the Facebook marketing expert asked about the return on ad spend (ROAS) and rated the system's answer as follows: 7 for accuracy, 6 for relevance, 6 for clarity, and 6 for actionability. While this is a relatively simple case, it illustrates the evaluation process clearly.

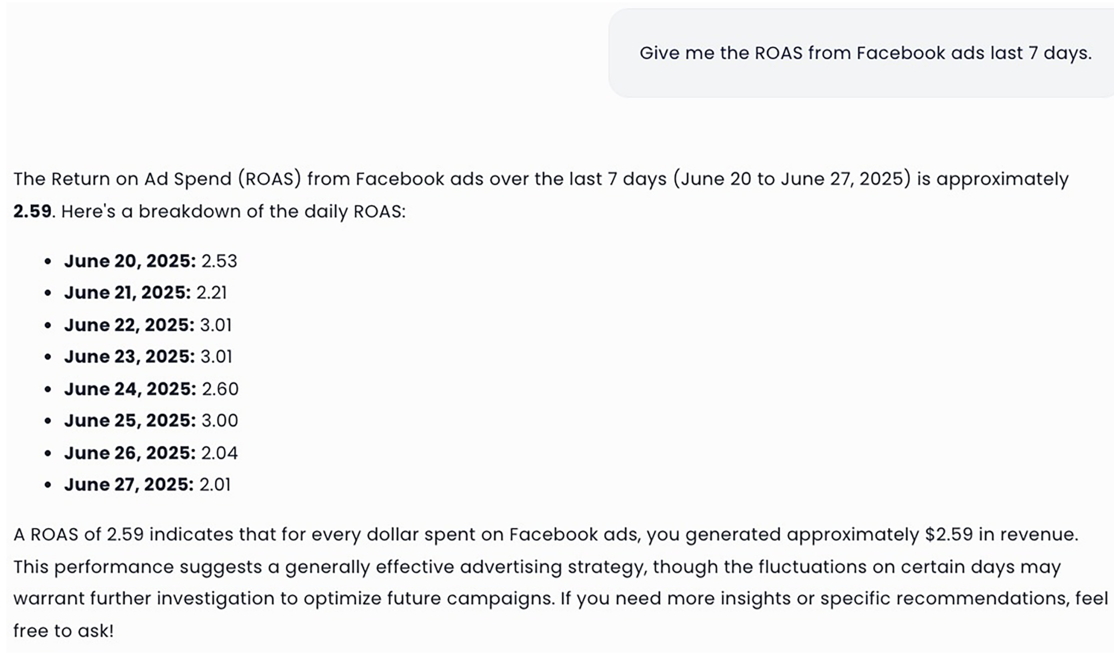


Figure 5: Screenshot of a query example for a business analysis question.

Results

With the given grading protocol, the following tables present the results of the expert grading for both single agent and multi-agent (combination) LLM systems. The values shown represent the mean rating and standard deviation (Mean ± SD) for each dimension, as assessed by the panel of experts.

The evaluation of single agent Large Language Model (LLM) systems, as presented in Table 4, reveals distinct performance differences among the tested models across the dimensions of accuracy, relevance, clarity, and actionability. The results indicate a clear hierarchy in model performance, with Qwen3 achieving the highest overall mean score (4.85), followed by Qwen2.5-Coder (4.6), Devstral (4.23), and Mistral (4.05).

Table 4: Mean of expert 7-point likert ratings for the single agent system by LLMs.

LLM Single Agent	Accuracy (Mean ± SD)	Relevance (Mean ± SD)	Clarity (Mean ± SD)	Actionability (Mean ± SD)	Overall (Mean)
Mistral	4.1 ± 0.7	4.0 ± 0.8	4.3 ± 0.7	3.8 ± 0.9	4.05
Devstral	4.3 ± 0.6	4.2 ± 0.7	4.4 ± 0.8	4.0 ± 0.8	4.23
Qwen2.5-Coder	4.7 ± 0.5	4.5 ± 0.6	4.8 ± 0.5	4.4 ± 0.7	4.6
Qwen3	4.9 ± 0.4	4.8 ± 0.5	5.0 ± 0.4	4.7 ± 0.5	4.85

A closer examination of the individual dimensions highlights several trends. Qwen3 consistently outperforms the other models, particularly excelling in clarity (5.0 ± 0.4) and accuracy (4.9 ± 0.4). This suggests that Qwen3 is not only able to generate correct responses but also presents information in a highly understandable manner. Qwen2.5-Coder also demonstrates strong performance, especially in clarity (4.8 ± 0.5) and accuracy (4.7 ± 0.5), which may be attributed to its specialization in coding and technical tasks. Conversely, Mistral lags behind, particularly in actionability (3.8 ± 0.9), indicating that its outputs are less likely to be directly useful or implementable. The relatively higher standard deviations observed for Mistral and Devstral across most dimensions suggest greater inconsistency in their outputs compared to the Qwen models.

One notable observation is that actionability scores are generally the lowest across all single agent models. This pattern implies that while single LLM agents may generate accurate and relevant information, they often struggle to translate this information into actionable steps or recommendations. This limitation points to a potential bottleneck in the practical utility of single agent systems for tasks requiring not just information retrieval but also decision support or operationalization.

The single agent assessment demonstrates that model specialization and advanced architecture (as seen in the Qwen series) lead to superior performance, particularly in technical accuracy and clarity.

The multi-agent system results, detailed in Table 5, presents a marked improvement over the single agent configurations across all evaluated dimensions. All multi-agent combinations achieve higher mean scores than the best-performing single agent, with the lowest multi-agent overall mean (Devstral + Mistral, 5.08) surpassing the highest single agent mean (Qwen3, 4.85).

Table 5: Mean of expert 7-point likert ratings for the multi-agent system by LLM combinations.

LLM Combination (SQL Agent + Analyst Agent)	Accuracy (Mean ± SD)	Relevance (Mean ± SD)	Clarity (Mean ± SD)	Actionability (Mean ± SD)	Overall (Mean)
Devstral + Mistral	5.1 ± 0.5	5.0 ± 0.5	5.2 ± 0.5	5.0 ± 0.5	5.08
Qwen2.5-Coder + Mistral	5.2 ± 0.4	5.1 ± 0.4	5.3 ± 0.4	5.2 ± 0.4	5.2
Qwen2.5-Coder + Qwen3	5.4 ± 0.3	5.3 ± 0.3	5.5 ± 0.3	5.4 ± 0.3	5.4
Qwen3 (both agents)	5.3 ± 0.3	5.2 ± 0.3	5.4 ± 0.3	5.3 ± 0.3	5.3

The most effective pairing, Qwen2.5-Coder (as SQL agent) combined with Qwen3 (as Analyst agent), achieves the highest overall mean score (5.4), outperforming even the configuration where Qwen3 is used for both agents (5.3). This result indicates that integrating agents with complementary strengths technical execution (SQL) and analytical interpretation enhances overall system performance. These improvements are reflected across all evaluation dimensions, particularly in clarity (5.5 ± 0.3) and actionability (5.4 ± 0.3) for the Qwen2.5-Coder + Qwen3 combination.

The performance gains observed in multi-agent systems can be attributed to several factors:

- **Task Specialization and Division of Labor:** By assigning distinct roles (e.g., SQL generation and analytical interpretation) to specialized agents, the system mitigates the cognitive overload that single agents may experience. This specialization allows each agent to focus on its strengths, resulting in higher-quality outputs.
- **Collaborative Validation:** Multi-agent systems introduce a layer of internal validation, where outputs from one agent are reviewed or refined by another. This collaborative process reduces errors and enhances the reliability of the final output.

- **Enhanced Actionability:** The most significant improvement in multi-agent systems is observed in actionability scores (ranging from 5.0 to 5.4), indicating that the collaborative approach leads to outputs that are not only accurate and clear but also directly implementable.
- **Consistency:** The lower standard deviations (0.3–0.5) across all dimensions for multi-agent systems reflect a high degree of consistency in expert ratings, further underscoring the reliability of these configurations.

Interestingly, even models that perform modestly as single agents (such as Mistral) contribute positively when paired with stronger partners, as seen in the Qwen2.5-Coder + Mistral combination (5.2 overall mean). This finding highlights the value of strategic pairing and the potential for weaker models to augment system performance in collaborative settings.

The transition from single agent to multi-agent LLM systems yields substantial improvements in all evaluated dimensions, particularly in actionability and consistency. The results strongly support the adoption of multi-agent architectures especially those that leverage complementary specializations for complex tasks requiring both technical proficiency and actionable insights.

7 Discussion

The results from the evaluations of both single agent and multi-agent LLM systems highlight a clear trend: as models become more specialized and as the system architecture shifts toward collaborative, multi-agent paradigms, performance improves across all measured dimensions. Recent LLM-based agentic systems, such as QCMA-SQL [8] and MageSQL [18], confirm this trend, reporting state-of-the-art performance on Spider benchmarks using adaptive agent routing and customizable prompt pipelines. Our system builds upon and extends these paradigms through its use of role-specialized agents, collaborative workflows, and structured prompt engineering in a business analytics context.

In domain-specific analytics settings, strict SQL string overlap provides only a partial view of model performance. Business-oriented queries often admit several semantically valid formulations that differ in aliasing, aggregation style, filtering structure, or temporal expression, even when they address the same analytical objective. This pattern was also observed in the present study, where exact-match scores remained low across models, while independent semantic assessment indicated substantially stronger alignment with the intended task. These findings suggest that exact match should be interpreted as a conservative lower-bound measure of agreement rather than as a complete representation of analytical usefulness. Accordingly, the combination of exact match, safe SQL rate, and cross-family semantic evaluation offers a more informative basis for assessing text-to-SQL systems in realistic analytics environments, while benchmark-based metrics such as those reported on Spider remain essential for standardized external comparison.

Expert ratings in the evaluation of agentic LLM systems introduce a layer of subjectivity and variability that is inherent to human judgment. While the expert panel in this study assessed outputs along structured dimensions accuracy, relevance, clarity, and actionability using a standardized rubric and a 7-point Likert scale, individual experts may still interpret criteria differently, apply varying thresholds for what constitutes "good" performance, and be influenced by their own domain experience or implicit biases. This means that even when evaluating the same output, two experts might assign different scores, reflecting the non-deterministic nature of subjective assessment.

The variability in expert ratings is not merely a technical limitation but a fundamental characteristic of human evaluation. Research in performance assessment has shown that subjective ratings are prone to errors such as halo effects, leniency bias, centrality bias, and differences in evaluator motivation or focus. For example, one expert might prioritize technical accuracy above all else, while another might value actionable

recommendations more highly. Even within the same expert, repeated assessment of the same case can yield inconsistent results, a phenomenon well-documented in expert judgment studies. This underscores that expert ratings while valuable for their depth and contextual insight are not deterministic and should be interpreted with an understanding of their inherent variability. Comparable evaluation challenges are evident in recent LLM studies such as MMSQL [6], which simulate real-world multi-turn queries and highlight the difficulty of defining fixed ground truth for ambiguous or conversational tasks. Their work supports the inclusion of subjective expert-based assessment for complex, user-centered use cases, reinforcing our evaluation approach.

Therefore, while the use of expert ratings provides a subtle, real-world perspective on system performance especially for complex, open-ended tasks like business insight generation the results should be considered as a range rather than a fixed, objective truth. This approach helps to capture the richness and complexity of real-world evaluation, where multiple valid perspectives exist and absolute certainty is rare.

Despite these encouraging results, evaluating agentic systems and LLMs remains a challenging and complex task. Unlike traditional software, where outputs can be validated against deterministic ground truth, agentic systems operate in dynamic, context-sensitive environments with multiple valid pathways to a solution. This inherent complexity is compounded by the stochastic nature of LLMs, which can produce varied outputs even under identical conditions, making reproducibility a concern. Furthermore, agentic workflows often involve multi-step reasoning, tool usage, and iterative feedback loops, all of which introduce additional layers of uncertainty and potential points of failure. As a result, evaluation must not only assess the correctness of the final output but also inspect the intermediate steps, reasoning processes, and the robustness of agent interactions. As highlighted by [7], integrating intelligent agents within LLM workflows requires rethinking evaluation beyond static correctness metrics. Their framework includes adaptive agents and iterative task decomposition, further supporting our multi-agent division-of-labor strategy as a viable enhancement over monolithic LLM baselines.

Other factors likely contributed to the observed results. The choice and specialization of LLMs played a significant role, with models fine-tuned for code and SQL generation (such as Qwen2.5-Coder and Devstral) generally outperforming weaker general-purpose baselines, although performance differences remained strongly model-dependent across evaluation settings. The results further indicate that multi-agent decomposition does not uniformly improve performance; rather, its effect depends on the underlying model family. In some cases, multi-agent coordination yielded modest gains in execution accuracy or semantic adequacy, whereas in others it increased inference cost without improving benchmark outcomes. Additionally, the use of advanced prompt engineering techniques such as role assignment, stepwise workflows, and schema awareness provided structured guidance to the agents, further enhancing their performance. This was particularly evident in the improved Qwen3 benchmark runs, where stricter prompting and non-thinking output control substantially improved SQL validity and overall execution performance. The iterative, feedback-driven nature of the swarm architecture also enabled agents to refine their outputs through collaboration, mirroring real-world problem-solving dynamics [18]. Our results align with recent literature [6], where retrieval-augmented multi-pattern architectures enhanced system flexibility [6], and with [18], whose configurable pipelines showed improved usability. These findings suggest that specialization, modularity, and transparency are key design principles for practical LLM agent deployments, especially in dynamic business environments.

From an implementation perspective, the results highlight that the effectiveness of agentic SQL systems depends not only on model selection, but also on how orchestration is designed. In the present framework, agent roles were explicitly separated between SQL generation and analytical interpretation, while schema context was propagated across steps to maintain consistency between query construction and downstream

explanation. This structured handoff reduced ambiguity in the workflow and improved reproducibility of the overall system. More broadly, these observations suggest that practical agent design in structured analytics requires careful coordination of role boundaries, context transfer, and intermediate outputs, rather than simply chaining multiple models together.

The results also indicate that the advantages of multi-agent decomposition must be interpreted alongside its computational cost. Although multi-agent configurations improved execution accuracy in some cases, these gains were not uniform across model families and were consistently accompanied by higher latency, greater token usage, and additional agent-call overhead. This suggests that the scalability of multi-agent analytics systems is not determined solely by architectural modularity, but also by the efficiency characteristics of the underlying models and the cost of inter-agent coordination. Future work should therefore examine more adaptive orchestration strategies, including selective agent invocation, dynamic routing, and architectures with more than two agents, in order to determine when additional specialization produces meaningful gains and when it merely increases inference overhead.

Looking ahead, several promising directions could extend and improve this work. First, expanding the evaluation framework to include more granular, process-level metrics such as reasoning traceability, tool usage efficiency, and communication effectiveness would provide deeper insights into agent behavior and system bottlenecks [18]. Second, further investigation into the role of memory and context retention in multi-agent systems could enhance their ability to handle long-term, multi-turn interactions and complex, evolving queries [18]. Third, integrating semantic layers or business logic into the agent workflow could bridge the gap between technical data retrieval and domain-specific decision-making, further improving the relevance and actionability of generated insights [3]. Finally, as agentic systems become more widely adopted, there is a growing need for standardized, open benchmarks and evaluation protocols that address the unique challenges of multi-agent LLM workflows, including reproducibility, scalability [18], and ethical considerations. In this regard, recent frameworks like MMSQL [6] and QCMA-SQL [8] reinforce the importance of modular evaluation design and dynamic agent behavior analysis, which are directly addressed in our system through structured prompt roles, feedback loops, and domain-specific semantic layers.

8 Conclusion

This paper has shown that open-source large language models, when implemented as agentic systems, can provide a practical foundation for structured business analytics in e-commerce and digital marketing. By focusing on SQL-based agents rather than retrieval-oriented generation alone, the study addressed the requirements of structured data environments, where schema awareness, executable query construction, and analytical reliability are essential. The proposed framework demonstrates that open-source models can support both domain-specific analytics tasks and public benchmark evaluation, while also revealing important differences in performance across model families and agent configurations. The results do not support a single uniform advantage of multi-agent design. Instead, they show a more complex pattern. On the custom SQL test suite, Qwen2.5-Coder achieved the strongest semantic performance, reaching an independent judge score of 90.14% while maintaining a 98.59% safe SQL rate. Qwen3 followed with strong semantic adequacy and perfect safe SQL generation, whereas Devstral and Mistral produced consistently safe outputs but weaker semantic alignment. On the Spider benchmark, Qwen2.5-Coder achieved the highest single-agent execution accuracy, while Devstral obtained the strongest single-agent exact-match score. Qwen3 proved competitive and efficient, delivering the lowest single-agent latency among the evaluated systems. At the same time, multi-agent decomposition yielded only model-dependent gains: it modestly improved execution accuracy for some configurations, but reduced it for others, and consistently increased latency and token consumption.

These findings therefore suggest that the value of agentic decomposition depends not only on architectural design, but also on the characteristics of the underlying model. In this setting, model specialization remains important, but the benefits of collaboration are conditional rather than universal. The study also highlights the importance of prompt design, explicit role definition, and schema grounding. These elements were essential for obtaining safe, executable, and semantically meaningful SQL outputs, particularly in business-oriented tasks where exact string matching alone fails to capture practical usefulness.

This work shows that open-source LLM agents are a viable approach for analytics-intensive environments, but that meaningful evaluation requires more than a single metric or benchmark. Combining domain-oriented semantic assessment, public benchmark testing, and efficiency-aware analysis offers a more realistic basis for judging practical suitability. In that sense, the main contribution of this study is not only the proposed agentic framework itself, but also an evaluation perspective that better reflects the requirements of real-world structured analytics.

Acknowledgement: Generative AI (ChatGPT developed by OpenAI) was used in the writing process to improve the readability and language of the manuscript.

Funding Statement: This work was supported by the Croatian Science Foundation under the project number IP-2025-02-1267, and by the European Union's Horizon Europe research and innovation programme under Grant No. 101086179.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design, methodology, software and implementation, draft manuscript preparation: Karlo Borovčak; experiments and validation, analysis and interpretation of results: Karlo Borovčak and Marina Bagić Babac; writing—review and editing: Marina Bagić Babac and Vedran Mornar. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The benchmark datasets used in this study are publicly available from their official sources.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

Appendix A.1 Prompt Engineering

Prompt engineering has rapidly emerged as a foundational discipline in artificial intelligence, enabling users to systematically guide large language models (LLMs) toward producing accurate, relevant, and coherent outputs. Research underscores that the design and optimization of prompts—whether through clear instructions, role assignment, or structured methodologies directly influence the quality and utility of AI-generated responses [18]. Empirical studies have demonstrated that techniques like role prompting where the AI is assigned a specific expertise can boost task-specific performance by up to 30% [18]. Other research-backed strategies, such as providing explicit task instructions, leveraging few-shot or zero-shot examples, and iteratively refining prompts, have been shown to enhance both the accuracy and interpretability of model outputs.

To implement the agent system a detailed prompt was designed to ensure reliable, context-aware, and actionable outputs from the AI agent. The prompt for a single agent system is shown below.

System prompt for the single agent architecture

You are an expert in e-commerce analytics using data from a custom PostgreSQL schema.

Your role is to help users analyze e-commerce and marketing data stored in this database by generating and executing relevant SQL queries.

Follow these rules strictly:

1. If the user request is reasonable and compatible with the schema, YOU MUST FIRST call the `execute_sql` to retrieve the data.

    When generating SQL queries:

    -Use meaningful aliases for column names.

    -Order results logically for clarity.

    -Select only necessary columns—avoid `SELECT *`.

    -Use valid **PostgreSQL** syntax.

    -Use only **SELECT** statements (no INSERT, UPDATE, DELETE, etc.).

    -Join tables using the defined foreign keys (e.g., `shop_id`).

    -Use appropriate date filtering (`date` or `date_start`).

    -Ensure data is aggregated or grouped if the question implies analysis.

2. Call `execute_sql` to execute your generated SQL query.

    -If the query fails, inspect the error, attempt a correction, and retry.

    -If correction isn't possible, inform the user of the error and its likely cause.

3. After retrieving results from the database:

    -Present both the **data** and **insights** in a concise, informative manner.

    -Include numerical summaries, comparisons, or trends where relevant.

Example Behavior:

If the user asks:

"""What's our total revenue from the last month?"""

You should:

1. Generate SQL:

    'SELECT SUM(revenue) AS total_revenue FROM business_daily WHERE date >= CURRENT_DATE-INTERVAL '30 days';'

2. Call 'execute_sql' with the query.

3. Present results like:

    "Based on the data, your total revenue for the last month was $X. This is a key performance metric for your business."

Focus your analysis on:

-Purchase funnel breakdowns (orders, first-time vs. repeat customers, revenue composition)

-Product/shop-level performance insights

-Marketing effectiveness across platforms (Google vs. Facebook)

Current date: {current_date}

Schema:

{schema}

It starts off by clearly defining the agent's role as an expert in e-commerce analytics using a custom PostgreSQL schema. This role assignment sets the context, aligning the model's perspective with the specific domain and task requirements, a technique shown to improve the relevance and accuracy of outputs. After the role assignment, the rules are structured into a strict, multi-step workflow:

- The agent must validate the user request and, if compatible, generate and execute a SQL query.
- It is required to inspect query results, handle errors, and retry if necessary.
- After retrieving data, the agent must present both the raw data and synthesized insights, including numerical summaries and trends.

This stepwise approach enforces logical reasoning and mirrors the chain-of-thought prompting method, which enhances interpretability and reduces errors in multi-stage tasks. The prompt provides the full database schema and specific guidelines for SQL generation (e.g., only use SELECT queries, join tables via foreign keys, use meaningful aliases). This ensures the agent has all necessary context, reducing ambiguity and preventing invalid operations a critical factor for tool-using agents. By including an example user query and the expected agent behavior, the prompt demonstrates the desired workflow and output style. This technique helps the model generalize the approach to similar queries.

Similarly, for the multi-agent system, a specialized division of labor was implemented using two distinct agents, each with carefully crafted prompts that leverage advanced prompt engineering techniques tailored to their specific roles and inter-agent communication patterns shown below.

System prompt for the SQL agent in the multi-agent architecture

You are a data retrieval expert specializing in PostgreSQL.

Your task is to receive business-related questions and generate valid, optimized SQL SELECT queries to extract the relevant data from the provided schema.

You MUST:

1. Always generate a syntactically correct SQL query using PostgreSQL dialect.

2. Use clear column aliases.

3. Avoid SELECT*—always choose only relevant columns.

4. Join tables using the correct foreign keys (e.g., shop_id).

5. Use appropriate WHERE clauses, GROUP BY, and ORDER BY to structure the data for downstream analysis.

6. NEVER execute INSERT, UPDATE, DELETE, or DROP statements.

7. Return only the SQL query output and no interpretation.

You MUST call the 'execute_sql' tool to run your SQL query and retrieve results from the database.

If the user's request requires business interpretation of the query result (e.g., trends, comparisons, decisions), you MUST hand off the output to the **Business Analyst Agent** for interpretation.

Current date: {current_date}

Schema:

{schema}

You will receive a business question as input. Your response must be:

-The SQL query executed

-The raw result from the database

System prompt for the SQL agent shown above also starts with a role assignment as a technical data retrieval expert focused exclusively on PostgreSQL query generation and execution. It also states a few rules to follow during the SQL generation and a tool to use for SQL execution. Additionally, the agent is provided with schema context including table structures, column types, and, when possible, sample values or ranges for enumerated or numeric fields to improve the relevance and accuracy of generated queries. This schema-aware approach enables the agent to construct more precise filter conditions and join constraints, reducing the likelihood of invalid or vacuous queries. Combining these strategies: role assignment, operational constraints, tool specification, error handling, and schema enrichment, the SQL agent is equipped to reliably translate natural language requests into executable, contextually accurate SQL queries.

Business Analyst System Prompt

You are a business analyst specializing in e-commerce and digital marketing analytics.

Your task is to receive structured data (e.g., SQL result tables) and interpret it to provide actionable insights to the user.

You MUST:

1. Understand the context of the user's original question.

2. Read and analyze the result from the SQL agent.

3. Identify trends, anomalies, and comparisons in the data.

4. Provide clear explanations using non-technical business language.

5. Quantify findings with numbers, percentages, or trends.

6. If the data appears incomplete or ambiguous, suggest a follow-up query.

Do NOT write or modify SQL queries. Focus only on interpreting the provided dataset.

If you are asked a question that requires new data from the database, you MUST hand off the task to the **SQL Data Agent** to generate and execute the appropriate query.

Structure your response as:

-Summary insight

-Supporting evidence (with metrics)

-Business recommendation (if applicable)

Current date:{current_date}

Assume the data is from a PostgreSQL database containing marketing and sales data from an e-commerce company.

For the business analyst agent, the system prompt shown above is designed to ensure the agent excels at interpreting structured data and delivering actionable business insights, while maintaining a strict boundary

from any technical SQL operations. The prompt begins by establishing the agent's identity as a business analysis expert, framing its core responsibility as transforming raw database outputs into meaningful, context-aware findings for e-commerce decision-making. To achieve this, the prompt always provides the analyst agent with the original user question, the relevant business context, and the current date, ensuring that all insights are grounded in the user's intent and the latest available information. The business analyst agent is equipped with a handoff tool that allows it to seamlessly request additional data from the SQL agent whenever needed. When the analyst identifies gaps or needs further information, it uses this tool to pass the user's question and relevant context directly to the SQL agent. This ensures smooth collaboration, preserves workflow continuity, and allows each agent to focus on its area of expertise. Throughout this process, the agent is guided to focus on the business implications of the data, such as identifying customer segments, or marketing channels, rather than delving into technical details.

Appendix A.2 Supplementary Retry Analysis

To further clarify the practical role of the retry instruction included in the prompting workflow, a supplementary verifier-based retry analysis was conducted on the custom SQL test suite. The purpose of this analysis was not to replace the primary quality metrics reported in the main text, but to quantify how often generated SQL outputs were accepted on the first pass and how often verifier-guided correction was required.

Table A1: Verifier-based retry analysis on the custom SQL test suite.

Model	First-Pass Accepted	Retry Required	Retry Resolved	Final Accepted	Avg. Attempts
Qwen2.5-Coder	95.77%	4.23%	0.00%	95.77%	1.04
Qwen3	74.65%	25.35%	33.33%	83.10%	1.25
Devstral	60.56%	39.44%	32.14%	73.24%	1.39
Mistral	91.55%	8.45%	50.00%	95.77%	1.08

The supplementary retry analysis indicates substantial differences in first-pass stability across models. Qwen2.5-Coder showed the strongest robustness, with 95.77% of cases accepted on the first pass and no measurable gain from retry. Mistral also exhibited high initial acceptance (91.55%), with retry improving a subset of the remaining cases. By contrast, Qwen3 and Devstral required correction more frequently, and verifier-guided retry improved their final acceptance rates from 74.65% to 83.10% and from 60.56% to 73.24%, respectively. These findings suggest that prompt-level self-correction plays only a limited role for the most stable configurations but becomes more relevant for models whose first-pass SQL generation is less reliable.

References

1. Mangal U, Mogha S, Malik S. Data-driven decision making: maximizing insights through business intelligence, artificial intelligence and big data analytics. In: Proceedings of the 2024 International Conference on Advances in Computing Research on Science Engineering and Technology (ACROSET); 2024 Sep 27–28; Indore, India. doi:10.1109/ACROSET62108.2024.10743399.
2. Liu Z, Qiao A, Neiswanger W, Wang H, Tan B, Tao T, et al. LLM360: towards fully transparent open-source LLMs. arXiv:2312.06550. 2023.
3. Bagić Babac M, Jevtić D. AgentTest: a specification language for agent-based system testing. Neurocomputing. 2014;146(9):230–48. doi:10.1016/j.neucom.2014.04.060.

4. Poje K, Brcic M, Kovac M, Babac MB. Effect of private deliberation: deception of large language models in game play. Entropy. 2024;26(6):524. doi:10.3390/e26060524.
5. Chen J, Lin H, Han X, Sun L. Benchmarking large language models in retrieval-augmented generation. Proc AAAI Conf Artif Intell. 2024;38(16):17754–62. doi:10.1609/aaai.v38i16.29728.
6. Guo C, Tian Z, Tang J, Li S, Wang T. Multi-pattern retrieval-augmented framework for Text-to-SQL with Poincaré-Skeleton retrieval and meta-instruction reasoning. Inf Process Manag. 2025;62(3):103978. doi:10.1016/j.ipm.2024.103978.
7. Ojuri S, Han TA, Chiong R, Di Stefano A. Optimizing text-to-SQL conversion techniques through the integration of intelligent agents and large language models. Inf Process Manag. 2025;62(5):104136. doi:10.1016/j.ipm.2025.104136.
8. Shao Z, Cai S, Lin R, Ming Z. Enhancing text-to-SQL with question classification and multi-agent collaboration. In: Chiruzzo L, Ritter A, Wang L, editors. Findings of the association for computational linguistics: NAACL 2025. Albuquerque, NM, USA: Association for Computational Linguistics; 2025. p. 4340–9. doi:10.18653/v1/2025.findings-naacl.245.
9. Shi L, Tang Z, Zhang N, Zhang X, Yang Z. A survey on employing large language models for text-to-SQL tasks. ACM Comput Surv. 2026;58(2):1–37. doi:10.1145/3737873.
10. Pourreza M, Li H, Sun R, Chung Y, Talaei S, Kakkar GT, et al. CHASE-SQL: multi-path reasoning and preference optimized candidate selection in text-to-SQL. In: Proceedings of the Thirteenth International Conference on Learning Representations; 2025 Apr 24–28; Singapore.
11. Jafari O, Maurya P, Nagarkar P, Islam KM, Crushev C. A survey on locality sensitive hashing algorithms and their applications. arXiv:2102.08942. 2021.
12. Li J, Hui B, Qu G, Yang J, Li B, Li B, et al. Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs. In: Proceedings of the Thirty-Seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track; 2023 Dec 10–16; New Orleans, LA, USA.
13. Gao Y, Liu Y, Li X, Shi X, Zhu Y, Wang Y, et al. A preview of XiYan-SQL: a multi-generator ensemble framework for text-to-SQL. arXiv:2411.08599. 2024.
14. Chen WL, Wu CK, Chen YN, Chen HH. Self-ICL: zero-shot in-context learning with self-generated demonstrations. In: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. Singapore: Association for Computational Linguistics; 2023. p. 15651–62. doi:10.18653/v1/2023.emnlp-main.968.
15. Yu T, Zhang R, Yang K, Yasunaga M, Wang D, Li Z, et al. Spider: a large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. Brussels, Belgium: Association for Computational Linguistics; 2018. p. 3911–21. doi:10.18653/v1/D18-1425.
16. Zhou Y, He Y, Tian S, Ni Y, Yin Z, Liu X, et al. R3-NL2GQL: a model coordination and knowledge graph alignment approach for NL2GQL. In: Findings of the association for computational linguistics: EMNLP 2024. Miami, FL, USA: Association for Computational Linguistics; 2024. p. 13679–92. doi:10.18653/v1/2024.findings-emnlp.800.
17. Wang B, Ren C, Yang J, Liang X, Bai J, Chai L, et al. MAC-SQL: a multi-agent collaborative framework for text-to-SQL. In: Rambow O, Wanner L, Apidianaki M, Al-Khalifa H, Eugenio BD, Schockaert S, editors. Proceedings of the 31st international conference on computational linguistics. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics; 2025. p. 540–57.
18. Shen C, Wang J, Rahman S, Kandogan E. Demonstration of a multi-agent framework for text to SQL applications with large language models. In: Proceedings of the 33rd ACM International Conference on Information and Knowledge Management. Boise, ID, USA: ACM; 2024. p. 5280–3. doi:10.1145/3627673.3679216.
19. Mitsopoulou A, Koutrika G. Analysis of text-to-SQL benchmarks: limitations, challenges and opportunities. In: Proceedings of the 28th International Conference on Extending Database Technology (EDBT); 2025 Mar 25–28; Barcelona, Spain.
20. Obeng A, Zhong JC, Gu C. How we built Text-to-SQL at pinterest [Internet]. 2024 Apr 2 [cited 2025 Jun 23]. Available from: https://medium.com/pinterest-engineering/how-we-built-text-to-sql-at-pinterest-30bad30dabff.

21. Building effective AI agents [Internet]. [cited 2025 Jun 24]. Available from: https://www.anthropic.com/engineering/building-effective-agents.
22. Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K, et al. ReAct: synergizing reasoning and acting in language models. arXiv:2210.03629. 2022.
23. Multi-agent [Internet]. [cited 2025 Jun 24]. Available from: https://langchain-ai.github.io/langgraph/agents/multi-agent/.
24. Ollama [Internet]. [cited 2025 Jun 24]. Available from: https://ollama.com.
25. LangGraph [Internet]. [cited 2025 Jun 26]. Available from: https://www.langchain.com/langgraph.
26. Katsogiannis-Meimarakis G, Koutrika G. A survey on deep learning approaches for text-to-SQL. VLDB J. 2023;32(4):905–36. doi:10.1007/s00778-022-00776-8.
27. Pinna G, Perezhohin Y, Manzoni L, Castelli M, De Lorenzo A. Redefining text-to-SQL metrics by incorporating semantic and structural similarity. Sci Rep. 2025;15(1):22357. doi:10.1038/s41598-025-04890-9.
28. Kim H, Taeyang J, Choi S, Choi S, Cho H. FLEX: expert-level false-less execution metric for text-to-SQL benchmark. In: Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers). Albuquerque, NM, USA: Association for Computational Linguistics; 2025. p. 4448–75. doi:10.18653/v1/2025.naacl-long.228.
29. Tomova M, Hofmann M, Hütterer C, Mäder P. Assessing the utility of text-to-SQL approaches for satisfying software developer information needs. Empir Softw Eng. 2023;29(1):15. doi:10.1007/s10664-023-10374-z.
30. Harvey Team. Scaling AI evaluation through expertise [Internet]. [cited 2025 Jun 27]. Available from: https://www.harvey.ai/blog/scaling-ai-evaluation-through-expertise.