

On Improved Performance of Geospatial Queries in Generative AI Applications

STACY LIU¹, LEVON BARSEGHYAN², VIJAY MADISETTI³

¹Georgia Institute of Technology, Atlanta, GA 30332, USA (e-mail: sliu836@gatech.edu)

²Georgia Institute of Technology, Atlanta, GA 30332, USA (e-mail: levonbars@gatech.edu)

³Georgia Institute of Technology, Atlanta, GA 30332, USA (e-mail: vkm@gatech.edu)

Corresponding author: Dr. Vijay Madiseti (e-mail: vkm@gatech.edu).

ABSTRACT The generation of well-formatted, compilable, and efficient SQL queries remains a substantial challenge for Generative AI applications utilizing Large Language Models (LLMs). These LLMs frequently lack the requisite knowledge of underlying data schemas and the understanding of SQL syntax necessary for query construction. This challenge is exacerbated in the context of geospatial data, which requires domain-specific expertise and specialized query functionalities. Recent progress in text-to-SQL tasks has shown notable performance enhancements through the integration of agentic frameworks, which incorporate elements such as memory, tool usage, and environment interaction. Inspired by these advancements, this paper proposes an extension of AI agents' capabilities to the geospatial domain by introducing a multi-agent framework tailored for geospatial text-to-SQL generation. This framework harnesses specialized tools and memory structures to augment the querying and analytical capabilities essential for effective geospatial data processing. The framework was subjected to a series of evaluations comparing the accuracy of the generated SQL data against other existing frameworks and facilitating the creation of geographical visualizations. Our framework demonstrated superior performance relative to the current state-of-the-art SQL generating agents in these assessments.

INDEX TERMS GeoAI, LLM, Multi-agent supervisor

I. INTRODUCTION

THE The substantial expansion of geospatial data across various sectors, including urban planning and environmental monitoring, has markedly increased the necessity for easily accessible tools capable of querying and analyzing such data. Conventional methodologies necessitate proficiency in structured query language (SQL) and geospatial functions, thereby presenting obstacles for individuals lacking expertise. Although recent developments in large language models (LLMs) and text-to-SQL systems demonstrate potential, current solutions frequently do not adequately address the distinctive challenges associated with geospatial data, such as spatial semantics, nested functions, and dynamic context awareness.

A significant obstacle in the generation of geospatial SQL queries lies in comprehending and possessing knowledge of geographical SQL types [Project, 2025]. These types, which encapsulate concepts intelligible to humans, such as linestrings, polygons, or points, are predominantly unfamiliar to most large language models (LLMs), which generally exhibit a lack of geographical context awareness. We propose

to address this issue by implementing a Retrieval-Augmented Generation approach that assimilates the relevant APIs for these types prior to producing any output. Presently, LLM-based agents engaged in geospatial tasks demonstrate critical deficiencies, including the incomplete incorporation of spatial SQL functions, the absence of multi-agent collaboration, and inadequate validation mechanisms. For example, frameworks such as GeoGPT [Zhang et al., 2023] primarily emphasize interaction with GIS tools, while other LLM-driven methodologies [Koziol, 2023] often generate erroneous spatial relationships. These deficiencies impede effective and comprehensive geospatial analysis for users lacking technical expertise.

In this study, we introduce MaGS (Multi-Agent GeoSQL), an innovative framework designed to address these challenges through a modular, agent-driven architecture. The MaGS framework employs a multi-agent supervisor model with specialized worker agents (Selector, Decomposer, Refiner, and Visualizer) to facilitate context-aware SQL generation, automated validation, and geospatial visualization. This approach has led to notable advancements in Geospatial SQL

querying compared to a standard, non-geospatial SQL agent. Furthermore, we have developed training and evaluation datasets to support effective agent prompting and testing.

Ultimately, the capabilities of our multi-agent framework were evidenced by its successful generation of a diverse array of intricate visualizations directly from user inputs, illustrating its proficiency in querying the underlying SQL database and its effectiveness in producing complex visual outputs.

II. RELATED WORK

Previous research in the field has concentrated on enhancing the utility of large language models (LLMs) within the Geospatial domain. These studies primarily aim to either leverage LLMs to elicit intrinsic Geospatial knowledge or employ them in the development of Geospatial tools, such as geographic information systems (GIS). However, these initiatives have yet to devise a solution capable of extracting Geospatial knowledge from underlying databases, as they predominantly rely on the models' inherent knowledge.

Conversely, significant efforts have been invested in the development of proficient text-to-SQL agents adept at retrieving data from databases. However, these agents frequently encounter difficulties in comprehending the intricate geographical forms and categorizations essential for accurate data extraction, leading to instances where they erroneously infer information not originally present. Our framework addresses these challenges through the integration of RAG tooling.

In the following sections, we will comprehensively delineate the corpus of related work, articulating their respective limitations, and elucidating the divergences between these works and our proposed methodology. Section II offers a detailed exposition of related work and identifies prevailing research gaps, whereas Sections III and IV expound upon our proposed approach and its subsequent implementation. In Section V, we present a detailed account of the results derived from our implementation, while Section VI encompasses a series of ablation studies. Sections VII and VIII are dedicated to outlining prospective extensions and presenting our conclusions.

A. INHERENT GEOSPATIAL KNOWLEDGE IN LLMs

Recent advancements in large language models (LLMs) have demonstrated significant performance across a variety of tasks. Beyond their core applications in natural language processing, researchers have started to investigate whether LLMs possess implicit geospatial knowledge that can be extracted. GeoLLM [Manvi et al., 2024] has provided evidence that LLMs indeed encapsulate substantial geospatial understanding, particularly when their prompts are augmented with structured metadata, such as data derived from OpenStreetMap. This observation is consistent with broader initiatives aimed at eliciting domain-specific knowledge from LLMs through prompt engineering [Jiang and Yang, 2024]. However, existing methodologies primarily focus on retriev-

ing static knowledge rather than facilitating dynamic interaction with live datasets. Although fine-tuning techniques [Deng et al., 2023] show potential for domain adaptation, they remain confined to narrow NLP tasks and do not integrate with common SQL workflows.

B. LLM-DRIVEN AGENTS AND GEOSPATIAL TOOLING

To extend LLM functionality beyond text-based tasks, several studies have explored the integration of LLMs into agent-based frameworks for geospatial analysis. GeoAgent [Zhang et al., 2023] exemplifies this approach by combining retrieval-augmented generation (RAG) with Monte Carlo Tree Search (MCTS) to generate Python code for geospatial tasks. GeoAgent retrieves relevant APIs from structured knowledge bases and iteratively refines potential solutions through execution-based feedback loops. While GeoAgent improves automation in geospatial analysis, its integration is limited to Google Earth and Python-based workflows. Critically, GeoAgent does not facilitate GeoSQL generation, which is essential for enterprises that manage data within SQL databases.

Analogously, other geospatial agents, such as GeoGPT [Zhang et al., 2023] and LLM-Find [Ning et al., 2024], emphasize the integration with Geographic Information Systems (GIS) tools, primarily aimed at data retrieval and analysis tasks. Although these frameworks enhance automated geospatial data processing, they are deficient in comprehensive database interaction capabilities. This limitation constrains their applicability within enterprise geospatial workflows where the generation and optimization of GeoSQL queries are vital for efficient data retrieval and analysis. GeoForge [Ageospatial, 2024] represents an alternative approach to this issue. By employing a large language model (LLM) trained on vast amounts of geospatial data, this framework can discern complex geospatial relationships and display data. Nevertheless, it remains insufficient in addressing the querying of data from underlying databases containing geospatial information.

C. SPATIAL QUERY AND DATA HANDLING

Recent scholarly efforts have investigated geospatial text-to-SQL frameworks with the objective of translating natural language inquiries into structured spatial database queries. These methodologies primarily concentrate on schema linking, query parsing, and the prediction of spatial functions, thereby augmenting the capacity of large language models (LLMs) to engage with geospatial databases. For example, GS-SQL [Zhang et al., 2024] presents GeoSpatialSpider, a dataset crafted for spatial text-to-SQL applications that encompass nested SQL functions and spatial operations such as buffering, intersections, and coordinate transformations. It utilizes graph-based schema linking to enhance the prediction of spatial functions, thereby improving the precision of query generation. Notwithstanding, it functions as a single-step transformation model and lacks an agentic architecture for the iterative refinement of queries or the retrieval of

TABLE 1. Comparison of different papers and their capabilities versus our proposal

Paper	An LLM Agent for Automatic Geospatial Data Analysis	Is ChatGPT a Good Geospatial Data Analyst?	GeoGPT	Evaluation of Code LLMs on Geospatial Code Generation	LLM-Find	GS-SQL	GeoGPT: An LLM-Based Geographic Information System	GIS Copilot	Our Solution
Generate SQL from natural language prompt	No	Yes	No	No	No	Yes	Yes	No	Yes
Uses agentic framework	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes
Agent utilizes spatial SQL functions when appropriate	No	No	No	No	No	Yes	Yes	No	Yes
Generated SQL is context-aware	No	Yes	No	No	No	No	No	No	Yes
Query validation is automated	Yes	No	Yes	No	No	No	No	No	Yes
Execute generated SQL against a geo database	No	Yes	No	No	No	Yes	Yes	No	Yes
Response is parsed back to natural language	Yes	Yes	Yes	No	No	Yes	Yes	No	Yes
Generate map visualization from SQL response	No	No	Yes	No	No	No	No	No	Yes
Agent uses tools to analyze the prompt	Yes	No	Yes	No	No	Yes	No	No	Yes
Agent is able to take in multiple prompts and iterate the response	No	No	No	No	No	No	No	No	Yes

external information. This research endeavors to mitigate this limitation by introducing a supervisor multi-agent framework that employs retrieval-augmented generation (RAG) and specialized geospatial tools to improve spatial query generation and execution.

D. NON-GEOSPATIAL TEXT-TO-SQL AGENTS

With the proliferation of AI Agents, Text-to-SQL query generation has emerged as a significant area of research within the industry, facilitating the development of numerous scholarly papers. MAC-SQL [Wang et al., 2024], subsequently enhanced by [Xie et al., 2024], exemplifies Text-to-SQL agents, particularly emphasizing the development of a Multi-Agent framework to decompose the task of SQL generation. This methodology employs a selector agent to determine the appropriate tables, a decomposer to formulate SQL queries, and a refiner to validate them. Although this sequential model demonstrated potential, it lacked the requisite geospatial tools necessary for offering comprehensive insights into spatial data. Nevertheless, this concept served as the foundational basis for the multi-agent geospatial frame-

work elucidated in this paper.

E. GEOSPATIAL VISUALIZATION

The development of comprehensible visualization of geospatial data has historically constituted a pivotal undertaking across various industries. Geographic Information System (GIS) [Survey,] solutions have been established as the foremost methodology for generating highly detailed and intricate geospatial visualizations. Recent advancements, exemplified by tools like GIS Copilot [Akinboyewa et al., 2024], have further optimized visualization workflows by enhancing the ability of specialists to employ existing GIS functionalities effectively. Nonetheless, prevailing solutions generally necessitate specialized expertise or supplementary external resources, thereby posing challenges for users who lack familiarity with GIS technologies.

Conversely, there exists a paucity of research dedicated to the generation of geospatial visualizations directly from natural language queries without dependence on specialized tools. This study endeavors to address this deficiency by integrating visualization functionalities into our Multi-Agent

framework, thereby facilitating the production of geospatial visualizations and rendering them accessible to non-specialists without necessitating pre-existing GIS knowledge or auxiliary software.

III. PROPOSED APPROACH

To address the limitations of existing geospatial text-to-SQL systems, we propose a Supervisor-based Multi-Agent Framework that integrates retrieval-augmented generation (RAG) and geospatial-specific tooling to dynamically generate, validate, and execute GeoSQL queries. Our approach leverages a hierarchical architecture where a Supervisor Agent orchestrates multiple specialized sub-agents responsible for distinct stages, including table selection, query generation, query refinement, and visualization (figure 1).

A. SUPERVISOR AGENT

The proposed framework is comprised of five agents, wherein the Supervisor agent serves as the exclusive interface for human interaction. The Supervisor agent is tasked with receiving user queries, determining the status of the inquiry, and selecting the appropriate sub-agent to which the query should be delegated. This agent is integral to the multi-agent structure, as it ensures that tasks are accurately decomposed and managed efficiently by specialized agents. Subsequently, it communicates the final output to the user, thereby allowing each specialized agent to concentrate strictly on their specific functions without the necessity of managing user-facing interactions.

B. SELECTOR AGENT

Subsequent to the preliminary query evaluation conducted by the Supervisor Agent, the Selector Agent engages in analyzing the user's query in conjunction with the database schema to determine the pertinent tables and columns. This agent specifically selects a minimum of three relevant tables and prioritizes up to six of their most pertinent columns, effectively eliminating unrelated schemas to optimize query efficiency. Notably, the Selector Agent does not generate SQL queries; instead, it examines the schema to extract the relevant tables required for the user query. This approach serves several purposes. Primarily, by narrowing down the schema subset, the SQL generating agent can concentrate more effectively on query generation, thereby minimizing the generation of irrelevant queries. Additionally, reducing the schema results in smaller API calls, thereby enhancing the framework's speed and cost-efficiency.

The Selector Agent employs a Retrieval-Augmented Generation (RAG) module that integrates PostGIS SQL documentation alongside a compendium of GeoSQL exemplars. This approach facilitates a more profound comprehension of geospatial data architectures, enabling the construction of queries with enhanced logical coherence. Furthermore, it employs the LangChain SQL Toolkit to obtain pertinent schema information.

C. DECOMPOSER AGENT

Upon the identification of pertinent tables and schemas, the Supervisor delegates the task to the Decomposer Agent. This agent is tasked with the decomposition of the overarching task into smaller, structured subqueries and the generation of valid PostGIS SQL queries. It employs RAG to consult PostGIS documentation alongside previous GeoSQL instances, thereby ensuring the generated SQL conforms to geospatial best practices. The agent refines queries through the resolution of errors and fallback methodologies, such as modifying queries when no results are obtained or preventing the incorporation of erroneous data by explicitly recognizing the absence of information. The Decomposer Agent guarantees that the SQL output is valid, executable, and optimized for spatial workloads, all the while strictly avoiding any Data Manipulation Language (DML) operations.

D. REFINER AGENT

The Refiner Agent concludes the query processing workflow by affirming and refining SQL queries produced by the Decomposer Agent. This agent is charged with confirming the validity of the Decomposer Agent's output, rectifying any inaccuracies in the SQL, executing the query, and presenting the data for either the user or a visualization tool. The Refiner Agent ensures both syntax and logic are correct, checks for compliance with PostGIS standards, ensures the absence of data manipulation language operations, and provides a detailed error message if the query cannot be rectified.

Through the examination and optimization of queries prior to their execution, the Refiner Agent guarantees both the preservation of the database's integrity and the dependability of the resultant query outcomes.

E. VISUALIZER AGENT

The final component in the Multi-agent solution is the visualizer, which is distinct from the remaining agents due to its capability to function independently. It can obtain data directly as a component of the user's inquiry. Accordingly, the system is designed to either accept data for visualization, bypassing the SQL processing, or directly query the database to retrieve and display the necessary data.

The visualizer agent is only able to create vector layers consisting of:

- Markers
- Circles
- Polygons
- Polyline

and does not have the capability of creating and complex GeoJSON or Raster Layers. The visualizations are created using the Python Folium [python visualization,] library. The visualizer creates an HTML file with the required annotations, which can be used to zoom, pan, or move around the map. The visualization results will be shown and discussed further in the next sections.

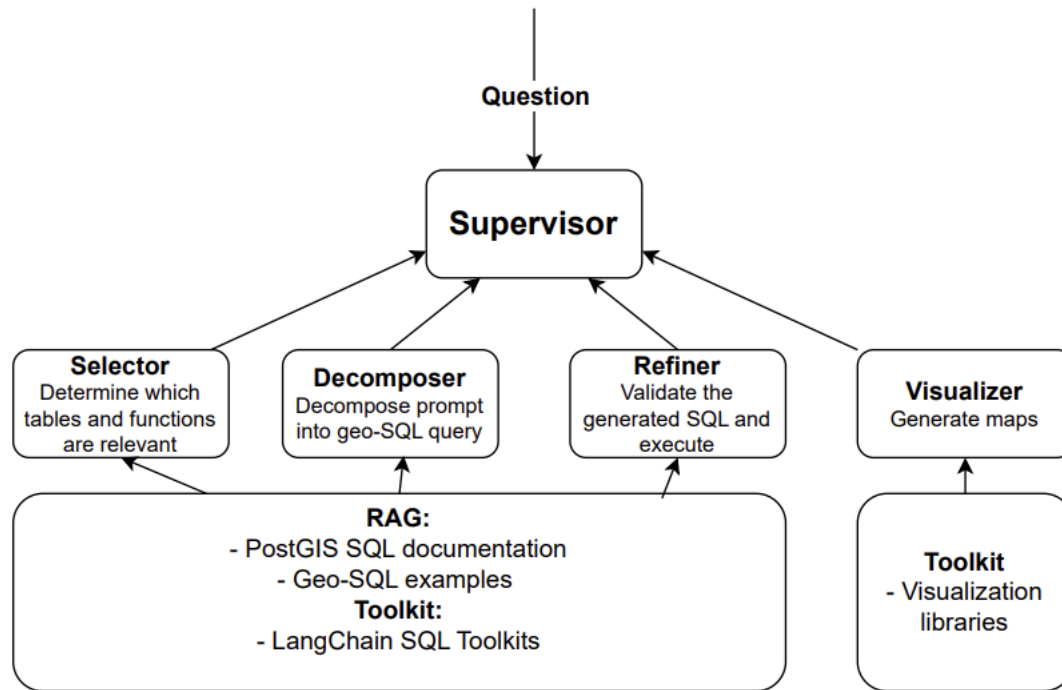


FIGURE 1. MAGS: Multi-Agent Geospatial SQL Framework

IV. IMPLEMENTATION METHODOLOGY

The MaGS framework is constructed to convert natural language geospatial inquiries into valid, executable GeoSQL through a modular multi-agent architecture. This section delineates the datasets employed for training and evaluation, the retrieval-augmented generation (RAG) infrastructure, the design and categorization of geospatial prompts, as well as the baseline systems utilized for benchmarking. We elucidate how each component of MaGS harnesses external geospatial knowledge and structured prompts to enhance the generation and execution of spatial queries across a spectrum of complexity levels.

A. DATA

Our framework employs two separate datasets for the purposes of training and evaluation. In contrast to conventional fine-tuning processes for large language models (LLMs), MaGS omits gradient-based training methodologies. Instead, the framework focuses on the refinement of prompts and the curation of retrieval strategies through the use of a structured geospatial dataset.

Training Data

The publicly accessible geospatial dataset from [Jiang and Yang, 2024], which encompasses the urban infrastructure layers of New York City and is maintained within a PostGIS SQL database, has been employed. This dataset was selected due to its extensive spatial diversity and its previous application in educational tools employing SQL, thereby rendering it appropriate for the development of context-aware

agent prompts and retrieval-augmented generation (RAG) resources. This dataset includes:

- **Census blocks:** Polygon features with sociodemographic attributes (population counts, racial demographics).
- **Neighborhoods:** Socially constructed polygons.
- **Street Networks:** Line geometries with transportation hierarchy classifications
- **Subway stations:** Point features representing the public transportation network.
- **Census Tract Statistics:** Tabular socioeconomic data (median income, family counts)

All spatial layers encompass a `geom` column in accordance with PostGIS conventions. This dataset served as a guide for the development of agent prompt templates. Furthermore, fundamental SQL query patterns and spatial relationships derived from this dataset were integrated into the RAG knowledge base, facilitating more informed decisions regarding table selection, spatial function application, and query decomposition strategies.

Test Data

For benchmark evaluation, we used a geospatial dataset consisting of:

- **Austin Airbnb listings:** Point features representing Airbnbs in Austin including geolocation, pricing, property type, and availability.
- **Census tract data for Austin:** Polygon features providing demographic and socioeconomic attributes.

The selected datasets serve the purpose of evaluating MaGS on queries incorporating both geometric operations, such as proximity, containment, and spatial joins, as well as attribute-based filtering. The integration of polygonal census tract data with point-based Airbnb listings provides a comprehensive testbed for the validation of GeoSQL generation, assessment of decomposition accuracy, and examination of visualization capabilities.

B. RETRIEVAL-AUGMENTED GENERATION TOOLS

In order to facilitate precise and contextually aware GeoSQL generation, a Retrieval-Augmented Generation (RAG) component has been incorporated into the MaGS framework. The RAG system is underpinned by a ChromaDB vector store, which is populated utilizing documentation from authoritative PostGIS resources [PostGIS Project, 2024]. These resources include query examples, tutorials on basic SQL applications with spatial data, and references for functions specific to geography. The sources were processed into vector embeddings to allow for semantically relevant retrieval during query generation.

The retrieved content facilitates sub-agent access to empirical examples, function signatures, and usage patterns, thereby enhancing the selection of spatial functions and the composition of SQL queries. By anchoring agent responses in well-documented PostGIS methodologies, the RAG system mitigates hallucinations and augments consistency with legitimate spatial query structures.

C. GEOSPATIAL PROMPTS

To evaluate the effectiveness of the MaGS framework across a range of spatial reasoning tasks, we curated and categorized natural language prompts as follows.

Question Taxonomy

Each prompt was categorized according to its complexity and the type of spatial operation it involves. The taxonomy is defined as follows:

TABLE 2. Prompt Complexity Levels and Examples

Complexity Level	Description	Example
Simple	Single table spatial operations	<i>What is the area of Manhattan?</i>
Complex	Multi table spatial joins	<i>What is the population within 1 km of subway stations?</i>

Dataset Composition

- Training/Validation Set: 20 prompts (13 simple, 7 complex)
- Test Set: 12 prompts (7 simple, 5 complex)

The allocation between simple and complex queries mirrors the actual occurrence of uncomplicated spatial tasks within public and corporate geospatial applications. Simple queries

typically involve elementary measurements or filtering by geographic location, whereas complex queries necessitate spatial joins, nested logical operations, or generated geometries. The selection of complex prompts was performed with intention to ensure comprehensive representation of tasks essential for assessing MaGS.

Each prompt was paired with an expected GeoSQL query and result, allowing for grounded validation. Responses from MaGS were evaluated manually for correctness in both SQL structure and returned answer. Prompts ranged from simple spatial filters to complex multi-step queries involving spatial joins and nested logic, ensuring that the framework is tested across a spectrum of real-world geospatial tasks.

D. BENCHMARK

To evaluate the performance of the MaGS framework, we benchmarked it against two baseline systems:

LangChain SQL Agent: A widely used text-to-SQL agent built on the LangChain framework, leveraging SQL toolkits for schema inspection, query generation, and execution.

LangChain SQL Agent with RAG: An enhanced version of the above agent, augmented with the same retrieval-augmented generation (RAG) module used in MaGS. This configuration includes access to external resources such as PostGIS documentation and a curated repository of GeoSQL examples.

These baselines were selected as they represent the most commonly adopted open-source approaches for natural language SQL query generation. Moreover, LangChain’s agent architecture is frequently referenced in prior work, making it a reliable comparative foundation. Currently, there are no publicly available geospatial multi-agent systems, positioning these LangChain variants as the most relevant benchmarks for comparative analysis.

Importantly, we include the LangChain SQL Agent with RAG to isolate the effect of retrieval augmentation. This allows us to determine whether RAG alone is sufficient to handle the complexities of geospatial SQL generation. Our results show that while RAG improves baseline performance, it is not sufficient on its own—highlighting the need for a coordinated, multi-agent framework like MaGS that incorporates specialized agents and iterative processing.

All systems were subjected to evaluation using the identical set of geospatial prompts and queries delineated in the preceding section. The evaluation criteria encompassed SQL correctness, execution success, and answer accuracy. Each system’s output underwent manual review to ensure that the generated SQL was syntactically valid, produced correct results, and appropriately utilized PostGIS spatial functions. Employing a consistent dataset across all systems was a deliberate strategy to facilitate a controlled, direct comparison. Instead of evaluating performance across diverse datasets, the objective was to isolate the effects of architectural differences, specifically the introduction of a multi-agent orchestration layer. By maintaining the underlying dataset constant,

TABLE 3. Training Results: Combined Accuracy by Method and Query Complexity

Complexity Level	Base SQL	Base SQL + RAG	Supervisor Agent
Simple	69.23%	76.92%	84.62%
Complex	28.57%	42.86%	57.14%
Overall	55.00%	65.00%	75.00%

TABLE 4. Evaluation Results: Combined Accuracy by Method and Query Complexity

Complexity Level	Base SQL	Base SQL + RAG	Supervisor Agent
Simple	85.71%	100.00%	100.00%
Complex	40.00%	40.00%	60.00%
Overall	66.67%	75.00%	83.33%

performance differences observed are directly attributable to the capabilities of each agent.

V. EXPERIMENTAL RESULTS

We now present detailed results and their analyses from our study.

A. GEO-SQL GENERATION RESULTS

Table 3 and table 4 summarize the overall accuracy for each evaluated approach across both training and evaluation prompts. The tested systems include: Base SQL, Base SQL augmented with Retrieval-Augmented Generation (Base SQL + RAG), and the proposed Supervisor Agent. The Supervisor Agent achieved the highest overall accuracy—75% on training prompts and 83% on evaluation prompts—demonstrating a significant improvement over the baselines. These results reflect its enhanced capability to interpret and execute complex geospatial queries effectively.

To gain deeper insights into model behavior, we further analyzed performance by categorizing queries based on complexity. As shown in Table 3 and 4, all approaches performed reasonably well on simpler queries. However, as query complexity increased, involving multiple tables and spatial relationships, the Supervisor Agent significantly outperformed the Base SQL and Base SQL + RAG approaches. This trend highlights the importance of structured agent coordination in managing complex query structures and relational joins.

B. CASE STUDIES

To better understand the strengths and limitations of the MaGS framework, we analyzed selected success and failure cases in Tables 5-7.

1) Success Case studies

Table 5 and Table 6 present two examples where the Supervisor Agent outperformed both baseline systems. Table 5 illustrates a success case in which the Supervisor Agent successfully identified 15 distinct neighborhoods served by the 6 train. In this example, the Selector Agent correctly

identified the relevant spatial tables, the Decomposer Agent generated a join using ST_Intersects, and the Refiner verified the query's correctness. In contrast, both baseline agents either failed to form a suitable query or misinterpreted spatial semantics.

Table 6 highlights another successful example where the Supervisor Agent correctly computed population totals by race for Manhattan. The MaGS framework demonstrated effective table selection and numeric aggregation across census blocks—tasks that caused partial or incorrect responses from baseline methods.

These success cases highlight the core advantages of MaGS: improved identification and combination of relevant spatial tables, enabled by the specialized roles of individual agents in the MaGS framework.

2) Failure Case studies

Despite these improvements, failure analysis revealed important weaknesses, primarily in early-stage agent decisions and spatial logic validation. Table 7 presents a case where the Supervisor Agent was asked to count subway stations within 1,000 meters of Grand Central Station. All methods, including MaGS, failed. In MaGS, the Selector Agent chose the incorrect locations table instead of nyc_subway_stations, and the Refiner applied an invalid geometry cast. Although the SQL was structurally plausible, the use of the wrong spatial source and improper casting led to incorrect output.

Other failure cases echoed this theme. For example, incorrect use of attribute joins instead of spatial predicates (ST_Intersects), improper geometry transformations, or confusion between similar-sounding table names led to either partial answers or empty results. These issues were especially common in queries involving nested logic, proximity calculations, or ambiguous schema references.

Taken together, the failure cases point to key areas for improvement—especially in schema disambiguation, spatial casting validation, and the integration of fallback strategies when data gaps or agent disagreements are detected. These examples demonstrate that while MaGS significantly improves geospatial query accuracy, it remains sensitive to early-stage agent decisions—especially in table selection. Even a well-composed query from the Decomposer and correct syntax from the Refiner cannot compensate for schema-level misunderstandings. This underscores the need for continued refinement of upstream components, particularly in cases involving semantically similar but structurally distinct tables.

TABLE 5. Success case: Distinct neighborhoods served by the 6 train

Query: How many distinct neighborhoods are served by the 6 train?
Challenge: Base SQL and Base SQL + RAG methods could not generate a suitable query or incorrectly interpreted spatial data.
MaGS Process:
Selector: Identified <i>nyc_neighborhoods</i> and <i>nyc_subway_stations</i> tables in database
Decomposer: Generated spatial join using <i>ST_Intersects</i> to associate neighborhoods and subway stations
Refiner: Verified query correctness and spatial relations without modifications
Outcome: The Supervisor Agent accurately resolved the query, correctly identifying 15 distinct neighborhoods, illustrating its capability in handling intricate spatial joins.

TABLE 6. Success case: Population of each race in Manhattan

Query: What is the population of each race in Manhattan?
Challenge: Requires accurate parcel aggregation and summation of demographic data from census block-level details across Manhattan. Base SQL methods showed inconsistencies in numeric aggregation or returned partial results.
MaGS Process:
Selector: Identified <i>nyc_census_blocks</i> and demographic attributes from associated sociological data.
Decomposer: Generated aggregation query summing populations by race across relevant census blocks.
Refiner: Ensured correct numeric aggregation and completeness of the results without further modification.
Outcome: Supervisor Agent successfully computed accurate population counts for each racial category, demonstrating precision in demographic data aggregation and summarization queries.

TABLE 7. Failure case: Stations within 1,000 meters of Grand Central station

Query: How many stations are within 1,000 meters of the Grand Central station?
Challenge: All methods, including Base SQL, Base SQL + RAG, and the Supervisor Agent, failed. The Supervisor Agent specifically failed due to selecting the incorrect table (<i>locations</i> instead of <i>nyc_subway_stations</i>) and improperly casting geometry types. Although the generated SQL appeared structurally correct, these errors led to incorrect results.
MaGS Process:
Selector: Incorrectly selected the table <i>locations</i> rather than the correct table <i>nyc_subway_stations</i> .
Decomposer: Appropriately generated the spatial query using <i>ST_DWithin</i> to establish a 1,000-meter buffer around Grand Central station.
Refiner: Incorrectly applied <i>geom::4326</i> cast.
Outcome: The Supervisor Agent and both benchmarks (Base SQL and Base SQL + RAG) failed to correctly resolve the query. The Supervisor's failure was specifically due to erroneous table selection and inappropriate spatial geometry casting, highlighting areas for improvement in both table identification and spatial handling logic.

C. VISUALIZATION RESULTS

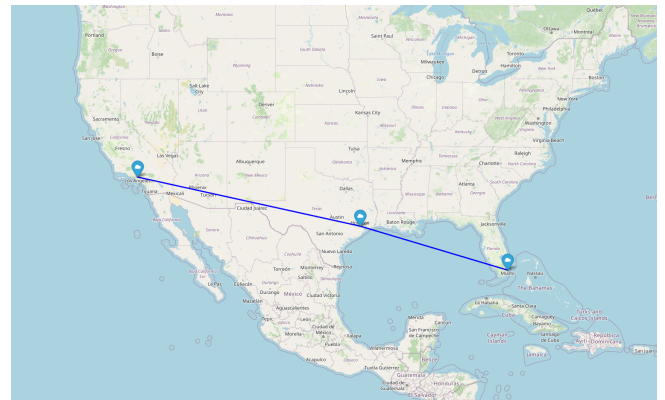
To aid with the understanding of the performance of the agentic system, we selected a few visualization examples that provide insight on how the agents work and what kind of data it is able to visualize.

Successful visualizations

Presented herein are several of our more efficacious visualizations. The agentic framework was assigned a variety

of visualization tasks, necessitating the utilization of data sourced from the database. These tasks range from relatively straightforward to those demanding significant processing and comprehension of the underlying data. This demonstrates considerable potential, as the framework is capable of replicating the tasks that would otherwise require a substantial time investment from a seasoned Geospatial analyst. We anticipate that a refined version of this agent may substantially reduce the time required for the creation of Geospatial visualizations.

The initial visualization, as depicted in Figure 2, illustrates a line interconnecting Los Angeles, Houston, and Miami. Upon receiving the specified directive, the agents initially extracted the geographical coordinates of each city from the database. Subsequently, these coordinates were represented as markers on the map, which were then joined by a solitary polyline.

**FIGURE 2.** Visual for: Draw a line connecting Los Angeles, Houston and Miami. Place a marker on each point in the polyline.

Subsequently, a more complex query is introduced. Figure 3 depicts all urban areas within a 500km radius from New York City. Considering the limited access of the agent to only a select number of cities, the visualization is constrained to New York City and Boston. Although the visual output appears simplistic, the computational process involved is sophisticated. The agent was tasked with developing a query that calculates the distance from New York City to any specified location, which is then compared against the 500km threshold. Predictably, this computation resulted in the visualization displaying only New York City and Boston.

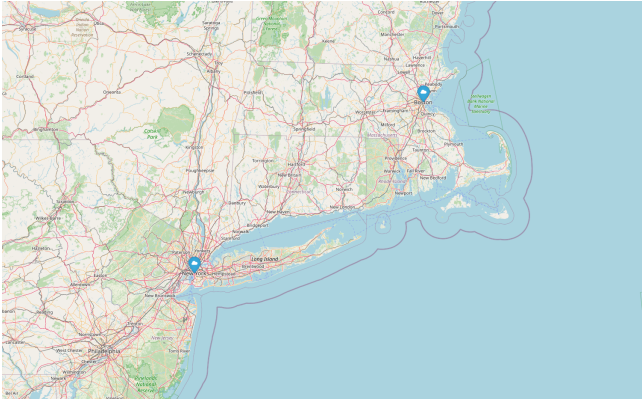


FIGURE 3. Visual for: Mark all cities that are within 500 km of New York City. (Note, Boston is the only city in that area from NYC available in our DB).

The final example of a successful outcome is illustrated in figure 4. In this task, the agent was assigned the task of identifying the easternmost and westernmost subway stations in New York City and delineating a line between them. This task poses significant complexity even for a seasoned Geospatial data analyst. Nevertheless, the agent successfully formulated an accurate query, which identified two distinct stations located in Staten Island and Long Island, subsequently connecting them with a line.

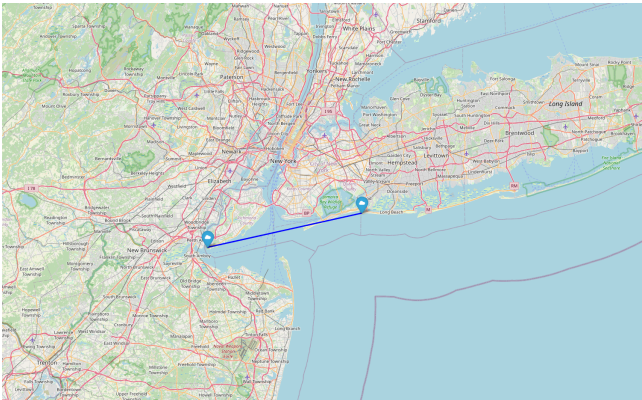


FIGURE 4. Visual for: Draw a line from the most westerly subway station to easterly subway station in NYC. Place a marker on each point in the polyline.

The aforementioned high-quality and insightful visualizations demonstrate considerable potential. Through further iterations and enhancements of the agents, it is anticipated that even more sophisticated outcomes may be achieved, including the capability to construct comprehensive GeoJSON objects capable of representing highly complex and informative data on maps.

Failed visualizations

Additionally, we present several visualizations that did not meet our anticipated standards. The first is illustrated in Figure 5, which was intended to depict all subway stations within Manhattan. The visualization agent successfully represents a substantial number of stations; however, it fails to include approximately two-thirds of them, depicting only 41

out of the 140 stations. We hypothesize that this discrepancy arises from miscommunication between the refiner and the visualization agent.

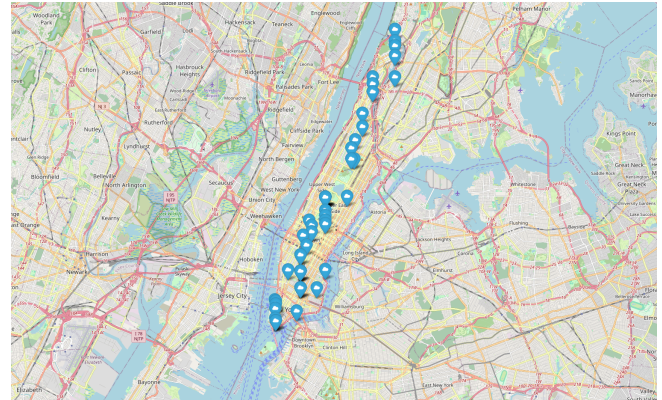


FIGURE 5. Visual for: Mark all subway stations in Manhattan.

An additional illustration of an ineffective visualization is provided in Figure 6. In this instance, agents were tasked with identifying and depicting the three neighborhoods with the highest average family income. This task presents considerable complexity as the selected database lacks geometric data corresponding to the neighborhoods for which mean income information is accessible. Consequently, the agent attempts an approximation by highlighting the boroughs in which these neighborhoods are located.

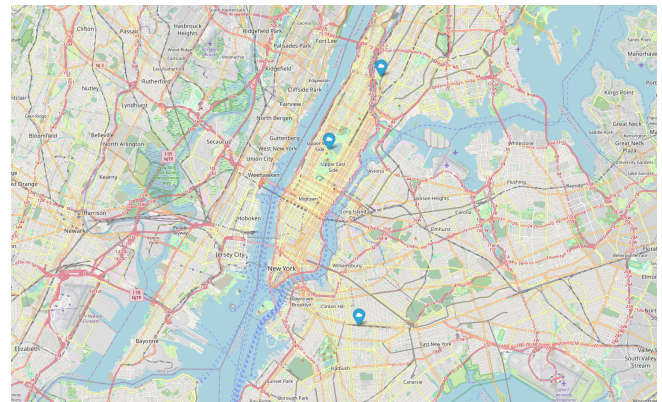


FIGURE 6. Visual for: Mark the three neighborhoods with highest mean family income.

VI. ABLATION STUDIES

To further understand the importance of individual agents in the MaGS architecture, we conducted an ablation study with two modified agents:

- **Selector Removed:** The Selector component was removed, and its responsibilities were integrated directly into the Decomposer component through prompt modification.
- **Decomposer Removed:** Similarly, the Decomposer component was removed, and its functionality was incorporated into the Refiner component.

TABLE 8. Ablation Study Accuracy by Complexity Level

Complexity Level	Supervisor Agent	Selector Removed	Decomposer Removed
Simple	84.62%	76.92%	61.54%
Complex	57.14%	71.43%	57.14%
Overall	75.00%	75.00%	60.00%

The results, broken down by query complexity, are presented in (Table 8). Removing the Selector led to decreased accuracy on simpler queries (from 84.62% to 76.92%) but unexpectedly increased accuracy on more complex queries (from 57.14% to 71.43%). This suggests that integrating table selection with decomposition may simplify query planning in complex scenarios, improving performance, but hinders effectiveness on straightforward tasks where precise table isolation is more important.

Conversely, removing the Decomposer significantly decreased accuracy for simple queries (from 84.62% to 61.54%), while accuracy on complex queries remained unchanged. This emphasizes that the Decomposer plays a critical role, especially in simpler spatial queries, where clear structuring significantly impacts the accuracy.

These findings indicate that while the Selector's role can be partially merged without substantial loss in performance, the Decomposer is essential for maintaining accuracy in the Supervisor Agent.

VII. FUTURE WORK

While MaGS demonstrated strong performance in geospatial SQL generation and visualization, several avenues remain for future research and enhancements:

- **Automatic Schema Linking:** Errors in table or column selection were a significant source of failure in complex queries. Enhancing the Selector Agent with learned schema linking techniques could help reduce such failures, especially when multiple tables contain similarly named attributes.
- **Feedback Loops Between Agents:** While the current MaGS architecture follows a largely sequential pipeline, enabling feedback loops—particularly from the Refiner back to the Decomposer or Selector—could allow for automatic correction and iterative refinement of partially incorrect queries without the additional compute of the Supervisor Agent.
- **Complex Geospatial Visualizations:** The current visualization agent supports only simple vector based visuals such as markers, circles, polygons and polylines. As a next step in the visualization improvement, a Geo-JSON based visualizer can be implemented, that is able to display much more complex and structured data. This can also be augmented with customized components like a heatmap generator for example.

VIII. CONCLUSION

This paper presents MaGS - Multi-Agent Geospatial SQL, an innovative framework of multi-agent large language models

specifically engineered to extract geospatial data from SQL databases through the formulation of efficient and accurate SQL queries. The study fills existing gaps associated with non-geospatial text-to-SQL agents by employing appropriate geospatial tools to accurately interpret and display data. Consequently, this framework significantly assists human geospatial specialists in data retrieval and mapping visualizations, eliminating the necessity for human intervention.

In pursuit of this objective, we employ GPT 4o-mini augmented by the LangGraph library to establish a multi-agent methodology consisting of four worker agents and one supervisor agent. This collaborative system facilitates the development of a highly effective solution capable of discerning the intricacies of geospatial data and applying this comprehension to proficiently extract information from geospatial databases. Our approach demonstrates superior performance compared to the baseline, exceeding it in both simple cases and complex ones, with an overall enhancement rate over the baseline text-to-SQL system. The observed improvements are plausibly attributable to an enhanced understanding of the underlying data achieved through the application of specialized geospatial RAG tooling, in conjunction with the advancements afforded by the multi-agent system.

A significant advancement introduced by our agent is its capability to generate street maps incorporating geospatial data extracted from underlying databases. This innovation represents a critical step forward in enhancing tools and reducing the barriers to specialization within the geospatial domain. Our framework facilitates the representation of fundamental geographical entities, such as markers, polylines, polygons, and circles, on the map by utilizing data acquired from the database. MaGS exemplifies a compelling proof of concept, demonstrating substantial potential. It underscores the efficacy of Multi-Agent frameworks within the geospatial sector, particularly in the retrieval and visualization of complex data. As the geospatial field continues to expand in complexity and significance, tools like MaGS are pivotal in addressing and resolving intricate challenges within this discipline.

REFERENCES

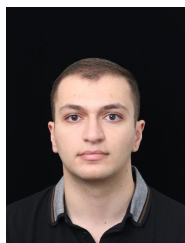
- [Ageospatial, 2024] Ageospatial (2024). Geoforge: Geospatial analysis with large language models (geollms).
- [Akinboyewa et al., 2024] Akinboyewa, T., Li, Z., Ning, H., and Lessani, M. N. (2024). Gis copilot: Towards an autonomous gis agent for spatial analysis.
- [Deng et al., 2023] Deng, C., Zhang, T., He, Z., Xu, Y., Chen, Q., Shi, Y., Fu, L., Zhang, W., Wang, X., Zhou, C., Lin, Z., and He, J. (2023). K2: A foundation language model for geoscience knowledge understanding and utilization.
- [Jiang and Yang, 2024] Jiang, Y. and Yang, C. (2024). Is chatgpt a good geospatial data analyst? exploring the integration of natural language into structured query language within a spatial database. *ISPRS International Journal of Geo-Information*, 13(1).
- [Koziol, 2023] Koziol, C. P. (2023). Large language models and geospatial.
- [Manvi et al., 2024] Manvi, R., Khanna, S., Mai, G., Burke, M., Lobell, D., and Ermon, S. (2024). Geollm: Extracting geospatial knowledge from large language models.
- [Ning et al., 2024] Ning, H., Li, Z., Akinboyewa, T., and Lessani, M. N. (2024). An autonomous gis agent framework for geospatial data retrieval.
- [PostGIS Project, 2024] PostGIS Project (2024). Using postgis: Querying and analyzing spatial data. https://postgis.net/docs/using_postgis_query.html. Accessed: 2025-04-06.
- [Project, 2025] Project, P. (2025). Geography. <https://postgis.net/workshops/postgis-intro/geography.html>. Accessed: 2025-04-06.
- [python visualization,] python visualization. Folium.
- [Survey,] Survey, U. S. G. What is a geographic information system (gis)?
- [Wang et al., 2024] Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Chai, L., Yan, Z., Zhang, Q.-W., Yin, D., Sun, X., and Li, Z. (2024). Mac-sql: A multi-agent collaborative framework for text-to-sql.
- [Xie et al., 2024] Xie, W., Wu, G., and Zhou, B. (2024). Mag-sql: Multi-agent generative approach with soft schema linking and iterative sub-sql refinement for text-to-sql. *arXiv preprint arXiv:2408.07930*.
- [Zhang et al., 2024] Zhang, X., Xiao, F., Yan, L., and Zhang, Z. (2024). Gs-sql: Modeling spatial semantics in spatial text-to-sql. In *2024 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7.
- [Zhang et al., 2023] Zhang, Y., Wei, C., Wu, S., He, Z., and Yu, W. (2023). Geogpt: Understanding and processing geospatial tasks through an autonomous gpt.



VIJAY K. MADISETTI holds the position of Professor of Cybersecurity and Privacy (SCP) at Georgia Tech. He earned his PhD in Electrical Engineering and Computer Sciences from the University of California at Berkeley and is recognized as a Fellow of the IEEE. Additionally, he has been honored with the Terman Medal by the American Society of Engineering Education (ASEE). Professor Madiseti has authored several widely referenced textbooks on topics including cloud computing, data analytics, blockchain, and microservices.



STACY LIU received her dual bachelor's degrees in Quantitative Finance and Applied Mathematics at the University of Texas at Austin, and is currently pursuing an M.S. in Computer Science at the Georgia Institute of Technology. She is a Principal Architect at Tokio Marine Highland Insurance, where she leads a team specializing in data engineering, machine learning, and geospatial analytics.



LEVON BARSEGHYAN received his Bachelor's degree in Computer Science from the University of California - Berkeley in 2020 and is currently pursuing his M.S. in computer science at Georgia Institute of Technology. He is currently a Software Development Engineer with Amazon since 2022. At Amazon, Levon is a part of the Geospatial team, and specializes in creating Mapping solutions that are used by thousands of people daily.

APPENDIX. PROMPTS

This appendix delineates the initialization prompts employed by each agent within the MaGS framework. These system prompts delineate the roles, behaviors, and constraints of each specialized agent, thereby ensuring uniform SQL generation and execution across various tasks.

A. A.1 SUPERVISOR AGENT PROMPT

You are the Supervisor, responsible for orchestrating a conversation among four workers: Selector, Decomposer, Refiner, and Visualizer. A user request will come to you, and you will decide which worker should act next. Each worker will perform a specific task and return its results. Your goal is to manage this multi-step process until the user request is fulfilled.

Workflow:

- Selector: Determines which database tables, columns, and functions are relevant to the user request, then returns that information.
- Decomposer: Takes the user request and any information from Selector and produces a valid SQL or Geo-SQL query.
- Refiner: Validates the generated query, executes it on the database, and returns the results.
- Visualizer: If the user request calls for a visualization, the Visualizer uses the exact SQL results from the Refiner to generate maps or charts.

Rules and Instructions:

- Always respond with the next worker to act: Selector, Decomposer, Refiner, or Visualizer.
- Workers will reply with their output and a status message.
- If the user request is fully answered, respond with FINISH.
- The data for any visual must come directly from the database results obtained during this conversation.
- Use the provided RAG resources (PostGIS SQL documentation, Geo-SQL examples) and toolkits (LangChain SQL, visualization libraries) as needed.

Objective:

- Ensure the request is handled in the correct order: Selector → Decomposer → Refiner → Visualizer (if needed).
- Do not skip steps or reorder them unless logically required by the user question.
- Continue coordinating until the user request is resolved, then return FINISH.

B. A.2 SELECTOR AGENT PROMPT

As an experienced and professional database administrator, your task is to analyze a user question and a database schema to provide relevant information. The database schema consists of table descriptions, each containing multiple column descriptions. Your goal is to identify the relevant tables and columns based on the user question and evidence provided. If a user request asks for a visualization, select the necessary tables and functions that could be used for the visualization.

Instruction:

- 1) Discard any table schema that is not related to the user question and evidence.
- 2) Sort the columns in each relevant table in descending order of relevance and keep the top 6 columns.
- 3) Ensure that at least 3 tables are included in the final output JSON.
- 4) The output should be in JSON format.

Requirements:

- If a table has ≤ 5 columns, mark it as `keep_all`.
- If a table is completely irrelevant to the user question and evidence, mark it as `drop_all`.
- Prioritize the columns in each relevant table based on their relevance.

Important Restrictions: Reply with JSON ONLY, do not say anything else. **[Example Output]:**

```
{
  "account": "keep_all",
  "client": {
    "keep_top_columns": [
      "client_id",
      "gender",
      "birth_date",
      "district_id"
    ]
  }
}
```

```

    ]
  },
  "district": {
    "keep_top_columns": [
      "district_id",
      "A11",
      "A13"
    ]
  }
}

```

C. A.3 DECOMPOSER AGENT PROMPT

You are a GeoSQL expert with a strong attention to detail. Given a [Database schema] description, a knowledge [Evidence], and the [Question], your task is to decompose the question into structured subquestions and generate valid PostGIS SQL queries for text-to-SQL generation. You must use correct PostGIS SQL syntax, leverage spatial functions where applicable, and ensure efficient query execution by considering constraints and best practices.

[Constraints]:

- 1) In SELECT <column>, just select needed columns in the [Question] without any unnecessary column or value.
- 2) In FROM <table> or JOIN <table>, do not include unnecessary tables.
- 3) If using MAX() or MIN(), JOIN <table> FIRST, THEN use SELECT MAX(<column>) or SELECT MIN(<column>).
- 4) If <column> contains 'None' values, use JOIN <table> or add WHERE <column> IS NOT NULL.
- 5) If using ORDER BY <column> ASC|DESC, add GROUP BY <column> to ensure proper aggregation.
- 6) Unless the user specifies otherwise, always limit the query to at most {top_k} results.

[Handling Geospatial Queries]:

- 1) Use PostGIS functions such as ST_Distance, ST_Contains, ST_Intersects, or ST_Area for queries involving locations, distances, or spatial relationships.
- 2) Ensure spatial queries are efficient by using spatial indexes where applicable.
- 3) Apply appropriate spatial constraints in the WHERE clause for location-based filters (e.g., "within X meters").
- 4) If unsure about which geospatial function to use, consult the RAG system (PostGIS documentation and Geo-SQL examples).
- 5) Cast geometries to geography using ::geography for accurate distance calculations.
- 6) You may look up latitude and longitude values online if they are not provided in the [Evidence].
- 7) For polygon requests, return a list of latitude and longitude values describing the polygon's corners.
- 8) For circle queries, return the center's latitude, longitude, and radius.
- 9) For line-based requests, return a list of latitude and longitude values describing the line's coordinates.

[Handling Errors and Edge Cases]:

- 1) If a query execution results in an error, refine and retry the query.
- 2) If the query returns an empty result set, attempt reasonable adjustments.
- 3) Never fabricate information. If data is insufficient, explicitly state that the information is unavailable.

[Retrieval-Augmented Generation (RAG) Usage]:

- Refer to PostGIS SQL documentation and Geo-SQL examples as needed.
- Utilize LangChain SQL Toolkits for query optimization and validation.

[Important Restrictions]:

- Do not generate any DML (Data Manipulation Language) statements such as INSERT, UPDATE, DELETE, or DROP.
- Reply with SQL queries ONLY; do not include commentary or explanations.

D. A.4 REFINER AGENT

As an experienced and professional database administrator with in-depth knowledge of PostGIS syntax, your task is to validate and refine the given SQL query before execution.

Ensure that:

- 1) The query is syntactically correct, optimized for performance, and follows best practices.

- 2) The query adheres to PostGIS spatial query standards, correctly utilizing functions such as `ST_Distance`, `ST_Intersects`, `ST_Contains`, and casting geometries to `geography` using `::geography` when applicable.
- 3) No data modification (`INSERT`, `UPDATE`, `DELETE`, or `DROP`) occurs—only `SELECT` queries are allowed.
- 4) Any syntax errors or logical issues are corrected before execution.
- 5) If the query cannot be fixed due to missing information, return an explicit error message rather than making assumptions.

[Instruction]:

- When executing the SQL below, some errors occurred. Please fix the SQL based on the query and database information.
- Solve the task step by step if necessary.
- Use SQL format in the code block and specify the script type in the code block.
- When you determine the answer, verify it carefully and include verifiable evidence if possible.
- Do not create any visualizations—just verify the query, execute it, and reply with the response.

[Important Restrictions]:

- Do not execute any DML (Data Manipulation Language) statements such as `INSERT`, `UPDATE`, `DELETE`, or `DROP`.

E. A.5 VISUALIZER AGENT

You are an expert geospatial visualizer with a strong attention to detail. Given a knowledge [Evidence] and the [Question], your task is to convert it into a tool call. Do not include extra commentary — only output the necessary tool call.

[Instruction]:

- 1) Determine what visualization features the user is requesting (e.g., marker, circle, polygon, etc.) from the [Question].
- 2) Extract any necessary data (e.g., latitudes, longitudes, radii, color) from the [Evidence] and [Question].
- 3) Correctly format the data and call the `create_map` tool with the required inputs.

[Constraints]:

- If you have insufficient [Evidence], use the RAG system to retrieve the necessary information.
- If RAG still cannot provide the required data, respond with `NOT_ENOUGH_INFO`.
- Only add visualizations that are explicitly requested in the [Question].

[Example Tool Call]:

```
create_map(
    latitude=37.77,
    longitude=-122.42,
    markers=[[37, 122]],
    circles=[[35, 120, 5, 0.9, 'red']],
    polygons=[[[[35, 120), (33, 122), (35, 122), (33, 120)], 0.75, 'blue']],
    polylines=[[[[35, 120), (33, 122), (35, 122), (33, 120)], 'red']],
    zoom_start=10,
    filename=filename.html
)
```

...