SpatialSQL: A Multi-Agent System for Interactive and Observable Spatial Text-to-SQL

Yang Song
Beihang University
Beijing, China
songyangbuaa@buaa.edu.cn

Yu Sun
Nankai University
Tianjin, China
sunyu@nankai.edu.cn

Xieyang Wang
Nanjing University of Aeronautics and Astronautics
Nanjing, China
xieyang@nuaa.edu.cn

Qian Tao
Beihang University
Beijing, China
qiantao@buaa.edu.cn

Tengfei Xu
Beihang University
Beijing, China
txu59721@gmail.com

Jianqiu Xu
Nanjing University of Aeronautics and Astronautics
Nanjing, China
jianqiu@nuaa.edu.cn

Shuyue Wei
Shandong University
Jinan, China
weishuyue@sdu.edu.cn

Yongxin Tong*
Beihang University
Beijing, China
yxtong@buaa.edu.cn

ABSTRACT

With significant advancements in Large Language Models (LLMs) in recent years, text-to-SQL has emerged as a prominent paradigm for querying databases, aiming to transform natural language into executable database queries. Despite the outstanding performance of previous works on relational databases, few studies have addressed spatial queries, where both data and queries are related to spatial information. Compared to text-to-SQL tasks on relational databases, generating spatial SQL from natural language remains challenging due to the complexity of geometric operations and spatial reasoning. To tackle these issues, this paper presents SpatialSQL, a multi-agent system for interactive and observable spatial text-to-SQL generation. At the core of SpatialSQL, several specially designed agents are coordinated to interpret user intent, construct spatial SQL queries, and iteratively refine them based on execution feedback. Additionally, SpatialSQL integrates a platform for monitoring and managing the entire interaction lifecycle, including chat histories, SQL execution traces, and user feedback, through an administrative console. To support interactive exploration, SpatialSQL surfaces intermediate reasoning artifacts and execution results, enabling users to inspect and guide the query generation process. We demonstrate SpatialSQL through a unified user interface and management console that together support transparent query construction and system observability.

PVLDB Reference Format:
Yang Song, Yu Sun, Xieyang Wang, Qian Tao, Tengfei Xu, Jianqiu Xu, Shuyue Wei, and Yongxin Tong. SpatialSQL: A Multi-Agent System for Interactive and Observable Spatial Text-to-SQL. PVLDB, 19(12): XXX-XXX, 2026.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/SongY123/SpatialText2SQL/tree/multi-agent.

*Corresponding author.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:XX.XX/XXX.XX

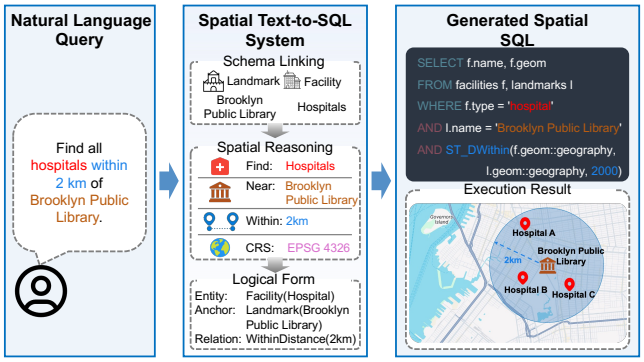


Figure 1: Workflow of a spatial text-to-SQL system

1 INTRODUCTION

Text-to-SQL tasks aim to transform natural language queries into SQL for database querying, reducing the barriers to data analytics and decision-making [7]. The recent emergence of Large Language Models (LLMs), such as GPT-4 [8] and Qwen3 [9], has significantly improved text-to-SQL performance [4, 10], spurring a growing research focus on text-to-SQL.

However, existing text-to-SQL systems primarily focus on relational queries, providing limited support for spatial databases [4, 7], which are widely used across various application domains, including geography, oceanography, urban planning, transportation, and trajectory analytics [11]. Prior works have explored natural language interfaces and spatial semantics modeling for spatial databases [5, 12], but they provide limited support for interactive and observable spatial text-to-SQL generation. As illustrated in Fig. 1, spatial queries require complicated reasoning over geometric types, spatial predicates, and coordinate systems, as defined in the SQL/MM standard [1]. The large function space of spatial DBMSs such as PostGIS [2] and the scarcity of public spatial text-to-SQL benchmarks further widen this gap.

To address these challenges, we present SpatialSQL, a multi-agent system for spatial text-to-SQL generation that extends our prior work NALSpatial [6]. Unlike NALSpatial, which relies on a pre-constructed query corpus and task-specific model training, SpatialSQL adopts an LLM-based agent orchestration workflow without task-specific training. SpatialSQL coordinates multiple spatially specialized agents to interpret spatial intents and construct

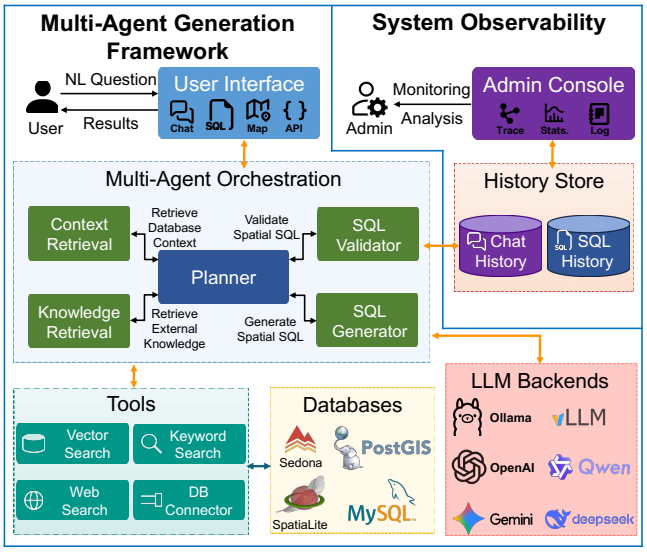


Figure 2: System architecture of SpatialSQL

spatial SQL queries for different tasks, such as range queries, nearest neighbor queries, and spatial join queries. In addition, SpatialSQL records user-agent dialogue history and user feedback, and converts them into structured supervision signals, supporting continuous system improvement. Besides, SpatialSQL adopts a dual-interface design to serve both end users and developers. For end users, it provides a unified interface supporting conversational interaction, SQL inspection, and map visualization to facilitate data exploration. For developers, it features a dedicated admin console for operational monitoring, trace inspection, and failure diagnosis. Under the hood, the backend implements the reasoning pipeline, which empowers the agents with external tools and provides unified connectivity across heterogeneous spatial database management systems.

In summary, the contributions are as follows:

- SpatialSQL is the first interactive and observable multi-agent system for spatial text-to-SQL generation.
- SpatialSQL features a role-specialized multi-agent framework that coordinates context retrieval, knowledge retrieval, SQL generation, and SQL validation to better support geospatial semantics and complex spatial queries.
- SpatialSQL includes an end-to-end observability and management plane that records intermediate reasoning artifacts, tool invocation traces, SQL versions, and execution results for monitoring, diagnosis, and system improvement.

2 SYSTEM DESIGN

2.1 System Architecture

As shown in Fig. 2, the system consists of two parts: the multi-agent generation framework and the system observability module.

The multi-agent generation framework serves as the core of SpatialSQL. It accepts natural language queries from the user interface and coordinates retrieval, planning, SQL generation, and validation to produce executable spatial SQL. To support this process, the framework interacts with external tools, spatial databases, and LLM backends. Further details are presented in Sec. 2.2.

The observability module provides monitoring and analysis support for developers and administrators. It includes an admin console together with a history store that records chat history and SQL history. These components support trace inspection, failure diagnosis, and subsequent system refinement, as described in Sec. 2.3.

2.2 Multi-Agent Generation Framework

Specialization of Agent Roles. Role specialization in SpatialSQL is motivated by the complexity of spatial text-to-SQL, which comprises several tightly coupled steps that are difficult to handle robustly within a single component. A user request often requires interpreting spatial intent, retrieving relevant context and domain knowledge, formulating an executable spatial SQL query, and validating it against schema and execution constraints. SpatialSQL therefore decomposes the overall task into cooperating agent roles, each handling a specific subtask and producing intermediate artifacts for the next step. This modular design clarifies responsibilities and makes the generation process easier to orchestrate, monitor, and refine. Accordingly, SpatialSQL defines the following five roles:

(1) *Planner* serves as the central coordinator of the multi-agent framework. It interprets the user query, determines whether additional context or knowledge is required, and orchestrates the calls to other agents. The *Planner* also aggregates intermediate artifacts and returns the final spatial SQL.
(2) *Context Retrieval Agent* retrieves and structures context for the current query. It aggregates relevant conversation history, prior SQL, and execution outcomes from the persistence layer, and fetches schema metadata and representative database values when needed.
(3) *Knowledge Retrieval Agent* acquires external evidence required for spatial text-to-SQL. It invokes the tool layer (*e.g.*, vector search, keyword search, and web search) to obtain schema hints, domain knowledge, or related references.
(4) *SQL Generator Agent* constructs spatial SQL queries based on the user request, retrieved context, and supporting evidence. It produces SQL drafts that include spatial predicates and functions as needed, together with the key assumptions used during generation.
(5) *SQL Validator Agent* checks candidate SQL queries for executability. It verifies syntax and schema consistency, and it performs checks for spatial queries, including the proper use of spatial predicates and functions, geometry type compatibility, and coordinate reference system consistency when spatial transformations are involved.

Agent Orchestration. SpatialSQL follows an iterative orchestration loop for spatial text-to-SQL. The *Planner* coordinates context acquisition, evidence retrieval, SQL construction, validation, and triggers additional iterations when necessary. Figure 3 illustrates the agent orchestration of SpatialSQL.
Main Path: Given a user query, the *Planner* initiates a processing round by requesting a compact context package from the *Context Retrieval Agent*, which summarizes relevant chat turns, prior SQL, and recent execution outcomes, and optionally includes schema metadata and representative database values. The *Planner* then prompts the *Knowledge Retrieval Agent* to complement this package with external evidence via the tool layer. Conditioned on the combined context and evidence, the *SQL Generator Agent* produces one or more candidate spatial SQL drafts. The drafts are then checked by

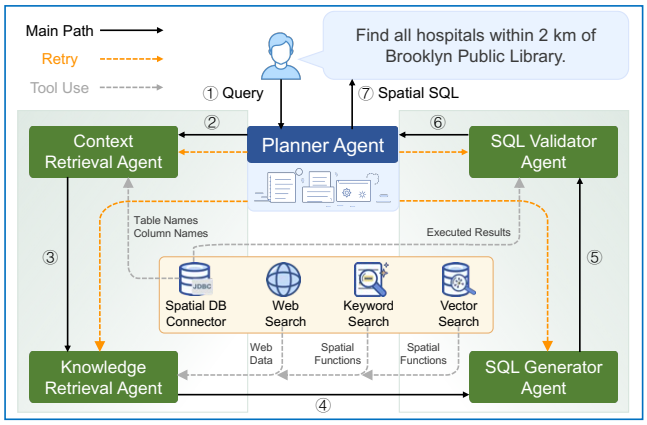


Figure 3: Multi-agent orchestration of SpatialSQL

the *SQL Validator Agent* to ensure they are executable and satisfy key schema and spatial constraints. Upon validation, the query is executed through the database connector, and the results are returned to the user interface.

Failure and Retry: When the *Planner* detects insufficient evidence to proceed, it triggers additional retrieval by requesting more context from the *Context Retrieval Agent* or acquiring external evidence through the *Knowledge Retrieval Agent*. Alternatively, if the *SQL Validator Agent* identifies an incorrect or non-executable SQL query, it returns diagnostic feedback to the *Planner*, which re-orchestrates the next iteration by adjusting retrieval, generation, and validation steps. In both cases, the *Planner* performs targeted retries by updating inputs or constraints and re-invoking generation and validation. The loop terminates when a valid query is produced, when additional user clarification is required, or when a predefined maximum number of retries is reached. We note that the multi-agent workflow introduces additional latency and token overhead, but SpatialSQL reduces this cost by compressing conversation history, invoking agents only when needed, and limiting the number of retries.

Tools. The tool layer provides a unified interface between the multi-agent generation framework and the external environment. It is introduced to (i) decouple agent logic from heterogeneous tool backends, (ii) standardize tool inputs and outputs into reusable artifacts, and (iii) support traceable and replayable tool invocations for system monitoring and debugging. To this end, SpatialSQL integrates a set of tools in the tool layer, described as follows:

(1) *Vector Search*: A vector search tool used to fetch relevant evidence from indexed resources, such as schema descriptions, documentation of spatial databases, and previously curated examples. It is typically invoked when the query requires semantic similarity search or when keyword signals are insufficient.

(2) *Keyword Search*: A keyword search tool used to locate exact or fuzzy matches, such as table and column names, spatial function names, or domain specific entities. It complements vector search by providing precise matching for queries that reference columns, tables, spatial functions, or entities.

(3) *Web Search*: A web search tool used to obtain external information when local resources are insufficient, for example, domain background or definitions referenced by the user query, or relevant examples for resolving SQL errors when the agents cannot repair a query locally. Retrieved content is treated as supporting evidence and is incorporated into subsequent reasoning steps.

(4) *Database Connector*: A database connector tool provides access to spatial databases to supply database context and query execution support. It retrieves spatial database clues needed during generation, such as representative values and geometry related metadata, and it executes candidate SQL to support validation and returns results and execution feedback to guide repair steps when validation fails.

2.3 System Observability Module

System observability is essential for SpatialSQL because the spatial text-to-SQL process spans multiple components, and failures are often difficult to diagnose from the final output alone. This section describes how SpatialSQL improves transparency and supports debugging by collecting key observability signals, recording them as structured traces, and presenting them in an observability console.

Observability Signals. SpatialSQL organizes observability into three categories: (i) a system overview that reports operational statistics across sessions, chats, and SQL queries, (ii) interaction histories that record conversations and SQL executions together with tool-invocation logs, and (iii) user feedback during chats.

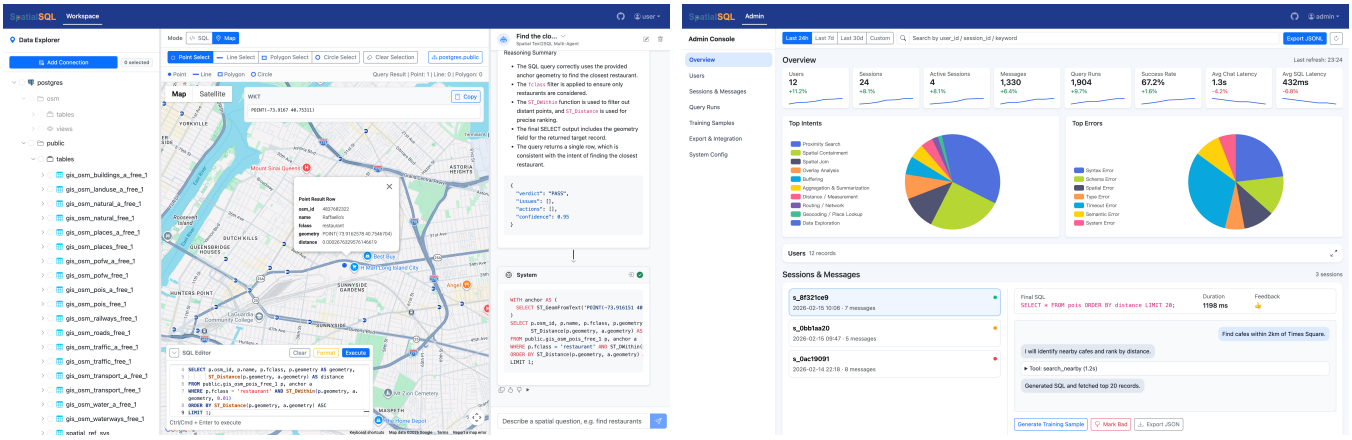
Trace Recording. System observability in SpatialSQL aims to make the spatial text-to-SQL process transparent and diagnosable. Accordingly, SpatialSQL collects signals from the system overview, interaction histories, and user feedback to support both runtime monitoring and post hoc analysis. To enable inspection and replay, SpatialSQL aligns these signals along the session and turn timeline and records a unified trace for each request that links the conversation, tool invocations, and SQL executions. With these traces, developers can replay failed sessions to trace any failure back to its specific flawed reasoning step.

Observability Console. SpatialSQL surfaces these signals and traces through an observability console for monitoring, diagnosis, and data curation. An overview dashboard visualizes aggregated statistics at the session, chat, and SQL-query levels, helping operators assess system status and detect anomalous patterns. For debugging, a trace inspector organizes records by session and turn, presenting conversations, tool invocations, and SQL executions in a unified view. This drill-down workflow helps developers localize failures and inspect intermediate evidence used during generation and execution. The console also links user feedback to corresponding traces and supports quality annotation, curating high-quality data for optimizing the multi-agent system.

3 DEMONSTRATION PLAN

We demonstrate SpatialSQL through two complementary views: the *user interface* for interactive spatial text-to-SQL and the *admin console* for monitoring, analysis, and data curation.

Setup. SpatialSQL is deployed as a client-server web application. The backend is implemented as a FastAPI server, which orchestrates five specialized agents through AgentScope [3] and connects to spatial databases via a unified connector. The frontend provides the user interface and the administrative console, communicating with the backend through RESTful APIs and Server-Sent Events to support real-time interaction between users and agents. We load



(a) User interface (b) Admin interface

Figure 4: SpatialSQL interfaces for interactive spatial text-to-SQL and system observability

a geospatial dataset of urban features, including facilities, landmarks, and transportation infrastructure, into a database instance to support spatial query exploration in an urban analytics scenario.

User Interface for Interactive Spatial Text-to-SQL. As shown in the user interface of Fig. 4a, the system adopts a resizable multi-panel layout. The left panel provides a hierarchical schema tree for browsing connected databases and selecting relevant tables or views as context. The center panel supports both *SQL mode* for syntax-highlighted query editing and execution, and *Map mode* for spatial visualization, WKT geometry drawing, and region-of-interest specification through Google Maps. Execution results are shown in a collapsible paginated preview pane and can be inspected as tables or map overlays. The right panel provides a chat interface where users can ask questions, follow the multi-agent generation process, and insert generated SQL into the editor with one click.

Admin Interface for System Observability. As shown in the admin console of Fig. 4b, the interface is organized as a dashboard with sidebar navigation and a scrollable main content area. A global toolbar provides time-range filters and search functionality, while the main sections present high-level views of system activity, query execution, training data management, data export, and configuration. Among these components, the Sessions & Messages section is the central workspace for monitoring and improving user interactions. It allows administrators to inspect session histories, review user feedback, and assess the quality of responses. The resulting interaction data can then be used to further improve the agents.

4 CONCLUSION

In this demonstration, we present SpatialSQL, a multi-agent system for interactive and observable spatial text-to-SQL. SpatialSQL organizes spatial query translation as an orchestration loop that integrates role-specialized agents for spatial SQL generation. Beyond returning spatial SQL and results, SpatialSQL captures signals from interaction and execution and makes them accessible through the admin console for inspection. While our implementation targets common spatial databases, its modular architecture provides a blueprint for extending agent-based query interfaces to other specialized data systems, such as time-series and graph databases.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (NSFC) (Grant Nos. T2588101, 62425202, and 62336003), the Beijing Natural Science Foundation (Z230001), the Fundamental Research Funds for the Central Universities No. JKF-2026007844235, and the State Key Laboratory of Complex & Critical Software Environment (SKLCCSE). Qian Tao's work is partially supported by Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing. Shuyue Wei's research is supported by Shandong Provincial Natural Science Foundation (No. ZR2026QC1215).

REFERENCES

[1] 2016. ISO/IEC 13249-3:2016 Information technology - Database languages - SQL multimedia and application packages, Part 3: Spatial. Retrieved June 23, 2026 from https://www.iso.org/standard/60343.html
[2] 2026. PostGIS. Retrieved June 23, 2026 from https://postgis.net/docs/manual-3.6
[3] Dawei Gao, Zitao Li, Yuexiang Xie, et al. 2025. AgentScope 1.0: A Developer-Centric Framework for Building Agentic Applications. *CoRR* abs/2508.16279 (2025).
[4] Dawei Gao, Haibin Wang, Yaliang Li, et al. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *PVLDB* 17, 5 (2024), 1132–1145.
[5] Jingjing Li, Wenlu Wang, Wei-Shinn Ku, et al. 2019. SpatialNLI: A Spatial Domain Natural Language Interface to Databases Using Spatial Comprehension. In *SIGSPATIAL*. 339–348.
[6] Mengyi Liu, Xieyang Wang, Jianqiu Xu, et al. 2025. NALSpatial: A Natural Language Interface for Spatial Databases. *IEEE Transactions on Knowledge and Data Engineering* 37, 4 (2025), 2056–2070. https://doi.org/10.1109/TKDE.2025.3525587
[7] Xinyu Liu, Shuyu Shen, Boyan Li, et al. 2025. A Survey of Text-to-SQL in the Era of LLMs: Where are we, and where are we going? *IEEE Transactions on Knowledge and Data Engineering* 37, 10 (2025), 5735–5754.
[8] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023).
[9] Qwen Team. 2025. Qwen3 Technical Report. *CoRR* abs/2505.09388 (2025).
[10] Bing Wang, Changyu Ren, Jian Yang, et al. 2025. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. In *ACL*. 540–557.
[11] Yuxiang Wang, Shuyuan Li, Yongxin Tong, et al. 2026. Efficient shapley-based data valuation for federated trajectories. *Frontiers of Computer Science* 20, 9 (2026), 2009621.
[12] Xu Zhang, Feiyang Xiao, Liang Yan, and Zhiqing Zhang. 2024. GS-SQL: Modeling Spatial Semantics in Spatial Text-to-SQL. In *IJCNN*. 1–7.