BridgeScope: A Universal Toolkit for Bridging Large Language Models and Databases

Lianggui Weng*, Dandan Liu*, Rong Zhu#, Bolin Ding, Jingren Zhou
Alibaba Group
{lianggui.wlg,ldd461759,red.zr,bolin.ding,jingren.zhou}@alibaba-inc.com

Abstract

As large language models (LLMs) demonstrate increasingly powerful reasoning and orchestration capabilities, LLM-based agents are rapidly adopted for complex data-related tasks. Despite this progress, the current design of how LLMs interact with databases exhibits critical limitations in usability, security, privilege management, and data transmission efficiency. To address these challenges, we introduce BridgeScope, a universal toolkit that bridges LLMs and databases through three key innovations. First, it modularizes SQL operations into fine-grained tools for context retrieval, CRUD execution, and ACID-compliant transaction management. This design enables more precise, LLM-friendly controls over database functionality. Second, it aligns tool implementations with database privileges and user-defined security policies to steer LLMs away from unsafe or unauthorized operations, which not only safeguards database security but also enhances task execution efficiency by enabling early identification and termination of infeasible tasks. Third, it introduces a proxy mechanism that supports seamless data transfer between tools, thereby bypassing the transmission bottlenecks via LLMs. All of these designs are database-agnostic and can be transparently integrated with existing agent architectures. We also release an open-source implementation of BridgeScope for PostgreSQL. Evaluations on two novel benchmarks demonstrate that BridgeScope enables LLM agents to interact with databases more effectively. It reduces token usage by up to 80% through improved security awareness and uniquely supports data-intensive workflows beyond existing toolkits. These results establish BridgeScope as a robust foundation for next-generation intelligent data automation.

1 Introduction

Background. Large language models (LLMs) have been widely applied in data-related tasks, such as data manipulation, data cleaning, and exploratory data analysis [20], to enable more automated and intelligent solutions. Due to the intrinsic complexity of these tasks, they are typically decomposed into subtasks that are manageable for the LLM, through expert-crafted frameworks [20]. For instance, a typical LLM-based NL2SQL pipeline involves three subtasks, *i.e.*, schema linking, SQL generation, and SQL revision [5]. However, such decompositions are often highly tailored to specific task scenarios and lack generalizability across broader classes of tasks. As the landscape of data-related tasks grows in both scale and complexity, relying solely on manually crafted frameworks is becoming increasingly impractical.

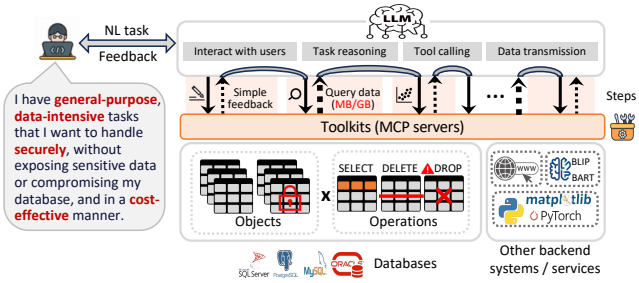


Figure 1: Architecture of a general agent.

Recently, advanced LLMs (*e.g.*, GPT-4o [4], DeepSeek R1 [3], and Claude-4 [1]) have demonstrated impressive capabilities in multi-step reasoning and task planning [18]. In parallel, the introduction of the model context protocol (MCP) [16] has standardized LLM interactions with external systems, which substantially enhances LLMs' capabilities by providing access to a wide range of domain-specific tools. Together, these advances are ushering in a new era of *fully automated*, *general-purpose* agents that can flexibly orchestrate and execute diverse tasks, particularly data-related ones, with minimal human intervention.

As illustrated in Figure 1, such a *general agent* employs an LLM as an intelligent manager, which: 1) accepts and responds to users' natural language (NL) tasks; 2) reasons over the task and orchestrates manageable steps to resolve the task (*e.g.*, querying necessary data or performing data analysis), shown as pink-shaded blocks; 3) invokes appropriate tools (black arrows) to handle each step; and 4) bridges and coordinates intermediate data flow from one tool to the next, possibly across different backends (*e.g.*, from web search services to local Python execution environments). Towards this direction, both research efforts [7, 17] and production systems such as the Manus agent [9] and the ChatGPT agent [2] have achieved state-of-the-art breakthroughs.

Challenges. Current efforts in data-related scenarios primarily focus on optimizing LLM-generated pipelines for complex tasks [7, 17]. However, the orthogonal direction of designing database toolkits, which fundamentally shape the agents' capabilities in handling data, remains largely unexplored. Database toolkits within the MCP ecosystem [10] are typically rudimentary, offering only a generic execute_sql tool for executing SQL statements and a get_schema tool for schema inspection. As we analyzed below, such toolkits are inadequate for real-world data-related tasks, which in turn constrain the agents' capabilities and even expose the task-solving process

*: Equal Contribution, #: Corresponding Author

 16th Annual Conference on Innovative Data Systems Research (CIDR '26). January 18-21, 2026, Chaminade, USA.

to security risks, excessive resource consumption, and potential failures:

• **C1. Coarse-Grained Tooling.** The universal execute_sql tool is both insecure and cumbersome. On one hand, it grants agents unrestricted access to all user data and database operations, increasing the risk of sensitive data exposure or unintentional execution of destructive commands (*e.g.*, DROP DATABASE) due to LLM hallucinations or prompt injection attacks [12]. On the other hand, overloading a single tool with diverse responsibilities from querying data to managing transactions greatly complicates LLMs' usage and heightens the risk of tool-selection errors.

• **C2. Privilege-Unaware Planning.** Current agents lack intrinsic awareness of users' privileges in the database, and may therefore generate execution pipelines exceeding authorized operations (*e.g.*, querying unauthorized tables). Although the database engine will eventually reject such operations, generating infeasible plans incurs non-negligible side effects, including additional LLM calls and token usage for replanning, as well as increased resource consumption for tool execution, especially when privilege violations occur late in the task execution workflow.

• **C3. Direct Transmission of Voluminous Data.** The current agent architecture (Figure 1), which relies on LLMs to exchange data between tools, is particularly problematic for data-intensive workflows. In such settings, the LLM's limited context window can be quickly exhausted and finally cause task failures. Moreover, hallucinations during data transmission can introduce errors, which further compromise the reliability of this approach.

Our Contributions. In response to Challenges C1–C3, we present BridgeScope, a systematic database toolkit designed to enhance both the effectiveness and efficiency of general-purpose LLM agents in data-related tasks. *To the best of our knowledge, BridgeScope is the first toolkit specifically developed for this domain*, providing a solid foundation for advancing the data-handling capabilities of both research prototype and production-grade agents.

At a higher level, BridgeScope offers three core functionalities to support arbitrary data-related tasks: 1) *context retrieval*, which obtains task-related contextual information (*e.g.*, database schema and column exemplars) as essential background for subsequent LLM interactions; 2) *SQL execution*, which enables general-purpose CRUD operations (Create, Read, Update, and Delete) over databases; and 3) *transaction management* for end-to-end ACID-compliant task execution. With each functionality implemented via a dedicated set of *fine-grained* tools, BridgeScope enables more precise and LLM-friendly control over database operations (as per Challenge C1). Instead of generic tool implementations, BridgeScope customizes tools aligning with both database-side user privileges and user-side security policies. Specifically, it includes only user-permitted database objects, along with their associated privilege information, in the context retrieval results, and selectively exposes only those tools capable of executing authorized, low-risk SQL statements (*e.g.*, exposing solely the select tool for read-only users). This design keeps LLMs aware of their operational boundaries, confines task planning to authorized scopes, and facilitates early detection and termination of infeasible tasks (addressing Challenge C2). Moreover, BridgeScope goes beyond passive restriction by incorporating rule-based verifications to strictly filter out unauthorized access or high-risk operations inadvertently issued by the LLM, providing an extra layer of protection. (see Sections 2.2–2.4 for details)

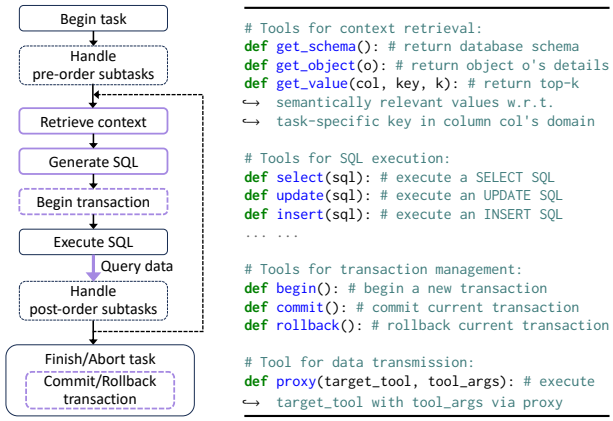


Figure 2: The abstracted workflow of data-related tasks (left) and fine-grained BridgeScope toolkit (right).

Beyond basic functionalities, BridgeScope encapsulates a *proxy* mechanism to support inter-tool data transmission (as per Challenge C3). With this approach, LLMs can delegate tool execution and data routing to a proxy tool, which autonomously forwards intermediate results from upstream tools to downstream ones to trigger subsequent steps, entirely bypassing the LLM in the data path. This design not only avoids overwhelming the LLM's context window with large-scale data transfers but also enables seamless integration into the evolving MCP ecosystem, thereby substantially broadening the agent's capabilities to support a wider range of data-intensive tasks. (see Section 2.5 for details)

BridgeScope is database-agnostic and can be flexibly implemented for various database systems. We have released an open-source implementation for PostgreSQL [15]. To rigorously evaluate BridgeScope, we introduce two novel benchmarks: one extends the NL2SQL benchmark BIRD [6] to include complex database modifications, and the other involves building machine learning pipelines using data extracted from the database. Experimental results show that BridgeScope not only offers clear advantages in support database operations such as transaction management, but also enhances LLM's security awareness, which contributes up to 80% less token costs by intercepting infeasible or unauthorized tasks. Furthermore, BridgeScope pioneers support for data-intensive workflows beyond the reach of existing toolkits. (see Section 3 for details)

2 BridgeScope: Design of the Toolkit

BridgeScope abstracts essential functionalities from general data-related workflows and formulates core principles to guide tool design (Section 2.1). For each functionality, a suite of fine-grained tools is proposed (Sections 2.2–2.5) to jointly address Challenges C1–C3 discussed in Section 1. Some implementation remarks of BridgeScope are provided in Section 2.6.

2.1 Functionalities and Design Principles

Functionalities Abstraction. We begin by examining key database operations involved in data-related workflows. As shown in

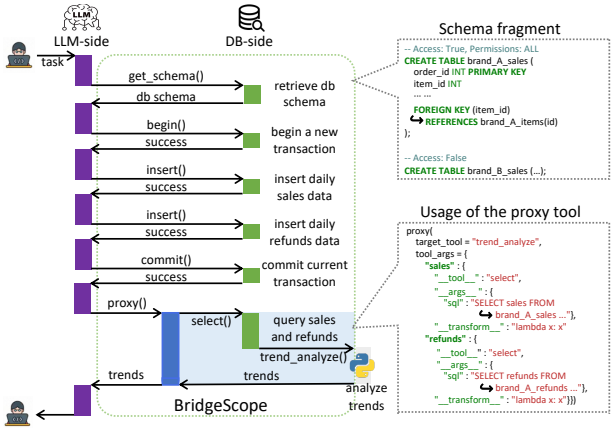


Figure 3: An illustrative example of how BridgeScope supports the chain store workflow.

Figure 2(left), a typical workflow usually starts with preliminary steps that do not require database access (*e.g.*, intent resolution). When data querying or manipulation is needed, essential contextual information, such as the database schema, is retrieved to enable accurate SQL formulation (**F1. context retrieval**). A valid, privilege-compliant SQL statement is then constructed and executed to fulfill the task objective (**F2. SQL execution**). For operations that modify the underlying database, a transaction must be initiated, and all modifying SQL statements are executed within its context to ensure ACID compliance (**F3. transaction management**). After execution, the resulting data or feedback may subsequently be routed to follow-up tools (**F4. data transmission**) for further processing or analysis [17]. If any step fails, all changes to the database are rolled back; otherwise, the workflow concludes with persisted database updates. Notably, one or more of these steps may be iterated multiple times within a single task.

For instance, consider a chain store scenario in which a Brand A manager daily updates sales and refund records, and analyzes recent sales trends. An LLM agent can support this workflow through a structured sequence of operations: 1) retrieving database metadata such as table schemas and column names relevant to sales and refund records (**F1**); 2) atomically inserting the day's new records into the database (**F2** and **F3**); 3) querying recent sales and refund data to gather inputs for analysis (**F2**); and 4) forwarding the retrieved data to an advanced machine learning tool for trend detection (**F4**).

Design Principles. From the above workflow and Challenges C1–C3 stated in Section 1, we distill four key principles for tool design:

1) Fine-Grained Tooling. Rather than relying on a generic execution tool, tools should be fine-grained to align with specific LLM–database interaction patterns. This enables more flexible functionality controls and lower LLM's cognitive load for tool selection, as per Challenge C1.

2) Robust Security Guarantees. Security must be enforced at two complementary levels: *1) (**database-side**)* Modern database systems such as PostgreSQL [14] and MySQL [13] provide fine-grained access controls over both actions (*e.g.*, SELECT, INSERT, UPDATE, DELETE, and *etc*) and data objects (*e.g.*, tables, views, columns, and *etc*). Tool calls must strictly respect these native privilege mechanisms in all operations; *2) (**user-side**)* Tools should also support user-defined security policies that further restrict the LLM's effective privileges to a safe subset of the user's own permissions, thereby preventing access to sensitive data and blocking potentially destructive operations.

3) High Efficiency. Toolkit design must prioritize efficiency in both time and computational resources, especially given the high cost of data-intensive tasks. It should empower the LLM to complete feasible tasks rapidly while promptly identifying and terminating infeasible or unsafe ones.

4) Transparency. Tools must operate transparently: they should neither disrupt existing workflows nor require modifications to other components. This ensures seamless integration into the diverse and evolving MCP ecosystem, and thereby enables broader support for heterogeneous data-related tasks.

Guided by these principles, BridgeScope implements each core functionality (**F1–F4**) through a set of dedicated, operation-specific tools, as illustrated in Figure 2(right). The remainder of this section details their design and demonstrates how they collectively support the chain store workflow introduced in Figure 3.

2.2 Context Retrieval

BridgeScope supports the retrieval of two key facets of task-related context: *database schemas* and *column exemplars*.

Schema Retrieval. Database schemas define the structure and relationships among database objects (*e.g.*, tables, columns, indexes, and foreign-key dependencies) and are essential for accurate SQL generation [5]. To accommodate databases of varying scale, BridgeScope employs an adaptive schema retrieval strategy:

1) For databases with a manageable number of named objects (*i.e.*, fewer than a user-specified threshold n), BridgeScope provides a get_schema tool that returns a standardized, LLM-readable representation of the entire schema (see Figure 3). This allows the LLM to grasp the complete database structure in a single tool call.

2) Otherwise, BridgeScope adopts a hierarchical approach to enable targeted, token-efficient retrieval: the get_schema tool returns only the names of top-level objects (*e.g.*, tables and views), while detailed information for a specific object o (*e.g.*, the columns, indexes, and constraints) is fetched on demand via get_object(o).

BridgeScope employs two complementary mechanisms to steer LLM planning in compliance with the security requirements outlined in Section 2.1. First, database-side privileges are made explicit to the LLM by augmenting schema representations with privilege annotations for each object. As illustrated in Figure 3, annotations above table schemas indicate that the store manager has full access to `brand_A_sales` but is denied access to `brand_B_sales`. More granular privileges (*e.g.*, read-only access to specific columns) can be expressed analogously. Second, to further restrict the LLM's access to sensitive data (*e.g.*, the employee salary table) within the user's granted privileges, BridgeScope enables users to define object-level access policies via configurable white- and black-lists. The schema retrieval tools then expose only those objects explicitly permitted by the user, filtering out restricted ones from the LLM's view. Together, these mechanisms provide the LLM with clear awareness of its access boundaries, guiding it toward secure, compliant planning and reducing hallucinated or unauthorized operations.

Column Exemplars Retrieval. Due to synonyms, spelling variations, and domain-specific terminology in database columns, LLMs often struggle to generate predicates aligned with actual data, leading to incomplete or inaccurate query results. By referencing column exemplars, however, LLMs can formulate semantically accurate SQL predicates. For instance, when analyzing sales trends for women's clothing, examining sample values in the category column enables the LLM to correctly generate predicate category = "women" instead of an unsupported variant like category = "women's wear". To support this capability, BridgeScope provides the get_value(col, key, k) tool, which retrieves the top-k values from column col that are most semantically relevant to a task-specific keyword key (*e.g.*, "women"). Notably, this targeted retrieval dramatically reduces the LLM's context burden compared to exhaustive enumeration of all distinct column values.

2.3 Security-Gauranteed SQL Execution

BridgeScope adopts a two-level mechanism that enforces user-specific tool restrictions to help secure and efficient task execution.

1) Action-Level Tool Modularization. BridgeScope modularizes SQL execution into a set of fine-grained tools and selectively exposes them to the LLM. Let $\mathcal{A}$ be the set of database actions (*e.g.*, SELECT, INSERT, UPDATE, DELETE, and *etc*). For each action $a \in \mathcal{A}$, BridgeScope instantiates a dedicated tool T_a that exclusively executes SQL statements performing a (*e.g.*, the select tool handles only SELECT queries). Let the privilege set of user u be $\mathcal{P}_u \subseteq \mathcal{A} \times \mathcal{O}$, where $\mathcal{O}$ denotes the set of database objects. The agent for user u is granted access to tool T_a only if there exists at least one object $o \in \mathcal{O}$ such that $(a, o) \in \mathcal{P}_u$. For example, a chain store sales assistant with read-only access to sales data receives only the select tool, whereas the manager with full CRUD privileges is granted all relevant tools, including the insert tool for updating daily sales records. As with object-level restrictions (Section 2.2), users may further refine tool access via white- or black-lists, *e.g.*, blocking the drop tool to prevent destructive actions like DROP DATABASE. These dual restrictions give the LLM an inherent understanding of its operational boundaries and effectively discourage unauthorized attempts and risky actions.

2) Object-Level Tool Verification. Although BridgeScope exposes permitted objects and privileges to the LLM via the get_schema tool, access violations may still occur due to LLM hallucinations or prompt injection attacks. To guarantee safe execution, BridgeScope enforces per-object verification during tool execution, ensuring that: 1) the user holds privileges to operate the object; and 2) the object complies with the user's configured security policies (*e.g.*, blacklist). This tool-side enforcement not only intercepts unauthorized operations before they reach the database, reducing unnecessary load, but also extends the database's native security model by blocking user-prohibited actions that might otherwise be syntactically valid under the user's base privileges.

2.4 Transaction Management

BridgeScope follows traditional database transaction controls and provides three distinct tools: begin, commit, and rollback, to manage transactions explicitly. The ACID properties of each transaction (*i.e.*, the sequence of operations between begin() and commit()) are guaranteed inherently by the database engine. As illustrated in Figure 3 for the chain store scenario, the agent invokes begin() to initiate a transaction, atomically inserts both sales and refunds records, and calls commit() to finalize the changes upon success. Our experiments (Section 3.2) show that these explicit transaction tools substantially improve the LLM's ability to manage transactions, thereby resulting in more robust and reliable workflows.

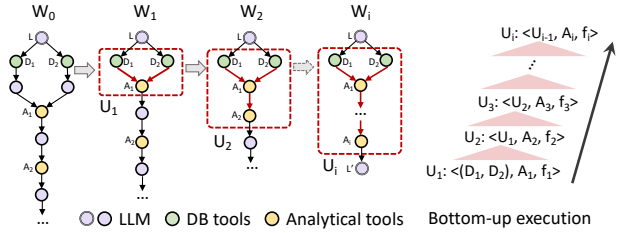


Figure 4: Illustration of proxy units.

2.5 Data Transmission with Proxy

Data-related tasks usually involve a *data producer* tool that generates a large volume of data for use by a downstream *data consumer* tool. To streamline data transmission, BridgeScope introduces an advanced *proxy* mechanism that automatically routes data between producer and consumer tools, freeing LLMs from any involvement.

Foundations. The proxy mechanism operates on each *proxy unit* which specifies a data flow as a triple $\langle p, c, f \rangle$. Here, p denotes a (or a list of) data producer(s) that supply the required data, c is the target data consumer, and f is an adaptation function that transforms the outputs of p to the input format expected by c.

This mechanism supports arbitrarily complex data flows by allowing each proxy unit to act as a data producer for higher-level recursive proxy units. As illustrated in Figure 4, consider a typical data-related workflow without proxies, exemplified by W_0, where the LLM serves as an intermediary between every pair of tools to transfer data, the data flow from database tools D_1 and D_2 to analytical tool A_1 can be abstracted as a proxy unit $U_1 = \langle (D_1, D_2), A_1, f_1 \rangle$. Treating U_1 as a producer, its output to analytical tool A_2 forms another proxy unit $U_2 = \langle U_1, A_2, f_2 \rangle$. This recursion continues until data transfer requires LLM-specific reasoning. In this example, the entire workflow between the initial and final LLM reasoning steps (*i.e.*, L and L') can be abstracted as a hierarchical proxy unit U_i.

During execution, proxy units are processed in a bottom-up manner, with leaf units (*i.e.*, U_1) handled first. Specifically, each data producer (*i.e.*, D_1 and D_2) is invoked to collect the required data, the adaptation function f_1 is then applied, and the transformed result is passed directly to the consumer A_1. The output of A_1 is subsequently propagated upward through the proxy hierarchy to serve as input for the next-level unit U_2. This process continues until the outermost proxy unit U_i is processed and the result from A_i is finally returned to the LLM (L_i).

Benefits of Proxy. Data transmission during the execution of proxy units occurs directly between tools, bypassing the LLM bottlenecks described in Challenge 3 (Section 1). Beyond this core advantage, the proxy mechanism offers additional practical benefits.

1) Task Execution Efficiency. By eliminating LLM-mediated data transmission steps, the proxy mechanism significantly streamlines task execution and shortens the critical path (*e.g.*, by condensing the workflow W_0 into the much simpler W_i shown in Figure 4). It also enables parallel execution of multiple data producers (*e.g.*, database tools D_1 and D_2), further accelerating task completion.

2) Adaptability. Through the built-in adaptation function f, the proxy mechanism automatically converts data formats and remains transparent to downstream tools. As emphasized in Section 2.1, this is crucial for seamless integration within the expansive MCP ecosystem and enables BridgeScope to interoperate effortlessly with any domain-specific MCP servers to support diverse task scenarios.

Implementations. BridgeScope implements the proxy mechanism through a dedicated proxy tool, supported by a carefully designed prompt that enables the LLM to recognize when to invoke the tool and to generate proxy units autonomously. The tool accepts two arguments to define a proxy unit: target_tool, which specifies the downstream tool c, and tool_args, a dictionary that maps each input of target_tool to its data producer(s) p and a transformation function f for format adaptation. When the LLM invokes the proxy tool, the corresponding proxy unit executes, and the output of target_tool is returned. This design elevates the LLM from a passive data router to an active orchestrator of proxy units, while delegating data flow management entirely to the proxy tool.

Figure 3 illustrates the use of the proxy tool in the chain store task. Here, target_tool is set to the trend_analyze tool, which analyzes sales trends using a machine learning model. The tool takes two inputs: a list of sales records, sales, and a list of refund records, refunds. Each input is produced by a select tool that executes an SQL query sql to fetch recent sales or refunds data. Since the query results already match the expected format, an identity transformation is applied. When the proxy tool is invoked, it executes the data selection, runs the trend analysis, handles all inter-tool data transfer, and returns the sales trends generated by trend_analyze.

2.6 Remarks of BridgeScope

All tools in BridgeScope are designed to be database-agnostic. They rely on a unified set of primitive database interfaces: executing generic SQL, retrieving metadata, fetching user privileges, and controlling transactions, which are supported by virtually all mainstream database systems. Regarding cross-database discrepancies (*e.g.*, differences in SQL dialects), an abstraction layer with database-specific adapters handles them, and the core tool implementation logic remains unchanged. This decoupling of tool logic from database specifics makes migration to new databases straightforward. To showcase this, we release an open-source implementation of BridgeScope for PostgreSQL [15]. This database-agnostic design enables LLMs to interact with any data source using a consistent set of tools, without managing system-specific variations, and significantly enhances their effectiveness in multi-datasource scenarios.

Our implementation also includes a carefully designed prompt that enables more efficient and ACID-compliant interactions between the LLM and the database. This prompt can be incorporated into any general-purpose agent to improve its handling of database-related steps. As shown in Section 3, the combination of BridgeScope's toolkit and this prompt achieves superior performance in supporting data-related tasks.

3 Experimental Evaluation

This section comprehensively evaluates BridgeScope to address the following questions: 1) How does fine-grained tool modularization impact the handling of data-related tasks (Section 3.2)? 2) How does privilege-aware tooling help intercept infeasible tasks and reduce unnecessary token consumption (Section 3.3)? 3) What benefits does the proxy mechanism provide (Section 3.4)? Notably, we have tested scenarios that involve privilege violations and operations that exceed user security policies, all of which were successfully intercepted by BridgeScope's rule-based security controls. Therefore, we omit further security evaluation in the following of this section.

3.1 Experimental Setup

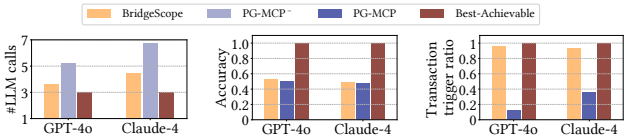
We implement two prototype general-purpose agents based on the ReAct framework [19], using GPT-4o and Claude-4 as the underlying LLMs. Throughout the evaluation, we refer to each agent by the name of its underlying LLM.

Baseline. We compare BridgeScope with a baseline adapted from the official MCP server for PostgreSQL [10], referred to as PG-MCP. PG-MCP offers two tools: get_schema for schema retrieving and execute_sql for SQL execution. This design is representative and widely adopted in current MCP servers for databases.

Benchmarks. Existing benchmarks are too simplistic to reflect the full functional requirements of real-world data-related tasks. To enable a more rigorous evaluation of BridgeScope, we construct two novel benchmarks that comprehensively cover the core functionalities (*i.e.*, **F1–F4**) of general data-related workflows, as abstracted in Section 2.1, which can thereby serve as first-order proxies for assessing the real-world performance of BridgeScope. Both benchmarks are made publicly available [15].

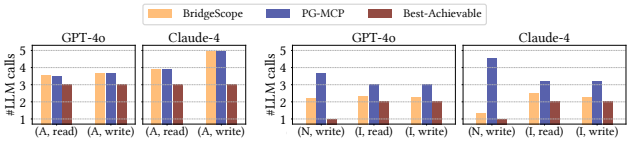
1) BIRD-Ext. This benchmark extensively tests functionalities **F1**, **F2**, and **F3** (*i.e.*, context retrieval, SQL execution, and transaction management). We extend the state-of-the-art NL2SQL benchmark BIRD [6], which originally focuses on read-only SELECT queries, by incorporating complex data manipulation operations (*i.e.*, INSERT, UPDATE, and DELETE). For each operation type, we select 50 original SELECT tasks, modify the involved tables, and adapt the natural language descriptions to create new tasks. The extended benchmark comprises 150 randomly sampled SELECT tasks and 150 newly synthesized tasks. It preserves the complexity and diversity of BIRD while placing greater emphasis on user privileges and transaction management in the context of data modifications.

2) NL2ML. This benchmark targets scenarios that involve large-scale data transmission (**F4**). It simulates an end-to-end model training workflow on the California Housing dataset [11], which comprises a single house table with 10 columns and 20,000 rows. NL2ML includes 30 natural language tasks across three complexity levels (10 tasks each): 1) basic data querying and model training, 2) additional data processing (*e.g.*, normalization), and 3) further house price prediction, which correspond to one, two, and three layers of proxy unit abstraction, respectively. For example, the second level requires data to pass through querying, processing, and training steps. Unlike prior benchmarks [8] focused on small-scale data processing, NL2ML captures real-world demands for large-scale data transmission and complex workflow orchestration.



(a) Context retrieval (b) SQL execution (c) Transaction management

Figure 5: Performance *w.r.t.* tooling granularity.



(a) Feasible tasks (b) Infeasible tasks

Figure 6: Average number of LLM calls for BIRD-Ext.

3.2 Coarse-Grained vs. Fine-Grained Tooling

This experiment investigates how the granularity of tools for each core functionality of LLM-database interactions, as described in Section 2.1, affects the handling of BIRD-Ext tasks.

1) Context Retrieval. To isolate the impact of explicit context retrieval tools, we compare BridgeScope with a variant of PG-MCP that offers a single execute_sql tool for both SQL execution and context retrieval, termed PG-MCP⁻. Figure 5(a) shows the average number of LLM calls required to complete each BIRD-Ext task using these toolkits, which directly reflects practical cost and efficiency. We observe that BridgeScope reduces LLM calls by over 30%, approaching the *best-achievable* value of three calls (one each for context retrieval, SQL execution, and result finalization). This gain arises because LLMs frequently hallucinate incorrect schema details and predicates when using PG-MCP-S, which causes futile retries. In contrast, BridgeScope's explicit context retrieval tools guide LLMs to gather necessary information in advance, ensuring more reliable and efficient SQL generation.

2) SQL Execution. The main advantage of fine-grained SQL execution tools in BridgeScope is their support for flexible security controls. Their benefits in execution efficiency will be discussed later in Section 3.3. Figure 5(b) compares the accuracy of agents on BIRD-Ext tasks when using BridgeScope's fine-grained SQL execution tools versus PG-MCP's single execute_sql tool. We observe that agents equipped with either BridgeScope or PG-MCP achieve comparable accuracy on these tasks. This result indicates that action-level modularization of SQL tools does not compromise task completeness, thus confirming its practical viability.

3) Transaction Management. The granularity of transaction management tools affects task control and ACID compliance, rather than efficiency or cost. Accordingly, we compare the ratio at which agents using BridgeScope and PG-MCP correctly initiate transactions on BIRD-Ext tasks. Figure 5(c) demonstrates that agents with BridgeScope's explicit transaction tools always initiate transactions correctly. The ratio falls slightly short of the theoretically best-achievable value of 1 due to cases where the LLM decides to abort a task before SQL execution. In contrast, agents using PG-MCP rarely recognize the need for transaction management. This underscores the importance of surfacing key functionalities from generic execution tools to enhance the LLM's awareness of executing necessary operations.

3.3 Effectiveness of Privilege-Aware Tooling

To evaluate how BridgeScope facilitates data-related tasks under different user privileges, we simulate three typical roles from production databases on the BIRD-Ext benchmark: **1) Administrator (A)**, with full privileges for data querying and manipulation; **2) Normal User (N)**, with read-only (SELECT) privileges; and **3) Irrelevant User (I)**, with privileges limited to accessing task-unrelated tables. Tasks are categorized as either **read** (involving only data querying) or **write** (requiring data manipulation). Figure 6 compares the average number of LLM calls required by BridgeScope and PG-MCP to complete or, if infeasible for insufficient privileges, abort a task. Each bar corresponds to a specific combination of user role and task type (*e.g.*, (A, read) for an administrator performing a read task). Results for (N, read) are omitted as they closely resemble those for (A, read). For reference, we also report the *best-achievable* lower bound on LLM calls: 3 calls for feasible tasks, as estimated in Section 3.2; 2 calls when a user accesses tables outside their privilege scope (one for schema retrieval and one for aborting the task); and 1 call when a read-only user attempts a write task (since only the select tool is available). Table 1 summarizes the corresponding token costs. We highlight two main observations:

Table 1: Token usage for BIRD-Ext.

Agent	Toolkit	Feasible tasks		Infeasible tasks		
		(A, read)	(A, write)	(N, write)	(I, read)	(I, write)
GPT-4o	BridgeScope	7,359	6,543	3,072	3,585	3,292
	PG-MCP	7,282	6,746	6,391	5,124	5,091
Claude-4	BridgeScope	10,102	14,652	2,194	4,515	4,632
	PG-MCP	9,803	15,111	12,109	6,496	7,093

Table 2: Effectiveness of the proxy mechanism.

Metric	Agent	BridgeScope	PG-MCP	PG-MCP-S
Task Completion Rate	GPT-4o	1.0	0.0	1.0
	Claude-4	1.0	0.0	1.0
Token Usage (On Average)	GPT-4o	13,449.7	-	21,047.6
	Claude-4	15,622.3	-	22,353.1
#LLM Calls (On Average)	GPT-4o	3.37	-	5.07
	Claude-4	3.40	-	5.07

1) When users have sufficient privileges, BridgeScope and PG-MCP incur similar numbers of LLM calls and token costs. This indicates that the privilege annotations and modular design of SQL tools in BridgeScope do not impose additional cognitive burdens on the LLM.

2) For infeasible cases where users lack necessary privileges, BridgeScope reduces LLM reasoning steps by 23% to 71%, with more pronounced improvements for Claude-4 due to its stronger reasoning capabilities. These results approach the best-achievable bound, with the smallest gap of only 10%. Token costs correspondingly decrease by 30% to 82%. These gains are attributed to BridgeScope's privilege annotations and modular SQL tools, which help the LLM better understand user privileges and abort unattainable tasks before any SQL execution. The advantage is most evident when read-only users attempt write tasks (*i.e.*, (N, write)). In this case, BridgeScope exposes only the select tool to such users, so the LLM can promptly recognize that data manipulation is impossible.

3.4 Effectiveness of Proxy

We evaluate the proxy mechanism on the NL2ML benchmark. To handle these tasks, agents are equipped with additional tools for data processing (*e.g.*, Z-score normalization) and for training and inference with machine learning models (*e.g.*, linear regression and random forest). Table 2 compares BridgeScope, PG-MCP, and a variant of PG-MCP across three metrics: task completion rate (the percentage of successfully completed tasks), token usage, and the number of LLM calls. We note the following:

1) PG-MCP fails to complete any NL2ML task because it relies on the LLM's limited context window to route data, which is quickly exhausted and causes task failures. In sharp contrast, BridgeScope's proxy mechanism enables direct data routing between tools and allows NL2ML tasks, even the most complex ones requiring three levels of proxy unit abstraction (see Section 3.1), to be completed with nearly the minimum possible number of three LLM calls (one each for context retrieval, proxy execution, and result finalization). This result demonstrates that modern LLMs can effectively orchestrate complex proxy units and underscores the practical effectiveness of the proxy mechanism.

2) Further examining PG-MCP-S, a trivial version of PG-MCP that operates on a reduced house table containing only 20 randomly sampled rows, we find that although it completes all NL2ML tasks, its token usage and number of LLM calls still exceed those of BridgeScope on the full table. This result reveals that PG-MCP is extremely sensitive to data size and remains viable only for very small data transmissions, whereas BridgeScope supports reliable task execution at any scale.

3) For reference, we also consider PG-MCP integrated into an idealized agent with unlimited reasoning capabilities and an unbounded context window, which can in principle handle all NL2ML tasks. However, the agent still requires at least two transfers of the house table to handle each task, which leads to no fewer than 1,500,000 token costs (750,000 per transfer). In contrast, BridgeScope reduces token usage by more than two orders of magnitude (13,449.7 versus $\geq$ 1,500,000), which could translate to substantial cost savings. This result shows that BridgeScope supports large-scale data transmission with stable computational costs. It therefore offers clear practical advantages for real-world data-related tasks.

4 Conclusions

This paper presents BridgeScope, a universal toolkit that bridges LLMs and diverse databases towards greater usability, efficiency, and security guarantees. Through fine-grained tool modularization, multi-level security controls, and a novel proxy-based data routing paradigm, BridgeScope overcomes longstanding barriers in agentic database interactions. Evaluations on two novel benchmarks confirm that LLM agents equipped with BridgeScope achieve substantially higher task completion rates and execution efficiency on data-related tasks. All components of BridgeScope are database-agnostic and transparent to both LLM agents and the MCP ecosystem, positioning BridgeScope as a cornerstone for the next generation of trustworthy, scalable, and intelligent data-related automation.

References

[1] Claude 4. 2025. https://www.anthropic.com/news/claude-4.
[2] ChatGPT Agent. 2025. https://openai.com/index/introducing-chatgpt-agent/.
[3] DeepSeek-R1. 2025. https://api-docs.deepseek.com/news/news250120.
[4] GPT-4o. 2025. https://openai.com/index/hello-gpt-4o/.
[5] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024. The Dawn of Natural Language to SQL: Are We Fully Ready? *PVLDB* 17, 11 (2024), 3318–3331.
[6] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *NeurIPS* 36 (2024).
[7] Peng Li, Yeye He, Dror Yashar, Weiwei Cui, Song Ge, Haidong Zhang, Danielle Rifinski Fainman, Dongmei Zhang, and Surajit Chaudhuri. 2023. Table-gpt: Table-tuned gpt for diverse table tasks. *arXiv:2310.09263* (2023).
[8] Yuyu Luo, Nan Tang, Guoliang Li, Chengliang Chai, Wenbo Li, and Xuedi Qin. 2021. Synthesizing natural language to visualization (NL2VIS) benchmarks from NL2SQL benchmarks. In *SIGMOD*. 1235–1247.
[9] Manus. 2025. https://manus.im/.
[10] Awesome MCP. 2025. https://github.com/punkpeye/awesome-mcp-servers.
[11] Cameron Nugent. 2018. California Housing Prices. https://www.kaggle.com/datasets/camnugent/california-housing-prices.
[12] Hacker News of Replit. 2025. https://news.ycombinator.com/item?id=44625119.
[13] Oracle Corporation. 2025. MySQL. https://www.mysql.com/.
[14] PostgreSQL. 2025. https://www.postgresql.org/.
[15] BridgeScope Repository. 2025. https://github.com/duoyw/bridgescope/.
[16] Introduction to MCP. 2025. https://modelcontextprotocol.io/introduction.
[17] Matthias Urban and Carsten Binnig. 2023. CAESURA: Language Models as Multi-Modal Query Planners. *arXiv:2308.03424* (2023).
[18] Hui Wei, Zihao Zhang, Shenghua He, Tian Xia, Shijia Pan, and Fei Liu. 2025. Plangenllms: A modern survey of llm planning capabilities. *arXiv:2502.11221* (2025).
[19] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *ICLR*.
[20] Xuanhe Zhou, Junxuan He, Wei Zhou, Haodong Chen, Zirui Tang, Haoyu Zhao, Xin Tong, Guoliang Li, Youmin Chen, Jun Zhou, et al. 2025. A Survey of LLM x DATA. *arXiv:2505.18458* (2025).