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 proliferating 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 resolve these challenges, we introduce BridgeScope, a universal toolkit bridging 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, enabling more precise and LLM-friendly functionality controls. Second, it aligns tool implementations with both database privileges and user security policies to steer LLMs away from unsafe or unauthorized operations, improving task execution efficiency while safeguarding database security. Third, it introduces a proxy mechanism for seamless inter-tool data transfer, bypassing LLM transmission bottlenecks. 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 operate databases more effectively, reduces token usage by up to 80% through improved security awareness, and uniquely supports data-intensive workflows beyond existing toolkits, establishing 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 [15], 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 using expert-crafted frameworks [15]. For instance, a typical LLM-based NL2SQL pipeline involves three subtasks, *i.e.*, schema linking, SQL generation, and SQL revision [2]. However, such decompositions are highly specific to task scenarios and lack generalizability to broader task classes. As the landscape of data-related tasks grows in both scale and complexity, relying solely on manually crafted frameworks becomes increasingly impractical.

Recently, advanced LLMs (*e.g.*, GPT-4o, DeepSeek R1, and Claude-4) have demonstrated impressive capabilities in multi-step reasoning and task planning [13]. In parallel, the introduction of the model context protocol (MCP) [11] has standardized LLM interactions with external systems, which greatly enhances LLMs' abilities through access to broad domain-specific tools. These advances are together ushering in a new era of *fully automated, general-purpose* agents

*: Equal Contribution, #: Corresponding Author

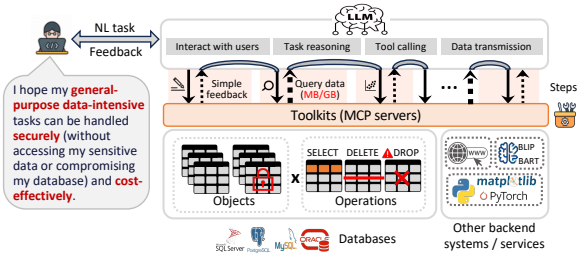


Figure 1: Architecture of a general agent.

capable of flexibly orchestrating and executing diverse, particularly data-related, tasks 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 backends). Towards this direction, both research [4, 12] and production advances like Manus agent [6] and ChatGPT agent [1] have achieved state-of-the-art breakthroughs.

Challenges. Current efforts in data-related scenarios focus on optimizing LLM-generated pipelines for complex tasks [4, 12]. 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 [7] are typically rudimentary, offering only a generic execute_sql tool for executing SQL statements and a get_schema tool for schema inspection. As analyzed below, such toolkits are inadequate for real-world data-related tasks, which in turn constrain the capabilities of agents and even expose the task-solving process 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, raising the risk of sensitive data exposure or unintentional destructive commands execution (*e.g.*, DROP DATABASE) due to LLM hallucinations or prompt injection attacks [9]. On the other hand, relying on a single tool for diverse operations, from querying data to managing transactions, greatly complicates LLMs' usage, increases their cognitive load, and heightens the risk of tool-selection errors.
- **C2. Privilege-Unaware Planning.** Current agents lack intrinsic awareness of users' privileges in the database, and thereby may generate pipelines exceeding authorized operations (*e.g.*, querying unauthorized tables). Although the database engine will eventually reject such operations, generating infeasible plans incurs non-negligible overhead for the LLM, especially when privilege violations occur late in the task execution workflow.

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

Our Contributions. In response to these challenges (C1–C3), we present BridgeScope, a systematic database toolkit that improves both the effectiveness and efficiency of general-purpose LLM agents on data-related tasks. *To the best of our knowledge, BridgeScope is the first dedicated toolkit design in this domain*, establishing a foundation for future prototype and production-grade agent development.

At a higher level, BridgeScope offers three core functionalities to support arbitrary data-related tasks: 1) *context retrieval*, which obtains task-related context (*e.g.*, database schema and column exemplars) as essential background for LLM's interactions; 2) *SQL execution*, which enables general-purpose CRUD operations (Create, Read, Update, and Delete); 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 toolkits, BridgeScope customizes tools aligning with both database-side user privileges and user-side security policies. Specifically, it includes only user-permitted objects with related privilege information in context retrieval results, and selectively exposes authorized, low-risk SQL execution tools (*e.g.*, just the select tool for read-only users). This keeps LLMs aware of their operational boundaries, confines task planning to authorized scopes, and enables early identification and termination of infeasible tasks (addressing Challenge C2). Additionally, BridgeScope employs rule-based verifications to strictly filter out unauthorized access and risky operations, providing an extra layer of protection. (see Sections 2.2–2.4 for details)

Beyond basic functionalities, BridgeScope encapsulates a *proxy* mechanism to support inter-tool data transfer (as per Challenge C3). With this approach, the LLM delegates data routing and execution management to a proxy tool, which retrieves necessary data and directly forwards it to downstream tools to trigger further operations, without any involvement from the LLM. This design not only mitigates the limitations of routing data through the LLM, but also allows seamless integration within the evolving MCP ecosystem, greatly expanding the agent's capabilities to 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 provide an open-source implementation of BridgeScope for PostgreSQL [10]. For rigorous evaluation, we introduce two novel benchmarks: one extends the NL2SQL benchmark BIRD [3] to include complex database modifications, and the other involves training machine learning models using data extracted from the database. Experimental results show that BridgeScope not only offers clear advantages in managing database operations such as transaction management, but also enhances LLM's security awareness, which contributes up to 80% less token costs by intercepting infeasible tasks. Furthermore, BridgeScope pioneers support for data-intensive workflows beyond the reach of existing toolkits. (see Section 3 for details)

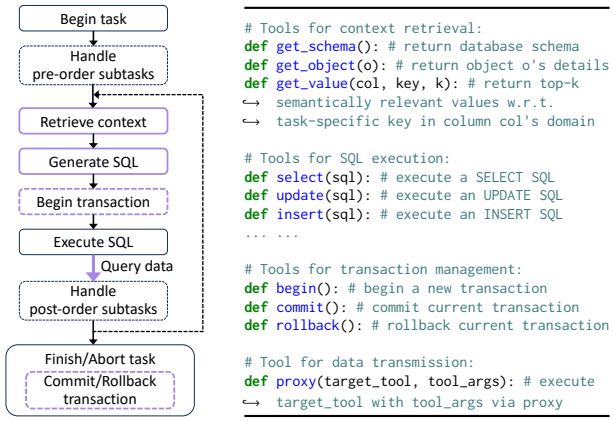


Figure 2: The abstracted workflow of data-related tasks (left) and fine-grained BridgeScope toolkit (right).

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 then proposed (Sections 2.2–2.5) to jointly address challenges C1–C3 (Section 1). Some remarks are provided in Section 2.6.

2.1 Functionalities and Design Principles

Functionalities Abstraction. We begin by examining key database operations in typical data-related workflows. As shown in Figure 2(left), upon task submission, the LLM first handles all pre-order subtasks not requiring database access (*e.g.*, user intent resolution). When data manipulation or querying is needed, it retrieves essential background information, such as the database schema, required to accurately formulate SQL statements (**F1. context retrieval**). It then generates and executes a valid, privilege-compliant SQL statement that fulfills the required goal (**F2. SQL execution**). For operations that modify the underlying database, the LLM must also initiate a transaction and execute these statements within the transaction context to ensure the ACID properties (**F3. transaction management**). After execution, feedback or resulting data are returned to the LLM and may be routed to follow-up tools (**F4. data transmission**) for subsequent processing or analysis [12]. If any step fails, all changes to the database are rolled back; otherwise, the workflow concludes with persisted database updates. Notably, one or more steps may iterate multiple times within a single task.

For instance, in a chain store scenario where a Brand A manager daily updates sales/refunds records and analyzes recent sales trends, an LLM agent can facilitate this through a structured workflow: 1) retrieving relevant metadata such as table schemas and column names for sales and refunds records (F1); 2) atomically inserting daily records into the database (F2 and F3); 3) querying recent sales and refunds data from the database (F2); and 4) passing the results to an advanced machine learning tool (F4) for trend detection.

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 adopting a generic execution tool, tools should be fine-grained for specific LLM-database

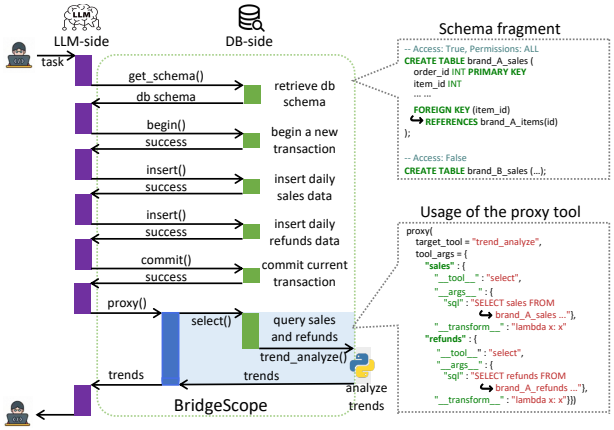


Figure 3: An illustrative example of how BridgeScope supports the chain store scenario.

interactions to enable more flexible functionality controls and lower LLM's cognitive load for tool selection, as per Challenge C1.

2) Robust Security Guarantees. Robust security must be enforced at dual levels: *1) (**database-side**)* Modern database systems like PostgreSQL and MySQL offer users granular privileges for actions (*e.g.*, SELECT, INSERT, UPDATE, DELETE, and *etc*) and objects (*e.g.*, tables, views, columns, and *etc*). Tools must ensure all operations adhere to these privileges; *2) (**user-side**)* Tools should support user-defined security policies to limit the LLM's privileges to a specific subset of the user's, thereby restricting its access to sensitive data and preventing destructive operations.

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

4) Transparency. Tools should operate transparently, neither interrupting nor requiring adaptations from other tools, thereby enabling seamless integration within the diverse and expanding MCP ecosystem to support a broad range of data-related tasks.

Guided by these principles, BridgeScope implements each core functionality (F1–F4) using a set of dedicated tools, each corresponding to a specific database operation, as shown in Figure 2(right). The rest of this section details the design of these tools and explains how they support the chain store task, as demonstrated 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. Schemas specify the structure and relationships among database objects (*e.g.*, tables, columns, indexes, and foreign-key dependencies), which are crucial for accurate SQL generation [2]. To support schema retrieval, BridgeScope adopts an adaptive strategy to accommodate varying database scales:

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 standardized, LLM-readable representations of all objects (see Figure 3). This enables the LLM to capture the complete database structure in a single tool call.

2) Otherwise, a hierarchical approach is adopted for more targeted and token-saving schema retrieval: get_schema() returns only 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) are obtained by issuing get_object(o) as needed.

BridgeScope employs two complementary mechanisms to steer LLM planning complying with the security requirements detailed in Section 2.1. First, database-side privileges are made explicit to the LLM by augmenting schema outputs with privilege annotations for each object. As illustrated in Figure 3, the annotations above the table schemas indicate that the store manager has full access to the `brand_A_sales` table, but is restricted from `brand_B_sales`. More granular privileges (*e.g.*, on specific columns) can be articulated similarly. Second, to limit LLM's access to sensitive data (*e.g.*, the employee salary table) within the user's privileges, BridgeScope allows object-level restrictions from the user side via configurable white- and black-lists. The schema retrieval tools then expose only user-permitted objects to the LLM. Together, these mechanisms provide the LLM with clear awareness of access boundaries, guiding it toward compliant planning and less hallucinated access.

Column Exemplars Retrieval. Due to synonyms, spelling variations, and domain-specific terms in database columns, LLMs often struggle to generate predicates aligned with actual data, leading to incomplete or inaccurate query results. By referencing column exemplars, LLMs can formulate semantically correct SQL predicates more accurately, *e.g.*, generating predicate category="women" rather than category="women's wear" by examining the category column when analyzing sales trends for women's clothes. To support this, BridgeScope provides a tool get_value(col, key, k), which retrieves the top-k most semantically relevant values to a task-specific key (*e.g.*, "women") within column col's domain. Notably, this targeted retrieval approach greatly reduces LLM context load compared to exhaustive enumeration of all 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 tools and selectively exposes them to the LLM based on both database-side privileges and user-side security configurations. Let the privilege set of user u be $\mathcal{P}_u \subseteq \mathcal{A} \times \mathcal{O}$, where $\mathcal{A}$ denotes the set of database actions (*e.g.*, SELECT, INSERT, UPDATE, DELETE, and *etc*) and $\mathcal{O}$ denotes the set of database objects. For each possible action a, BridgeScope instantiates a dedicated tool T_a to exclusively handle SQL statements acting a (*e.g.*, the select tool for executing only SELECT statements). The agent for user u is granted access to T_a only if $(a, o) \in \mathcal{P}_u$ for some objects o. For instance, a chain store sales assistant with read-only access to the sales data receives only the select tool; while the manager is granted full CRUD tools, including the insert tool for daily sales records update. As with object-level restrictions (Section 2.2), users can further restrict LLM's tool access using white- or black-lists, *e.g.*, by 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 its attempts to perform unauthorized or risky operations.

2) Object-Level Tool Verification. Although BridgeScope exposes permitted objects and privileges to the LLM via the get_schema

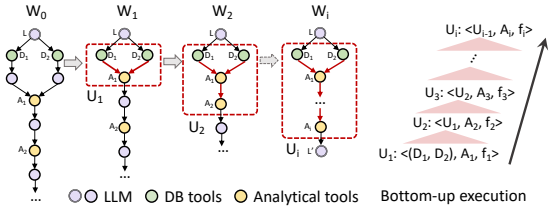


Figure 4: Illustration of proxy units.

tool, access violations can still occur due to LLM hallucinations or prompt injections. To guarantee safe access, BridgeScope enforces per-object verification during tool execution, ensuring that: 1) the user has privileges to operate the object; 2) the object complies with users' security policies. This tool-side check not only intercepts unauthorized operations, reducing the burden on the database, but also complements the database's native security mechanisms by additionally blocking dangerous actions specified by users.

2.4 Transaction Management

BridgeScope follows traditional database transaction controls and provides distinct begin, commit, and rollback tools to manage transactions. The ACID properties of each transaction (*i.e.*, between begin() and commit()) are guaranteed inherently by the database engine. As Figure 3 illustrates in the chain store scenario, the agent can invoke begin() to atomically insert both sales and refunds data, and call commit() to finalize the transaction upon success. Our experiments (Section 3.2) show that such explicit transaction tools substantially improve the LLM's ability to manage transactions, resulting in more robust and reliable workflows.

2.5 Data Transmission with Proxy

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

Foundations. The proxy mechanism operates on each *proxy unit* that specifies how data flows, formalized as a triple $\langle p, c, f \rangle$, where p denotes a (or a list of) data producer(s) to collect necessary 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 flow by allowing each proxy unit to act as a data producer for higher-level recursive proxy units. As Figure 4 illustrates, in a typical data-related workflow without the proxy, as exemplified by W_0, where the LLM intermediates between each 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 further forms $U_2 = \langle U_1, A_2, f_2 \rangle$. 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, the 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 necessary data, after which f_1 is applied and the transformed result is directly passed to the consumer A_1. The output of A_1 is then propagated upward in the proxy hierarchy to serve as input for the higher-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 transfer in the execution of proxy units occurs directly between tools, bypassing LLM bottlenecks as detailed in Challenge 3 (Section 1). Beyond this core advantage, the proxy mechanism brings further practical benefits.

1) Task Execution Efficiency. By eliminating LLM-mediated data transfer steps, the proxy mechanism significantly streamlines task execution and shortens the critical path (*e.g.*, condensing the workflow W_0 to the much simpler W_i in Figure 4). Moreover, it enables parallel execution of multiple data producers (*e.g.*, database tools D_1 and D_2), which further accelerates task completion.

2) Adaptability. By incorporating the function f for automatic data format adaptation, the proxy mechanism operates transparently *w.r.t.* downstream tools. As highlighted in Section 2.1, this is crucial for integration within the expansive MCP ecosystem and enables BridgeScope to be seamlessly used with any domain-specific MCP servers to support diverse task scenarios.

Implementations. BridgeScope implements the proxy mechanism via a proxy tool, complemented with a delicately crafted prompt that enables the LLM to recognize when to invoke it and autonomously generate proxy units as needed. The tool takes 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 invoked, the proxy unit executes and the output of target_tool is returned. This approach 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 for the chain store task. Here, target_tool is set to a trend_analyze tool, which analyzes sales trends via a machine learning model with lists of sales and refunds records (sales and refunds) as arguments. Each argument is produced by a select tool with an input sql querying recent sales or refunds data. As the results are already in the required format, an identity transformation is applied. Upon invocation, the proxy tool automates data selection, trend analysis, and inter-tool data transfer, and returns the sales trends produced by trend_analyze.

2.6 Remarks of BridgeScope

All tools in BridgeScope are built on a unified set of database interfaces (*e.g.*, executing a generic SQL) that can be implemented for any database system. This decouples tool logic from database specifics and makes migration to new databases straightforward. To showcase this, we release an open-source implementation for PostgreSQL [10]. This database-agnostic design enables LLMs to interact with any data source using a consistent set of tools, without handling system-specific variations, greatly enhancing their capabilities in multi-datasource scenarios.

Our implementation also includes a carefully crafted prompt that enables more efficient and ACID-compliant LLM-database interactions. 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 prompt achieves superior performance in supporting data-related tasks.

3 Experimental Evaluation

This section comprehensively evaluates BridgeScope to answer 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 costs (Section 3.3)? and 3) What benefits can the proxy mechanism bring (Section 3.4)? Notably, we tested scenarios involving privilege violations and operations exceeding users' security policies, all of which were successfully intercepted by BridgeScope's rule-based security controls. Therefore, we omit further security evaluation in the following results.

3.1 Experimental Setup

We implement two prototype general-purpose agents based on the ReAct [14] framework, with GPT-4o and Claude-4 as the underlying LLMs, respectively. Throughout the evaluation, we refer to the two agents by their underlying LLMs.

Baseline. We compare BridgeScope with a baseline adapted from the official MCP server for PostgreSQL [7], 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 simple to reflect the full functional requirements of real-world data-related tasks. For rigorous evaluations of BridgeScope, we synthesize the core requirements of these tasks and construct two novel benchmarks as below. Both benchmarks are made publicly available [10].

1) BIRD-Ext. We extend the state-of-the-art NL2SQL benchmark BIRD [3] originally focused on read-only SELECT SQL queries with complex data manipulation operations (*i.e.*, INSERT, UPDATE, and DELETE). For each type of operation, we choose 50 original SELECT tasks, modify the involved tables, and adapt the NL description to produce new tasks. The extended benchmark includes 150 randomly sampled original SELECT tasks and 150 newly synthesized tasks, which not only preserves the complexity and diversity of BIRD, but also adds stronger focuses on operation semantics, user privileges, and transaction management for data modifications.

2) NL2ML. This benchmark simulates end-to-end model training on the California Housing dataset [8], which has a single house table of 10 columns and 20,000 rows. It comprises 30 NL tasks at 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, corresponding to one, two, and three layers of proxy unit abstraction, respectively. For example, the second level involves routing data through extraction, processing, and model training steps. Unlike prior benchmarks [5] focused on small-scale tasks, this benchmark reflects practical needs for large-scale data transfer and complex workflow orchestration.

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) presents the average number of LLM calls they need to complete each task, which directly reflects the practical cost and efficiency. We observe that BridgeScope reduces LLM calls by over 30%, approaching the *best-achievable* value of 3 calls (one each for context retrieval, SQL execution, and result finalization). This gain arises because LLMs often hallucinate incorrect schema details and predicates with PG-MCP-S, causing futile retries. In contrast, BridgeScope's explicit context retrieval tools guide LLMs to gather necessary information beforehand, ensuring more reliable and efficient SQL generation.

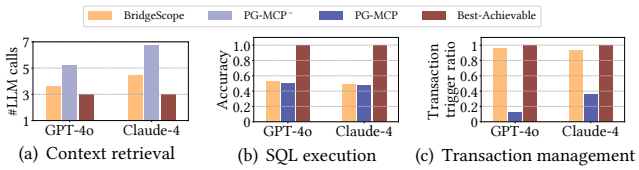


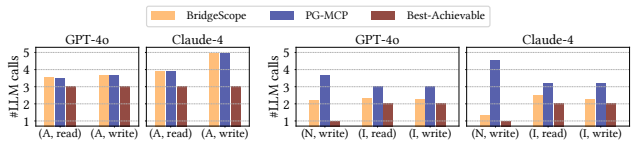
Figure 5: Performance *w.r.t.* tooling granularity.

2) SQL Execution. The main advantage of fine-grained SQL execution tools in BridgeScope is their support for flexible security controls, with its benefits in execution efficiency later discussed in Section 3.3. Figure 5(b) compares the accuracy of agents handling BIRD-Ext tasks with BridgeScope's fine-grained tools and PG-MCP's single execute_sql tool. We observe that agents equipped with either BridgeScope or PG-MCP achieve comparable accuracy on BIRD-Ext tasks. This indicates that the action-level modularization of SQL tools does not introduce unintended side effects on 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 with BridgeScope and PG-MCP correctly initiate transactions on BIRD-Ext tasks. Figure 5(c) demonstrates that agents with explicit transaction tools in BridgeScope always initiate transactions correctly, with minor gaps to the theoretically best-achievable ratio of 1 due to LLM's misjudgments to abort the tasks before SQL execution. In contrast, agents with 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 data query and manipulation privileges; **2) Normal User (N)**, with read-only (SELECT) privileges; and **3) Irrelevant User (I)**, with privileges limited to task-unrelated tables. Tasks are categorized as either **read**, involving only data queries, 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 the tasks. Each bar corresponds to a specific combination of user and task types (*e.g.*, (A, read) for an administrator performing a read task). Results for (N, read) are omitted as they closely resemble (A, read). For reference, we also report the *best-achievable* lower bound on LLM calls, as estimated in Section 3.2 for feasible tasks, or as the minimum



(a) Feasible tasks (b) Infeasible tasks

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

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

LLM steps needed to abort infeasible ones. Table 1 summarizes the corresponding token costs. We have two main observations:

1) When users have sufficient privileges, BridgeScope and PG-MCP incur similar numbers of LLM calls and token costs. This suggests that the privilege annotation and SQL tool modularization in BridgeScope do not add cognitive burden for the LLM.

2) For infeasible cases where users lack necessary privileges, BridgeScope reduces the number of LLM reasoning steps by 23%–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%. Correspondingly, token costs decrease by 30%–82%. These gains are attributed to BridgeScope's privilege annotation and SQL tool modularization, 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)). BridgeScope exposes only the select tool to these users, allowing the LLM to promptly recognize its inability to manipulate data.

3.4 Effectiveness of Proxy

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

1) PG-MCP struggles to complete any NL2ML task because it relies on 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 (each for context retrieval, proxy execution, and result finalization). This demonstrates that modern LLMs can abstract complex proxy units and highlights 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 with 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 reveals that PG-MCP is extremely sensitive to data size and is only viable for very small data transfers, whereas BridgeScope enables reliable task execution regardless of data size.

3) For reference, we also consider PG-MCP incorporated in an idealized agent with unlimited reasoning capabilities and context window, which can, in principle, handle all NL2ML tasks. However, the agent still incurs at least two transfers of the house table to handle each task, totaling no less than 1,500,000 tokens (750,000 per transfer). In contrast, BridgeScope reduces token usage by more than two orders of magnitude (13,449.7 vs. $\geq$ 1,500,000), which could translate to significant cost savings. This result indicates that BridgeScope can robustly handle large-scale data transfers with stable computational costs, highlighting its advantages for practical 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 also confirm that, with BridgeScope, LLM agents can handle data-related tasks with notably higher task completion rate and execution efficiency. All components in 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] ChatGPT Agent. 2025. https://openai.com/index/introducing-chatgpt-agent/.
[2] 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.
[3] 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).
[4] 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).
[5] 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.
[6] Manus. 2025. https://manus.im/.
[7] Awesome MCP. 2025. https://github.com/punkpeye/awesome-mcp-servers.
[8] Cameron Nugent. 2018. California Housing Prices. https://www.kaggle.com/datasets/camnugent/california-housing-prices.
[9] Hacker News of Replit. 2025. https://news.ycombinator.com/item?id=44625119.
[10] BridgeScope Repository. 2025. https://github.com/duoyw/bridgescope/.
[11] Introduction to MCP. 2025. https://modelcontextprotocol.io/introduction.
[12] Matthias Urban and Carsten Binnig. 2023. CAESURA: Language Models as Multi-Modal Query Planners. *arXiv:2308.03424* (2023).
[13] 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).
[14] 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*.
[15] 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).