DataLab: A Unified Platform for LLM-Powered Business Intelligence

Luoxuan Weng[†‡*], Yinghao Tang[†], Yingchaojie Feng[†], Zhuo Chang[§‡*], Peng Chen[‡], Ruiqin Chen[‡], Haozhe Feng[‡], Chen Hou[‡], Danqing Huang[‡], Yang Li[‡], Huaming Rao[‡], Haonan Wang[‡], Canshi Wei[‡], Xiaofeng Yang[‡], Yuhui Zhang[‡], Yifeng Zheng[‡], Xiuqi Huang[†], Minfeng Zhu[‖], Yuxin Ma[¶], Bin Cui[§], Wei Chen[†]

[†]*State Key Lab of CAD&CG, Zhejiang University*, [‡]*Tencent Inc.*, [‖]*Zhejiang University*,
[¶]*Southern University of Science and Technology*, [§]*School of Computer Science, Peking University*
[†]{lukeweng, yinghaotang, fycj, huangxiuqi, chenvis}@zju.edu.cn, [‡]{pengchen, ruiqinchen, aidenhzfeng, rickhou, daisyqhuang, thomasyngli, huamingrao, nanthanwang, caydenwei, felixxfyang, yukiyhzhang, yifengzheng}@tencent.com,
[‖]minfeng_zhu@zju.edu.cn, [¶]mayx@sustech.edu.cn, [§]{z.chang, bin.cui}@pku.edu.cn

***Abstract*—Business intelligence (BI) transforms large volumes of data within modern organizations into actionable insights for informed decision-making. Recently, large language model (LLM)-based agents have streamlined the BI workflow by automatically performing task planning, reasoning, and actions in executable environments based on natural language (NL) queries. However, existing approaches primarily focus on individual BI tasks such as NL2SQL and NL2VIS. The fragmentation of tasks across different data roles and tools lead to inefficiencies and potential errors due to the iterative and collaborative nature of BI. In this paper, we introduce DataLab, a *unified* BI platform that integrates a *one-stop* LLM-based agent framework with an *augmented* computational notebook interface. DataLab supports a wide range of BI tasks for different data roles by seamlessly combining LLM assistance with user customization within a *single* environment. To achieve this unification, we design a domain knowledge incorporation module tailored for enterprise-specific BI tasks, an inter-agent communication mechanism to facilitate information sharing across the BI workflow, and a cell-based context management strategy to enhance context utilization efficiency in BI notebooks. Extensive experiments demonstrate that DataLab achieves state-of-the-art performance on various BI tasks across popular research benchmarks. Moreover, DataLab maintains high effectiveness and efficiency on real-world datasets from Tencent, achieving up to a 58.58% increase in accuracy and a 61.65% reduction in token cost on enterprise-specific BI tasks.**

***Index Terms*—Business Intelligence, LLM, Data Analysis**

I. Introduction

Business intelligence (BI) aims to transform large volumes of data into actionable insights for informed decision-making [1]. A typical BI workflow includes multiple stages such as data preparation, analysis, and visualization. It requires the collaboration of data engineers, scientists, and analysts using various specialized tools (*e.g.*, Visual Studio Code, Power BI, Tableau), which can be highly tedious and time-consuming [2]. Therefore, modern organizations require advanced techniques to automate and optimize this workflow.

Recent advancements in autonomous agents powered by large language models (LLMs) [3] offer the potential to streamline the BI workflow (Figure 1). By receiving instructions in natural language (NL), LLM-based agents can perform task planning, reasoning, and actions in executable environments. This can significantly reduce the complexity of many BI tasks, such as code generation [4], text-to-visualization translation [5], and automated insight discovery [6].

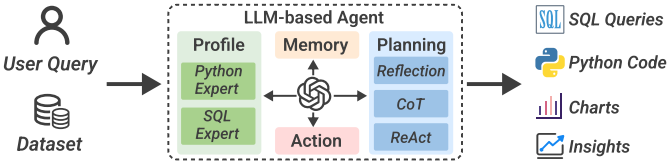

Fig. 1. General workflow of LLM-based agents for BI tasks.

However, previous works on LLM-based agents for BI primarily focus on individual tasks or stages without considering the BI workflow as a whole. The separation of BI tasks across different data roles and tools impedes seamless information flow and insight exchange, adding to communication costs, delays, and errors [7], [8]. For example, data analysts using GUI-based platforms (*e.g.*, Power BI) often rely on data engineers working with development tools (*e.g.*, PyCharm) to prepare data for analysis or visualization. This reliance necessitates back-and-forth communication between analysts and engineers due to the iterative and collaborative nature of BI [1]. Such procedures highlight the limitations of existing fragmented and fixed agent pipelines [9]. Consequently, this leads to a significant gap among different roles, tasks, and tools, which hinders timely and informed decision-making.

To bridge this gap, we aim to unify the BI workflow with a *one-stop* LLM-based agent framework in a *single* environment that satisfies the requirements of various data roles. However, achieving this unification in practical enterprise settings is non-trivial due to the following challenges:

C1: Lack of domain knowledge incorporation. Existing studies usually leverage clean and synthesized research benchmarks to build and evaluate agents [10]. However, BI tasks typically involve large and dirty real-world datasets with many ambiguities [11]. For example, column names in business data tables may have unclear semantic meanings [12], and user queries often include enterprise-specific jargon [10]. To miti-

* Work done during an internship at Tencent TEG Big Data Platform.

gate these issues, incorporating extensive domain knowledge is essential to enhance agents' understanding of input data and improve their performance on practical BI tasks. While some approaches adopt fine-tuning [13] or continued pre-training [14] to augment agents' domain-specific capabilities, acquiring the necessary large and up-to-date training data corpora remains challenging in BI scenarios due to their complexity and dynamic nature.

C2: Insufficient information sharing across tasks. Different tasks are typically managed by corresponding LLM-based agents to achieve optimal performance [15]. As a complex BI query may encompass multiple tasks, information sharing among the involved agents is critical. For example, the data retrieved by a *SQL writing agent* must be accurately conveyed to a *chart generation agent*. Therefore, an effective and efficient inter-agent communication mechanism is essential to align their understanding of the overall analysis goals, current data context, and executed actions. However, many existing multi-agent frameworks, such as ChatDev [16] and CAMEL [17], use unstructured natural language for communication. This can lead to distortions [18] and is inadequate for handling the complexity of BI tasks, which commonly involve diverse information types (e.g., data, charts, texts).

C3: Demand for adaptive LLM context management. LLM-based agents depend on their *context windows* (*i.e.*, limited input tokens for NL understanding, reasoning, and generation) to complete tasks. Necessary contexts must be provided to ensure a successful and seamless workflow. Meanwhile, in a unified BI platform, vast amounts of multi-modal contexts (*e.g.*, code snippets and their execution results, charts and their specifications) are intertwined and often relate to diverse data tables. Obviously, only relevant portions of these contexts are pertinent to specific tasks and should be selectively provided to the agents [19]. Therefore, adaptive context management tailored to prior states and current user needs is crucial for maintaining system efficiency and cost-effectiveness.

In this paper, we introduce DataLab, a *unified* environment that supports various data tasks throughout the BI workflow, thereby serving different data roles whether they use Markdown, SQL, Python, or no-code, all within a *single* computational notebook. We use notebooks as the foundational system due to their popularity in data science [7] and their iterative nature for the BI workflow [1]. DataLab adopts an LLM-based agent framework to integrate LLM assistance seamlessly, and a notebook interface to enable user customization flexibly.

To improve agents' performance on enterprise-specific BI tasks (for **C1**), we develop a *Domain Knowledge Incorporation* module, a systematic approach for automated knowledge generation, organization, and utilization. It leverages data processing scripts (*e.g.*, Python code, SQL queries) within the entire enterprise to extract the knowledge associated with databases/tables/columns, thereby uncovering their common usage patterns. To facilitate information sharing across different tasks (for **C2**), we design an *Inter-Agent Communication* module, a structured mechanism that goes beyond pure NL to enhance the information representation capabilities of agents. It also formulates the information sharing process among agents with a finite state machine (FSM) for a more controlled and efficient flow of communication. Finally, to manage LLM contexts within multi-modal notebooks (for **C3**), we propose a *Cell-based Context Management* module that represents inter-cell dependencies using directed acyclic graphs (DAGs). These dependency graphs are dynamically updated in response to user modifications. This enables the adaptive selection of pertinent contexts based on specific task requirements, thereby enhancing agents' context utilization efficiency.

In summary, the major contributions of our work are:

- We present DataLab, a platform that unifies the BI workflow with the integration of a one-stop LLM-based agent framework and a computational notebook interface, to bridge the gap among different roles, tasks, and tools.
- We develop a systematic approach for domain knowledge incorporation to enhance LLM-based agents' performance on enterprise-specific BI tasks in practical settings.
- We introduce a structured communication mechanism to formulate the information sharing process among different agents to facilitate their cross-task performance.
- We propose an adaptive context management strategy to improve agents' context utilization abilities within computational notebooks for efficiency and cost-effectiveness.
- We extensively evaluate DataLab on both research benchmarks and real-world business datasets from Tencent, demonstrating its performance on various BI tasks.

II. Background

In this section, we briefly introduce the core stages and roles in a modern BI workflow. We then provide an overview of LLM-based agents, focusing on their applications in BI tasks.

A. BI Workflow

The BI workflow encompasses several critical stages, namely data collection, storage, preparation, analysis, and visualization. **Data Collection** [20] refers to the initial step of gathering raw data from various sources like databases and spreadsheets. Once gathered, **Data Storage** [21] typically uses data warehouses or data lakes to organize the collected data for efficient retrieval. This may also involve combining data into a unified format (*i.e.*, data integration). Subsequently, **Data Preparation** [22] ensures data consistency, correctness, and quality. This usually includes cleaning, structuring, and enriching raw data into a format suitable for further analysis. Following preparation, **Data Analysis** [23] applies statistical and analytical techniques to extract insights, aiming to uncover patterns, trends, and correlations in the data. Finally, **Data Visualization** [5] presents analyzed data in visual formats like charts, graphs, and dashboards, which makes complex data easier to understand and interpret for decision-makers.

The data roles involved in the BI workflow are specific to different organizations. Among them, data engineers, scientists, and analysts are usually indispensable. **Data Engineers** are tasked with data collection, storage, and preparation, constructing and administering data pipelines to ensure that data

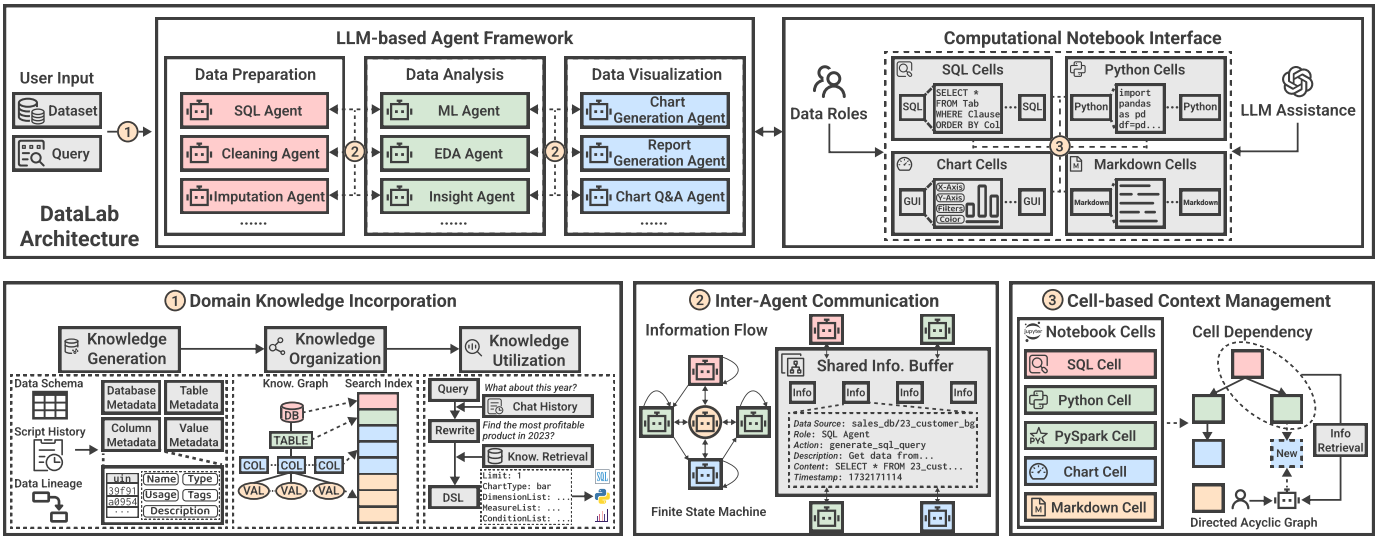

Fig. 2. Overview of DataLab and its three critical modules.

is accurately cleansed and structured for subsequent analysis. They typically use SQL and Python for data processing, and use cloud computing platforms like AWS for data storage and ingestion. **Data Scientists** engage in data preparation and analysis, applying advanced statistical and machine learning methodologies to extract insights and forecast trends from intricate datasets. They are familiar with Python/R and popular data science libraries like `Pandas`. **Data Analysts** concentrate on data analysis and visualization, analyzing data to discern patterns and conveying findings through detailed visual reports and dashboards. They use SQL to query data, and rely on BI platforms like Tableau to perform and share their analyses.

In modern enterprises, a complex BI workflow requires the collaboration of multiple data roles across various stages. The current fragmentation of tools for data preparation, analysis, and visualization introduces frictions and delays in timely decision-making. Therefore, an integrated and unified platform can serve as a shared environment for distinct user groups, facilitating the efficiency, transparency, and productivity of BI.

B. LLM-based Agents for BI

LLM-based agents are autonomous systems powered by LLMs that can perceive environments, execute tasks, make decisions, and interact with users in complex contexts [3]. These agents comprise profiling, memory, planning, and action modules, which respectively define the agent's role, facilitate operations in dynamic environments through recall and future action planning, and convert decisions into outputs [24]. In BI scenarios, agents receive users' NL queries and then perform data preparation, analysis, and visualization. By interpreting execution results, they can iteratively complete many BI tasks such as data transformation [25] and insight generation [6]. Moreover, equipped with vision language models (VLMs) [26], agents can even generate GUI operations for enterprise applications (*e.g.*, BigQuery, Airflow) [2], which further augments their capabilities for more complex BI workflows. Below, we list some typical tasks that can be streamlined by LLM-based agents at each BI stage:

- Data Collection: **Table Generation** [27], **Table Augmentation** [28], **Table Summarization** [29].
- Data Storage: **Data Warehousing** [2], **Data Integration** [30], **Data Orchestration** [2].
- Data Preparation: **NL2SQL** [31], **NL2DSCode** [4].
- Data Analysis: **NL2Insight** [32], **Table Q&A** [33].
- Data Visualization: **NL2VIS** [5], **Chart Q&A** [34].

However, most existing LLM-based agents are limited to individual tasks and do not meet the diverse user requirements of a complex BI workflow. Moreover, they often neglect the integration of enterprise-specific knowledge, resulting in unsatisfactory performance on proprietary business datasets. This lack of generalizability and customizability highlights the need for a structured and adaptive agent framework for BI.

III. Overview

Architecture Overview. As illustrated in Figure 2, DataLab consists of two primary components: (1) *LLM-based Agent Framework* and (2) *Computational Notebook Interface*.

- **LLM-based Agent Framework.** In DataLab, multiple agents are designed for different BI tasks based on user requirements. To achieve this, we first identify several common BI procedures and abstract them into *data tools* that can be called upon by agents during inference. Example tools include a Python sandbox for code execution and a Vega-Lite environment for visualization rendering. Accompanied by other auxiliary components like memory modules, each BI agent is represented as a DAG for high flexibility and easy extensibility. Within the DAG, nodes depict reusable components (*e.g.*, LLM APIs, tools) and edges depict their connections (*e.g.*, file transfer across tools). Figure 3 illustrates an example agent workflow for NL2VIS. Additionally, we add a *proxy agent* to the framework, which serves as

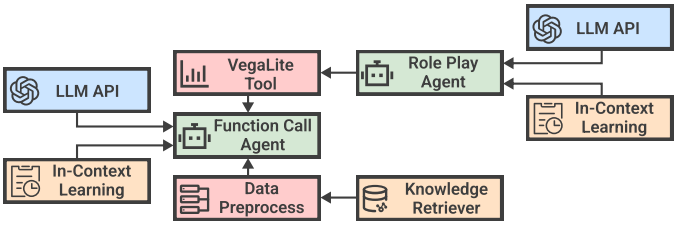

Fig. 3. Example agent workflow for NL2VIS.

a hub to directly interact with users and allocate tasks to each specialized agent based on user queries. These agents collaborate with each other to complete a wide array of tasks for various data roles throughout the BI workflow.

- **Computational Notebook Interface.** DataLab's notebook interface (Figure 4) serves as a unified, interactive, and collaborative environment for different data roles to complete their specialized tasks. To achieve this, we augment JupyterLab (a widely used notebook interface) to support (1) multi-language cells and (2) on-the-fly LLM assistance. First, DataLab wrangles SQL, Python/PySpark, Markdown, and Chart cells altogether, allowing both technical and non-technical users to easily adopt their familiar workflows on the notebook. Going beyond traditional notebooks that only support Python and Markdown, DataLab notebooks directly connect to backend databases for SQL query execution, and feature GUI-based dashboards [35] similar to Tableau for visualization authoring. Second, we integrate our LLM-based agent framework seamlessly into each notebook cell. Users can get LLM assistance both at notebook- and cell-level. Specifically, users toggle an input box and type their analytic queries, which are then processed by the agents in our framework. These agents can create new cells or modify existing ones in the notebook. Users can subsequently examine the results and make further customizations flexibly.

DataLab Workflow. Upon receiving an NL query and the associated dataset, DataLab initially analyzes the dataset and interprets the query, incorporating domain knowledge ① before feeding them into LLMs. Then, DataLab leverages various agents to complete the task, which may involve information sharing with each other through a structured communication mechanism ②. Subsequently, the corresponding result will be presented in the notebook. Users can either accept, edit, or reject the result and continues the BI workflow. Meanwhile, a context management strategy ③ automatically generates and maintains cell dependencies within the notebook to promote further agent calls. Next, we provide an overview of the three critical modules in DataLab.

- **Domain Knowledge Incorporation.** This module takes a data table's schema, its associated script history (*e.g.*, SQL queries, Python code), and its data lineage information as input. Specifically, the schema provides a basic overview of the table and its columns, including their names and types. The associated data processing scripts, which are created by professionals and executed every day within the

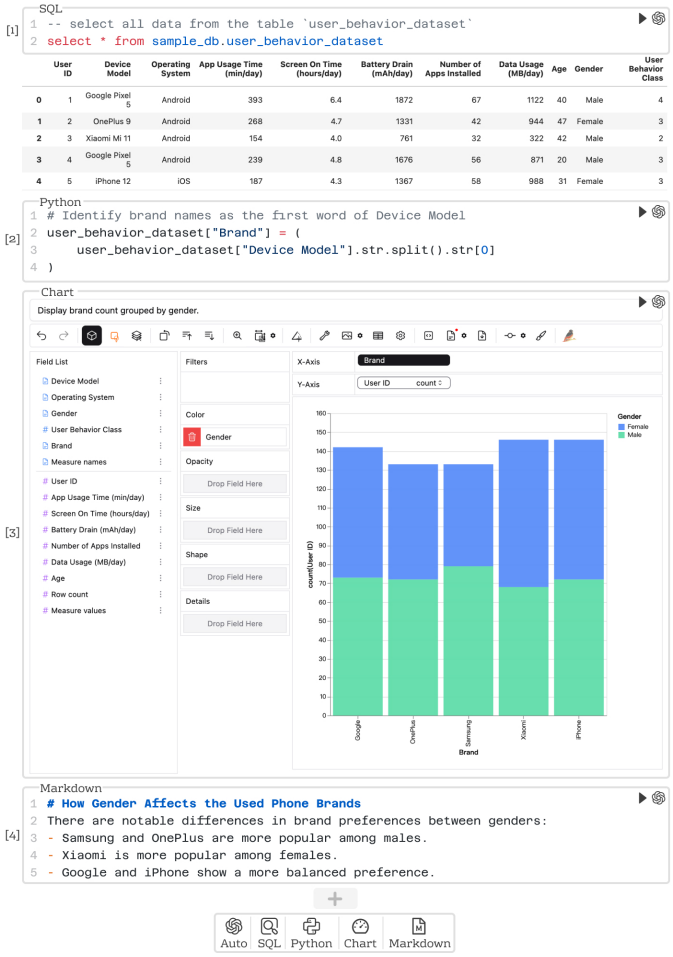

Fig. 4. The notebook interface of DataLab.

organization, reflect the semantic meanings and common usage patterns of the table and its columns. And the data lineage information [36], which reveals interrelationships among distinct tables and columns across the organization, can serve as an auxiliary resource for domain knowledge extraction. Based on the input, the module leverages LLMs to automatically generate the *knowledge components* (*e.g.*, descriptions, usages) of databases, tables, columns, and certain values. These knowledge components are then organized in a knowledge graph to facilitate further retrieval and utilization, which translate ambiguous user queries into structured domain-specific languages (DSLs) for improved agent performance on enterprise-specific BI tasks.

- **Inter-Agent Communication.** This module formulates the information flow process among different agents as an FSM to enable more control over their communications, with nodes representing agents and edges representing inter-agent information transition directions. Upon task completion, each agent's outputs are formatted into structured *information units* [18], comprising key characteristics such as the associated table's identifier and a concise description of the agent's executed actions. The module also maintains a shared information buffer for all agents to proactively

exchange information based on the FSM to improve communication efficiency.

- **Cell-based Context Management.** This module identifies cell dependencies within a notebook based on variable references and constructs a DAG, where nodes represent cells and edges denote their dependencies. Notably, data variables in Python or SQL cells are meticulously tracked, such as `DataFrames` and `SELECTs`. Given a user query, the module traverses the DAG to locate relevant cells, performs pruning based on task types, and retrieves information from the shared buffer. Then, the original cells and their corresponding information units are fed to the proxy agent as necessary contexts to facilitate task completion.

IV. Domain Knowledge Incorporation

In this section, we introduce DataLab's *Domain Knowledge Incorporation* module, which encompasses three stages, namely knowledge generation, organization, and utilization.

A. Knowledge Generation

Ambiguities are pervasive in real-world BI scenarios, manifesting both in the underlying databases and users' NL queries. For example, consider the query, *'show me the income of TencentBI this year'*, which involves three columns: *'prod_class4_name'*, *'shouldincome_after'*, and *'ftime'*. The semantic relationships between these column names and the user's request are often vague, leading to LLMs' suboptimal performance on such tasks. To mitigate these issues, existing approaches integrate table schema [5] into prompts and adopt retrieval-augmented generation (RAG) [10] to improve LLMs' domain-specific capabilities. We categorize three types of domain knowledge commonly utilized for BI tasks:

- **Metadata:** Information about data structure and attributes, such as table and column names, types, descriptions, and common usage patterns.
- **Business Logic:** Rules and processes that dictate how data is used and interpreted within the business.
- **Jargon:** Specialized terminologies and acronyms specific to the industry or organization.

Traditionally, such knowledge has been manually constructed by domain experts, which is tedious and time-consuming. Through an extensive examination conducted at Tencent, it was observed that, while 85% of the tables lack comprehensive metadata, they are predominantly linked to various SQL or Python scripts utilized for data processing. These scripts reveal common usage patterns within practical business contexts. Additionally, for those tables lacking adequate processing scripts, data lineage information, which elucidates their connections to other tables or columns throughout the organization, provides an alternative resource for metadata imputation. Therefore, inspired by LLMs' exceptional code understanding and reasoning abilities, we propose an LLM-based knowledge generation approach (Algorithm 1) that leverages script history to abstract and summarize knowledge components through meticulously-designed prompting techniques. This automated approach comprises a Map-Reduce process with a self-calibration mechanism [37] to generate high-quality knowledge for databases, tables, and columns.

Algorithm 1 LLM-based Knowledge Generation

Input: Schema $\mathcal{S}$, Script History $\mathcal{H}$,
Lineage Information $\mathcal{L}$, Score Threshold $\mathcal{T}$
Output: Database/Table/Column Knowledge $\mathcal{D}$, $\mathcal{T}$, $\mathcal{C}$

// Filter out duplicated or similar scripts
$\mathcal{H} \leftarrow preprocess(\mathcal{H})$
Map Phase:
$map\_res \leftarrow [\,]$
for each historical script $h_i \in \mathcal{H}$ **do**
  while $s_i < \mathcal{T}$ **do**
    // Generate knowledge individually
    $d_i, t_i, c_i \leftarrow \mathcal{LLM}(h_i, \mathcal{S}, \mathcal{L})$
    // Self-calibration
    $s_i \leftarrow \mathcal{LLM}(d_i, t_i, c_i)$
  end while
  $map\_res.append([d_i, t_i, c_i])$
end for
Reduce Phase:
// Synthesize knowledge collectively
$\mathcal{D}, \mathcal{T}, \mathcal{C} \leftarrow \mathcal{LLM}(map\_res, \mathcal{S}, \mathcal{L})$
return $\mathcal{D}, \mathcal{T}, \mathcal{C}$

Knowledge Components. Considering the previously defined knowledge categories, metadata and business logic can be deduced from data processing scripts, as both SQL queries and Python code support data manipulation operations like filtering and aggregation. Business logic is essential for computing *derived columns* which, though absent in the original table, hold significant value in business contexts. In contrast, jargon primarily exists in user queries or organization wikis (*i.e.*, documents), necessitating enterprise-specific glossaries for management and application. The *knowledge components* that our automated approach can generate are outlined below:

- **Database Level:** *description, usage, tags.*
- **Table Level:** *description, usage, organization, key column names, key derived attribute names, tags.*
- **Column Level:** *description, usage, type, tags, derived column information (name, description, usage, calculation logic, related columns, tags).*

These knowledge components are structured using JSON formats to improve LLMs' generation performance.

Map Phase. Given a data table, we take its schema $\mathcal{S}$, its script history $\mathcal{H}$, and its lineage information $\mathcal{L}$ as input. During the map phase, each distinct historical script h_i is individually processed using an LLM as the mapping model to produce corresponding knowledge components. The LLM is prompted to carefully analyze the script's semantic content and logical structure, aiming to extract critical information relevant to the specific business context. To mitigate LLMs' hallucination issues, focus is restricted to the involved databases, tables, and columns. This process results in the generation of a set of knowledge components associated with the script h_i.

Self-Calibration. Within each iteration of the map phase, we integrate a self-calibration mechanism that leverages LLMs' self-reflection abilities [38] to evaluate the intermediate results using a numerical score ranging from 1 to 5. Specifically, we instruct the LLM to consider multiple aspects of the knowledge components (*e.g.*, correctness, comprehensiveness, clarity) and provide several manually crafted in-context examples to demonstrate the scoring criteria. Should the rating score s_i fall below the predefined threshold $\mathcal{T}$, the knowledge generation process must be repeated. Therefore, this feedback loop ensures the generation quality of each iteration.

Reduce Phase. During the reduce phase, we aim to synthesize the individual results derived from each historical script to produce the final sets of knowledge components $\mathcal{D}$, $\mathcal{T}$, and $\mathcal{C}$ for the involved database, table, and columns, respectively. The LLM is instructed to meticulously scrutinize, aggregate, and summarize the information from all separate results to ensure a consistent and conflict-free collective result.

For each data table at Tencent, we execute the above Map-Reduce process to generate a comprehensive and high-quality set of knowledge components, which can significantly benefit many downstream BI tasks.

B. Knowledge Organization

We employ a knowledge graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ to systematically organize the knowledge generated by our automated approach (*i.e.*, metadata and business logic) and the manually crafted enterprise-specific glossaries (*i.e.*, jargon).

As depicted in Figure 5, the knowledge graph adopts a tree-based structure for knowledge organization. The nodes $\{\mathcal{V}\}$ are structured into five primary types: *database*, *table*, *column*, *value*, and *jargon*, each comprising various components (*e.g.*, *description*, *usage*) and uniquely identified by a *name*. To address the common challenge of terminological inconsistencies in user queries (*e.g.*, synonyms, acronyms), an additional node type, *alias*, has been introduced. This node type contains alternative terms associated with the official *name* of other node types and can be dynamically updated in real-world applications. The relationships between these nodes are represented by edges $\{\mathcal{E}\}$, which delineate both *logical relationships* among the primary node types and *associative relationships* between *alias* nodes and other primary nodes.

To facilitate efficient knowledge retrieval, we develop a task-aware indexing mechanism for graph nodes, utilizing Elasticsearch [39] for full-text search and StarRocks [40] for embedding search. This supports both lexical and semantic matching of knowledge nodes in response to user queries. The indexing structure is designed as triplets ($\{name, content, tag\}$), where the $content$ field is a concatenation of knowledge components specified based on the various requirements of downstream tasks. For instance, some tasks necessitate the *calculation logics* while others only need *descriptions* for successful completion. By dynamically selecting the appropriate index, we ensure that knowledge retrieval is both efficient and effective.

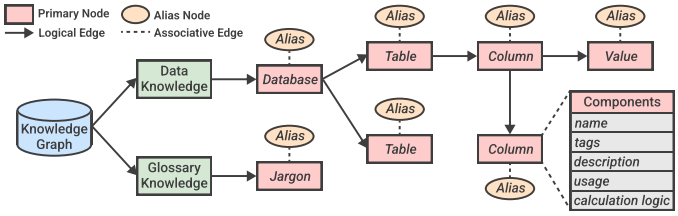

Fig. 5. Structure of the knowledge graph.

C. Knowledge Utilization

As shown in Figure 2①, given a user query $\mathcal{Q}$, we initially rewrite it to enhance clarity and detail. We then retrieve its relevant knowledge from the knowledge graph $\mathcal{G}$. Following this, the query is translated into a DSL specification, facilitating downstream tasks like NL2SQL and NL2VIS.

Query Rewrite. In addition to ambiguities, user queries can also be incomplete or underspecified, especially in multi-round interactions. For example, queries might omit prior context with phrases like *'what about'*. To ensure effective knowledge retrieval, the original query is enhanced and rewritten into a clearer and more detailed form, incorporating relevant prior information when available. Notably, temporal references (*e.g.*, *'last year'*) are also standardized based on the current time.

Algorithm 2 Knowledge Retrieval

Input: User Query $\mathcal{Q}$, Knowledge Graph $\mathcal{G}$
Output: Knowledge Nodes $\mathcal{V}_\mathcal{Q}$

$\mathcal{V}_\mathcal{Q} \leftarrow \emptyset$
Coarse-Grained Retrieval:
$\mathcal{V}_\mathcal{Q} \leftarrow lex\_search(\mathcal{Q}, \mathcal{G}) + sem\_search(\mathcal{Q}, \mathcal{G})$
Fine-Grained Ordering:
for each node $v_i \in \mathcal{V}_\mathcal{Q}$ **do**
  if $v_i.type == alias$ **then**
    $v_i \leftarrow backtrack(v_i)$ *// Backtrack to a primary node*
  end if
  // Compute a weighted matching score
  $score_i \leftarrow \omega_1 \cdot lex\_eval(\mathcal{Q}, v_i) + \omega_2 \cdot sem\_eval(\mathcal{Q}, v_i) + \omega_3 \cdot LLM\_eval(\mathcal{Q}, v_i)$
end for
$\mathcal{V}_\mathcal{Q}.sort(score_i)$ *// Rank by matching score*
return $\mathcal{V}_\mathcal{Q}.topK$

Knowledge Retrieval. To enhance LLMs' domain specific performance by integrating relevant knowledge into their context alongside the query, the selection and ordering of knowledge nodes from the knowledge graph are crucial. We employ a coarse-to-fine approach (Algorithm 2) to ensure comprehensive and precise knowledge retrieval.

- **Coarse-Grained Retrieval:** We perform lexical and semantic searches to retrieve a coarse set of knowledge nodes via token matching and embedding similarity between the query and each node's indexing triplet. We set a rather loose threshold to maximize recall. For *alias* nodes, we trace back to identify the nearest primary nodes (*i.e.*, database/table/column/value/jargon nodes).

- **Fine-Grained Ordering:** To prioritize the retrieved nodes, we implement a scoring mechanism that calculates a weighted matching score for each node, assessing its relevance to the query. This involves a three-stage evaluation: token-based (*i.e.*, lexicon), embedding-based (*i.e.*, semantics), and LLM-based (*i.e.*, overall relevance) [41]. Each stage yields a normalized score, with specific calculation methods and weights tailored to different BI tasks. The final set of knowledge nodes $\mathcal{V}_{\mathcal{Q}}$ is determined by sorting the initial node set according to these scores and selecting the top-K nodes, where K is set to a relatively large value to ensure a comprehensive coverage.

DSL Translation. The final step involves translating the query into a DSL specification, a common routine in BI scenarios [10]. This JSON structure specifies the relevant data and processing requirements, including fields such as *'MeasureList'* (*i.e.*, numerical columns), *'DimensionList'* (*i.e.*, categorical columns), and *'ConditionList'* (*i.e.*, filters). We prompt an LLM for DSL translation, providing detailed instructions and in-context examples to improve its performance. The generated DSL specification is validated using JSON Schema [42] to ensure syntactic and semantic correctness. This specification can then be directly converted to high-level languages like SQL and Vega-Lite based on predefined rules, or be used to enhance free-form code generation for complex tasks like NL2Insight, thereby facilitating LLMs' performance in practical business settings.

We also introduce a fallback strategy to address scenarios where relevant knowledge is scarce, especially for in-the-wild tables. Specifically, we develop a **Data Profiling** module that systematically extracts information from the table. This module consists of two stages: (1) heuristics-based analysis, which identifies and calculates each column's name, data type (*e.g.*, *float*, *string*), basic statistics (*e.g.*, *min*, *max*), and a random sample list, and (2) LLM-based interpretation, which feeds the extracted information to an LLM to generate a semantic description of each column and the overall table. Together, these stages produce a comprehensive summary of the table, aiding in the DSL translation process.

V. Inter-Agent Communication

In this section, we introduce DataLab's *Inter-Agent Communication* module, which facilitates efficient communication among multiple agents to complete complex BI tasks.

Workflow. As shown in Figure 6, upon receiving a user query, the proxy agent initiates an analysis to formulate an execution plan (defined by an FSM), which comprises multiple subtasks allocated to various agents (*Steps 1-2*). It then dynamically manages the communication among involved agents based on task progression by retrieving information from a shared buffer and forwarding it to the agents to support subtask execution (*Steps 5-6*). Upon completion of the subtasks, the proxy agent stores the agents' outputs in the buffer (*Steps 3-4*). Finally, once all subtasks are completed, the proxy agent generates a final answer and returns it to the user (*Step 7*).

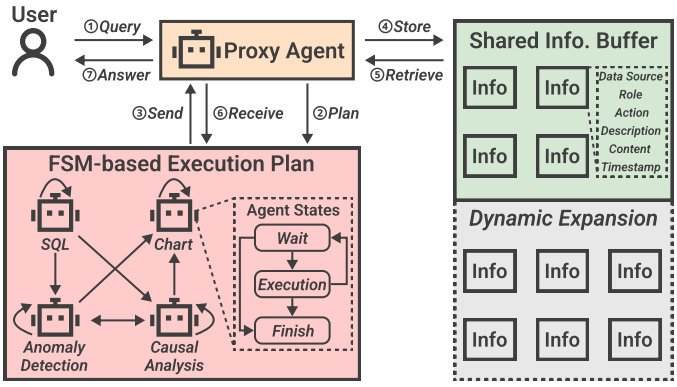

Fig. 6. Workflow of *Inter-Agent Communication*.

Information Format Structure. A critical consideration in multi-agent collaboration is *what 'language' agents use to communicate*. In BI scenarios, the information exchanged among agents is diverse, encompassing types such as SQL queries, Python code, and charts. This variety makes traditional multi-agent frameworks [16], [17] that rely on natural language for information sharing inadequate. To address this, we design a structured information format tailored to the unique characteristics of BI scenarios.

An *information unit* comprises six fields: *Data Source*, *Role*, *Action*, *Description*, *Content*, and *Timestamp*. *Data Source* identifies the dataset manipulated by the agent (*e.g.*, a table identifier). *Role* indicates the identity of the agent (*e.g.*, SQL Agent). *Action* specifies the agent's behavior (*e.g.*, `generate_sql_query`). *Description* provides a summary of the agent's executed actions for the task (*e.g.*, writing a SQL query to extract specific data from the table). *Content* details the agent's output (*e.g.*, a generated SQL query). *Timestamp* records the completion time of the task. To maintain consistent communication, all agents produce messages in this format.

Information Sharing Protocol. Another important question to consider is *how to ensure efficient information sharing among agents*. One extreme approach is that each agent receives information solely from its predecessor based on a plan chain [43]. While this minimizes data volume, it may cause the agent overlooking essential context (*e.g.*, background information pertinent to the task). Conversely, allowing any agent unrestricted access to the information of all others is inefficient, as irrelevant context can degrade LLMs' reasoning quality [44] and introduces additional computational overhead [45] during inference. Therefore, each agent should only receive the most relevant information for task completion.

To this end, we introduce two mechanisms to achieve this:

- **Shared Information Buffer** is a location for agents to store and retrieve information. Upon task completion, an agent deposits the produced information into this buffer via the proxy agent. This mechanism decouples information producers from consumers, thereby reducing synchronization overhead. Consequently, inter-agent communication becomes asynchronous and non-blocking,

allowing producers to continue processing without awaiting the retrieval of information by consumers, and vice versa. On the other hand, to manage the substantial information volumes often associated with multi-agent collaboration, we employ a dynamically growing buffering mechanism. Specifically, when the buffer reaches capacity, it automatically increases its size (*i.e.*, doubling the current capacity). Additionally, outdated information is periodically cleared. For example, if an agent's information is updated based on execution feedback, the original information is removed. This ensures that the buffer can maintain high performance while efficiently adapting to changing workloads.

- **Selective Retrieval** determines the information an agent receives from others. Inspired by the message-passing mechanism in TCP/IP [46], we design an FSM-based approach to implement this. Specifically, the proxy agent analyzes the user query and generates an FSM based on task requirements and each agent's abilities to orchestrate multi-agent information sharing, with nodes depicting agents and edges depicting information transition directions. Each agent operates within three states: *Wait, Execution,* and *Finish*. When an agent (*i.e.*, acting as a *'client'*) needs to execute a subtask, the proxy agent (*i.e.*, acting as a *'server'*) selects its relevant information from the shared buffer based on the FSM, and forwards it to the client agent. Upon receipt, the agent transitions from the *Wait* state to the *Execution* state, performs the necessary action, and produces structured information, which is then sent back to the proxy agent. The proxy agent, upon receiving the response, stores it in the buffer, and the client agent reverts to the *Wait* state. This loop continues until all subtasks are completed, at which point all agents transition to the *Finish* state and release their resources.

VI. Cell-based Context Management

In this section, we introduce DataLab's *Cell-based Context Management* module, which adaptively manages contexts in a notebook to ensure system efficiency and cost-effectiveness.

Thanks to the *Inter-Agent Communication* module (Section V), DataLab can handle complex BI tasks that require the collaboration of multiple agents. Specifically, user queries will trigger the presentation or modification of cells in the notebook, each corresponding to specific information units in the shared buffer. However, the previous module primarily aims at facilitating *individual* task completion for *single* data roles. In contrast, real-world BI scenarios often involve *multiple* data roles working on *different* tasks and collaborating within a unified notebook. This typically results in a multitude of multi-language cells (*e.g.*, SQL, Python, Chart) generated or altered by either agents or users themselves. When addressing a new user query, it is crucial to efficiently provide the proxy agent with the necessary contexts from the notebook. Simply supplying all cells and their associated information units is impractical due to inefficiency and high token costs. Therefore, we aim to identify the *minimum set* of relevant cells to minimize token usage without compromising agent performance. To achieve this, we model cell dependencies within the notebook as a DAG based on variable references, and propose an adaptive context retrieval mechanism.

Algorithm 3 DAG Construction

Input: Notebook Cells $\mathcal{C}$
Output: Dependency DAG $\mathcal{G}$

$v\_hash, cell\_refs \leftarrow \emptyset, \emptyset$
// Identify new variables in each cell
for each cell $c \in \mathcal{C}$ **do**
  if $c.type == Python$ **then**
    $ast \leftarrow construct\_ast(c)$
    $new\_v \leftarrow find\_global\_variables(ast)$
    $v\_hash[new\_v] \leftarrow c$
  else if $c.type == SQL$ **then**
    $data\_v \leftarrow find\_data\_variable(c)$
    $v\_hash[data\_v] \leftarrow c$
  end if
end for
// Find referenced cells for each cell
for each cell $c \in \mathcal{C}$ **do**
  $external\_v \leftarrow find\_external\_variables(c)$
  $cell\_refs[c] \leftarrow find\_ref\_cells(external\_v, v\_hash)$
end for
return $\mathcal{G} \leftarrow construct\_dag(cell\_refs)$

DAG Construction. As shown in Algorithm 3, given notebook cells $\mathcal{C}$, the DAG construction process includes two steps:

- **Identify new variables.** For Python cells, we construct an abstract syntax tree (AST) from the raw code to find *global* variables that are accessible across the entire notebook (*e.g.*, function/class definitions, package imports). We exclude local variables as they are only visible within their scope. For SQL cells, any `SELECT`'s output is stored in a data variable (*e.g.*, `DataFrame`) for future use, and thus represents a new variable. Conversely, Markdown and Chart cells do not produce variables that can be referenced elsewhere, and are therefore omitted. We store the variable-cell associations using a hash table.
- **Find referenced cells.** Based on the hash table, we locate each cell's referenced cells by identifying its *external* variables that are defined in other cells. For Python and SQL cells, this can be easily achieved with ASTs. For Chart cells, the underlying data variable serves as the reference point. As Markdown cells do not associate with any variables, they are excluded from this step. Consequently, a DAG of the notebook can be constructed using the extracted cell references.

The DAG keeps updating whenever a cell is created, modified, or deleted, provided that the changes pass the syntax check. This ensures real-time maintenance of cell dependencies.

Context Retrieval. Based on the cell dependency DAG and an input query, relevant cells are identified through graph traversal. This process supports queries at both *cell-level*

and *notebook-level*. For cell-level queries, the search is initiated within an existing cell, allowing for the straightforward identification of all ancestral cells via the DAG. Notebook-level queries, conversely, are formulated without specifying an existing cell, which typically rely on LLMs to automatically create new cells. In such cases, we first determine the related data variable either from explicit user input or through LLM prediction. Then, we locate the initial cell c_s where the data variable is defined. To ensure through coverage, all descendant cells of c_s are considered. Additionally, since Markdown cells lack references, our selection is guided by the textual similarity between cell content and the query. This process yields a comprehensive set of relevant cells $\mathcal{C}_r$ for each query.

Subsequently, $\mathcal{C}_r$ is pruned based on task types. Specifically, we employ LLMs to determine the task type contained in the query and identify the involved cell types. For example, in NL2DSCode tasks, only Python cells are considered. Accordingly, we filter out irrelevant cell types, resulting in a pruned set that constitutes the *minimum set* of relevant cells.

We then retrieve the associated information from the shared buffer for each relevant cell generated or altered by agents. The final necessary contexts for the query are determined by combining the retrieved information units with the original relevant cells, thereby providing a concise yet sufficient background for the proxy agent to understand and address the query.

VII. Experiment

In this section, we evaluate DataLab's performance using both public research benchmarks and proprietary business datasets from Tencent, including End-to-End Performance Comparison, Sensitivity Analysis, *Domain Knowledge Incorporation* Evaluation, *Inter-Agent Communication* Evaluation, and *Cell-based Context Management* Evaluation.

A. End-to-End Performance

To demonstrate the capabilities of DataLab as a unified BI platform, we first compare its end-to-end performance with SOTA LLM-based baselines on four typical BI tasks. We utilize **GPT-4** [59] as the foundation model for all methods.

1) Settings: The **NL2SQL** task converts natural language to SQL queries, typically marking the start of a BI workflow. We use two benchmarks (*i.e.*, Spider [47] and BIRD [50]) and compare with two baselines (*i.e.*, DAIL-SQL [48] and DIN-SQL [49]). We use the *Execution Accuracy (EX)* as the evaluation metric, which measures the execution equivalence of the generated SQL queries with the ground truth.

The **NL2DSCode** task converts natural language to data science code using Python libraries like `NumPy` and `Pandas`, which happens frequently throughout the BI workflow. We use two benchmarks (*i.e.*, DS-1000 [4] and DSEval [1] [53]) and compare with two baselines (*i.e.*, CoML [51] and Code Interpreter [52]). *Pass Rate* is used as the evaluation metric, which divides the number of passed problems by all problems.

The **NL2VIS** task converts natural language to data visualizations based on either Python libraries like `Matplotlib` or visualization grammars like `Vega-Lite`. We use two benchmarks (*i.e.*, nvBench [55] and VisEval [58]) and compare with two baselines (*i.e.*, LIDA [56] and Chat2Vis [57]) [2]. For nvBench, we use the *EX* metric for evaluation, which measures the equivalence of the generated visualizations with the ground truth based on the presented data values and chart types [60]. For VisEval, we use the *Pass Rate* metric to measure the ratio of valid or legal results divided by all queries, and the *Readability Score* judged by GPT-4V(ision) [61] to measure the overall quality of the generated visualizations [58].

The **NL2Insight** task converts analysis goals to data insights in an end-to-end manner, which requires LLMs' comprehensive problem-solving abilities. We use two benchmarks (*i.e.*, InfiAgent-DABench [54] and InsightBench [32]) and compare with two baselines (*i.e.*, AutoGen [15] and AgentPoirot [32]). For InfiAgent-DABench, we calculate the *Accuracy* of problems with correct answers to all problems. For InsightBench, we use the summary-level *LLaMA-3-Eval* and *ROUGE-1* scores as the evaluation metrics, which measure the alignment of the generated insights against the ground truth based on LLaMA-3 judgment and unigram overlap, respectively [32].

2) Results: As shown in Table I, DataLab achieves comparable performance or even surpass the SOTA LLM-based baselines (most of which only focus on a single task) on all four BI tasks. Specifically, DataLab outperforms all baselines on benchmarks including BIRD, DS-1000, DSEval, InsightBench, and VisEval, spreading over each critical BI stage. This demonstrates the superiority of DataLab in unifying the BI workflow with a single LLM-based agent framework.

For tasks that require the generation of symbolic languages (*e.g.*, NL2SQL, NL2DSCode, NL2VIS), DataLab consistently performs well primarily due to the intermediate DSL specifications generated by our *Domain Knowledge Incorporation* module. Although most research benchmarks lack the necessary information for extracting table/column knowledge, DataLab adopts a meticulously-designed data profiling strategy as an alternative (Section IV-C) to fully utilize the provided data schema, enabling LLMs to correctly identify and associate the semantic relationships between data columns and NL queries, which are crucial to generate high-quality DSLs. Consequently, compared to merely feeding the original pure NL queries, these intermediate DSLs can significantly improve LLM-based agents' performance on generating higher-level languages like SQL queries, Python code, or Vega-Lite specifications, as also shown in previous works [62].

For more complex tasks (*e.g.*, NL2Insight) that typically require multi-step reasoning and/or the collaboration of multiple agents, DataLab also achieves a satisfactory performance. Notably, it outperforms AutoGen, a popular multi-agent framework, by up to 5.06% on DABench and 19.35% on InsightBench. This performance gain can be attributed to two key factors: the agents' improved understanding of the involved datasets due to data profiling and the incorporation

[1] We only evaluate DSEval-LeetCode and -SO due to implementation issues.

[2] As the two baselines lack support for NL queries related to multiple data tables, we only evaluate on single-table queries for fair comparison.

TABLE I
PERFORMANCE OF DATALAB ON RESEARCH BENCHMARKS

BI Stage	Task	Benchmark	Metric	Method & Performance		
Data Preparation	NL2SQL	Spider [47]	Execution Accuracy	DataLab (Ours) 80.70 -2.54%	DAIL-SQL [48] **83.60**	DIN-SQL [49] 82.80
		BIRD [50]	Execution Accuracy	DataLab (Ours) **61.33** +9.71%	DAIL-SQL 57.41	DIN-SQL 55.90
	NL2DSCode	DS-1000 [4]	Pass Rate	DataLab (Ours) **53.80** +21.72%	CoML [51] 44.20	Code Interpreter [52] 51.60
		DSEval [53]	Pass Rate	DataLab (Ours) **80.99** +12.64%	CoML 71.90	Code Interpreter 80.58
Data Analysis	NL2Insight	DABench [54]	Accuracy	DataLab (Ours) 75.10 +5.06%	AutoGen [15] 71.48	AgentPoirot [32] **75.88**
		InsightBench [32]	LLaMA-3-Eval	DataLab (Ours) **0.37** +19.35%	AutoGen 0.31	AgentPoirot 0.35
			ROUGE-1	DataLab (Ours) 0.33 +17.86%	AutoGen 0.28	AgentPoirot **0.35**
Data Visualization	NL2VIS	nvBench [55]	Execution Accuracy	DataLab (Ours) 53.90 +0.13%	LIDA [56] **54.71**	Chat2Vis [57] 53.83
		VisEval [58]	Pass Rate	DataLab (Ours) **75.99** +5.67%	LIDA 74.66	Chat2Vis 71.91
			Readability Score	DataLab (Ours) 3.73 +0.81%	LIDA **3.77**	Chat2Vis 3.70

NOTE: Percentage gain is calculated relative to the weaker SOTA baseline marked with underlines.

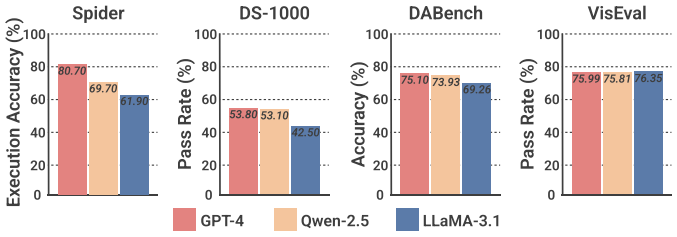

Fig. 7. Performance of DataLab using various underlying LLMs.

of our structured communication mechanism. This mechanism standardizes inter-agent information sharing, enabling a more comprehensive and thorough insight discovery process, especially when provided with high-level analytical objectives.

B. Sensitivity Analysis

To evaluate DataLab's robustness, we experiment with both closed- and open-source LLMs (*i.e.*, **GPT-4**, **Qwen-2.5** [63], and **LLaMA-3.1** [64]) on the above tasks using benchmarks including Spider, DS-1000, DABench, and VisEval.

As shown in Figure 7, DataLab consistently achieves satisfactory performance on all tasks, albeit with some sensitivity to the underlying LLMs. Proprietary models like GPT-4 typically exhibit superior instruction following and code generation abilities, surpassing open-source models like Qwen-2.5 and LLaMA-3.1. For code-intensive tasks like NL2DSCode and NL2Insight, LLaMA-3.1 experiences notable performance drops, especially on DS-1000, due to its relatively weaker code generation capabilities. We further evaluate DS-1000 using vanilla LLaMA-3.1 and achieve a pass rate of 36.90% (lower than 42.50% when integrated with DataLab). Another interesting fact is that, all three LLMs perform similarly on VisEval, with LLaMA-3.1 surprisingly being the best. These findings indicate that DataLab maintains a consistent performance across tasks, despite variations in LLMs, attributed to our data profiling and communication mechanisms. The data profiling mechanism enhances agents' understanding of input data, while the inter-agent communication module enables efficient error handling and iterative refinement, leading to overall performance improvements.

C. Effect of Domain Knowledge Incorporation

1) Knowledge Generation: As described in Section IV-A, this module aims to automatically generate knowledge components of data tables and columns. Deployed at Tencent for one month, **2,426** databases, **262,041** tables, and **2,708,884** columns (averaging 10.3 columns per table) have been successfully processed for knowledge generation, with an average time cost of 45.2 seconds per table. These statistics exhibit the practical application of our approach at a large enterprise.

To evaluate the quality of the generated knowledge, we collect a real-world dataset comprising 50 tables and 629 columns. Each table and column is annotated by domain experts for a ground truth of their semantic meanings. We then compare the *Sentence Embedding Similarity (SES)* between the generated descriptions and the ground truth using M3-Embedding [65], with 1 being identical and 0 being irrelevant. Results show that, the average *SES* scores are **0.712** (60% above 0.7) for tables and **0.677** (53% above 0.7) for columns, indicating the practical utility of the generated knowledge.

TABLE II
ABLATION STUDY ON DOMAIN KNOWLEDGE INCORPORATION

Task / Metric	S1	S2	S3
Schema Linking / Recall @5 (%)	41.02	71.79	**79.49**
NL2DSL / Accuracy (%)	32.52	61.66	**91.10**

TABLE III
ABLATION STUDY ON INTER-AGENT COMMUNICATION

Metric	S1	S2	S3
Success Rate (%)	73.00	85.00	**92.00**
Accuracy (%)	56.00	79.00	**84.00**

Overall, the real-world deployment and quality evaluation demonstrate the efficiency and effectiveness of the knowledge generation process of this module in practical settings.

2) Downstream Tasks: To assess the real-world impact of this module, we evaluate the following downstream tasks:

- The **Schema Linking** task seeks to select relevant tables and columns from the database schema based on NL queries, providing a basis for further analysis [66]. It requires LLMs to precisely capture the semantic relationships between user input and elements of the schema.
- The **NL2DSL** task converts NL queries to DSLs, which have been commonly adopted in commercial BI platforms and are crucial for many downstream tasks [10], [67]. In DataLab, DSLs are used as intermediates for generating SQL queries, Python code, and visualizations.

Due to common issues like ambiguities and jargon in real-world BI scenarios, both tasks require LLMs' deep understanding of domain knowledge. For Schema Linking, we collect a real-world dataset comprising 439 query-table-column pairs, and use *Recall @5* for evaluation. For NL2DSL, we compile another dataset comprising 326 query-DSL pairs, and measure the overall *Accuracy*. We then employ this module to generate knowledge for each involved table and column. For comparison, we design the following three experiment settings:

- **S1 (w/o knowledge)**: This setting provides NL queries along with a brief data schema generated by `Pandas`, but no additional knowledge, serving as a baseline. It is commonly adopted by most existing LLM-based agents.
- **S2 (w/ partial knowledge)**: Compared to S1, this setting additionally provides the generated *description*, *usage*, and *tags* of data tables and columns. It accounts for almost all successful cases in our practical deployment.
- **S3 (w/ all knowledge)**: Compared to S2, this setting further provides all generated knowledge of data tables and columns (see Section IV-A). It accounts for approximately 40% successful cases in our practical deployment.

As shown in Table II, DataLab's performance on both tasks improves significantly when provided with enterprise-specific knowledge. Specifically, the *Recall @5* of Schema Linking increases by **38.47%**, and the *Accuracy* of NL2DSL improves by up to **58.58%**. Even with only partial knowledge (S2), the performance still exhibits a significant increase of 30.77% for Schema Linking and 29.14% for NL2DSL compared to the baseline (S1). During our deployment at Tencent, we observe that many real-world business tables lack sufficient information required for generating comprehensive knowledge, often limited to table and column *descriptions*, *usage*, and *tags*. While understanding the semantic meanings of ambiguous table/column names can largely enhance LLMs' performance, the absence of other knowledge - especially *calculation logic* of derived columns for NL2DSL - can impede their capabilities in certain scenarios. This explains the performance difference between S2 and S3. Despite this, the promising results of S2 guarantee a minimum acceptable level of performance, demonstrating the module's effectiveness and robustness for downstream tasks in real-world BI scenarios.

D. Effect of Inter-Agent Communication

We experiment with a complex BI scenario that involves multiple tasks performed by distinct agents: NL2SQL, NL2DSCode, NL2VIS, Anomaly Detection, Causal Analysis, and Time Series Forecasting. We compile a dataset from practical settings at Tencent, consisting of 2 databases, 10 tables, and 111 columns. For each table, we meticulously design 10 complex questions derived from real-world business queries, totaling to 100 samples. Each question requires multi-step reasoning and multi-agent collaboration, ensuring a rigorous evaluation of our inter-agent communication mechanism.

We evaluate this module's efficiency and effectiveness by respectively calculating the *Success Rate* and *Accuracy* of the agents' responses across all questions. The *Success Rate* measures the ratio of questions that can be successfully solved within a maximum of 5 calls per agent, while the *Accuracy* measures the ratio of correct answers among all questions. For comparison, we employ three experiment settings:

- **S1 (w/o FSM)** [18]: This setting removes the FSM-based information sharing protocol. Therefore, each agent receives *all* information from the shared buffer.
- **S2 (w/o information formatting)** [15]: This setting removes the information format structure and adopts *pure natural language* for inter-agent communication.
- **S3 (w/ both)**: This setting keeps both techniques.

As illustrated in Table III, DataLab's performance on complex BI tasks improves by **19.00%** in *Success Rate* and **28.00%** in *Accuracy* with our inter-agent communication mechanism. Without the FSM-based information sharing protocol (S1), performance significantly degrades. Error analysis reveals that most failures involve more than 3 agents, resulting in overwhelming and irrelevant information that hinders LLMs' reasoning, thereby leading to incorrect outputs [19]. Additionally, the absence of the information format structure (S2) leads to a 7% decrease in *Success Rate* and a 5% drop in *Accuracy*, highlighting the importance of structured prompts in enhancing LLM comprehension and reducing information sharing ambiguities. This is critical in BI scenarios where

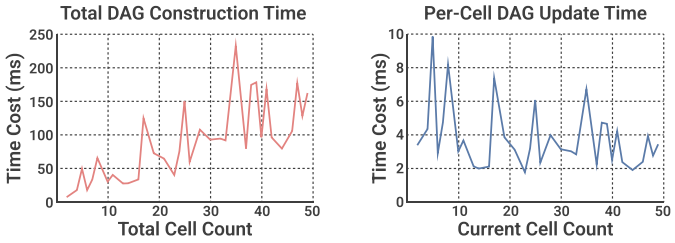

Fig. 8. Time cost of DAG construction.

complex tasks often require iterative error handling for data processing and structured summaries for lengthy outputs.

E. Effect of Cell-based Context Management

1) DAG Construction: To evaluate the efficiency of the DAG construction process, we collect 50 DataLab notebooks containing multi-language cells from practical settings, with cell counts ranging from 2 to 49. We measure the *Time Cost* of DAG construction both at notebook-level and cell-level. The initial construction encompasses all cells upon notebook opening, whereas subsequent updates generally involve a single cell. Our goal is to ensure a reasonable cold-start time while maintaining real-time responsiveness for subsequent updates.

As shown in Figure 8, DAG construction and updating maintain low time costs, at less than **250** and **10 milliseconds**, respectively. The total time for DAG construction increases with cell count, reaching a maximum of 232.22 milliseconds for 35 cells. In contrast, the per-cell time for DAG updating averages to 3.78 milliseconds, peaking at 9.84 milliseconds for 5 cells. Time costs are affected not only by cell count but also by lines of code, accounting for observed fluctuations. Given that a typical DataLab notebook contains fewer than 50 cells, these results demonstrate the efficiency of DAG construction.

2) Task Completion: For each notebook in our collected dataset, we derive 3 real-world user queries, which involve NL2SQL, NL2DSCode, and NL2VIS tasks, totaling to 150 samples. We evaluate this module's performance and cost-effectiveness using two metrics: *Accuracy* and *Token Cost per Query*. For comparison, we conduct an ablation study with two experiment settings: **S1 (w/o DAG)** and **S2 (w/ DAG)**.

As illustrated in Table IV, DataLab achieves a satisfactory *Accuracy* under both settings. Further analysis reveals that certain Markdown cells may contain critical information for task completion, which are occasionally failed to retrieve by our context retrieval mechanism due to limitations of embedding similarity [68]. This accounts for the slight 4.67% drop in *Accuracy* under S2. Conversely, S2 significantly reduces the *Token Cost per Query* by **61.65%** compared to S1. This is achieved by identifying the *minimum set* of relevant cells based on DAGs. These results demonstrate the cost-effectiveness of this module while maintaining acceptable performance.

TABLE IV
ABLATION STUDY ON CELL-BASED CONTEXT MANAGEMENT

Metric	S1	S2
Accuracy (%)	**86.67**	82.00
Token Cost per Query (K)	10.69	**4.10**

VIII. Related Work

Business Intelligence Platforms. BI platforms support users in analyzing business data for decision-making [8], [12]. Representatives like Tableau [69], Power BI [70], and Databricks [71] provide GUIs to support user interactions for data transformation and dashboard generation. These platforms also integrate natural language interfaces [72], [73] to lower the burden of manual operation. Quamar *et al.* [1] proposed an ontology-based method based on business models to provide semantic information and reasoning capability for query interpretation. The emergence of LLMs has further enhanced the domain knowledge integration and visualization generation abilities of BI platforms [74], [75]. Compared with these existing tools that mostly target data analysis scenarios, we go beyond to support more comprehensive stages (*e.g.*, data collection and preparation) and roles (*e.g.*, data engineers and scientists) in BI scenarios, which significantly reduces the cost of platform switch and requirement communication.

LLM-based Data Analysis. LLMs have demonstrated significant capabilities in semantic understanding and logical reasoning, facilitating complex data analysis tasks within conversational interfaces [6], [22]. Early empirical studies [76] investigate how data analysts interact with LLMs and identify challenges like contextual data retrieval, prompt refinement, and code adaptation. Table-GPT [77] fine-tunes LLMs on synthesized table-task data to enhance their table-understanding abilities. Inspired by Chain-of-Thought prompting [43], Chat2Query [23] decomposes the NL2SQL task into multiple steps to improve generation quality. Similarly, InsightPilot [78] automates the discovery of data insights and synthesizes them into high-level overviews. Chat2Data [79] leverages domain knowledge based on vector databases to mitigate LLMs' hallucination issues. While most existing works focus on individual tasks and generate results in an end-to-end manner, we introduce a unified platform for various data tasks in a computational notebook workspace. It supports user intervention of intermediate results, enabling more flexible human-LLM collaboration for data analysis in BI scenarios.

IX. Conclusion

In this paper, we present DataLab, a unified platform for business intelligence that combines an LLM-based agent framework with a computational notebook interface. DataLab features a domain knowledge incorporation module, an inter-agent communication mechanism, and a cell-based context management strategy. These components enable seamless integration of LLM assistance with user customization, making DataLab well-suited for practical BI scenarios. Extensive experiments on both research benchmarks and real-world business datasets demonstrate the effectiveness of DataLab.

REFERENCES

[1] A. Quamar, F. Özcan, D. Miller, R. J. Moore, R. Niehus, and J. T. Kreulen, "Conversational BI: an ontology-driven conversationsystem for business intelligence applications," *Proc. VLDB Endow.*, vol. 13, no. 12, pp. 3369–3381, 2020.

[2] R. Cao, F. Lei, H. Wu, J. Chen, Y. Fu, H. Gao, X. Xiong, H. Zhang, Y. Mao, W. Hu, T. Xie, H. Xu, D. Zhang, S. Wang, R. Sun, P. Yin, C. Xiong, A. Ni, Q. Liu, V. Zhong, L. Chen, K. Yu, and T. Yu, "Spider2-v: How far are multimodal agents from automating data science and engineering workflows?" *CoRR*, vol. abs/2407.10956, 2024.

[3] T. Xie, F. Zhou, Z. Cheng, P. Shi, L. Weng, Y. Liu, T. J. Hua, J. Zhao, Q. Liu, C. Liu, Z. Liu, Y. Xu, H. SU, D. Shin, C. Xiong, and T. Yu, "Openagents: An open platform for language agents in the wild," in *COLM*, 2024.

[4] Y. Lai, C. Li, Y. Wang, T. Zhang, R. Zhong, L. Zettlemoyer, W. Yih, D. Fried, S. I. Wang, and T. Yu, "DS-1000: A natural and reliable benchmark for data science code generation," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, 2023, pp. 18 319–18 345.

[5] Y. Wu, Y. Wan, H. Zhang, Y. Sui, W. Wei, W. Zhao, G. Xu, and H. Jin, "Automated data visualization from natural language via large language models: An exploratory study," *Proc. ACM Manag. Data*, vol. 2, no. 3, p. 115, 2024.

[6] L. Weng, X. Wang, J. Lu, Y. Feng, Y. Liu, and W. Chen, "Insightlens: Discovering and exploring insights from conversational contexts in large-language-model-powered data analysis," *CoRR*, vol. abs/2404.01644, 2024.

[7] R. Kosara, "Notebooks for data analysis and visualization: Moving beyond the data," *IEEE Computer Graphics and Applications*, vol. 43, no. 1, pp. 91–96, 2023.

[8] V. V. Meduri, A. Quamar, C. Lei, V. Efthymiou, and F. Ozcan, "BI-REC: guided data analysis for conversational business intelligence," *CoRR*, vol. abs/2105.00467, 2021.

[9] S. Hong, Y. Lin, B. Liu, B. Liu, B. Wu, D. Li, J. Chen, J. Zhang, J. Wang, L. Zhang, L. Zhang, M. Yang, M. Zhuge, T. Guo, T. Zhou, W. Tao, W. Wang, X. Tang, X. Lu, X. Zheng, X. Liang, Y. Fei, Y. Cheng, Z. Xu, and C. Wu, "Data interpreter: An LLM agent for data science," *CoRR*, vol. abs/2402.18679, 2024.

[10] A. Su, A. Wang, C. Ye, C. Zhou, G. Zhang, G. Zhu, H. Wang, H. Xu, H. Chen, H. Li, H. Lan, J. Tian, J. Yuan, J. Zhao, J. Zhou, K. Shou, L. Zha, L. Long, L. Li, P. Wu, Q. Zhang, Q. Huang, S. Yang, T. Zhang, W. Ye, W. Zhu, X. Hu, X. Gu, X. Sun, X. Li, Y. Yang, and Z. Xiao, "Tablegpt2: A large multimodal model with tabular data integration," *CoRR*, vol. abs/2411.02059, 2024.

[11] P. B. Chen, F. Wenz, Y. Zhang, M. Kayali, N. Tatbul, M. J. Cafarella, Ç. Demiralp, and M. Stonebraker, "BEAVER: an enterprise benchmark for text-to-sql," *CoRR*, vol. abs/2409.02038, 2024.

[12] J. Lian, X. Liu, Y. Shao, Y. Dong, M. Wang, Z. Wei, T. Wan, M. Dong, and H. Yan, "Chatbi: Towards natural language to complex business intelligence SQL," *CoRR*, vol. abs/2405.00527, 2024.

[13] M. R. J, K. VM, H. Warrier, and Y. Gupta, "Fine tuning LLM for enterprise: Practical guidelines and recommendations," *CoRR*, vol. abs/2404.10779, 2024.

[14] A. Ibrahim, B. Thérien, K. Gupta, M. L. Richter, Q. G. Anthony, E. Belilovsky, T. Lesort, and I. Rish, "Simple and scalable strategies to continually pre-train large language models," *Trans. Mach. Learn. Res.*, vol. 2024, 2024.

[15] Q. Wu, G. Bansal, J. Zhang, Y. Wu, S. Zhang, E. Zhu, B. Li, L. Jiang, X. Zhang, and C. Wang, "Autogen: Enabling next-gen LLM applications via multi-agent conversation framework," *CoRR*, vol. abs/2308.08155, 2023.

[16] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun, "Chatdev: Communicative agents for software development," in *ACL*. Association for Computational Linguistics, 2024, pp. 15 174–15 186.

[17] G. Li, H. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, "CAMEL: communicative agents for "mind" exploration of large language model society," in *NeurIPS*, 2023.

[18] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber, "Metagpt: Meta programming for A multi-agent collaborative framework," in *ICLR*. OpenReview.net, 2024.

[19] F. Shi, X. Chen, K. Misra, N. Scales, D. Dohan, E. H. Chi, N. Schärli, and D. Zhou, "Large language models can be easily distracted by irrelevant context," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 202. PMLR, 2023, pp. 31 210–31 227.

[20] S. E. Whang, Y. Roh, H. Song, and J. Lee, "Data collection and quality challenges in deep learning: a data-centric AI perspective," *VLDB J.*, vol. 32, no. 4, pp. 791–813, 2023.

[21] D. Kang, R. Jiang, and S. Blanas, "Jigsaw: A data storage and query processing engine for irregular table partitioning," in *SIGMOD Conference*. ACM, 2021, pp. 898–911.

[22] S. Chen, H. Liu, W. Jin, X. Sun, X. Feng, J. Fan, X. Du, and N. Tang, "Chatpipe: Orchestrating data preparation pipelines by optimizing human-chatgpt interactions," in *SIGMOD Conference Companion*. ACM, 2024, pp. 484–487.

[23] J. Zhu, P. Cai, B. Niu, Z. Ni, K. Xu, J. Huang, J. Wan, S. Ma, B. Wang, D. Zhang, L. Tang, and Q. Liu, "Chat2query: A zero-shot automatic exploratory data analysis system with large language models," in *ICDE*. IEEE, 2024, pp. 5429–5432.

[24] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, and J. Wen, "A survey on large language model based autonomous agents," *Frontiers Comput. Sci.*, vol. 18, no. 6, p. 186345, 2024.

[25] Z. Zhang, P. Groth, I. Calixto, and S. Schelter, "Directions towards efficient and automated data wrangling with large language models," in *ICDEW*. IEEE, 2024, pp. 301–304.

[26] J. Zhang, J. Huang, S. Jin, and S. Lu, "Vision-language models for vision tasks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 8, pp. 5625–5644, 2024.

[27] S. Arora, B. Yang, S. Eyuboglu, A. Narayan, A. Hojel, I. Trummer, and C. Ré, "Language models enable simple systems for generating structured views of heterogeneous data lakes," *Proc. VLDB Endow.*, vol. 17, no. 2, pp. 92–105, 2023.

[28] K. Chen and N. Koudas, "Unstructured data fusion for schema and data extraction," *Proc. ACM Manag. Data*, vol. 2, no. 3, p. 181, 2024.

[29] P. Li, Y. He, D. Yashar, W. Cui, S. Ge, H. Zhang, D. R. Fainman, D. Zhang, and S. Chaudhuri, "Table-gpt: Table fine-tuned GPT for diverse table tasks," *Proc. ACM Manag. Data*, vol. 2, no. 3, p. 176, 2024.

[30] Z. Chen, Z. Gu, L. Cao, J. Fan, S. Madden, and N. Tang, "Symphony: Towards natural language query answering over multi-modal data lakes," in *CIDR*. www.cidrdb.org, 2023.

[31] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-sql empowered by large language models: A benchmark evaluation," *Proc. VLDB Endow.*, vol. 17, no. 5, pp. 1132–1145, 2024.

[32] G. Sahu, A. Puri, J. A. Rodriguez, A. Drouin, P. Taslakian, V. Zantedeschi, A. Lacoste, D. Vázquez, N. Chapados, C. Pal, S. Rajeswar, and I. H. Laradji, "Insightbench: Evaluating business analytics agents through multi-step insight generation," *CoRR*, vol. abs/2407.06423, 2024.

[33] R. Mouravieff, B. Piwowarski, and S. Lamprier, "Learning relational decomposition of queries for question answering from tables," in *ACL*. Association for Computational Linguistics, 2024, pp. 10 471–10 485.

[34] Y. Han, C. Zhang, X. Chen, X. Yang, Z. Wang, G. Yu, B. Fu, and H. Zhang, "Chartllama: A multimodal LLM for chart understanding and generation," *CoRR*, vol. abs/2311.16483, 2023.

[35] Y. Yu, L. Shen, F. Long, H. Qu, and H. Chen, "Pygwalker: On-the-fly assistant for exploratory visual data analysis," *CoRR*, vol. abs/2406.11637, 2024.

[36] M. Tang, S. Shao, W. Yang, Y. Liang, Y. Yu, B. Saha, and D. Hyun, "SAC: A system for big data lineage tracking," in *ICDE*. IEEE, 2019, pp. 1964–1967.

[37] K. Tian, E. Mitchell, A. Zhou, A. Sharma, R. Rafailov, H. Yao, C. Finn, and C. D. Manning, "Just ask for calibration: Strategies for eliciting calibrated confidence scores from language models fine-tuned with human feedback," in *EMNLP*. Association for Computational Linguistics, 2023, pp. 5433–5442.

[38] Z. Ji, T. Yu, Y. Xu, N. Lee, E. Ishii, and P. Fung, "Towards mitigating LLM hallucination via self reflection," in *EMNLP (Findings)*. Association for Computational Linguistics, 2023, pp. 1827–1843.

[39] C. Gormley and Z. Tong, *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*. " O'Reilly Media, Inc.", 2015.

[40] StarRocks, "Starrocks: A high-performance analytical database," https://www.starrocks.io/, 2024.

[41] D. C. Chiang and H. Lee, "Can large language models be an alternative to human evaluations?" in *ACL*. Association for Computational Linguistics, 2023, pp. 15 607–15 631.

[42] F. Pezoa, J. L. Reutter, F. Suárez, M. Ugarte, and D. Vrgoc, "Foundations of JSON schema," in *WWW*. ACM, 2016, pp. 263–273.

[43] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *NeurIPS*, 2022.

[44] Z. Xi, S. Jin, Y. Zhou, R. Zheng, S. Gao, J. Liu, T. Gui, Q. Zhang, and X. Huang, "Self-polish: Enhance reasoning in large language models via problem refinement," in *EMNLP (Findings)*. Association for Computational Linguistics, 2023, pp. 11 383–11 406.

[45] D. Li, Y. Ma, N. Wang, Z. Ye, Z. Cheng, Y. Tang, Y. Zhang, L. Duan, J. Zuo, C. Yang, and M. Tang, "Mixlora: Enhancing large language models fine-tuning with lora-based mixture of experts," 2024. [Online]. Available: https://arxiv.org/abs/2404.15159

[46] K. R. Fall and W. R. Stevens, *Tcp/ip illustrated*. Addison-Wesley Professional, 2012, vol. 1.

[47] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. R. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," in *EMNLP*. Association for Computational Linguistics, 2018, pp. 3911–3921.

[48] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-sql empowered by large language models: A benchmark evaluation," *Proc. VLDB Endow.*, vol. 17, no. 5, pp. 1132–1145, 2024.

[49] M. Pourreza and D. Rafiei, "DIN-SQL: decomposed in-context learning of text-to-sql with self-correction," in *NeurIPS*, 2023.

[50] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. C. Chang, F. Huang, R. Cheng, and Y. Li, "Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls," in *NeurIPS*, 2023.

[51] L. Zhang, Y. Zhang, K. Ren, D. Li, and Y. Yang, "Mlcopilot: Unleashing the power of large language models in solving machine learning tasks," in *EACL*. Association for Computational Linguistics, 2024, pp. 2931–2959.

[52] shroominic, "Open source implementation of the chatgpt code interpreter," https://github.com/shroominic/codeinterpreter-api, [Accessed 18-10-2024].

[53] Y. Zhang, Q. Jiang, X. XingyuHan, N. Chen, Y. Yang, and K. Ren, "Benchmarking data science agents," in *ACL*. Association for Computational Linguistics, 2024, pp. 5677–5700.

[54] X. Hu, Z. Zhao, S. Wei, Z. Chai, Q. Ma, G. Wang, X. Wang, J. Su, J. Xu, M. Zhu, Y. Cheng, J. Yuan, J. Li, K. Kuang, Y. Yang, H. Yang, and F. Wu, "Infiagent-dabench: Evaluating agents on data analysis tasks," in *ICML*. OpenReview.net, 2024.

[55] Y. Luo, N. Tang, G. Li, C. Chai, W. Li, and X. Qin, "Synthesizing natural language to visualization (NL2VIS) benchmarks from NL2SQL benchmarks," in *SIGMOD Conference*. ACM, 2021, pp. 1235–1247.

[56] V. Dibia, "LIDA: A tool for automatic generation of grammar-agnostic visualizations and infographics using large language models," in *ACL (demo)*. Association for Computational Linguistics, 2023, pp. 113–126.

[57] P. Maddigan and T. Susnjak, "Chat2vis: Generating data visualizations via natural language using chatgpt, codex and GPT-3 large language models," *IEEE Access*, vol. 11, pp. 45 181–45 193, 2023.

[58] N. Chen, Y. Zhang, J. Xu, K. Ren, and Y. Yang, "Viseval: A benchmark for data visualization in the era of large language models," *IEEE Transactions on Visualization and Computer Graphics*, 2024.

[59] OpenAI, "GPT-4 technical report," *CoRR*, vol. abs/2303.08774, 2023.

[60] G. Li, X. Wang, G. Aodeng, S. Zheng, Y. Zhang, C. Ou, S. Wang, and C. H. Liu, "Visualization generation with large language models: An evaluation," *CoRR*, vol. abs/2401.11255, 2024.

[61] Z. Yang, L. Li, K. Lin, J. Wang, C. Lin, Z. Liu, and L. Wang, "The dawn of lmms: Preliminary explorations with gpt-4v(ision)," *CoRR*, vol. abs/2309.17421, 2023.

[62] B. Wang, Z. Wang, X. Wang, Y. Cao, R. A. Saurous, and Y. Kim, "Grammar prompting for domain-specific language generation with large language models," in *NeurIPS*, 2023.

[63] Q. Team, "Qwen2.5: A party of foundation models," September 2024. [Online]. Available: https://qwenlm.github.io/blog/qwen2.5/

[64] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Rozière, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. M. Kloumann, I. Misra, I. Evtimov, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, and et al., "The llama 3 herd of models," *CoRR*, vol. abs/2407.21783, 2024.

[65] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, "M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation," in *ACL (Findings)*. Association for Computational Linguistics, 2024, pp. 2318–2335.

[66] W. Lei, W. Wang, Z. Ma, T. Gan, W. Lu, M. Kan, and T. Chua, "Re-examining the role of schema linking in text-to-sql," in *EMNLP*. Association for Computational Linguistics, 2020, pp. 6943–6954.

[67] A. Popovic, I. Lukovic, V. Dimitrieski, and V. Djukic, "A DSL for modeling application-specific functionalities of business applications," *Comput. Lang. Syst. Struct.*, vol. 43, pp. 69–95, 2015.

[68] H. Steck, C. Ekanadham, and N. Kallus, "Is cosine-similarity of embeddings really about similarity?" in *WWW (Companion Volume)*. ACM, 2024, pp. 887–890.

[69] Tableau, "Tableau einstein," https://www.tableau.com/, 2024.

[70] Microsoft, "Power bi," https://www.microsoft.com/en-us/power-platform/products/power-bi, 2024.

[71] Databricks, "Databricks data intelligence platform," https://www.databricks.com/, 2024.

[72] M. Tory and V. Setlur, "Do what I mean, not what I say! design considerations for supporting intent and context in analytical conversation," in *VAST*. IEEE, 2019, pp. 93–103.

[73] Y. Feng, X. Wang, B. Pan, K. Wong, Y. Ren, S. Liu, Z. Yan, Y. Ma, H. Qu, and W. Chen, "XNLI: explaining and diagnosing nli-based visual data analysis," *IEEE Trans. Vis. Comput. Graph.*, vol. 30, no. 7, pp. 3813–3827, 2024.

[74] X. Miao, Z. Jia, and B. Cui, "Demystifying data management for large language models," in *SIGMOD Conference Companion*. ACM, 2024, pp. 547–555.

[75] X. Zhou, X. Zhao, and G. Li, "Llm-enhanced data management," *CoRR*, vol. abs/2402.02643, 2024.

[76] B. Chopra, A. Singha, A. Fariha, S. Gulwani, C. Parnin, A. Tiwari, and A. Z. Henley, "Conversational challenges in ai-powered data science: Obstacles, needs, and design opportunities," *CoRR*, vol. abs/2310.16164, 2023.

[77] P. Li, Y. He, D. Yashar, W. Cui, S. Ge, H. Zhang, D. R. Fainman, D. Zhang, and S. Chaudhuri, "Table-gpt: Table fine-tuned GPT for diverse table tasks," *Proc. ACM Manag. Data*, vol. 2, no. 3, p. 176, 2024.

[78] P. Ma, R. Ding, S. Wang, S. Han, and D. Zhang, "Insightpilot: An llm-empowered automated data exploration system," in *EMNLP (Demos)*. Association for Computational Linguistics, 2023, pp. 346–352.

[79] X. Zhao, X. Zhou, and G. Li, "Chat2data: An interactive data analysis system with rag, vector databases and llms," *Proc. VLDB Endow.*, vol. 17, no. 12, pp. 4481–4484, 2024.