

From SQL to Knowledge Graphs: An LLM-Driven Multi-Agent Approach with Data Schema Improvement

Dinh-Khanh Pham
FPT Software, Vietnam

Quy-Anh Dang
Knovel Engineering, Singapore

Lam Mai Thanh
HCMC University of Technology and Education, Vietnam

Khanh Bui
University of Alberta, Canada

Truong-Son Hy
University of Alabama at Birmingham, United States

Abstract—RDBMS (Relational Database Management System) databases face several limitations, including slow execution with multi-hop queries and a lack of explainability by graphical interpretations. In contrast, Graph database offers a more intuitive and efficient data schema that performs faster execution on large datasets. Most existing RDBMS conversion pipelines focus on running traditional loading commands and relying on Cypher queries. However, the efficiency of using an LLM to generate an effective graph data schema, significantly reducing the ambiguity of the graph database, remains underexplored in the current research literature. This paper presents a novel algorithm that bridges RDBMS and graph database by using a novel LLM-powered ETL agent to standardize table and column names before saving them to the Data Mart. A Multi-Agent System generates a looping discussion between ETL, Analyzer, and Graph agents to optimize the final design through an iterative process of suggesting and scoring the graph database schema. We ensure that the final graph database meets three criteria before being accepted for data conversion: Accuracy, Groundedness, and Faithfulness. This system demonstrates an effective pipeline to automatically convert a tabular database into a graph database through a comprehensive end-to-end process. Our study highlights notable efficiency in using the converted graph database, which is measured on 1,081 samples of the BFSI dataset across three levels of complexity (easy, medium, and hard). Specifically, CypherAgent achieves an 85.6% accuracy for Q&A tasks using a Graph database, which is 12.12% higher than the accuracy achieved by an SQLAgent on the RDBMS database type PostgreSQL, for all queries. Additionally, the Graph database demonstrates faster performance, reducing latency by approximately 3 times. For easy, medium, and hard queries, the Graph database attains accuracies of 90.43%, 81.98%, and 80.06%, respectively, surpassing the RDBMS database by 17.8%, 4.2%, and 11.0%, respectively.

Index Terms—Graph Database Modeling, RDBMS-to-Graph Transformation, Large Language Models, LLM Agents, Multi-Agent Systems, Automated Schema Design, Knowledge Graph Construction, Query Answering, Explainable AI

I. INTRODUCTION

A graph database is a NoSQL database that uses graph structures to store data as nodes (entities) and edges (relationships), making it ideal for highly interconnected data [11].

This new database has a flexible schema that allows data to be represented with rich semantic meaning through nodes and relationships, similar to a human-conceptualized representation of complex data objects in the real-world application [7]. Therefore, it can be applied in various real scenarios for multiple industries such as computer science, social networks, e-commerce foundations, supply-chain systems, and medical applications [4] where graph database accepts a more natural-language network-structured compared to traditional databases such as relational databases [30, 31].

Graph databases offer substantial improvements in cost reduction and time efficiency over complex multi-hop traversals [12, 21]. Unlike relational databases, which frequently face performance bottlenecks and scalability issues with complex, multi-table join operations, graph databases are purpose-built for traversing highly interconnected data. Their native representation of various data domains by entities as nodes and relationships as directed edges allows traversal algorithms to operate in linear or near-linear complexity, even across deeply connected structures. They are also less memory-intensive compared to relational databases, contributing to their efficiency [19]. Graph databases also significantly outperform relational databases in terms of query speed [23, 29]. This highlights the superior performance and efficiency of graph databases in managing and querying connected data.

Additionally, efficiently handling complex data correlations is vital due to the diversity of topics in these datasets, which require sophisticated query optimization techniques [22]. Graph databases, with their inherent ability to manage complex relationships and deliver superior performance, align well with complex queries, especially multi-hop reasoning queries.

Most traditional conversions of relational databases to graph databases are based primarily on graphical query language (GQL) loading commands that cannot interpret the relationships among entities within the targeted output graph. The process of migrating data from relational databases to graph databases is complex and resource-intensive, and currently, there is no standardized or widely accepted solution for

this conversion [26]. Despite their transparent set of loading commands, these drawbacks represent a critical barrier to the broader adoption of graph databases.

We primarily propose a novel Multi-Agent System for automatically transforming an unnormalized structured database into a normalized graph database that leverages the knowledge of the structured data schema to yield a better graph database design with higher accuracy and faster answers. We demonstrate its efficiency on a BFSI dataset from a real financial company that contains many internal relationships between tables. We apply data anonymization and masking technologies to hide sensitive information and ensure data privacy policy.

We address the complexities and challenges of converting relational databases by leveraging the strength of the LLM-based Multi-Agent System. This closed multi-agent system comprises many specialized agents with particular skills to emulate these distinct roles with the required expertise and collaborate to considerably address complex tasks, especially ETL and database queries. Moreover, this system is an end-to-end solution that can automate and define the knowledge graph structure and ETL data in the same pipeline. To boost the accuracy of Graph database design, we apply a novel Multi-Agent design pattern, which can be reiterated over many refinement turns before reaching the final standard knowledge graph design.

In conclusion, to address the challenges of relational databases, our approach offers several key contributions as follows:

- **Integration of relational and Knowledge Graph databases:** We propose a method that effectively integrates relational databases with Knowledge Graphs, overcoming the limitations of current approaches and enabling seamless data management across different paradigms.
- **End-to-End Data Ingestion Pipeline:** We offer a comprehensive end-to-end data ingestion pipeline that integrates Knowledge Graph architecture with a general Data Ingestion solution, ensuring high accuracy and efficiency in data processing.
- **End-to-End Multi-Agent System:** We introduce an end-to-end Multi-Agent System that produces a round-table discussion with featuring specialized agents with distinct roles: ETL Agent, Analyzer Agent, and Graph Agent. This system automatically designs relationships and nodes for Knowledge Graph architectures.
- **Standard improvement of Knowledge Graph schema procedure:** We establish the first generalized procedure that iteratively evaluates and improves graph structures to be more precise and aligned with the domain-knowledge.

Our approach leverages the cognitive, reasoning, and self-reflective capabilities of Multi-Agent Systems combined with advanced LLM techniques to address existing limitations in database integration. Thanks to automating the design of Knowledge Graph architectures and providing a streamlined data ingestion pipeline that demonstrates a significant advancement in database management and optimization.

II. RELATED WORK

a) *Multi-Agent Systems.*: Recent advancements in LLM-based autonomous agents have received significant attention from industry and academia, particularly in software development [15]. These agent-centric systems, designed to specialize in various coding tasks, have propelled the field forward [8, 16, 17, 27, 33]. Typically, these systems assign distinct roles, such as Programmer, Reviewer, and Tester, to individual agents, each responsible for a specific phase of the code generation process, thereby improving the quality and efficiency of software development. In addition, a variety of benchmarks have been developed to assess the performance of these systems in real-world scenarios. For example, SWE-Bench evaluates the ability of multi-agent systems to address actual GitHub issues [18], while MetaGPT's SoftwareDev provides a comprehensive suite of software requirements from diverse domains, challenging agents to produce fully developed software solutions [16]. Similarly, [24] uses Agile Methodology and multiple agents with a Dynamic Code Graph Generator to improve code generation and modifications through updated dependency graphs. In our case, we aim to build upon these advancements by leveraging a smart end-to-end Multi-Agent system to bridge relational databases and Knowledge Graph databases, enhancing data management and integration through advanced LLM techniques.

b) *Relational vs Graph Databases.*: The comparison between SQL and Graph Databases reveals crucial differences in the management of complex and interconnected data. Relational databases, such as MySQL, excel in structured data scenarios with their robust schema enforcement and support for complex queries, including joins and transactions. However, their performance can degrade when handling large, interconnected datasets due to inefficient joins across extensive tables, leading to slower query execution [1, 14, 32]. In contrast, Graph Databases, such as Neo4j, leverage nodes and edges to efficiently model and query interconnected data, significantly reducing the need for complex joins. This design enables them to handle complex relationships, such as those found in social networks or data provenance systems, with superior performance [9, 20, 28]. Graph Databases offer a flexible schema that adapts to evolving data structures, which enhances their efficiency and scalability in dynamic environments [3, 5]. This flexibility reduces data retrieval times and improves scalability, especially in distributed systems. Additionally, transforming relational databases into Knowledge Graph databases can enhance data integration by combining SQL's structured querying capabilities with the flexible, relational modeling capabilities of Graph Databases, as noted in recent advancements [2, 4, 10]. This approach maximizes the strengths of both paradigms, optimizing data management and integration.

c) *Data Schema Improvement.*: Relational and graph-based data management paradigms differ substantially in how they treat schema: relational databases rely on fixed, table-based schemas, while knowledge graphs support richer

semantics, flexible class hierarchies, and dynamic relations, making schema design and evolution in KGs more complex but also more expressive. For relational databases, model-driven approaches to schema evolution, such as EvolveDB, leverage reverse-engineering of existing data dictionaries into abstract models, allow manual schema modifications, compute differences, and automatically derive migration scripts to evolve both schema and data over time [13]. Meanwhile, in the KG context, schema evolution encompasses operations like adding/removing classes, altering class hierarchies, and modifying properties, i.e. changes that often require propagating updates throughout the graph to preserve consistency. Bridging the gap between relational data and knowledge graphs has been a major research focus. Some works address this by direct mapping: for example Towards a Complete Direct Mapping From Relational Databases To Property Graphs defines a complete mapping process that transforms any relational database (both schema and data) into a property graph, ensuring information preservation, semantic preservation, and query preservation; and even supports translating SQL queries into equivalent graph-database (e.g. Cypher) queries [6]. Another recent work, Rel2Graph [34]: Automated Mapping From Relational Databases to a Unified Property Knowledge Graph, extends this effort: it automatically constructs a unified property knowledge graph from one or more relational databases and supports the mapping of conjunctive SQL queries into equivalent graph-pattern queries, measuring fidelity with an execution-accuracy metric. Once relational data is mapped into a graph representation, additional techniques address schema optimization for storage and query performance. For instance, [25] proposes a method to extract of properties describing different classes of RDF instances, exploit their hierarchy, and merge/reduce tables to improve performance in an RDB-backed RDF engine, improving both storage efficiency and query performance. Although existing schema improvement algorithms provide systematic ways of mapping between data schema and relational and graph databases. They overly depend on syntactic rules and pre-defined mapping operators that suffer several limitations. Therefore, it motivates us to create an LLM-powered Multi-Agent System capable of semantically standardizing table names, column names, and relationships.

III. MULTI-AGENT ETL SYSTEM

A. Problem Statement

The rapid growth of business analytics across multiple industries has created a strong demand for systems that can automate database operation, management, and knowledge discovery tasks traditionally performed by humans. In response, we propose a multi-agent system capable of handling multiple aspects of the database lifecycle, including ETL processes, data standardization, data transformation, and data mining.

Our multi-agent framework incrementally formalizes a knowledge graph schema that explicitly represents complex entity relationships, thereby enhancing semantic transparency and interpretability, while simultaneously enabling efficient query execution. In contrast, traditional relational (SQL)

databases, although effective for structured tabular data, suffer from several limitations in analytical workloads, including high computational costs for large-scale joins, implicit and schema-bound relational semantics, and sensitivity to rigid query syntax.

Knowledge graph (KG) databases, by comparison, represent data as interconnected nodes and edges, facilitating efficient graph traversals, semantic enrichment, and scalable handling of complex relational structures. These properties make KGs a more suitable foundation for intelligent, LLM-powered Multi-Agent database systems.

Therefore, we propose a novel system that can automate many stages in a data operational system. Specifically, our detailed architecture is as follows.

Our solution leverages the SQL database D_{standard} and the corresponding schema S to create a Graph Schema. The pipeline of transformations is wrapped up inside the Graph Schema for simplicity.

First, the standard data D_{standard} are loaded and standardized from structured data: D_{SQL} , CSV tabular data D_{csv} ; and raw data D_{raw} . The SQL and CSV tables will be normalized to their table names, field names, and field formats, ensuring interpretability after transformation by ETLAgent, which defines a prompt engineering to adjust the user column:

Algorithm 1: ETL and standardize dataset

- 1: **Input:** Structured data: CSV raw data D_{raw} , SQL database $D_{\text{SQL}}(\mathcal{T}, \mathcal{R}, \mathcal{C})$; and raw data D_{raw} .
 - 2: **Output:** Standard dataset D_{standard}
 - 3: **function** ELT(D_{SQL} , D_{csv} , D_{raw})
 - 4: $\{\mathcal{T}, \mathcal{R}, \mathcal{C}\}_{\text{SQL}} \leftarrow \text{Load}(D_{\text{SQL}})$ $\triangleright$ Load table, relation, column from SQL database
 - 5: $\{\mathcal{T}, \mathcal{C}\}_{\text{csv}} \leftarrow \text{Load}(D_{\text{csv}})$ $\triangleright$ Load table, column from CSV file
 - 6: $\{\mathcal{N}, \mathcal{E}, \mathcal{P}\}_{\text{Graph}} \leftarrow \text{Load}(D_{\text{raw}})$ $\triangleright$ Define node, edge, properties from raw data
 - 7: $D_{\text{standard}} \leftarrow \text{LLM}(\{\mathcal{T}, \mathcal{R}, \mathcal{C}\}, \{\mathcal{N}, \mathcal{E}, \mathcal{P}\})$ $\triangleright$ Asking ETLAgent to modify table name, column name, datatype (Fig.3)
 - 8: **return** D_{standard}
 - 9: **end function**
-

where $D_{\text{SQL}}(\mathcal{T}, \mathcal{R}, \mathcal{C})$ is the input SQL database comprising Tables $\mathcal{T}$, Relationships $\mathcal{R}$, and Columns $\mathcal{C}$. and $\mathcal{G} = (\mathcal{N}, \mathcal{E}, \mathcal{P})$ is the fine-grained knowledge graph including nodes $\mathcal{N}$, edges $\mathcal{E}$ and properties $\mathcal{P}$.

In the next stage, we produce a Graph generation pipeline that takes the standardized input from ETL Agent and uses Graph Agent to map tables $T_i \in \mathcal{T}$ to nodes $\mathcal{N}$ and relationships $\mathcal{R}$ to directed edges $\mathcal{E}$, with properties $\mathcal{P}$ capturing metadata. The obtained output is a proposal graph schema G_i . However, this graph did not meet the technical specifications, like structures and syntax. Therefore, we process a continuous

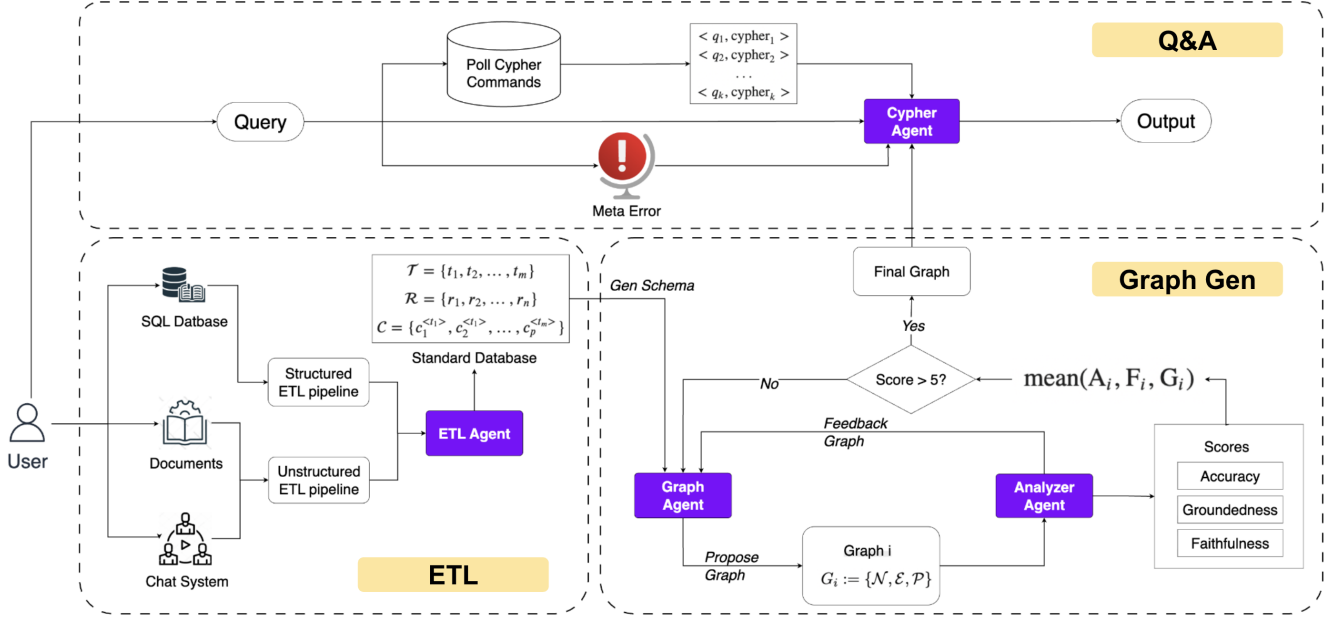


Fig. 1: The proposed solution is composed of three primary components. First, the ETL component is responsible for extracting, transforming, and loading both structured and unstructured data into a standardized database through dedicated Structured ETL pipelines. Second, the GraphGen component iteratively evaluates and refines the graph schema design to construct a final knowledge graph optimized for accuracy, groundedness, and faithfulness. Finally, the Q&A component enables the system to address business queries by generating and executing Cypher commands against the constructed graph.

loop for tuning the Graph Schema, relying on its previous version, until it obtains a score greater than τ_{thres} .

Formally, the transformation process can be expressed as GraphAgent as in Algorithm 2:

$$A_{\text{Graph}} : \langle \mathcal{D}_{\text{SQL}}(\mathcal{T}, \mathcal{R}, \mathcal{C}), \mathcal{G}_{i-1}(\mathcal{N}, \mathcal{E}, \mathcal{P}) \rangle \rightarrow \mathcal{G}_i(\mathcal{N}, \mathcal{E}, \mathcal{P}),$$

Algorithm 2: Graph generation pipeline

```

1: Input: Structured data: Table schema descriptions  $\mathcal{S}$ , SQL
   database  $\mathcal{D}_{\text{SQL}}(\mathcal{T}, \mathcal{R}, \mathcal{C})$ , CSV Tabular data  $\mathcal{D}_{\text{csv}}$ ; Raw
   unstructured data: text, pdf, powerpoint  $\mathcal{D}_{\text{raw}}$ ;
2: Output: Final graph knowledge representation  $\mathcal{G}_{\text{final}}$ 
3: procedure GENGRAPH
4:    $\mathcal{D}_{\text{standard}} \leftarrow \text{ETL}(\mathcal{D}_{\text{SQL}}, \mathcal{D}_{\text{csv}}, \mathcal{D}_{\text{raw}})$ 
5:    $\mathcal{G}_0 := \text{GraphAgent}(\mathcal{D}_{\text{standard}})$ 
6:   for  $i \leftarrow 1$  to max_iteration do
7:      $\mathcal{G}_i \leftarrow \text{GraphAgent}(\mathcal{D}_{\text{standard}}, \mathcal{G}_{i-1})$ 
8:      $\text{score}(\mathcal{G}_i) \leftarrow \text{SCOREGRAPH}(\mathcal{G}_i)$ 
9:     if  $\text{score}(\mathcal{G}_i) \geq \tau_{\text{thres}}$  then
10:       $\mathcal{G}_{\text{final}} \leftarrow \mathcal{G}_i$  ▷ Get final graph
11:      break
12:     end if
13:   end for
14: end procedure

```

To ensure the validity of the generated graph schema across schema theory, knowledge graph modeling principles, and domain-specific business semantics, we assign Analyzer Agent to evaluate the architecture of graph according to three aspects:

- **Accuracy:** The degree to which the nodes, attributes, and relationships in the Graph Schema correctly represent the structures and constraints of the original SQL database.
- **Groundedness:** The extent to which the Graph Schema conforms to graph database design principles, including syntactic correctness and semantic appropriateness.
- **Faithfulness:** The degree to which the Graph Schema faithfully captures the intended business logic and supports downstream business operations without introducing distortion.

Algorithm 3: Knowledge Graph Scoring

```

1: Input: Previous Graph Schema  $\mathcal{G}_{i-1} \triangleq \{\mathcal{N}, \mathcal{E}, \mathcal{P}\}$ , Stan-
   dard data  $\mathcal{D}_{\text{standard}}$ 
2: Output: Knowledge Graph scoring (accuracy, faithful-
   ness, groundedness)
3: function SCOREGRAPH( $\mathcal{G}_i$ )
4:    $\mathcal{G}_i := \{\mathcal{N}, \mathcal{E}, \mathcal{P}\}$ 
5:    $(A_i, F_i, G_i) \leftarrow \text{Analyzer}(\mathcal{G}_i)$  ▷ Accuracy,
   Faithfulness, and Groundedness
6:    $\text{score}(\mathcal{G}_i) = \text{mean}(A_i, F_i, G_i)$ 

```

```

7:   return score( $G_i$ )
8: end function

```

Heuristically, accuracy verifies that nodes, attributes, and relationships correctly reflect the original SQL schema. This aligns with the correctness of schema transformation and lossless information used in database migration research. Groundedness evaluates whether the schema follows standard graph modeling principles. We restrict the Graph Schema from unnecessary intermediaries, require meaningful relationship types, and ensure semantic clarity. These groundedness checks are equivalent to integrity constraints and schema validity used in the ontology of the original relational database schema. Finally, faithfulness evaluates whether the schema preserves domain business semantic meaning and supports downstream tasks without distortion of important business principles and context.

These three metrics generalize long-standing evaluation dimensions from both relational schema theory, knowledge graph modularity, and business understanding. These metrics are organized in a template of the Analyzer Agent prompt as in Figure 3 in the Appendix. Each metric is rated on a 5-point Likert scale. The final score is computed as the average of the three metrics. A schema is accepted if its score is $\geq \tau_{\text{thres}} = 5$. The tuning loop runs for a maximum of 10 iterations.

B. Overall Architecture

The architecture, as shown in Figure 1, is designed to transform structured SQL data into a Knowledge Graph Database (GraphDB) using a Multi-Agent system. Each agent has a distinct role to play, working collaboratively to ensure efficient and accurate conversion. The transformation process can be modeled as: $\mathcal{A}_{\text{total}} = \mathcal{A}_{\text{Graph}} \cup \mathcal{A}_{\text{Analyzer}} \cup \mathcal{A}_{\text{ETL}}$, where $\mathcal{A}_{\text{total}}$ represents the combined functionality of all agents. The system produces a validated GraphDB ready for advanced querying and analytics.

1) *System Prompt Pooling*: The System Prompt Pooling component serves as the central coordinator, distributing tasks to specialized agents: Graph Agent, Analyzer Agent, and ETL Agent. It provides prompts $\mathcal{P}_{\text{pool}} = \{p_1, p_2, \dots, p_m\}$, where each p_i contains instructions and parameters for agent tasks. The prompt assignment function is as follows: $\mathcal{P}_{\text{pool}} \rightarrow \{\mathcal{A}_{\text{Graph}}, \mathcal{A}_{\text{Analyzer}}, \mathcal{A}_{\text{ETL}}\}$, ensuring a precise execution of the data conversion process.

IV. DATASET

We propose a SQL database comprising many basic tables like Loans, LoanTypes, Transactions, Customers, Accounts, Cards, and Branches for the BFSI field. This database will be used as a foundation to develop our Data conversion and Business Question and Answering with the Multi-Agent system. The SQL database will be converted to a standard Graphical Database. Afterward, they will be used as source databases to evaluate the efficiency of the question and answering task on both SQL and Knowledge Graph.

A. BFSI Business Queries

We propose a new dataset named BFSI, which consists of 1,081 business queries that comprehensively span the operational aspects of the banking and finance domain. To capture a balanced range of difficulty, queries are segmented into three levels: easy (512 queries), medium (333 queries), and hard (236 queries), based on the complexity of their logical structure and the number of entities involved in the underlying graph database.

- **Hard level**: queries involve complex business questions that simultaneously reference at least two entities within the graph. These queries are designed to incorporate multiple challenging constraints, advanced filtering conditions, nested aggregations, and intricate calculation functions. They often require the use of multi-level groupings, join operations, and domain-specific computations, making them representative of real-world decision-making tasks faced by financial analysts and managers.
- **Medium level**: queries also relate to two entities but employ simpler logical constructs. Their focus lies on basic aggregation functions such as SUM, AVG, MIN, and MAX, without the need for sophisticated filtering or multi-step reasoning. These queries reflect common analytical tasks that help banks derive insights into customer behavior, product usage, or account-level summaries.
- **Easy level**: queries are restricted to a single entity and can be resolved using straightforward query syntax. They usually answer fundamental operational questions, such as entity counts, attribute filtering, or simple comparisons. These queries serve as the foundation for routine reporting and monitoring activities in banking systems.

By organizing the dataset into these three levels, we ensure that it not only reflects the breadth of financial business operations but also provides a structured benchmark for evaluating systems across varying degrees of query complexity.

V. EXPERIMENTS

A. Setup Evaluation Agents

To process the benchmarking process for the query, we initialize **CypherAgent** and **SQLAgent**. CypherAgent handles business queries based on text-2-cypher engine, which converts the input query into the final answer. This agent is aided by a graph database that is migrated from an unstandardized SQL database. On the other hand, SQLAgent is an ReAct SQL Agent that helps answer business queries using the text-2-sql engine. Thanks to the ReAct pattern, this agent is strengthened to Thought, Action, and Observation interleavedly. Another agent called **EvalAgent** is to evaluate the accuracy and latency of two answers from the same query input of CypherAgent and SQLAgent. To keep the equality, we use the same input for both agents, including database schema, relationships, and meta errors. Figure 7 in the Appendix shows the complete pipeline.

B. Results

We make a comprehensive comparison between the SQL and Cypher approaches at varying levels of query complexity. Our empirical results demonstrate that the Cypher-based approach consistently outperforms the SQL-based method in all evaluation metrics, with particularly notable improvements in both accuracy and computational efficiency, as in Table II

In the easy query category, which primarily comprises single-entity retrievals and basic filtering operations, Cypher achieves an accuracy of 90.43%, surpassing SQL’s 72.66% by a significant margin. This 17.77 percentage point improvement is accompanied by a 3.9 second reduction in latency (1.73 seconds vs. 5.63 seconds), demonstrating the inherent efficiency of graph-based query processing even for straightforward operations.

The performance disparity becomes more pronounced in the medium complexity category, where Cypher maintains its accuracy of 81.98%, while SQL’s performance achieved 77.78%. This substantial 4.2 percentage point improvement is particularly noteworthy as it represents queries involving moderate relationship traversals and multi-entity interactions. The latency differential also widens, with Cypher requiring only 1.94 seconds compared to SQL’s 6.91 seconds, highlighting the scalability advantages of graph processing.

For hard queries, which encompass complex multi-hop relationships and sophisticated pattern matching, both approaches exhibit decreased performance. However, Cypher maintains its superiority with 80.08% accuracy compared to SQL’s 69.07%. The latency measurements in this category reveal the most significant temporal efficiency gap, with Cypher operating at 2.64 seconds versus SQL’s 7.30 seconds, underscoring the graph database’s ability to handle complex relationship patterns more efficiently as in Figure 8 in the Appendix.

The aggregate results demonstrate Cypher’s robust performance advantages, achieving a mean accuracy of 85.57% compared to SQL’s 73.45%, with an average latency improvement of 4.39 seconds (corresponding to 68.8% faster) across all complexity levels (Table II). These findings provide compelling evidence for the efficacy of graph-based approaches in LLM-driven query generation systems, particularly within domains characterized by complex interconnected data structures. The results suggest that graph databases offer a more suitable foundation for natural language query processing in enterprise-scale applications.

VI. ABLATION STUDY

In addition, the performance comparison of Cypher components over the BFSI dataset demonstrates that Meta Error improves accuracy by 28.84% while greatly reducing token usage, and Few-shot Learning achieves the highest accuracy 85.57%, corresponding to 4.63% improvement, with the lowest latency. Both approaches demonstrate clear efficiency gains over the baseline because the Baseline Cypher usually gets syntax errors that require many retry turns. These errors lead to substantially higher latency and token consumption. Table III

Few-shot learning is effective: Initially, the query system relied on two base SQL Agent for querying relational data in PostgreSQL and Cypher Agent for handling graph-based data in Neo4j. These agents were designed without similar examples. Therefore, it lacks the experience to learn the correct business logic, especially in complex relational databases such as banking and finance. After adding the few-shot examples, it reinforces the business understanding of these agents a lot.

Meta error adding: After analyzing the initial version of the Multi-agent system, we realize that it usually commits common types of error in syntax and selects the right tables and columns. Therefore, we store all common types of errors in a meta error database. Adding them to Agent prompt will significantly reduce syntax errors and increase the percentage of runnable code.

In conclusion, our result demonstrates that the full system, combining Meta Error correction with few-shot grounding, substantially outperforms the baseline in both effectiveness (accuracy) and efficiency (token usage, latency). This supports our claim that a judicious integration of common errors and few-shot examples prompting is essential for reliably translating relational queries into correct graph queries in real-world, domain-specific databases.

VII. CONCLUSION & FUTURE WORK

We propose a novel system that systematically converts structured databases into knowledge graph databases while preserving critical fields and relationships across multiple tables. Our results demonstrate that employing a multi-agent system yields superior performance compared to a single large language model-based Question Answering system when applied to business queries. Notably, our proposed solution, CypherAgent, outperforms a traditional SQLAgent in terms of efficiency, as it leverages graph traversal for query answering rather than relying solely on indexing constraints. In addition, CypherAgent achieves higher accuracy by enhancing the semantic representation of relationships across multiple schemas. The overall token usage of CypherAgent is significantly lower than that of SQLAgent due to optimized prompting and a reduced number of reasoning steps. Furthermore, our experiments show that incorporating techniques such as Meta-Error correction and Few-Shot Learning strengthens syntactic robustness by integrating prior knowledge of common errors and domain-specific expertise. In general, our multi-agent solution establishes an end-to-end pipeline that automates key database operations including normalization, migration, and analysis. This system provides a scalable mechanism for extracting, loading, and transforming structured databases into graph-based representations. Empirical evidence indicates that graph databases are robust and effective in the BFSI domain, with strong potential for generalization across diverse industries and large-scale enterprise environments. These findings highlight the promise of multi-agent knowledge graph systems as a foundation for future intelligent data management and decision-support applications.

TABLE I: Performance comparison of CypherAgent and SQLAgent variants across different query complexities (*Easy*, *Medium*, *Hard*). CypherAgent incorporates three techniques: meta error (ME), which encodes common query mistakes; few-shot learning (FL) with $k = 3$ examples; and correct error (CE), which analyzes syntax errors and suggests fixes in subsequent rounds. Results are reported in terms of Accuracy (Acc, %), Latency (seconds per sample), and number of Tokens (per sample). CypherAgent-[ME]-[FL]-[CE] with GPT-4o achieves the best overall accuracy while maintaining competitive efficiency, whereas SQLAgent baselines show higher token usage and latency.

Algorithm	Backbone	Easy			Medium			Hard		
		Acc	Latency	Tokens	Acc	Latency	Tokens	Acc	Latency	Tokens
CypherAgent-[ME]	GPT4o	87.70	3.19	1143	77.18	2.38	1156	71.61	2.75	1177
CypherAgent-[ME]-[FL]	GPT4o	79.49	3.28	1309	76.28	3.81	1354	61.44	3.05	1411
CypherAgent-[ME]-[FL]-[CE]	GPT4o	90.43	1.73	1314	81.98	1.94	1364	80.08	2.64	1458
CypherAgent-[ME]-[CE]	GPT4o	80.27	1.66	1239	76.28	2.04	1285	61.02	3.23	1353
SQLAgent-[ReAct]	GPT4o	72.66	5.63	3188	77.78	6.91	2624	69.07	7.30	3284
SQLAgent-[ReAct]	Qwen-8B	64.70	2.17	2332	58.30	2.66	2355	48.70	2.56	2385
CypherAgent-[ReAct]	Qwen-8B	58.40	2.13	3547	46.90	2.65	3562	45.80	2.99	3578

TABLE II: Comparison between the best-performing CypherAgent and SQLAgent across different query complexities. CypherAgent consistently outperforms SQLAgent, achieving an average accuracy gain of +12.12% while reducing latency by over 3 times and total tokens by over 2 times. Results are reported in terms of Accuracy (%), Latency (seconds), and relative Accuracy Gain (%).

Query Complexity	Accuracy (%)		Latency (seconds)		Tokens		Accuracy Gain (%)
	Cypher	SQL	Cypher	SQL	Cypher	SQL	
Easy	90.43	72.66	1.73	5.63	1,314	3,188	+17.77
Medium	81.98	77.78	1.94	6.91	1,362	2,624	+4.20
Hard	80.08	69.07	2.64	7.30	1,458	3,284	+11.01
Mean	85.57	73.45	2.00	6.34	1,361	3,035	+12.12

TABLE III: Performance comparison of Baseline Cypher, Meta Error, and Few-shot Learning. Adding meta error can significantly increase the accuracy by 28.84%, whereas, Few-shot learning improves 4.63%.

Method	Accuracy	Latency (seconds)	Tokens	Accuracy Gain
Baseline Cypher	52.11%	2.482	3,558	—
Meta Error	80.94%	2.846	1,154	+28.84%
Few-shot Learning	85.57%	1.995	1,361	+4.63%

REFERENCES

- [1] Victor Aluko and Sherif Sakr. Big sql systems: an experimental evaluation. *Cluster Computing*, 22(4):1347–1377, December 2019.
- [2] Renzo Angles, Angela Bonifati, Stefania Dumbrava, George Fletcher, Alastair Green, Jan Hidders, Bei Li, Leonid Libkin, Victor Marsault, Wim Martens, Filip Murlak, Stefan Plantikow, Ognjen Savkovic, Michael Schmidt, Juan Sequeda, Slawek Staworko, Dominik Tomaszuk, Hannes Voigt, Domagoj Vrgoc, Mingxi Wu, and Dusan Zivkovic. Pg-schema: Schemas for property graphs. *Proceedings of the ACM on Management of Data*, 1(2):1–25, June 2023.
- [3] Renzo Angles and Claudio Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40(1), February 2008.
- [4] Maciej Besta, Robert Gerstenberger, Emanuel Peter, Marc Fischer, Michał Podstawski, Claude Barthels, Gustavo Alonso, and Torsten Hoefer. Demystifying graph databases: Analysis and taxonomy of data organization, system designs, and graph queries, 2023.
- [5] Angela Bonifati, Peter Furniss, Alastair Green, Russ Harmer, Eugenia Oshurko, and Hannes Voigt. Schema validation and evolution for graph databases, 2019.
- [6] Abdelkrim Boudaoud, Houari Mahfoud, and Azeddine Chikh. Towards a complete direct mapping from relational databases to property graphs, 2022.
- [7] Carlos J. Fernández Candel, Diego Sevilla Ruiz, and Jesús J. García-Molina. A unified metamodel for nosql and relational databases. *Information Systems*, 104:101898, 2022.
- [8] Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruob-

- ing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors in agents. *arXiv preprint arXiv:2308.10848*, 2023.
- [9] Yijian Cheng, Pengjie Ding, Tongtong Wang, Wei Lu, and Xiaoyong Du. Which category is better: Benchmarking relational and graph database management systems. *Data Science and Engineering*, 4, 12 2019.
 - [10] Alberto Hernández Chillón, Meike Klettke, Diego Sevilla Ruiz, and Jesús García Molina. A taxonomy of schema changes for nosql databases, 2022.
 - [11] Alejandro Corbellini, Cristian Mateos, Alejandro Zunino, Daniela Godoy, and Silvia Schiaffino. Persisting big-data: The nosql landscape. *Information Systems*, 63:1–23, 2017.
 - [12] Thi-Thu-Trang Do, Thai-Bao Mai-Hoang, Van-Quyet Nguyen, and Quyet-Thang Huynh. Query-based performance comparison of graph database and relational database. In *Proceedings of the 11th International Symposium on Information and Communication Technology, SoICT '22*, page 375–381, New York, NY, USA, 2022. Association for Computing Machinery.
 - [13] Torben Eckwert, Michael Guckert, and Gabriele Taentzer. Evolvedb: evolving relational database schemas in a model-driven way. *Software and Systems Modeling*, pages 1–26, 11 2025.
 - [14] Avrielia Floratou, Umar Farooq Minhas, and Fatma Özcan. Sql-on-hadoop: full circle back to shared-nothing database architectures. *Proc. VLDB Endow.*, 7(12):1295–1306, August 2014.
 - [15] Taicheng Guo, Xiuying Chen, Yaqi Wang, Ruidi Chang, Shichao Pei, Nitesh V. Chawla, Olaf Wiest, and Xiangliang Zhang. Large language model based multi-agents: A survey of progress and challenges. *arXiv preprint arXiv:2402.01680*, 2024.
 - [16] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024.
 - [17] Dong Huang, Qingwen Bu, Jie M Zhang, Michael Luck, and Heming Cui. Agentcoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*, 2023.
 - [18] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.
 - [19] Petri Kotiranta, Marko Junkkari, and Jyrki Nummenmaa. Performance of graph and relational databases in complex queries. *Applied sciences*, 12(13):6490, 2022.
 - [20] Petri Kotiranta, Marko Junkkari, and Jyrki Nummenmaa. Performance of graph and relational databases in complex queries. *Applied Sciences*, 12(13), 2022.
 - [21] Geonho Lee, Jeongho Park, and Min-Soo Kim. Chimera: A system design of dual storage and traversal-join unified query processing for sql/pgq. *Proc. VLDB Endow.*, 18(2):279–292, October 2024.
 - [22] Vikram Nathan, Jialin Ding, Tim Kraska, and Mohammad Alizadeh. Cortex: Harnessing correlations to boost query performance, 2020.
 - [23] Thomas Neumann, Matthias Bender, Sebastian Michel, Ralf Schenkel, Peter Triantafillou, and Gerhard Weikum. Distributed top-k aggregation queries at large. *Distributed and Parallel Databases*, 26:3–27, 2009.
 - [24] Minh Huynh Nguyen, Thang Phan Chau, Phong X. Nguyen, and Nghi D. Q. Bui. Agilecoder: Dynamic collaborative agents for software development based on agile methodology. *arXiv preprint arXiv:2406.11912*, 2024.
 - [25] George Papastefanatos, Marios Meimaris, and Panos Vassiliadis. Relational schema optimization for rdf-based knowledge graphs. *Information Systems*, 104:101754, 2022.
 - [26] Joshua Porter and Aleks Ontman. Importing relationships into a running graph database using parallel processing, 2020.
 - [27] Chen Qian, Xin Cong, Cheng Yang, Weize Chen, Yusheng Su, Juyuan Xu, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development. *arXiv preprint arXiv:2307.07924*, 2023.
 - [28] Cajetan Rodrigues, Mit Ramesh Jain, and Ashish Khanchandani. Performance comparison of graph database and relational database. *technical report*, 05 2023.
 - [29] Johan Sandell, Einar Asplund, Workneh Yilma Ayele, and Martin Duneld. Performance comparison analysis of arangodb, mysql, and neo4j: An experimental study of querying connected data, 2024.
 - [30] Phillip Schneider, Tim Schopf, Juraj Vladika, Mikhail Galkin, Elena Simperl, and Florian Matthes. A decade of knowledge graphs in natural language processing: A survey, 2022.
 - [31] Sushmi Thushara Sukumar, Chung-Horng Lung, Marzia Zaman, and Ritesh Panday. Knowledge graph generation and application for unstructured data using data processing pipeline. *IEEE Access*, 12:136759–136770, 2024.
 - [32] Yuanyuan Tian. The world of graph databases from an industry perspective, 2022.
 - [33] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.
 - [34] Ziyu Zhao, Wei Liu, Tim French, and Michael Stewart. Rel2graph: Automated mapping from relational databases to a unified property knowledge graph, 2023.