Works in Progress in Embedded Computing (WiPiEC) Journal, vol. 11, no. 1, 2025

Optimal Graph Model Schema Injection for Large Language Model-Generated Cypher Queries

Shady Hegazy, Nouman Nusrallah, Christoph Elsner
Siemens Foundational Technologies
Siemens AG
Munich, Germany
Firstname.lastname@siemens.com

Jan Bosch
Department of Computer Science and Engineering
Chalmers University of Technology
Göteborg, Sweden
Jan.bosch@chalmers.se

Helena Holmström-Olsson
Department of Computer Science and Media Technology
Malmö University
Malmö, Sweden
Helena.holmstrom.olsson@mau.se

Abstract—**Platform ecosystems have transformed the way value is created in different industries. The data traces of such ecosystems are typically represented through graph models and databases. Retrieval of relevant data from such databases requires writing extensively complex queries to travers such complex networks to fetch and slice the correct sub-graphs corresponding to the original business inquiry. Advances in generative artificial intelligence, namely large language models (LLMs), can provide a no-code interface to such complex databases by generating and executing database queries that fetch the correct and relevant data in response to user prompts and inquiries. However, for the LLM to generate the right query, data about the schema of the database and the underlying graph model must be provided. In this study, we present a pipeline for evaluating different techniques for injecting the database schema in the LLM prompts, in addition to preliminary evaluation results.**

Keywords-graph database; software ecosystem; large language models; graph algorithms

I. Introduction

Platform ecosystems have fundamentally transformed the way value is created and distributed across industries [1]. By enabling diverse actors such as developers, organizations, and users to co-create and exchange value around a shared technological platform, these ecosystems have become critical enablers of innovation and economic growth. The increasing digitization of platform activities has led to the generation of rich data traces, which are often represented using graph-based models and stored in graph databases [2]. These representations capture the intricate relationships among ecosystem participants, resources, and interactions, offering a powerful basis for analyzing ecosystem dynamics [3]. Despite the expressive power of graph databases, retrieving relevant data from them remains a technically challenging task. Business inquiries that require traversing complex networks and extracting specific sub-graphs often demand the formulation of sophisticated query languages such as Cypher or Gremlin. Writing such queries is not only time-consuming but also requires significant technical expertise, which limits access for non-technical stakeholders such as platform managers and decision-makers. This barrier hampers the ability of organizations to derive timely and actionable insights from their ecosystem data. Recent advances in generative artificial intelligence, particularly large language models (LLMs), offer a promising opportunity to bridge this gap. LLMs have demonstrated remarkable capabilities in understanding natural language and generating structured outputs, enabling the development of no-code interfaces for complex systems. By translating natural language inquiries into graph database queries, LLMs have the potential to make graph-based ecosystem analytics accessible to a broader range of users. However, enabling LLMs to generate accurate and relevant queries for graph databases presents unique challenges. Crucially, the model requires contextual information about the database schema and the underlying graph structure to formulate correct queries. Without this knowledge, even highly capable models often produce incomplete or invalid outputs.

This paper presents a pipeline for evaluating techniques to inject database schema information into LLM prompts to improve their ability to generate correct graph queries. We describe our approach for schema representation and prompt construction, as well as a set of experiments comparing different injection techniques. Preliminary results from these experiments demonstrate the impact of schema injection on query accuracy and provide insights into the design of LLM-driven interfaces for graph databases.

II. Methodology

This section presents the end-to-end workflow for transforming platform ecosystem data into a graph representation and enabling natural language to Cypher query translation

Manuscript received July 16, 2025; revised July 26, 2025; accepted July 25, 2025. Published September 2, 2025.
Issue category: Special Issue on DSD/SEAA 2025 on Works in Progress (WiP) Session, Salerno, Italy, Sept. 2025
Paper category: Short
DOI: doi.org/10.64552/wipiec.v11i1.102

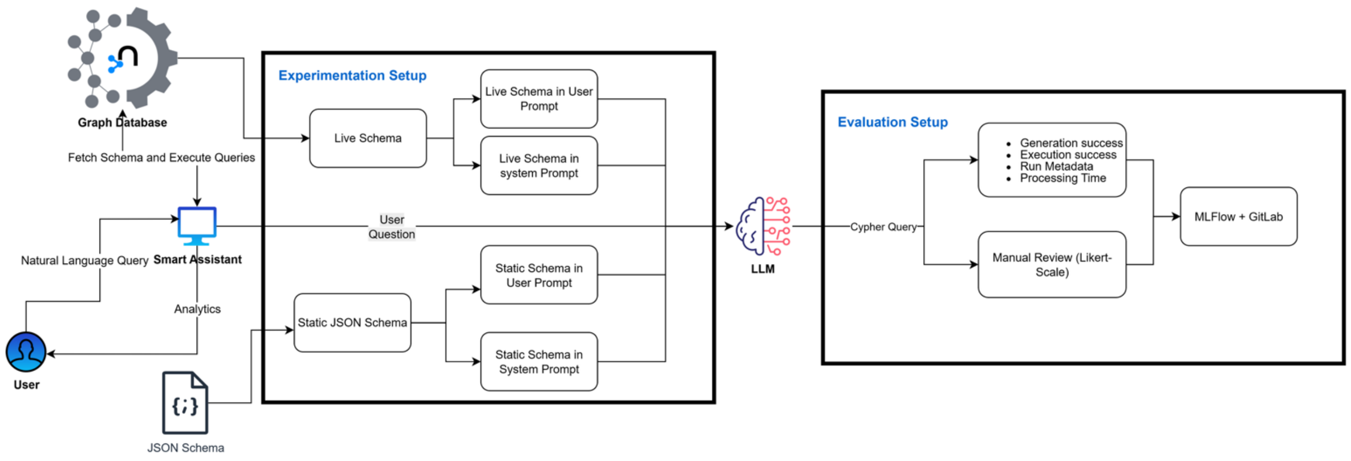


Figure 1. Architecture overview of the experimentation and evaluation pipeline including graph database integration with the Smart Assistant for natural language to Cypher queries.

using a Large Language Model (LLM). Figure 1 outlines the architecture overview of the workflow for the experimentation and evaluation pipeline.

A. *Data Extraction and Graph Construction*

Structured SECO data is collected from different data sources. The data covers entities such as products, owners, maintainers, visitors, devices, and usage logs. The sources cover different types of data including DevOps, financial, analytics data. An ETL pipeline was created and executed daily. The pipeline consists of the following steps:

- Extract: Connects to APIs, scrapes relevant data, and retrieves documentation and metadata.
- Transform: Drops irrelevant fields, merges multi-source records, parses URLs, and validates entity relationships.
- Load: Loads cleaned data into a Neo4j property graph model. Historical schema snapshots are stored for reproducibility.

B. *Generative AI Query Translation Pipeline*

The Smart Assistant enables natural language interaction for analytics. The system explores schema injection modes varying along the following variables:

- Schema Source: Either a live schema fetched from Neo4j via the APOC library, which pulls complete metadata of the graph, or the static JSON schema definition file.
- Prompt Placement: Schema injected into either the system prompt or the user prompt.

For each query, the pipeline: Fetches or loads the schema; constructs system and user prompts accordingly; submits the prompt pair to the LLM; parses the LLM response to extract Cypher queries; and executes queries on the graph and returns results.

C. *Graph Schema Model*

The SECO graph schema includes nodes for Visitors, Devices, Departments, Companies, and APIs, connected through relationships such as `MADE`, `USED`, `PART_OF`, and `ASSOCIATED_WITH`. Figure 2 shows a version of the graph data model schema.

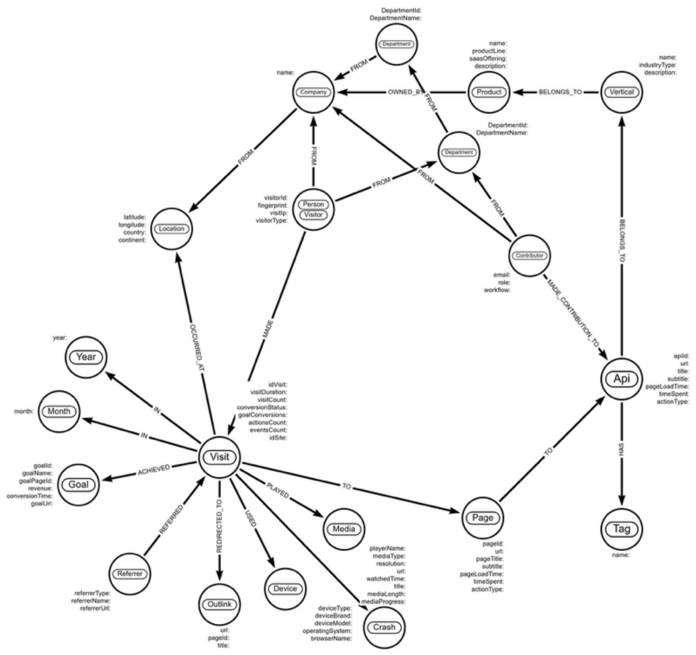


Figure 2. Graph database schema highlighting nodes and relationships.

D. *Feedback and Evaluation Workflow*

The entire experiment is tracked with MLflow for prompt versions, model runs, and metrics, as shown in Figure 3. GitLab CI/CD pipelines automate data preparation, schema snapshotting, and test coverage for each configuration. Additionally, A human-in-the-loop workflow is integrated and comprises the following steps:

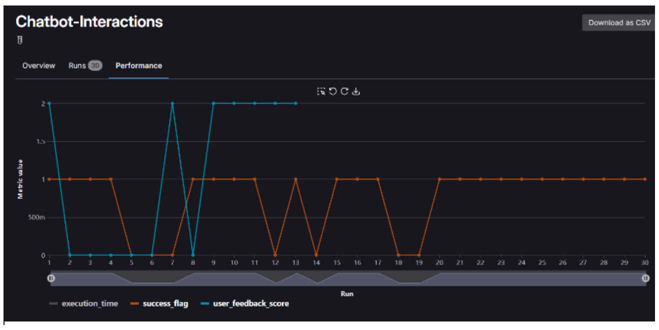


Figure 3. Performance logging of chatbot interactions in MLflow.

- Reviewers assess Cypher generation correctness and logical validity.
- Each query is rated on a 5-point Likert scale, with 1 indicating incorrect/irrelevant output and 5 indicating fully correct and usable queries, as shown in Table I.

TABLE I
LIKERT SCALE FOR HUMAN EVALUATION

Score	Interpretation
1	Incorrect or irrelevant Cypher
2	Major logical errors
3	Partially correct, needs edits
4	Mostly correct, minor edits
5	Perfectly correct and usable

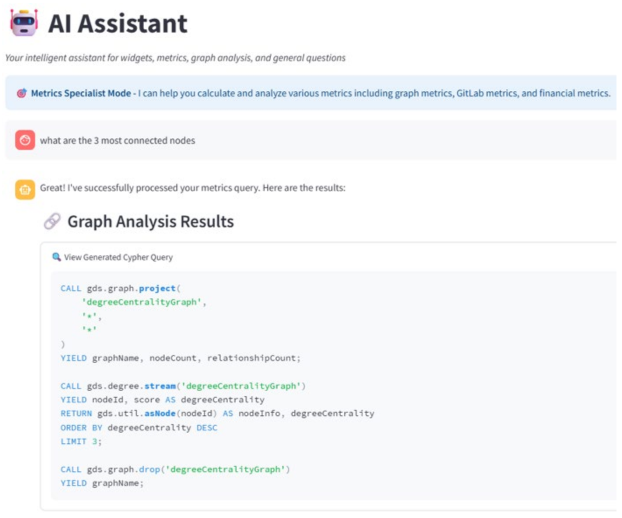
E. *Chatbot User Interface*

A user interface was developed based on Streamlit Chatbot UI framework, as shown in Figure 4. It includes the following aspects:

- Session storage: Each session logs user prompts, schema context, generated Cypher, execution results, and feed- back.
- Results persistence: Users can iteratively refine queries and view results in real-time. Previously generated visualizations are also persisted for further refinements.
- Historical tracking: Sessions are stored for later review and model improvement.
- Feedback elicitation: A widget is embedded in each response to allow for feedback elicitation enabling human-in-the-loop evaluation.

III. DISCUSSION

A preliminary evaluation of the different setups was carried out by executing a set of ten representative queries under the four experimental configurations. Human evaluation by domain experts were carried out for the 40 test cases. The 5-point Likert scale responses were aggregated so that it results in either a success or failure flag to facilitate comparison. The results suggest that including the database schema in the system prompt achieves higher consistency compared to embedding it in the user prompt. Additionally, using a well- defined static JSON schema generally performs better than fetching the schema live using the database own algorithmic functions. Simple entity-relation queries, such as filtering visitors by device brand or counting unique users, achieved near-perfect success across all configurations. For instance, the query *"List all Visitors who use a specific Device brand, like 'Apple'."* returned valid Cypher queries and expected record counts in all scenarios. In contrast, complex multi-hop or community-matching queries demonstrated higher variance. For example, the query *"Show all Companies with Departments that belong to the same community as Visitor"* succeeded when using the JSON schema but failed in all live schema scenarios. This indicates that more complicated traversal logic is sensitive to prompt placement and schema representation. Approximately 20% of test cases failed, primarily due to incomplete subgraph pattern matching or empty result sets when complex conditions were involved. Failures were more common in the JSON schema with user prompt scenario for temporal-spatial queries, such as "List all unique Visitors from Country X who made Visits in August 2024, along with the device types they used." On average, the query generation time per NLQ remained under 5 seconds, while Cypher execution times were acceptable for interactive analytics workloads. All logs were stored in structured CSV files to enable further analysis and reproducibility. These findings highlight the need for prompt engineering and potential domain fine-tuning to handle edge cases more reliably. Future work will expand the query set, integrate additional LLM models, and explore retrieval-augmented generation.


AI Assistant

Your intelligent assistant for widgets, metrics, graph analysis, and general questions

Metrics Specialist Mode - I can help you calculate and analyze various metrics including graph metrics, GitLab metrics, and financial metrics.

what are the 3 most connected nodes

Great! I've successfully processed your metrics query. Here are the results:

Graph Analysis Results

View Generated Cypher Query

```
CALL gds.graph.project(
    'degreeCentralityGraph',
    '*',
    '*'
)
YIELD graphName, nodeCount, relationshipCount;

CALL gds.degree.stream('degreeCentralityGraph')
YIELD nodeId, score AS degreeCentrality
RETURN gds.util.asNode(nodeId) AS nodeInfo, degreeCentrality
ORDER BY degreeCentrality DESC
LIMIT 3;

CALL gds.graph.drop('degreeCentralityGraph')
YIELD graphName;
```

Figure 4. Snapshot of the user interface.

REFERENCES

[1] A. Hein *et al.*, "Digital platform ecosystems," *Electron Markets*, vol. 30, no. 1, pp. 87–98, Mar. 2020, doi: 10.1007/s12525-019-00377-4.

[2] F. von Briel and P. Davidsson, "Digital platforms and network effects: Using digital nudges for growth hacking," in *Int. Conf. Inf. Syst., ICIS*, Association for Information Systems, 2019.

[3] P. Boldi and G. Gousios, "Fine-Grained Network Analysis for Modern Software Ecosystems," *ACM Trans. Internet Technol.*, vol. 21, no. 1, 2021, doi: 10.1145/3418209.