UDC 004.8:004.652:004.912 DOi 10.30837/ bi.2025.2(103).09

O. O. Sutiahin
NTU KhPI, Kharkiv, Ukraine, sutiahin.oleksandr@cs.khpi.edu.ua,
ORCID iD: 0009-0005-6527-455X

A MULTI-STAGE SELF-REVIEW FRAMEWORK FOR TRANSLATING NATURAL LANGUAGE INTO NEO4J CYPHER QUERIES

A Multi-Stage Self-Review Framework for Translating Natural Language into Neo4j Cypher Queries. The article presents a multi-stage self-review framework for automatically translating natural language questions into Cypher queries for the Neo4j graph database. The proposed approach integrates self-review mechanisms of large language models (LLMs), knowledge graph structure analysis, and multi-level validation of the syntax and semantics of generated queries. The framework includes three core stages: preliminary graph schema analysis, initial LLM-based query generation, and iterative self-review using specialized validation agents that detect logical, structural, and analytical inconsistencies. A prototype implementation is developed to evaluate the difference between query generation with and without the self-review mechanism. Experimental results demonstrate that incorporating self-review improves Cypher query correctness, reduces logical and structural errors, and enhances alignment with OLAP-oriented analytical requirements. The findings confirm the effectiveness of multi-stage self-review workflows for increasing the reliability of natural-language interfaces to graph-based analytical systems.

LLM, CYPHER, NEO4J, QUERY GENERATION, SELF-REVIEW, KNOWLEDGE GRAPH, OLAP, ANALYTICAL SYSTEMS, RAG, QUERY VALIDATION

О. О. Сутягін. Багатокрокова система саморецензування для перетворення природної мови у Cypher запити Neo4j. У статті представлено багатокрокову систему саморецензування для автоматизованого перетворення текстових запитів природною мовою у Cypher-запити до графової бази даних Neo4j. Робота поєднує механізми самоперевірки великих мовних моделей (LLM), аналіз структури графа знань та багаторівневу валідацію синтаксису й семантики згенерованих запитів. Запропонований підхід включає три основні етапи: попередній аналіз схеми графа, первинну генерацію запиту на основі LLM та ітеративну самоперевірку з використанням агентів валідації, які виявляють логічні, структурні та аналітичні помилки. Запропонована система впроваджена у прототипі програмного забезпечення, що виконує експериментальне порівняння генерації запитів із самоперевіркою та без неї. Результати експериментів показують, що використання self-review механізму забезпечує підвищення коректності Cypher-запитів, зменшення кількості логічних та структурних помилок і покращення відповідності сформованих запитів аналітичним OLAP-вимогам. Отримані результати підтверджують ефективність багатокрокового саморецензування для підвищення надійності текстового інтерфейсу до графових аналітичних систем.

LLM, CYPHER, NEO4J, ГЕНЕРАЦІЯ ЗАПИТІВ, САМОПЕРЕВІРКА, ГРАФ ЗНАНЬ, OLAP, АНАЛІТИЧНІ СИСТЕМИ, RAG, ВАЛІДАЦІЯ ЗАПИТІВ

1. Introduction

Text-to-SQL converts natural language into SQL queries, enabling non-experts to access databases without SQL knowledge [23]. While Large Language Models (LLMs) can interpret natural language, they are prone to errors and hallucinations. To address this, self-correction and verification techniques have been developed, including methods that refine LLM responses through generated feedback [24]. However, retraining LLMs is often impractical due to time constraints, so this work focuses on prompt-based teaching.

We introduce a self-verification approach for converting natural language to Cypher queries. Our multi-agent framework automatically generates and applies correction guidelines to remedy common graph database query errors, such as incorrect relationship directions and referencing nonexistent analytical entities. Unlike previous text-to-SQL efforts, our method targets the challenges of Cypher queries in Neo4j graph databases, particularly for OLAP tasks, by adapting the MAGIC framework for this context.

The framework operates in three logical stages, implemented via a multi-agent architecture: schema analysis, initial query generation, and iterative self-review at inference time. This prevents queries from referencing invalid or hallucinated entities.

Through a multi-agent architecture, our framework delegates schema analysis, query generation, review, and correction to specialized agents, mirroring human strategies and improving reliability over single-agent systems. Key contributions include:

– Establishing self-correction guideline generation for text-to-Cypher tasks in graph databases, adapting the MAGIC framework to handle graph-specific issues [24].

– Introducing a multi-agent validation system that integrates syntactic, semantic, and execution checks with real-time metadata verification.

– Developing an automated schema analysis pipeline for accurate and efficient LLM context injection.

– Implementing and empirically evaluating a complete multi-agent self-review pipeline for text-to-Cypher generation in Neo4j OLAP scenarios.

– The remainder of this paper covers related work (Section 2), methodology and architecture (Section 3), experimental setup (Section 4), and results (Section 5).

2. Literature review

2.1 LLM self-review methods

The proposed methodology operates on the principle that LLMs possess the capability to critically evaluate their own outputs when explicitly prompted to do so. By introducing a validation phase between initial response generation and final output delivery, we create an opportunity for the model to identify potential inconsistencies, logical fallacies, or factual inaccuracies that may have emerged during the initial generation phase. The method could be implemented by generating suitable prompts for LLMs to identify fallacies and correct them [8]. The authors proposed and evaluated several prompts on different LLMs for detecting, categorizing, and solving formal and non-formal fallacies step by step to decrease probability of logical reasoning errors.

This paper proposes and demonstrates that LLMs possess similar self-verification abilities [20]. The method operates in two stages: Forward Reasoning generates candidate answers using CoT, while Backward Verification masks original conditions and predicts them based on the proposed conclusion.

QueryGenie is a framework to generate SQL query from a sentence, using a self-review method [21]. It consists of different modules: a confirmation module that helps LLM to verify intent of generation SQL, query generation module and query validation method, which execute the query and recheck result. Despite the practical idea, there wasn't any practical evidence this framework was implemented, so its practical applicability remains unclear of QueryGenie.

Another app that helps with generation of SQL queries is MAGIC [22]. The authors created a multi-agent system that automates the creation of the self-correction guideline. These agents operate collaboratively in an iterative framework to analyze failures produced by a baseline large language model (LLM) on the training dataset. Through this iterative process, the system autonomously generates and refines self-correction guidelines specifically calibrated to address systematic errors made by the LLM. This approach emulates human error-analysis and guideline-development processes while maintaining complete autonomy from human intervention. This approach was chosen as the basis for our method.

2.2 Knowledge graph

The knowledge graph was developed to better manage knowledge by connecting entities—real-world objects—via semantically described edges. Entities are typically represented as triples (subject, predicate, object), and edges denote their relationships. There are two main construction approaches: top-down, which starts with ontology creation, and bottom-up, which begins with data extraction. To build a knowledge graph, study [1] suggests: identifying relevant domains and data sources, constructing an ontology (mainly for top-down methods using existing formats like OWL, XML, or RDF), extracting knowledge—often with machine learning, processing to eliminate redundancy and ambiguity, enriching the knowledge, and finally, developing, storing, visualizing, and deploying the knowledge graph. Neo4j is a widely used database for storing knowledge graphs.

A knowledge graph could be presented as G (E, R, T), where E, R and T represent the set of entities, relations, and knowledge triples, respectively [2]. For every knowledge triplet T ∈ T summarize knowledge of graph G and presented as T = (e_h, r, e_t), where e_h, e_t ∈ E and r ∈ R. This work also considers two definitions for reasoning path and entity path [2].

For reasoning path they use formula that represents set of connected sequences of triplets in graph: $path_G$ (e_l, e_{l+1}) = {$T_1, T_2, \ldots, T_l$} = {(e_1, r_1, e_2), (e_2, r_2, e_3), ..., (e_l, r_l, e_{l+1})}, where T_i ∈ T denotes the *i*-th triple in the path and *l* denotes the length of the path, i.e., *length* ($path_G$ (e_l, e_{l+1})) = *l*. Example: Consider a reasoning path between the entity "University" and the entity "Student" in a KG. The reasoning path is given by: $path_G$ (University, Student) = {(University, employs, Professor), (Professor, teaches, Course), (Course, enrolled_in, Student)}, and can be visualized as:

University $\rightarrow^{employs}$ Professor $\rightarrow^{teaches}$ Course $\rightarrow^{enrolled\_in}$ Student.

This path indicates that a “University" employs a “Professor," who teaches a “Course," in which a "Student" is enrolled. The length of the path is 3.

Second definition looks like: given a KG_G and a list of entities $list_e$ = [e_1, e_2, e_3, . . ., e_l], the entity path of $list_e$ is defined as a connected sequence of reasoning paths, which is denoted as

$path_G$ ($list_e$) = {$path_G$ (e_1, e_2), $path_G$ (e_2, e_3), ..., $path_G$ (e_{l-1}, e_l)} = {(e_s, r, e_t) | (e_s, r, e_t) ∈ $path_G$ (e_l, e_{l+1}) ∧ $1 \le i < l$}.

All those definitions are used to prompt LLM models with KG that will prevent LLM anomalies and improve performance of answering questions. The main idea is to construct KG that contains facts and relations between them. The whole process is divided into four stages: initialization, exploration, path pruning, and question answering.

Also, KG are used for improving understanding of facts and relations between them in LLMs. Those models often suffer from hallucinations because of lack of specific information contained in trained corpus and problems of using chain-of-thought. A possible solution to these issues is to integrate knowledge graphs (KGs) into LLMs. This study [3] provides a strategy for LLMs to communicate with KGs for improving reasoning. Authors developed three categories of interaction between LLM and KG: KG-enhanced LLM, LLM-augmented KG, synergized LLM + KG [3].

In the context of this work, these knowledge graph concepts are not used for explicit reasoning path extraction but serve as a theoretical foundation for schema-aware prompting. The proposed framework leverages structured graph representations primarily to constrain Cypher generation and prevent hallucinated entities rather than to perform symbolic graph reasoning.

3. Method

The BI4people project [4] aims to analyze data and make business decisions collaboratively by leveraging Online Analytical Processing (OLAP) to provide interactive analysis and data visualization. It introduces Collaborative Business Analysis (CBA) as a process of analyzing data and making business decisions collaboratively. Under the idea of developing the software-as-a-service model, the BI4people project tries to bring people together in a virtual space and encourage them to share their opinions and comments to collectively solve problems. The concept of CBI involves using social networks, quizzes, brainstorming sessions, and even simple chats [5]. Additionally, the reuse of other collaborators' comments or results is also considered part of CBI, which leads to a more comprehensive approach to BI. When creating a virtual space or forum for users to share their problems, comments, and solutions, it is essential to gather and analyze their data to unveil the intricate relationships between different pieces of information. It is crucial for the user to provide feedback on how they utilized the analysis carried out that can make a background for recommendations and reusing. BI4Tourism is a web-based application designed to empower users with insights into tourism data, thereby facilitating decision-making. At its core, BI4Tourism enables users to visualize and compare diverse tourism data, unraveling dependencies crucial for making strategic decisions [6].

For collaboration systems it's crucial to be able to investigate someone's similar question, process of research and result of it. In our application, we trace every action of the user during work on use cases and save it to the database. We need it to gather the context of user investigation to recommend other users with similar questions. Also, it's very important to persist in comments that are marked as answers because other users can participate in any use of case discussion. We use an OLAP cube to save our raw data and then our app queries it with different dimensions and measures, filters to retrieve information that is formatted to draw any chart. Among several implementations of OLAP cubes, we consider using Cube [7] because of easy installation and deployment locally. The schema of our graph is presented in Figure 1.

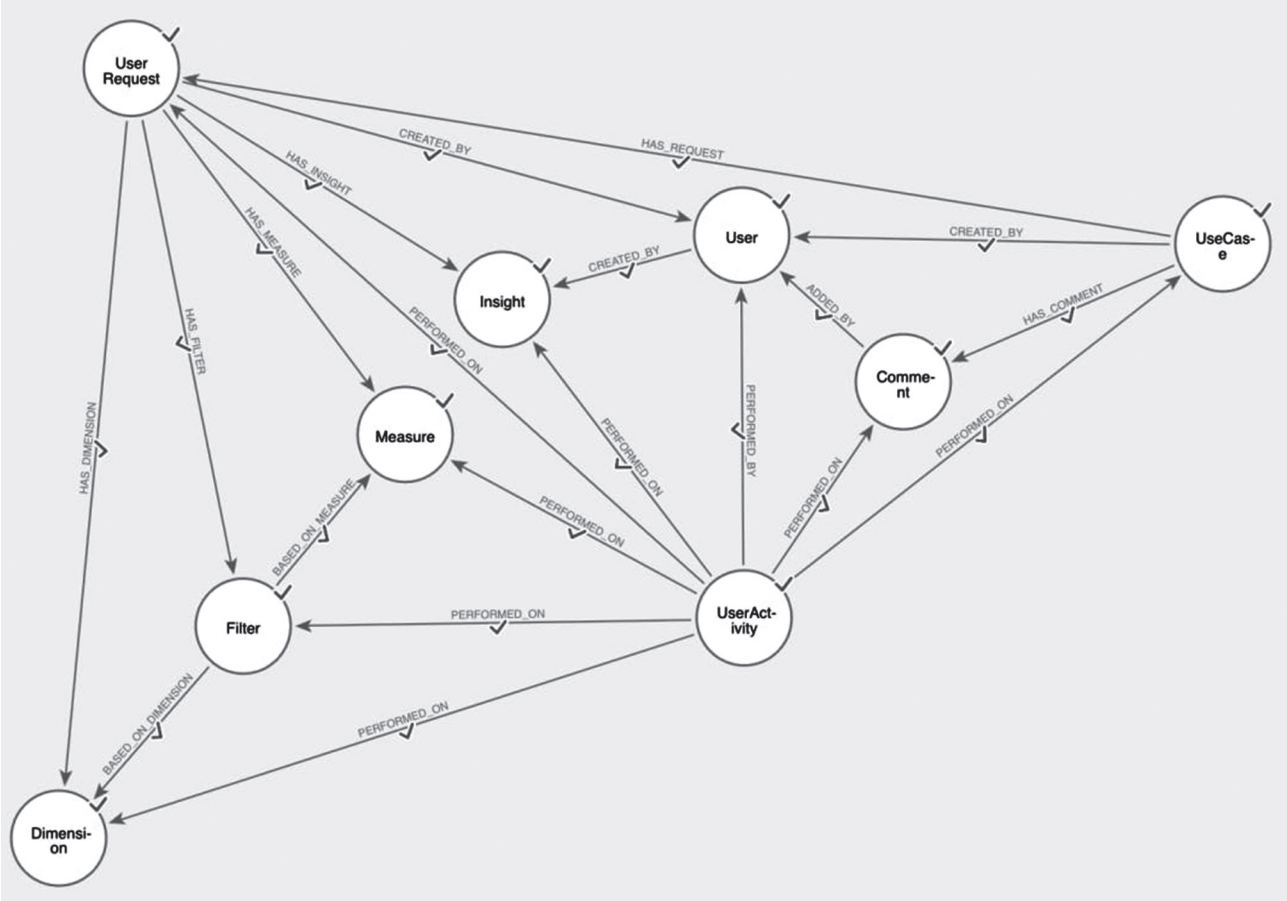

Fig. 1. Schema of knowledge graph for BI4Tourism in Neo4j database

3.1 Knowledge graph schema

The construction of the knowledge graph (KG) begins with the formal specification of its entities. Because the KG is intended to capture relationships among dimensions, measures, users, use cases and filters we instantiate each of these constructs as first-class entities within the graph. More generally, an entity in a KG denotes a categorical referent (for example, a person or a location). So in our system we establish that nodes of the knowledge graph could have one of the next types: dimension (E_d), measure (E_m), user

(E_u), use case (E_{uc}), comment (E_c), user request (E_{ur}), insight (E_i), filter (E_f) or user action (E_{ua}). Dimension usually represents values that describe the business context of the data, allowing users to slice and dice our OLAP cube by using filters. Filter contains operation type, filter value and unique id. Measures are data points within the cube, often numeric values from a fact table, by which we want to group data. User request's nodes gather all measures, filters and dimensions in one named request with help of which the user chose to investigate the chart. Insight entity represents which problem the user solved by investigating the result of the user request, it consists of comments that describe the solution and author of solution. Use case is the main question on which the user tries to find the answer. Use cases can have multiple user requests and each user request can have multiple insights. But one of the most important parts of our knowledge graph is the user action entity. It represents every action a user can make on a use case: write comment, change filter, add or update dimension or measure, add user request and insight. to this request.

After introducing nine classes of entities, we further defined ten classes of relations as shown in Figure 1. First class, we have relations between use case, user, user request and comment entities ($R_{uc\to ur}$ $R_{uc\to u}$, $R_{uc\to c}$) that represent "(one use case) created by (a specific author)", "(one use case) has a user request (a specific user request)" and "(one use case) has comment (a specific comment)". Second, we have a class of relations that connect dimensions to user request ($R_{ur\to d}$) and represent "(one user request) has dimension (a specific dimension)". Another class is connection between user request and measure ($R_{ur\to m}$) and means "(one user request) has measure (a specific measure)". Next one is about the relation between user and user request ($R_{ur\to u}$), it means that the user is the author of this user request, an example of it "(one user request) created by (a specific user)". And the next two types of relations are dependencies between filter and user request ($R_{ur\to f}$) and filter and dimension ($R_{d\to f}$), it could look like "(one user request) has a filter (a specific filter)" and "(one dimension) filtered by (a specific filter)". Also, we have a class of relation that links use case and user, reproducing whole user activity on this use case entity ($R_{ua\to u}$) and ($R_{ua\to uc}$, $R_{ua\to ur}$, $R_{ua\to c}$, $R_{ua\to d}$, $R_{ua\to m}$, $R_{ua\to i}$) for example "(one user action) performed on (one user request) by (one user)". Last type of relation is the connection between user requests and insights ($R_{ur\to i}$), it means that "(one user request) has an insight (a specific insight)".

3.2 Self-review Cypher generating method

We propose our method of generating Cypher query, based on self-review LLM method, to improve the experience of non-expert users in knowledge graphs. To achieve this aim, we introduced a multiagent system that consists of a schema review agent, feedback agent, correction, and Cypher generation agent.

The schema review agent should investigate the schema of Neo4j database, summarize it to a short description of nodes and relations, add more context about the database: domain of model, intent of different nodes and their relations. We need this information to share it with others and not overcharge their token window. Prompts for this agent are gathered in Appendix A

The Cypher generation agent has a database schema, context of the schema as input parameters to generate proper requests, based on user input text. The result of this agent's work should be presented as a ready to use Cypher query. Also, it consumes hints from the feedback agent to regenerate the query. Appendix C represents the prompt that the agent uses. And the next agent is the feedback one; it consumes schema context, user input, Cypher query and result of execution of this query. It analyzes those parameters with proper prompt in cycles, which is presented in Appendix D and sends a structured response for the correction agent to change the initial Cypher query. The correction agent helps to regenerate Cypher query based on the result of the feedback agent. The prompt for this agent is saved in Appendix E. The whole process is documented in Figure 2.

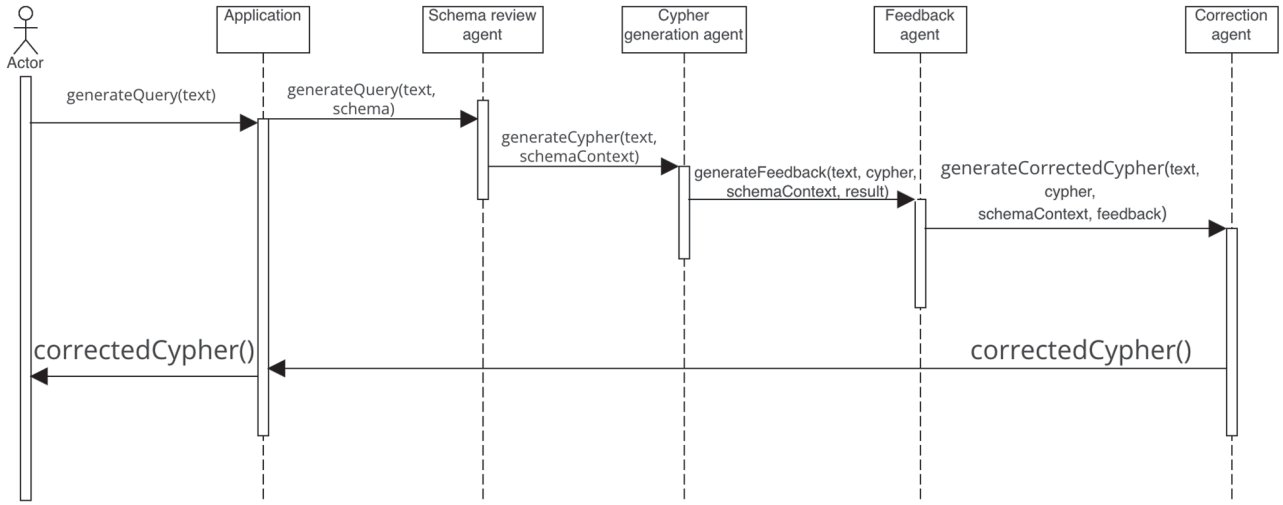

Fig. 2. Sequence diagram of communication between agents to get corrected cypher

4. Experiment

4.1 Experiment setup

We conducted an experiment to compare two ways of generating Cypher query from text with preprompted LLM: with self-review mechanism and without. Firstly, we had to choose a proper LLM, which will be used in the experiment. We compared the following LLMs: ChatGPT-4o [12], Groq [11], GPT-OSS-120B [9], Llama 3.3 70B [10] by context window, limit of requests per minute and speed of answer in Table 1.

Table 1

Results of comparison of LLMs

LLM name	Context token window	Limit of tokens/ minute	Price per 1M tokens
ChatGPT-4o	1,047,576	30,000	Input — $2.00 Output — $8.00
Groq	131,072	200,000	Free but depending on usage of the model
GPT-OSS-120B	131,072	250,000	Input — $0.15 Output — $0.60
Llama 3.3 70B	131,072	300,000	Input — $0.59 Output — $0.79

We selected GPT-OSS-120B as our primary language model based on its optimal balance of cost-efficiency and performance characteristics. Our implementation employs a self-review methodology requiring multiple sequential LLM requests per query within short time windows, making throughput capacity critical. This selection enabled economically feasible large-scale experimentation with iterative prompting without compromising result of quality or reproducibility.

We synthetically generate test data and approximately 100 tourism-related queries, grouped by different categories, based on real questions about tourism from Reddit [14]. We persisted the data in the Neo4j cloud database - Neo4j AuraDB [13].

In this study, we compared two prompts for generating Cypher queries from natural language: one that included self-review steps and one that did not. To identify which approach was more effective, we focused on key Retrieval-Augmented Generation (RAG) metrics, particularly answer correctness. We used the DeepEval framework [15] for our evaluation, based on evidence from GroUSE [16], a meta-evaluation tool for benchmarking evaluators. GroUSE reports that DeepEval outperforms similar frameworks such as RAGAS [17] on faithfulness, correctness, and answer relevance for grounded question answering. DeepEval relies on the LLM-as-a-judge paradigm, which has been shown to be more reliable than traditional statistical or human-based evaluation methods [18].

During implementation of evaluation in DeepEval framework we chose a prompt-based GPTEval method [19] which uses a chain of thoughts model to assess natural language generation output based on LLM-as-judge method. The framework prompts LLM to score some value for each evaluation aspect, based on the defined criteria, then LLM should add weights to those scores and summarize it. We defined our evaluation aspects for generation Cypher query based on two sets of prompts: for generation with feedback, we use Appendix A-E, but without self-review we use only prompts A-C.

The GPTEval scoring function is presented as a predefined set of discrete scores (e.g., 1 to 5) specified within the prompt, serving as the evaluation scale for subsequent assessments - $S = \{s_1, s_2, ..., s_n\}$. The token probability of each score $p(s_i)$ is the probability the LLM assigns to generating that score token, and the final score is:

$$score = \sum_{i=1}^{n} 2p(s_i) \times s_i \quad (1)$$

We configured the whole process of evaluation in the form of several categories of different queries, each containing different tests with data from our dataset, and wrote it in Python with the Deepeval framework.

4.2 Experiment results

Firstly, we grouped results of our tests for two methods of prompting by score each test received from the G-eval framework. As we can see on Figure 3-4 strategy of using LLM prompting with feedback loop shows better results than strategy without self-review, average of first is 0.65 score and second is 0.61 accordingly. The score is calculated with the GPTEval [19] method, described in previous section.

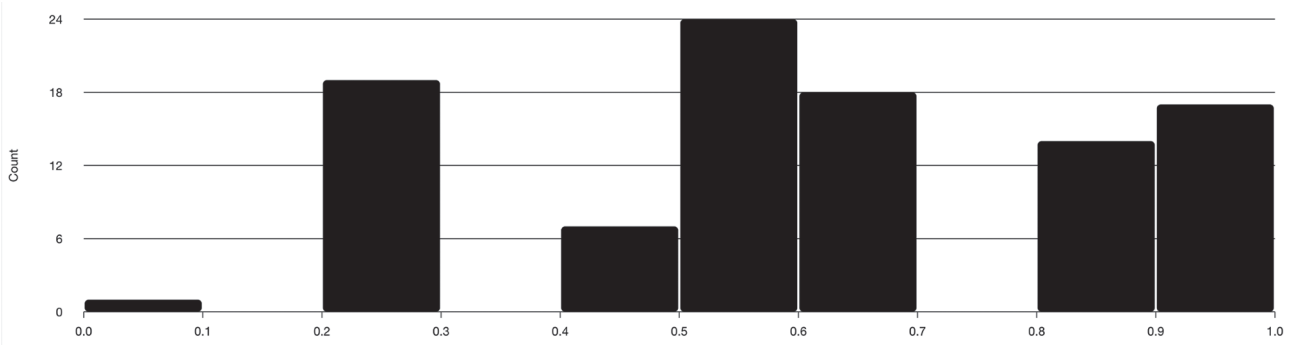

Fig. 3. Count of tests to their score in case of using LLM prompting without feedback loop

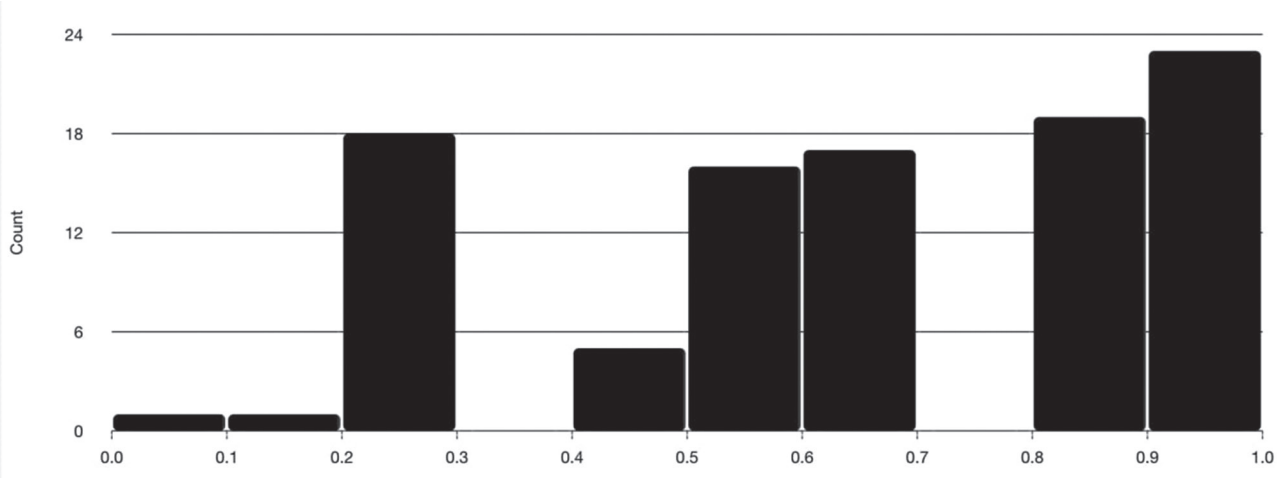

Fig. 4. Count of tests to their score in case of using LLM prompting with feedback loop

Next grouping key was the category of issues, which occurred during tests run. These reasons were generated by the G-eval framework using ChatGPT-4o [12] and then grouped by issue categories. It helps us to determine weak points of our prompt. The weakest points of both our prompts were incorrect logic of generating Cypher query, incorrect relationships, missing fields and filters. So, in this case the first and the second prompts made similar result.

Our last grouping field is the input category for average correctness score. For the prompt with feedback loop, multi-dimensional queries achieve the highest correctness, followed by Meta/Schema and Complex/Other, whereas in the prompt without feedback, Meta/Schema leads, with Aggregation/Statistics showing marked improvementю A notable divergence is observed in the multi-dimensional category, which drops from the top-performing position to mid-tier performance between prompts - a decrease of 22 percentage points. Conversely, Content/Insight demonstrates improvement, and Business Metrics rises above the threshold in the second condition.

4.3 Experiment conclusion

The comparative analysis demonstrates that the feedback loop prompting strategy consistently outperforms the non-feedback approach under the evaluated conditions, achieving a 6.5% relative improvement in average correctness score (0.65 vs. 0.61). These findings suggest that iterative self-review enhances output quality and cross-category robustness, though specialized solutions remain necessary for spatial and complex filtering operations.

5. Conclusion

This work introduces a multi-agent self-review framework for generating Cypher queries from natural language, tailored to the unique challenges of translating user questions into Neo4j graph database queries, especially in OLAP contexts. By adapting the MAGIC self-correction approach, our system leverages automated guideline creation and iterative validation to improve the reliability and accuracy of LLM-generated queries.

Key innovations include an automated schema analysis pipeline that efficiently represents Neo4j structures, a multi-agent design separating query generation, review, and correction, and real-time metadata verification to prevent referencing non-existent analytical entities. These elements address issues like complex graph traversal and improve upon monolithic or purely syntactic solutions.

While effective in Neo4j OLAP settings, the framework's limitations include its specificity to certain schemas, potential latency and token overhead from iterative correction, and lack of learning from past queries. Future work should explore broader graph database support, optimization for efficiency, and incorporating a repository of validated examples for improved performance.

References

[1] Tamašauskaitė, G., & Groth, P. (2022). Defining a Knowledge Graph Development Process Through a Systematic Review. ACM Transactions on Software Engineering and Methodology, 32, 1 - 40. https://doi.org/10.1145/3522586.

[2] Tan, Xingyu & Wang, Xiaoyang & Liu, Qing & Xu, Xiwei & Yuan, Xin & Zhang, Wenjie. (2024). Paths-over-Graph: Knowledge Graph Empowered Large Language Model Reasoning. 10.48550/arXiv.2410.14211.

[3] Shirui Pan, Linhao Luo, Yufei Wang, Chen Chen, Jiapu Wang, and Xindong Wu. 2024. Unifying large language models and knowledge graphs: A roadmap. IEEE Transactions on Knowledge and Data Engineering (2024).

[4] Business intelligence for the people. https://eric.univ-lyon2.fr/bi4people/index-en.html

[5] Muhammad, F., Darmont, J., Favre, C.: The Collaborative Business Intelligence Ontology (CBIOnt). 18e journées Business Intelligence et Big Data (EDA-22), Vol. B-18. Clermont- Ferrand, Octobre 2022, RNTI (2022)

[6] Cherednichenko, O., Sutiahin, O.: Development of Collaborative Business Intelligence Framework for Tourism Domain Analysis. In: Tekli, J., et al. (eds.) New Trends in Database and Information Systems. ADBIS 2024. CCIS, vol. 2186, pp. 270–281. Springer, Cham (2025). https://doi.org/10.1007/978-3-031-70421-5_21

[7] Cube. Available: https://cube.dev/, last accessed 2025/11/16

[8] Hong, Ruixin & Zhang, Hongming & Pang, Xinyu & Yu, Dong & Zhang, Changshui. (2024). A Closer Look at the Self-Verification Abilities of Large Language Models in Logical Reasoning. 900-925. 10.18653/v1/2024.naacl-long.52.

[9] OpenAI, "GPT-OSS-120B," OpenAI Platform Documentation. Available: https://platform.openai.com/docs/models/gpt-oss-120b, last accessed 2025/11/18.

[10] Meta and Ollama, "Llama 3.3 70B," Ollama Model Library. Available: https://ollama.com/library/llama3.3:70b, last accessed 2025/11/15.

[11] xAI, "Grok," xAI Official Documentation. Available: https://x.ai/, last accessed 2025/11/15.

[12] OpenAI, "GPT-4o," OpenAI Platform Documentation. Available: https://platform.openai.com/docs/models/gpt-4o, last accessed 2025/11/14.

[13] Neo4j, "Neo4j AuraDB: Fully Managed Graph Database," Neo4j Product Documentation. Available: https://neo4j.com/product/auradb/, last accessed 2025/11/10.

[14] Reddit, "Reddit - Dive into anything,". Available: https://www.reddit.com/, last accessed 2025/11/01.

[15] Confident AI: deepeval documentation. https://docs.confident-ai.com/ (2025), last accessed 2025/11/15.

[16] Muller, S., Loison, A., Omrani, B., Viaud, G.: GroUSE: A Benchmark to Evaluate Evaluators in Grounded Question Answering. arXiv preprint arXiv:2409.06595 (2024).

[17] Shahul Es, J.J., Espinosa-Anke, L., Schockaert, S.: Ragas: Automated Evaluation of Retrieval Augmented Generation. arXiv preprint arXiv:2309.15217 (2023)

[18] Gu, J., Jiang, X., Shi, Z., Tan, H., Zhai, X., Xu, C., Li, W., Shen, Y., Ma, S., Liu, H., Wang, Y., Guo, J.: A Survey on LLM-as-a-Judge. arXiv preprint arXiv:2411.15594 (2024)

[19] Liu, Y., Iter, D., Xu, Y., Wang, S., Xu, R., Zhu, C.: G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. arXiv preprint arXiv:2303.16634 (2023)

[20] Weng, Yixuan, et al. "Large language models are better reasoners with self-verification." Findings of the Association for Computational Linguistics: EMNLP 2023. 2023.

[21] Longfei Chen, Shenghan Gao, Shiwei Wang. 2025. QueryGenie: Making LLM-Based Database Querying Transparent and Controllable. In Adjunct Proceedings of the 38th Annual ACM Symposium on User Interface Software and Technology (UIST Adjunct '25). Association for Computing Machinery, New York, NY, USA, Article 49, 1–4. https://doi.org/10.1145/3746058.3758982

[22] Askari, Arian, Christian Poelitz, and Xinye Tang. "Magic: Generating self-correction guideline for in-context text-to-sql." In Proceedings of the AAAI Conference on Artificial Intelligence, vol. 39, no. 22, pp. 23433-23441. 2025.

[23] Qu, G., Li, J., Li, B., Qin, B., & Cheng, R. (2024). Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-sql generation. arXiv preprint arXiv:2405.15307.

[24] Aman Madaan, Niket Tandon, Luyu Gao, Sarah Wiegreffe, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. Advances in Neural Information Processing Syste

Date of submission of the article to the editorial board: 09.10.2025

Appendix A

Before any query generation occurs, we employ a dedicated schema analysis agent. The rationale for this preliminary step stems from the observation that LLMs often struggle to correctly interpret raw database schemas, leading to invalid node labels, incorrect relationship directions, and non-existent property references. The schema analyzer performs a comprehensive examination of the Neo4j graph structure, extracting:

- Node labels with their associated properties and data types
- Relationship types with directionality and cardinality
- Traversal patterns critical for OLAP analytics

	Prompt for mapping schema to table view for LLM consumption
1.	Analyze the following Neo4j graph schema with DETAILED focus on nodes and relationships: ## Raw Schema: {raw_schema} ## Required Analysis Structure: ### 1. NODES ANALYSIS For each node type in the schema, provide: - **Label**: Node label name - **Properties**: List all properties with their data types - **Purpose**: What this node represents in the domain - **Key Properties**: Properties that uniquely identify this node - **Usage Notes**: When and how to use this node in queries ### 2. RELATIONSHIPS ANALYSIS For each relationship type in the schema, provide: - **Pattern**: SourceNode -[:RELATIONSHIP_TYPE]-> TargetNode - **Direction**: Specify if directional or can be traversed both ways - **Properties**: List any properties on the relationship - **Semantic Meaning**: What this relationship represents - **Traversal Notes**: How to use this relationship in MATCH patterns - **Example Cypher Pattern**: Show a typical MATCH pattern using this relationship

	### 3. GRAPH STRUCTURE MAP - Draw a visual map of how all nodes connect through relationships - Show the complete graph topology - Identify primary traversal paths (e.g., UseCase -> UserRequest -> Measure) ### 4. DOMAIN MODEL INSIGHTS - What domain does this graph represent (OLAP, social network, etc.)? - What are the central/hub entities? - What are the analytical patterns supported? ### 5. QUERY GUIDANCE - Common MATCH patterns for this schema - Best practices for traversing relationships - Properties to use in WHERE clauses - Properties to RETURN for meaningful results Focus heavily on nodes and relationships - this is the most critical information for accurate Cypher query generation.

Fig. 5. Prompt for mapping schema to table view for LLM consumption

Appendix B

The primary query generation agent receives the user question and analyzed the schema context. The system prompts enforce strict adherence to the provided schema, instructing the model to use only documented relationship types and properties. For enhanced accuracy, we offer an alternative generation path incorporating the MAGIC (Multi-Agent Guideline Iteration for Correction) self-correction framework, which provides additional guideline documents containing learned patterns from previous query generation failures and instructs the model to engage in explicit step-by-step reasoning before query construction.

	Prompt for generating Cypher Query based on schema_context, question, hint
1.	Task: Generate Cypher queries to query a Neo4j graph database based on the provided schema definition and also provide the result of query. Instructions: Use only the provided relationship types and properties. Do not use any other relationship types or properties that are not provided in schema. If you cannot generate a Cypher statement based on the provided schema, explain the reason to the user. When generating queries: 1. Identify relevant measures and dimensions from the schema 2. Use only relationships from schema context and for appropriate nodes with right directions 2. Use proper node labels 3. Use proper relationship types for OLAP cubes (see schema for details) 4. Ensure queries are suitable for analytical/reporting purposes 5. When querying user requests, traverse through use cases to access their associated measures, dimensions, and filters 6. **IMPORTANT**: Results should usually be grabbed from UseCase and UserRequest entities as these are the primary analytical entities in the database Schema context: {schema_context} Question: {question} Hint: {hint} Generate the Cypher query. Return it in the following format: ```cypher YOUR CYPHER QUERY HERE ``` ``` | Variable | Description | |----------|-------------| | `base_system_prompt` | System message from Neo4jGPTQuery containing schema | | `schema_context` | Analyzed or raw schema context | | `question` | User's natural language question | | `hint` | Optional hint about the query |

Fig. 6. Prompt for generating Cypher Query based on schema_context, question, hint

Appendix C

The validation phase implements a multi-agent feedback loop, recognizing that single-pass generation often produces queries with subtle errors that may execute but return incorrect results. Generated queries are submitted to a specialized review agent that performs validation across four dimensions: OLAP-specific validation verifying that referenced measures and dimensions exist in the database, syntactic compliance examining Cypher patterns and relationship directions, common error detection identifying frequent mistakes such as incorrect quote usage, and logical correctness assessing whether the query semantically addresses the user's original question. The review agent receives real-time metadata from the Neo4j database, enabling it to detect hallucinated analytical entities that would otherwise pass syntactic validation but produce empty or incorrect results.

	Prompt for generating feedback of generated Cypher query
1.	You are an expert Cypher query reviewer for Neo4j. Analyze the generated Cypher query and provide constructive feedback. ## Review Focus Areas: ### 1. OLAP-Specific Validation - Are the referenced measures valid? (Check against schema) - Are the referenced dimensions valid? (Check against schema) - Does the query make sense for analytical purposes? - Are aggregations appropriate for the measures being queried? ### 2. Cypher Syntax & Schema Compliance - Cypher syntax correctness (MATCH, WHERE, RETURN patterns) - Relationship direction and types - ensure they match the schema - Node label usage - verify labels exist in schema - Property names - check they exist on the specified nodes/relationships ### 3. Common Issues - Using double quotes instead of single quotes for string literals - Incorrect relationship patterns - Missing RETURN clause - Wrong relationship directions for OLAP cube structure ### 4. Logical Correctness - Does the query answer the original question? - Are the joins/relationships logical for OLAP analytics? - Would this query produce meaningful business insights? Be specific and actionable. If the Cypher looks correct, say so briefly. Please review this Cypher query: Question: {question} Hint: {hint} Schema context: **Note:** measures_list and dimensions_list are retrieved from Neo4j database..{schema_context} {measures_list} {dimensions_list} Generated Cypher: ```cypher {generated_cypher} ``` Check if the measures and dimensions used in the query exist in the lists above. If they don't exist, this is a critical error that must be reported. Provide feedback on potential issues or improvements.

Fig. 7. Prompt for generating feedback of generated Cypher query

Appendix D

When the review agent identifies issues, a correction agent receives both the original query and the expert feedback. This separation of review and correction responsibilities follows the principle that critique and generation are distinct cognitive tasks, and specialized agents outperform single agents attempting both functions. Additionally, a post-execution analysis agent examines the actual results returned by Neo4j to determine whether they properly answer the user's question. If the analysis indicates a mismatch between results and user intent, the pipeline re-enters the correction phase with this additional feedback.

	Prompt for correction agent to update Cypher query
1.	You are an expert at analyzing database query results to determine if they answer the user's question. Your task is to: 1. Understand what the user is asking for 2. Examine the Cypher query that was executed 3. Review the actual results returned

	4. Determine if the results answer the user's question appropriately 5. Result should be from UseCase and UserRequest entities as these are the primary analytical entities in the database 6. Before suggest cypher query, you should check schema context and make sure the query is valid and makes sense for OLAP analytics Provide your analysis in this format: - **Matches Intent**: Yes/No - **Analysis**: Brief explanation of whether results answer the question - **Suggestions**: If the results don't match intent, suggest what should be changed (leave empty if results are good) Please analyze if these query results answer the user's question. ## User Question: {question} ## Schema Context: {schema_context} ## Cypher Query Executed: ```cypher {cypher} ``` ## Query Results: {result_data} ## Analysis Task: Does this query and its results properly answer the user's question? Consider: - Are the right columns/properties being returned? - Is the data relevant to what was asked? - Are the results complete or is something missing? - Would a user be asking this question be satisfied with these results? - Does the query align with the schema structure? Provide your analysis: ``` ## Response Parsing The response is parsed to determine if intent matches by looking for phrases like: - "matches intent: yes"

Fig. 8. Prompt for correction agent to update Cypher query