

Milestone 3 (30%)

Deadline: 15th of December at 23:59

Build an end-to-end **Graph-RAG Travel Assistant** that uses the Neo4j Knowledge Graph *from Milestone 2 as a grounding mechanism* for an LLM-based system. Your system must retrieve relevant nodes, relationships, and reviews from the KG, then use these results to guide the LLM's final answer. **You are allowed to add to the existing scheme and/or integrate the data with an external dataset.**

The project highlights how symbolic reasoning (KG) and statistical reasoning (LLMs) collaborate to reduce hallucination, increase factual accuracy, and enhance interpretability.

System Requirements

1. Input Preprocessing

a. Intent Classification

Classify what the user wants to do (e.g., ask a question, get recommendations, search for entities). This helps route the query to the appropriate retrieval strategy. You can use rule-based methods (keyword matching) or LLM-based classification. The intent determines which Cypher queries or retrieval methods to use. Each theme should have its own intent classifier adapted to its domain (e.g., hotel search, player performance analysis, flight route queries).

b. Entity Extractions

Extract relevant entities from user input (e.g., *entity names, locations, dates, attributes*). These entities are used to fill in the chosen Cypher query with the parameters.

Use *Named Entity Recognition (NER)* to identify theme-specific entities:

- **Hotel theme:** hotels, cities, countries, traveller types, demographics
- **FPL theme:** players, teams, positions, seasons, gameweeks, statistics
- **Airline theme:** flights, airports, passengers, journeys, routes

c. Input Embedding (depending on 2.b)

Convert the user's text input into a vector representation for semantic similarity search in the embedding-based retrieval approach. Only needed when you implement embedding-based retrieval (**section 2.b**). Use the same embedding model that was used to create node or feature vector embeddings in your KG.

2. Graph retrieval layer

You are required to implement **TWO** experiments. The first one is the baseline only, and the second adds the embeddings (section 2.b) to the baseline.

a. Baseline

1. Use Cypher queries to retrieve relevant information.

Write structured queries in Cypher (Neo4j's query language) to fetch nodes, relationships, and properties from the knowledge graph based on extracted entities. These are deterministic queries that use exact matches or filters.

Examples:

- **Hotel:** "Find hotels in Paris with rating > 4"
- **FPL:** "Get top players by position in season 2023"
- **Airline:** "Find flights from ORD to LAX with minimal delays"

2. At least 10 queries that answer 10 questions, based on the user input.

Create a library of at least 10 different Cypher query templates that can handle various question types. These queries should cover **different** intents and entity combinations. You'll select and parameterize the appropriate query based on intent classification and entity extraction.

Examples:

- **Hotel:** hotel search, review retrieval, amenity filtering, location-based queries, visa requirements
- **FPL:** player performance, team analysis, fixture queries, recommendations, statistics
- **Airline:** flight search, route analysis, delay queries, passenger satisfaction

3. Pass the extracted entities from the input to query the KG and retrieve the answer.

Use the entities extracted in (**step 1.b**) as parameters to fill in the Cypher query templates, then execute them to get relevant graph data.

Example:

- **Hotel:** If the user asks "*Hotels in Cairo*", extract "*Cairo*" as a location entity, then use it in a Cypher query
- **FPL:** If the user asks "*Top forwards in 2023*", extract position="FWD" and season="2023", then query players by position and season
- **Airline:** If the user asks "*Flights from ORD*", extract "ORD" as the departure airport, then query flights departing from that airport

b. Embeddings:

Implement semantic similarity search using vector embeddings. Choose **ONE** of the following approaches, and experiment with at least **TWO** different embedding models for comparison:

1. Node Embeddings

Create vector representations for each node in the graph. Similar nodes will have similar embeddings.

- **For themes with numerical data (FPL, Airline):** Use numerical feature vectors derived from node properties (*e.g., player stats, journey metrics*)
- **For themes with textual data (Hotel):** Use graph embedding techniques (*e.g., Node2Vec, GraphSAGE*) or text embeddings of node properties
- Store these in Neo4j's vector index for fast similarity search

2. Features Vector Embeddings

Create vector representations for feature combinations. This captures the semantic meaning of feature sets.

- **For themes with textual features (Hotel):** Combine multiple node properties into a single feature vector (*e.g., concatenate hotel name, location, amenities, average rating, review text*). Embed this combined representation.
- **For themes without textual features (FPL, Airline):** You may construct text descriptions from numerical properties, then embed them (*e.g., "Player: X, Position: Y, Points: Z" or "Journey: X, Class: Y, Food: Z, Delay: W"*). Or use the numerical features as they are directly.

3. LLM Layer

a. Combine the KG results from both the baseline and the embeddings

Merge the results from Cypher queries (baseline) and embedding-based retrieval into a unified context. This provides both structured and semantic information to the LLM. Combine retrieved nodes, relationships, and data from both methods. Remove duplicates and rank/prioritize results if needed.

b. Use structure prompt: context, persona, task

Structure your LLM prompt with three components:

- **Context**: The retrieved KG information (nodes, relationships, data)
- **Persona**: Define the assistant's role (e.g., "You are a helpful travel assistant" for Hotel, "You are an FPL expert" for FPL, "You are a flight information assistant" for Airline)
- **Task**: Clear instructions on what to do with the context (e.g., "Answer the user's question using only the provided information")

This structured approach improves answer quality and reduces hallucinations by explicitly grounding the LLM in the KG data.

c. You must compare at least three models (examples)

Test your system with at least three different LLMs to evaluate performance differences. Examples include GPT-3.5, GPT-4, Claude, Gemini (However, their APIs are paid, so be careful if you choose these models), or open-source models like Llama, Mistral, Gemma. Compare their accuracy, response quality, and cost. Use free models from HuggingFace or OpenRouter for prototype development.

d. The comparison must include qualitative and quantitative impressions

Evaluate models using both:

- **Quantitative**: Metrics like accuracy, response time, token usage, cost
- **Qualitative**: Human evaluation of answer quality, relevance, naturalness, and correctness

Create test cases and measure how well each model answers questions using the KG context. Document which model performs best for your use case.

4. Build a UI (e.g., Streamlit)

Create a user interface that demonstrates your Graph-RAG system in action. Streamlit is recommended for its simplicity. The UI should be functional and intuitive, but doesn't need to be complex. Focus on demonstrating the system's capabilities.

Select **one task or more** (*QA, booking assistant, recommender, etc.*). The UI should allow a user to:

a. View the KG-retrieved context

Display the raw information retrieved from the knowledge graph before it's processed by the LLM. Show users what nodes, relationships, and data were found. This increases transparency and trust.

b. View the final LLM answer

Display the LLM's final response to the user's query. The main output is the answer generated using the KG context.

c. Optionally display:

- Cypher Queries Executed:
 - Show the **actual** Cypher queries that were run to retrieve information. This helps users understand how the system found the information. Useful for debugging and transparency.
- Graph Visualization Snippets:
 - Visualize the retrieved subgraph (nodes and relationships) in a graph format.
 - Use libraries like NetworkX, Plotly, or Neo4j's built-in visualization to show the graph structure.
- Recommendations:
 - If implementing a recommender, display ranked recommendations with explanations.
 - Show why certain entities were recommended based on user preferences and KG data:
 - **Hotel:** hotel recommendations with explanations

- **FPL:** player recommendations for fantasy teams
 - Model Selection Dropdown (to switch between LLMs):
 - Allow users to select which LLM to use for generating answers.
 - Enables real-time comparison of different models' responses to the same query.
 - Retrieval Method Selection (baseline, embeddings, or both): Allow users to choose which retrieval method(s) to use. Enables comparison of baseline Cypher queries vs. embedding-based retrieval vs. hybrid approaches.
-

Experiments, Results, & Improvements

Your final presentation must include:

- **System Architecture:** Document the overall system design, components, and how they interact (preprocessing → retrieval → LLM → UI). Include architecture diagrams for each theme.
- **Retrieval Strategy and Examples:** Explain your Cypher queries and embedding approach. Show example queries and their results for each theme. Document both baseline and embedding-based retrieval strategies.
- **LLM Comparison:** Present quantitative metrics and qualitative analysis comparing the three+ LLM models you tested.
- **Error Analysis:** Document cases where the system failed, why it failed, and how it could be improved. Include theme-specific error cases.
- **Improvements Added:** Describe any enhancements you made to overcome the limitations.
- **Remaining Limitations:** Acknowledge what the system cannot do well and what would need further work. Include limitations specific to each theme.

Choose Your Task

Hotel

- Booking and Visa Assistant
- Hotel Recommender System

Airline

- Hybrid Travel Assistant
- Airline Company Flight Insights Assistant

FPL

- Team Formulation Recommender System
- FantasyTrivia

Select **one or more** tasks to implement. A hybrid assistant that combines multiple capabilities is recommended for demonstrating the full potential of Graph-RAG.

Note:

- *For this project, Question Answering System is the primary task, with the option to extend to other tasks.*
 - *You are free to define any creative task/usecase, as long as the core Graph-RAG pipeline is implemented.*
 - *You are not expected to reflect ALL the mentioned options; these are just examples of what you can do. Your design choice will reflect the robustness of your system. Advice: be creative with the task but don't overwhelm yourself with too many features. Start with a simple task and once you're done you can add more features.*
 - *You will be evaluated on your system; integration, design, and functionality, and attempts of improvement. There is no threshold for performance.*
-

Theme-Specific Considerations

- **Hotel Theme**
 - Focus on hotel search, reviews, recommendations, and visa requirements
 - Utilize textual review content for feature vector embeddings
 - Consider traveller demographics and preferences
 - **FPL Theme**
 - Focus on player performance, team analysis, and fantasy recommendations
 - No textual features - You may construct text from numerical properties for embeddings, or use the numerical features directly.
 - Consider seasons, gameweeks, and statistical metrics
 - **Airline Theme**
 - Focus on flight routes, delays, passenger satisfaction, and journey analysis
 - No textual features - You may construct text from numerical properties for embeddings, or use the numerical features directly.
 - Consider airports, routes, and passenger classes
 - The assistant application is from the perspective of the airline company, not from the passengers'. This could be in the form of gaining insights about poorly rated flights for example.
-

Additional Resources:

- [LangChain overview documentation](#)
- [Node embeddings - Neo4j Graph Data Science](#)
- [Sentence Similarity Models – Hugging Face](#)
- [HuggingFace Inference API](#)
- [google/gemma-2-2b-it · Hugging Face](#)
- [LangChain Docs Integration packages](#)
- [Neo4j Documentation](#)
- [Neo4j Vector Index - Docs by LangChain](#)
- [Introduction - Cypher Manual](#)
- [Build applications with Neo4j and Python - Neo4j Python Driver Manual](#)
- [Building A Graph & LLM-Powered RAG Application from PDF Documents](#)
- [Building RAG Applications With the Neo4j GenAI Stack: A Comprehensive Guide](#)
- [LangChain Graph RAG Tutorial](#)

- [Enhanced QA Integrating Unstructured Knowledge Graph Using Neo4j and LangChain](#)
- [GenAI Stack: The GenAI Stack will get you started building your own GenAI application.](#)

Timeline:

- This document is posted in the 3rd of December
 - The deadline to submit this milestone is 15th of December at 23:59
 - Evaluation and presentation will start on the 16th of December
 - Office hours will start on Saturday 6th of December till Thursday 11th of December
 - Office Hours reservation link will be posted on the CMS on Friday 5th of December at 13:00. **First-Come-First-Serve.**
 - Reservation for the evaluation will be posted on Wednesday, 10th of December at 11:00 am, and the deadline to choose your slot will be Friday, 12th of December at 23:59. **Please check the Evaluation and Presentation Guideline Document on the CMS.**
 - The final evaluation schedule will be posted on Sunday, 14th of December.
-

Submission Guidelines:

Please submit your GitHub repository, create a branch named Milestone3, and a **link to your presentation slides**, fulfilling the specified requirements, using the following [form](#) by December 15th at 23:59..

Ensure your repository remains private until the deadline. After the deadline, you will be required to make it public or add the course account (csen903w25-sys) as a collaborator for milestone grading.