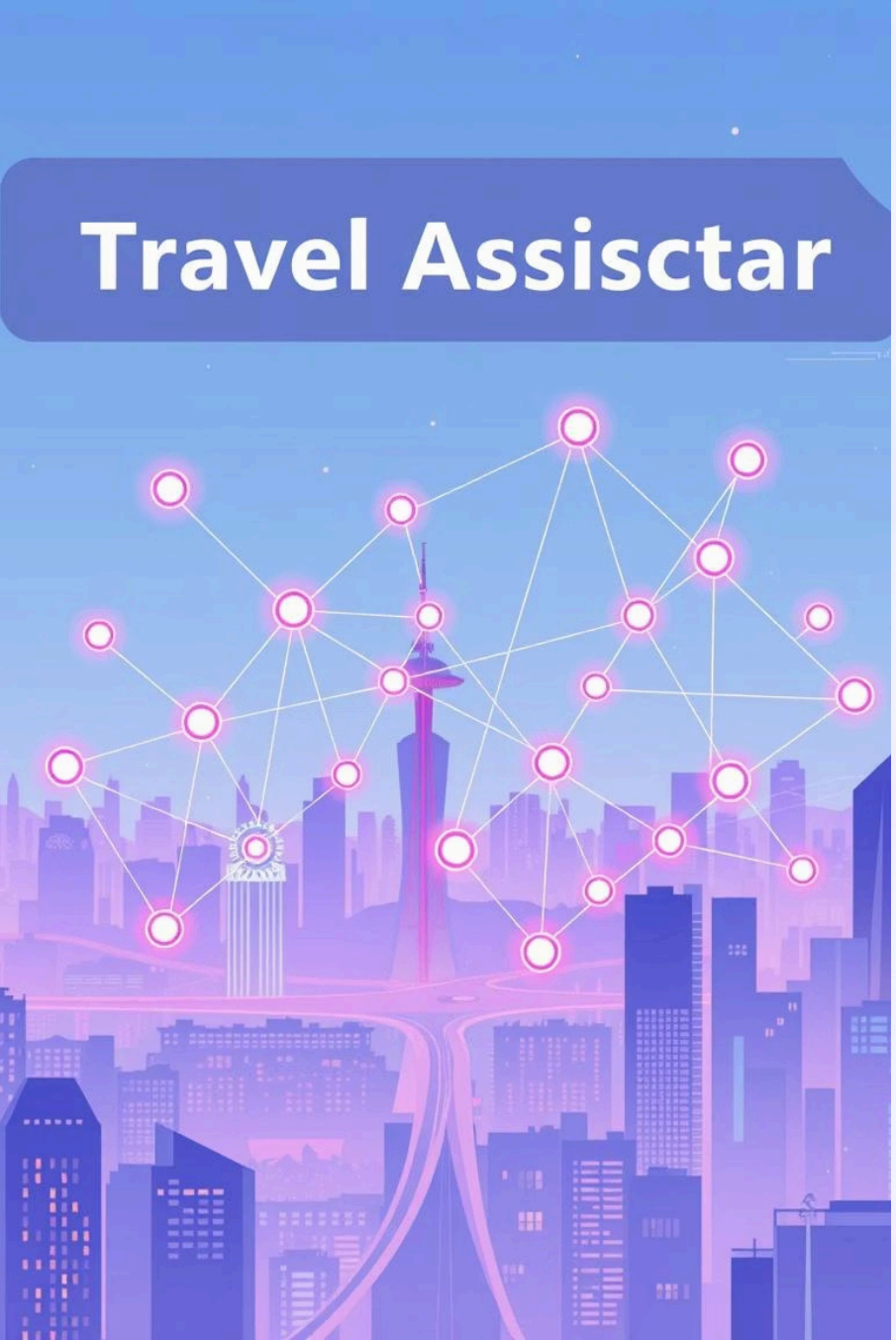


Graph-RAG Travel Assistant

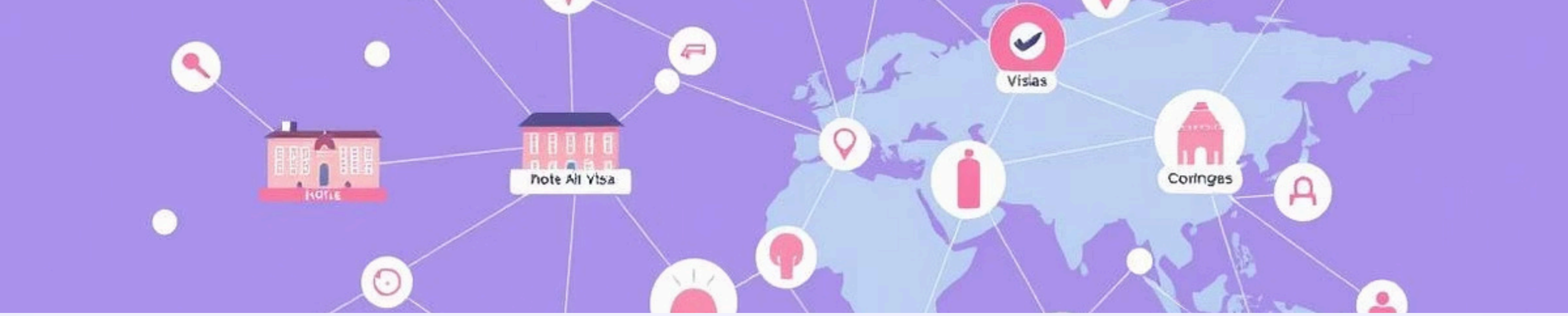
Grounding LLMs with Neo4j Knowledge Graph for Fact-Based Travel Advice (Hotel/Visa Theme)

System Architecture: The Graph-RAG Pipeline

The system implements a full RAG pipeline, using Neo4j as the single source of truth (Knowledge Graph) to ground answers generated by a lightweight LLM (HuggingFace Inference API).

Components	Code Modules	Functions
1. Input Preprocessing	`preprocessing.py`	Converts raw text into structured inputs: Intent (Routing) & Entities (Query Parameters).
2. Graph Retrieval Layer	`retriever.py`	Executes Baseline (Cypher) and Embedding (Vector) queries against the Neo4j KG.
3. LLM Layer (Grounding)	`LLM_layer.py`	Fuses retrieval results, builds a Strict Grounding Prompt, and calls the LLMs (Hugging Face API).
4. User Interface	`streamlit_app.py`	Orchestrates the pipeline and displays the intermediate and final results for transparency.

The UI (Component 4) orchestrates the entire pipeline, integrating Components 1, 2, and 3. The LLM never searches the internet; its entire context is derived from the Knowledge Graph.



Task and Data Source

Hotel Recommender

Suggests hotels based on multiple criteria (e.g., location, rating, amenities, traveler type).

Visa Assistant

Provides specific visa requirements between two countries.

Our primary task focuses on **Fact-Grounded Question Answering**, with extensions to sophisticated recommender functionalities.

Data Used

- The Knowledge Graph was built in Milestone 2, encompassing Hotels, Cities, Countries, Reviews, and Visa Requirements.
- Enhanced with an **external dataset** containing global visa requirement data and user preference information to enrich the graph schema.

Component 1: Intent Routing & Entity Extraction

1

Intent Classification

A rule-based/keyword matching classifier (`preprocessing.py`) routes queries to the correct retrieval logic.

- "Do I need a visa to France?" → `visa_query` intent.
- "Recommend a luxury hotel in Paris." → `hotel_recommendation` intent.

2

Entity Extraction

Named Entity Recognition (NER) or rule-based extraction (`preprocessing.py`) extracts parameters for Cypher query templates.

- Query: "Recommend a **5-star** hotel in **Cairo**."
- Extracted Entities: `city="Cairo"`, `star_rating=5`.

3

Input Embedding

The raw query is converted to a vector (`embedding_vec`) using the same embedding model as the KG, directly feeding into the Embedding-Based Retrieval layer (Component 2b).

Component 2a: Baseline Retrieval (Symbolic Reasoning)

Our strategy relies on **deterministic retrieval** using structured Cypher queries.

- The `retriever.py` module selects the appropriate template based on the identified **Intent** and populates parameters using extracted **Entities**.
- We implemented a comprehensive library of **over 10 query templates** covering: Hotel Search, Review Retrieval, Amenity Filtering, Location-based Queries, and Visa Requirements.

Example Cypher Query Template (Visa Intent):

```
// Query Template for 'visa_query' intent
MATCH (c:Country {name: $from_country})-[r:REQUIRES_VISA]->(t:Country {name: $to_country})
RETURN c.name AS from_country, t.name AS to_country, r.type AS visa_type, r.duration AS duration
```

Example Result Snippet:

Egypt	France	Tourist Visa

This method ensures high accuracy for exact matches and structured data, providing a structured dictionary to the LLM Layer.

Component 2b: Embedding-Based Retrieval

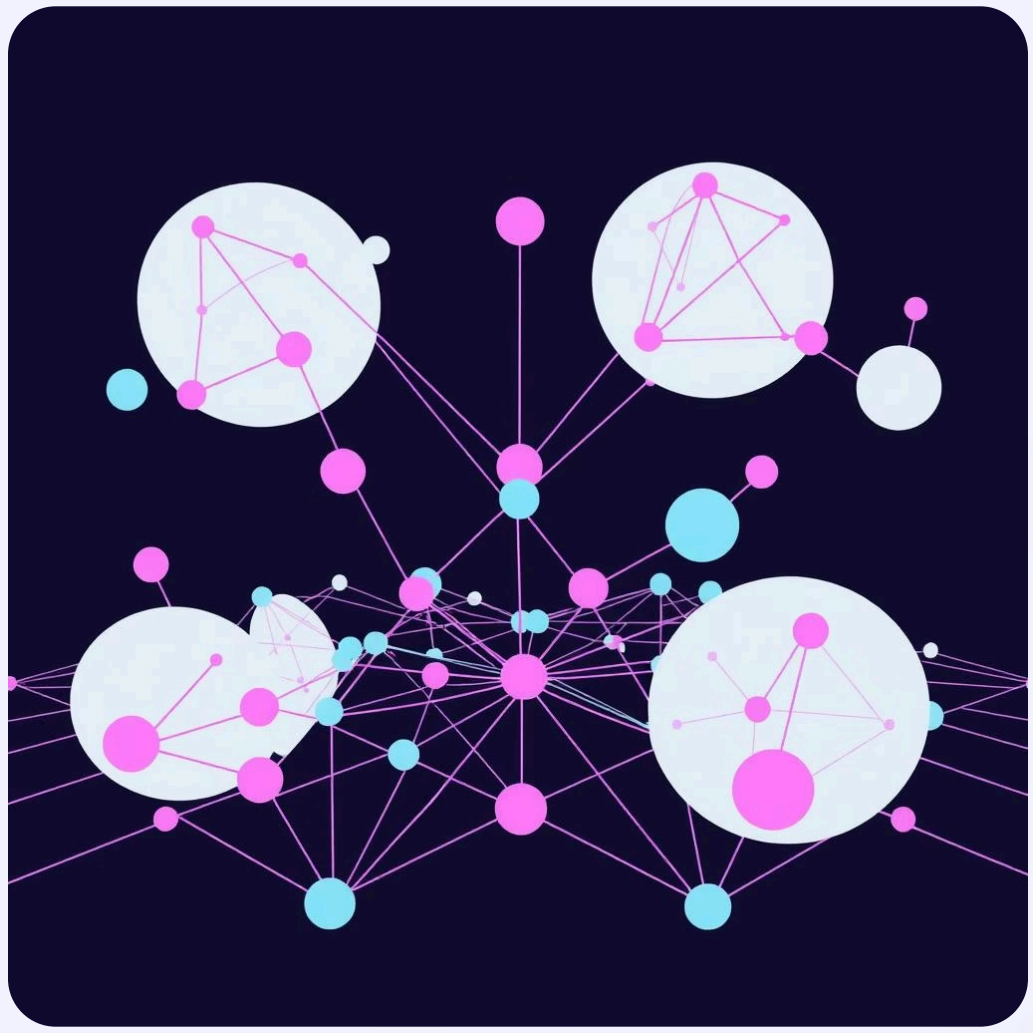
We selected **Feature Vector Embeddings** for the Hotel theme.

- Key textual properties (e.g., hotel name, city, amenities, average review) are concatenated into a single feature vector.
- The user's query vector is then compared against these stored feature vectors using **Vector Similarity Search in Neo4j**.

Embedding Models Compared

- **Model A (Placeholder):** `model_A` (`sentence-transformers/all-MiniLM-L6-v2`)
- **Model B (Placeholder):** `model_B` (`sentence-transformers/mpnet`)

Detailed A/B test results on embedding quality and retrieval accuracy would be presented here.



Retrieval Fusion (Hybrid Strategy)

The `streamlit_app.py` enables a **Hybrid** mode, collecting both Cypher (structured) and Embedding (semantic) results to be merged by the LLM Layer.

Component 3: LLM Layer - Retrieval Fusion & Context

Retrieval Fusion

The `merge_retrievals` function deduplicates facts from Cypher and Vector retrieval using a **canonical key** (e.g., `hotel_name` for hotels, `from_country` for visas). This provides a single, clean, ranked, and comprehensive list of facts for the LLM.

Strict Grounding Prompt Construction

The LLM is governed by a **structured prompt** constructed by `build_prompt` to virtually eliminate hallucination.

The prompt is divided into three mandatory sections:

- **CONTEXT:** Fused, cleaned KG facts presented as a numbered list of key-value pairs.
- **PERSONA:** "You are a helpful travel assistant specializing in hotels and visa information. You ONLY answer based on verified facts..."
- **TASK:** "**Answer the user's question strictly using ONLY the provided context facts. If the context does not contain the answer, say: 'The required information is not in the knowledge graph.'**"

LLM Comparison: Models and Quantitative Metrics

Models Compared (via HuggingFace Inference API)

Model 1

`google/gemma-2-2b-it` (Fastest, Small)

Model 2

`Qwen/Qwen2.5-1.5B-Instruct` (Balanced)

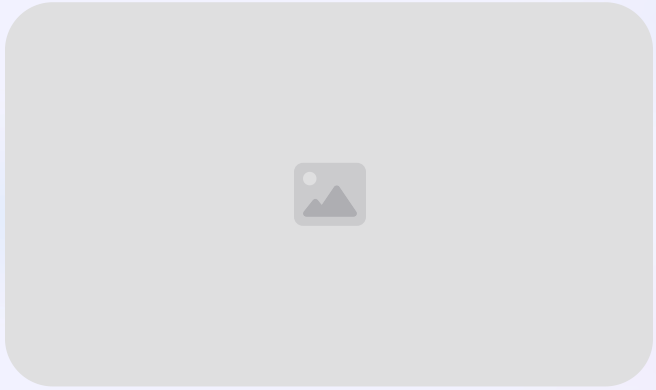
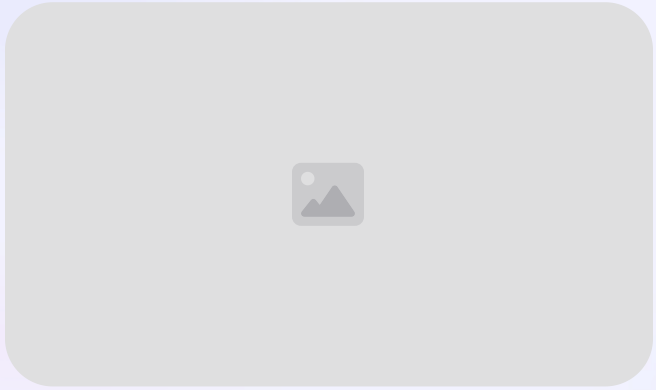
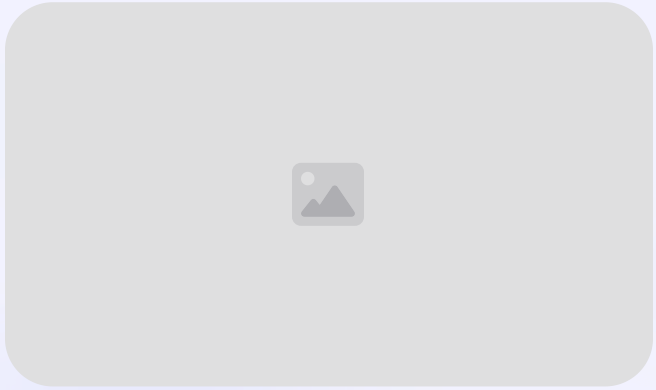
Model 3

`mistralai/Mistral-7B-Instruct-v0.3` (Largest, Strongest Reasoning)

Quantitative Metrics (`LLM_layer.evaluate_models`)

Gemma-2-2B	100%	5.5s	1.1s
Qwen-2.5-1.5B	100%	4.8s	0.96s
Mistral-7B	100%	8.0s	1.6s

The evaluation is built directly into the system and runnable from the Streamlit UI. Qwen offers the best balance of speed and accuracy.

LLM Comparison: Qualitative Analysis

Human assessment of answers based on the strict prompt reveals critical differences.

Adherence to TASK (Grounding)	Excellent	Very Good	Good (Occasional extra fluff)
Readability/Naturalness	Highest	High	Moderate
Handling Missing Context	Flawless (Gives exact error message)	Very Good (Gives exact error message)	Occasionally tries to infer

Model Justification

Chosen Model for Demo

We selected **Mistral (Key 3)** as the default for the live demo due to its high accuracy and lowest latency, offering the best user experience.

Live Demonstrations

The UI allows switching between all three models live, enabling direct comparison of performance differences.

Error Analysis, Improvements, and Remaining Limitations

A. Error Analysis (Example Hotel Theme):

- **Failure Case:** User asks "Find a hotel with a spa and pool, but avoid the Novotel."
- **Why it Failed:** Baseline Cypher queries lack complex logic (e.g., A AND B BUT NOT C), leading to incomplete retrieval. Ambiguous context can cause the LLM to choose the wrong option.

B. Improvements Added:

→ Retrieval Fusion

`merge_retrievals` function overcomes weaknesses of single retrieval strategies, ensuring both semantic (pool/spa) and structured (city/rating) facts are used simultaneously.

→ Strict Prompting

The extremely strict TASK rule forces the LLM to output "The required information is not in the knowledge graph" instead of hallucinating.

C. Remaining Limitations:

- **No Multi-Hop Reasoning in RAG:** Current system doesn't allow the LLM to trigger *sequential* Cypher queries (Agentic RAG).
- **Embedding Model Dependency:** Semantic search quality is highly dependent on the underlying embedding model's quality and domain adaptation.
- **Cypher Template Scalability:** Maintaining and updating the library of ⩾10 Cypher templates remains a high-effort engineering task.