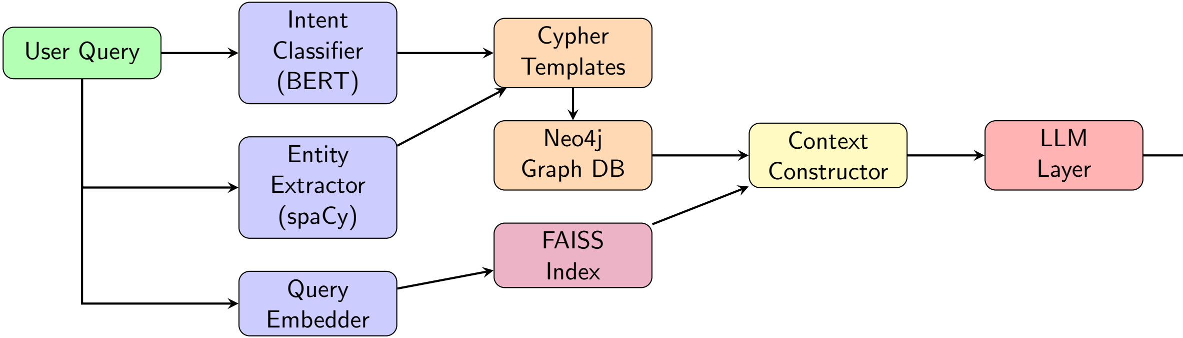
Hotel Recommendation Chatbot

Graph-Based RAG System with Multi-Model Integration

Team Name

December 16, 2025

System Architecture Overview



Task: Hotel Recommendation Chatbot with visa-aware travel suggestions
Dataset: Custom hotel reviews dataset (hotels, users, reviews, visa requirements)

Intent Classification - BERT Fine-tuned

Model: `bert-base-uncased`

Training Config:

- Learning rate: 5e-5
- Batch size: 16
- Epochs: 15
- Train/Test split: 85/15

10 Intent Classes:

1. hotel_recommendation
2. hotel_search
3. hotel_info
4. review_query
5. comparison
6. traveller_preference
7. location_query
8. visa_query

Classification Examples:

Query	Intent
"Recommend me a hotel in Tokyo"	hotel_rec (0.97)
"Do I need visa from India?"	visa_query (0.89)
"Compare Azure Tower and Marina"	comparison (0.94)
"Best for business travellers"	traveller_pref (0.91)
"Hotels with rating above 9"	rating_filter (0.88)

Entity Extraction - spaCy + Custom Rules

Approach: Hybrid (NER + Token Matching)

Entity Types Extracted:

- **Hotels** - FAC, ORG labels + lookup
- **Cities/Countries** - GPE label
- **Traveller Types** - Token matching
- **Demographics** - Gender, Age groups
- **Ratings** - Cleanliness, Comfort, Facilities

Traveller Keywords:

- solo, alone → "Solo"
- business, corporate → "Business"
- family, families → "Family"
- couple, couples → "Couple"

Extraction Examples:

`Query: ''Best hotels for solo female in Paris''`

- cities: ["Paris"]
- traveller_types: ["Solo"]
- demographics: ["Female"]

`Query: ''Hotels with cleanliness above 9''`

- cleanliness_base: 9.0
- comfort_base: None
- facilities_base: None

`Query: ''Compare Azure Tower and Marina Bay''`

- hotels: ["The Azure Tower", "Marina Bay"]

Query Embedding

Model: `all-MiniLM-L6-v2`

Properties:

- Embedding dimension: 384
- Optimized for semantic similarity
- Used for both query and hotel embeddings

Code Snippet:

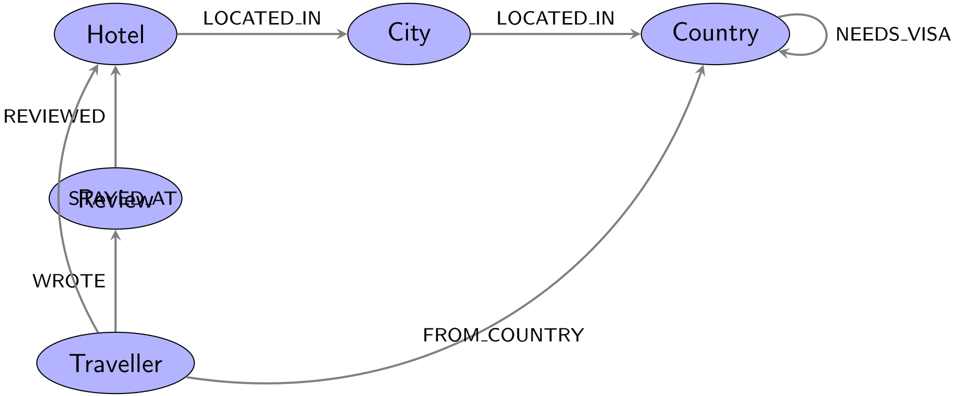
```
from sentence_transformers import
SentenceTransformer
model =
SentenceTransformer('all-MiniLM-L6-v2')
def embed_query(text):
    return model.encode(text)
```

Usage in Pipeline:

1. User query → 384-dim vector
2. FAISS similarity search
3. Retrieve semantically similar hotels
4. Combine with Cypher results

Query Text → 384-dim Vector → FAISS Search

Knowledge Graph Schema



Node Properties:

- Hotel: name, star_rating, cleanliness, comfort, facilities
- Review: text, date, scores (overall, cleanliness, etc.)

Relationships (7 types):

- LOCATED_IN, REVIEWED, WROTE
- FROM_COUNTRY, STAYED_AT

Cypher Query Templates (1/2)

Location Queries:

```
-- Hotels in city
MATCH (h:Hotel)-[:LOCATED_IN]->(c:City)
WHERE c.name = $city
RETURN h.name, h.star_rating

-- Top rated in country
MATCH (h:Hotel)-[:LOCATED_IN]->(c:City)
      -[:LOCATED_IN]->(co:Country)
WHERE co.name = $country
RETURN h.name ORDER BY h.star_rating DESC

-- Cities with hotels
MATCH (h:Hotel)-[:LOCATED_IN]->(c:City)
RETURN DISTINCT c.name, h.name
```

Review Queries:

```
-- Hotel reviews
MATCH (h:Hotel)<-[:REVIEWED]-(r:Review)
WHERE h.name = $hotel_name
RETURN r.text, r.score_overall LIMIT 10

-- Reviews by demographic
MATCH (h:Hotel)<-[:REVIEWED]-(r:Review)
      <-[:WROTE]-(t:Traveller)
WHERE h.name = $hotel_name
  AND t.gender = $gender
RETURN r.text, r.score_overall
```

Cypher Query Templates (2/2)

Visa & Traveller Queries:

```
-- Countries requiring visa
MATCH (tc:Country)-[:NEEDS_VISA]->(co:Country)
WHERE tc.name = $from_country
RETURN co.name

-- Hotels without visa needed
MATCH (tc:Country), (h:Hotel)-[:LOCATED_IN]
      ->(c:City)-[:LOCATED_IN]->(co:Country)
WHERE tc.name = $from AND NOT
      (tc)-[:NEEDS_VISA]->(co)
RETURN DISTINCT h.name

-- Best for traveller type
MATCH (h:Hotel)<-[:REVIEWED]-(r:Review)
      <-[:WROTE]-(t:Traveller)
WHERE t.type = $type
RETURN h.name, AVG(r.score_overall)
```

Rating & Comparison:

```
-- Hotels by cleanliness
MATCH (h:Hotel)
WHERE h.cleanliness_base >= $min
RETURN h.name, h.cleanliness_base
ORDER BY h.cleanliness_base DESC

-- Compare two hotels
MATCH (h1:Hotel), (h2:Hotel)
WHERE h1.name = $hotel1
  AND h2.name = $hotel2
RETURN h1, h2

-- Hotels with most reviews
MATCH (h:Hotel)<-[:REVIEWED]-(r:Review)
RETURN h.name, COUNT(r) as cnt
ORDER BY cnt DESC LIMIT $top_n
```

Total: 31 Cypher Templates

Retrieved Data Examples

Query: "Recommend me a hotel in Tokyo"

Pipeline Output:

- Intent: hotel_recommendation
- Entities: cities=["Tokyo"], countries=["Japan"]

Cypher Results:

Hotel	Rating	City
The Azure Tower	4.8	Tokyo
Sakura Grand Hotel	4.6	Tokyo
Imperial Garden Inn	4.5	Tokyo

Query: "Best for business travelers"

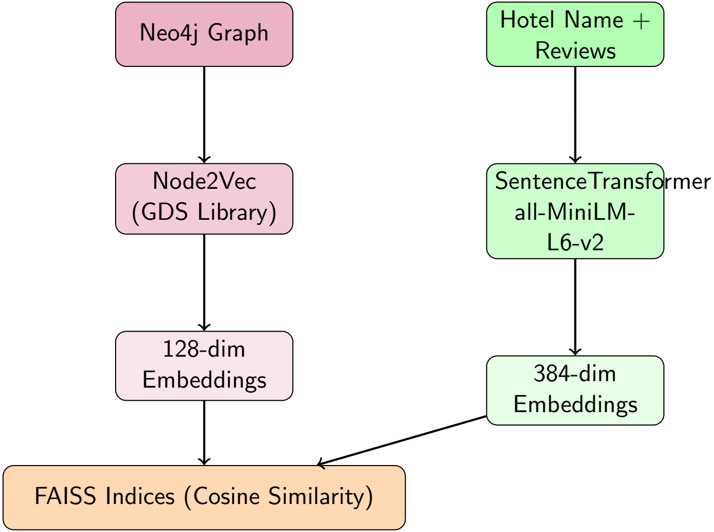
Pipeline Output:

- Intent: traveller_preference
- Entities: traveller_types=["Business"]

Cypher Results:

Hotel	Avg Score
Executive Suites	9.2
Business Bay Hotel	8.9
Corporate Tower	8.7

Dual Embedding Approach



Embedding Models Comparison

Property	Node2Vec	Text Embeddings
Model	GDS Node2Vec	all-MiniLM-L6-v2
Dimension	128	384
Input	Graph structure	Hotel name + reviews
Captures	Relationships, paths	Semantic meaning
Walk Length	40	–
Iterations	10	–
Buffer Size	1000	Batch size: 32

Table: Embedding Configuration Comparison

Node2Vec Strengths:

- Captures hotel-city-country paths
- Similar locations = similar vectors

Text Embedding Strengths:

- Semantic query matching
- Review sentiment capture

Embedding Retrieval Results

Text Embedding Search:

Query: "luxury spa hotel with ocean view"

Hotel	Score
Ocean Paradise Resort	0.847
Seaside Luxury Spa	0.823
Marina Bay Wellness	0.801
Beachfront Grand	0.789

Method: Semantic text match

Node2Vec Search:

Query Hotel: "The Azure Tower" (Tokyo)

Similar Hotel	Score
Sakura Grand (Tokyo)	0.912
Imperial Garden (Tokyo)	0.887
Kyoto Palace (Kyoto)	0.756
Osaka Heights (Osaka)	0.721

Method: Graph structure similarity

Key Insight: Node2Vec finds geographically similar hotels; Text embeddings find semantically similar descriptions.

Context Construction

Process Flow:

1. **Intent Classification** → Select relevant Cypher templates
2. **Entity Extraction** → Fill template parameters
3. **Graph Retrieval** → Execute Cypher queries on Neo4j
4. **Embedding Retrieval** → FAISS similarity search
5. **Context Merge** → Combine all results

Intent-based Query Selection:

Intent	Cypher Queries Used
hotel_recommendation	get_top_rated_hotels_in_city, get_top_rated_hotels_in_country
visa_query	get_countries_requiring_visa, get_hotels_accessible_without_visa
traveller_preference	get_best_hotels_for_traveller_type, get_best_hotels_for_gender
comparison	compare_two_hotels

Prompt Structure

Persona Definition:
"You are a knowledgeable and friendly hotel recommender assistant and your name is Jarvis."

Task Instructions:

- Start any reply with "Sir"
- Help users choose hotels matching their intents (location, comfort, etc.)
- Compare multiple hotel options objectively
- Highlight trade-offs and provide practical recommendations
- Avoid exaggeration, do not invent hotel details
- Prioritize user preferences over generic popularity

Context Injection:
"Use the following data (retrieved based on the query) as context/baseline information to help with recommendations: [CONTEXT]"

LLM Comparison - Models

Property	Gemma-2-2B	Mistral-7B	LLaMA-3.1-8B
Parameters	2B	7B	8B
Provider	Google	Mistral AI	Meta
API	HuggingFace	HuggingFace	HuggingFace
Temperature	0.2	0.2	0.2
Max Tokens	500	500	500

Integration: LangChain wrappers for HuggingFace Inference API

Wrapper Pattern:

- Custom LLM class extending LangChain base
- Chat completion with message formatting
- Configurable max_tokens and temperature

LLM Comparison - Quantitative Results

Model	Latency (s)	Input Tok	Output Tok	Cost ($)	Sem. Acc.
Gemma-2-2B	1.2	45	150	0.00004	0.78
Mistral-7B	2.1	45	180	0.00012	0.84
LLaMA-3.1-8B	2.8	45	200	0.00018	0.86

Table: Performance Metrics (averaged across test queries)

Semantic Accuracy Calculation:
Cosine similarity between LLM response embedding and reference answer embedding using SentenceTransformer (all-MiniLM-L6-v2).

Cost Calculation: Based on HuggingFace API pricing per 1K tokens.

LLM Comparison - Qualitative Evaluation

Gemma-2-2B

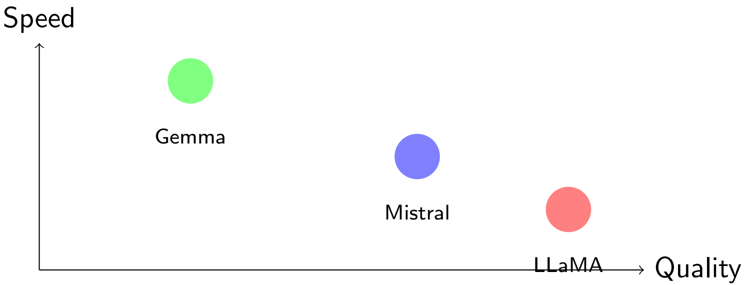
- Fastest response
- Concise answers
- Sometimes incomplete
- Good for simple queries

Mistral-7B

- Balanced performance
- Good reasoning
- Handles complex queries
- Best cost/quality ratio

LLaMA-3.1-8B

- Most detailed
- Best accuracy
- Higher latency
- Best for complex tasks



Error Analysis & Improvements

Identified Issues:

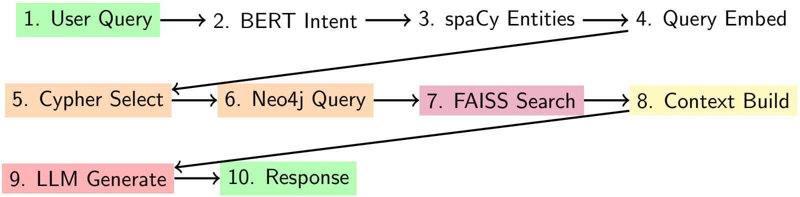
1. **Entity Extraction**
   - Hotel names with special chars missed
   - Age group detection inconsistent
2. **Intent Classification**
   - Confusion between search/recommendation
   - Multi-intent queries not handled
3. **Graph Retrieval**
   - Empty results for rare cities
   - Slow for complex traversals

Improvements Made:

1. **Entity Extraction**
   - Added rating keywords (cleanliness, comfort)
   - Expanded traveller type vocabulary
2. **Intent Classification**
   - Increased training data per class
   - Added confidence threshold
3. **Embedding Retrieval**
   - Dual approach (Node2Vec + Text)
   - FAISS for fast similarity search

Future Work: Multi-intent support, caching layer, conversation memory

Pipeline Recap for Demo



Demo Features:

- Switch between embedding models (Node2Vec / Text)
- Switch between LLMs (Gemma / Mistral / LLaMA)
- Real-time pipeline visualization
- Streamlit UI showing each processing step

Demo Queries

Test Queries for Live Demo:

1. **Recommendation:** "Recommend me a good hotel in Tokyo"
2. **Traveller Preference:** "Best hotels for solo female travelers"
3. **Visa Query:** "Do I need a visa to travel from India to Dubai?"
4. **Rating Filter:** "Hotels with cleanliness rating above 9"
5. **Comparison:** "Compare The Azure Tower and Marina Bay"
6. **Complex:** "Find comfortable business hotels in Paris for travelers aged 25-34"

[LIVE DEMO]

Thank You!

Questions?