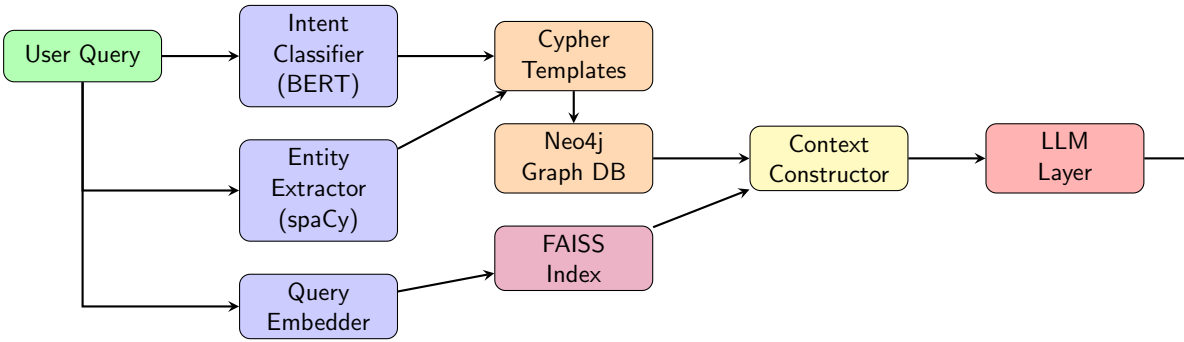
Hotel Recommendation Chatbot

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 travelers"	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 (Feature-Based Model)

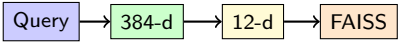
Text Encoder: `all-MiniLM-L6-v2`
Text-to-Feature Mapping:

- Query → 384-dim text embedding
- Ridge regression mapper
- Maps to 12-dim feature space
- Trained on hotel descriptions

Training Data:
"{name} hotel in {city}, {country}. {stars} star hotel with {reviews} reviews. Average score {avg}. Cleanliness {clean}..."

Query Processing Pipeline:

1. Extract location filter (city/country)
2. Extract star rating (luxury→5-star)
3. Encode query → 384-dim
4. Map to feature space → 12-dim
5. FAISS similarity search
6. Apply name boost (synonyms, fuzzy)
7. Apply attribute boost (clean, comfort)
8. Re-rank and filter results



Knowledge Graph Schema - Complete Overview

Node Types & Properties:

Node	Properties
Hotel	hotel_id, name, star_rating, avg_reviewer_score, review_count, avg_cleanliness, avg_comfort, avg_facilities, avg_location, avg_staff, avg_value
City	name
Country	name
Review	review_id, text, date, score_overall, score_cleanliness, score_comfort, score_facilities, score_location, score_staff, score_value
Traveller	traveller_id, age_group, gender, traveller_type

Relationships (7 types):

Relationship	Pattern
LOCATED_IN	Hotel → City
LOCATED_IN	City → Country
REVIEWED	Review → Hotel
WROTE	Traveller → Review
STAYED_AT	Traveller → Hotel
FROM_COUNTRY	Traveller → Country
NEEDS_VISA	Country → Country

Graph Statistics:

- 25 Hotels across 25 Cities
- 24 Countries with visa relations
- 500+ Reviews with scores
- 500+ Travellers with demographics

Key Design: Hotels enriched with computed averages from reviews for embedding features

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"

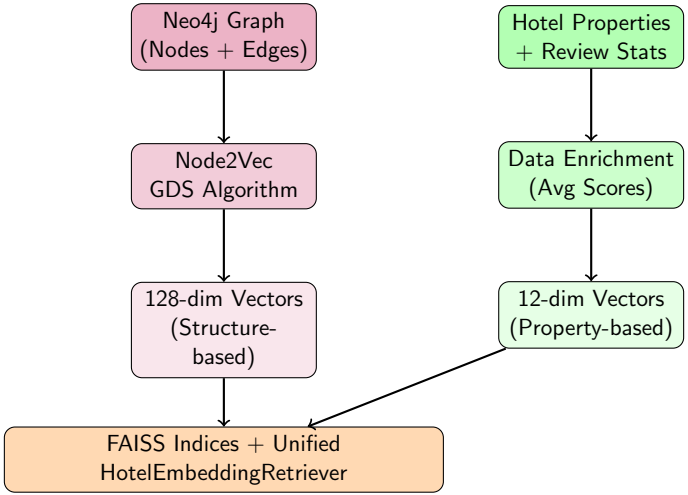
Pipeline Output:

- 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



Implementation: Two embedding models with unified black-box interface

Embedding Implementation Details

Node2Vec (Graph Structure):

- Uses Neo4j Graph Data Science (GDS)
- Random walks explore graph topology
- Parameters: walkLength=40, iterations=10
- Learns from node connectivity patterns
- All 5 node types embedded together

How it works:

1. Project graph to GDS catalog
2. Run node2vec.stream() algorithm
3. Extract 128-dim vectors per node
4. Build FAISS index for hotels only
5. Store hotel_id $\leftrightarrow$ index mapping

Feature-Based (Property Vectors):

- Aggregates review scores per hotel
- Computes: avg_cleanliness, avg_comfort, avg_facilities, avg_location, avg_staff, avg_value
- Normalizes features to [0,1] range
- Creates 12-dimensional feature vector

12 Feature Dimensions:

1. hotel_id (normalized)
2. star_rating
3. review_count (log-scaled)
4. avg_reviewer_score
5. avg_cleanliness
6. avg_comfort
7. avg_facilities
8. avg_location
9. avg_staff
10. avg_value

Embedding Models Comparison

Property	Node2Vec	Feature-Based
Algorithm	GDS Node2Vec	Property Aggregation
Dimension	128	12
Input	Graph structure (5 node types)	Hotel properties + review stats
Captures	Relationship paths, connectivity	Ratings, scores, computed averages
Walk Length	40	–
Iterations	10	–
Features	–	star_rating, avg_cleanliness, avg_comfort, avg_facilities, avg_location, avg_staff, avg_value

Table: Embedding Configuration Comparison

Node2Vec Strengths:

- Captures hotel-city-country paths
- Similar locations → similar vectors
- Traveller connection patterns
- Fast similarity lookup (4ms)

Feature-Based Strengths:

- Natural language query support
- Location/star rating filtering
- Attribute-based ranking ("best cleanliness")
- Synonym + fuzzy matching

Embedding Retrieval Results - Similar Hotels

Node2Vec Search:

Query Hotel: "Berlin Mitte Elite"

Similar Hotel	Score
Colosseum Gardens (Rome)	0.686
Aztec Heights (Mexico City)	0.663
Table Mountain View (Cape Town)	0.587

Finds: Hotels with similar graph connectivity patterns (shared traveller demographics, review patterns)

Feature-Based Search:

Query Hotel: "Berlin Mitte Elite"

Similar Hotel	Score
The Kiwi Grand (Wellington)	0.995
Han River Oasis (Seoul)	0.979
Kremlin Suites (Moscow)	0.977

Finds: Hotels with similar ratings and review scores (5-star, high cleanliness, similar avg scores)

Key Insight: Node2Vec captures structural similarity (traveller patterns); Feature-based captures property similarity (ratings/scores).

Embedding Retrieval Results - Query Search

Query: "best hotels in cleanliness"

Hotel	Clean	Score
Kyo-to Grand	9.39	1.090
The Maple Grove	9.26	1.249
Canal House Grand	9.23	1.284

Feature: Ranking query detection
Detects "best" + "cleanliness" → sorts by avg_cleanliness DESC

Query: "luxury hotel in Paris"

Hotel	Boost
L'Etoile Palace (Paris)	synonym:luxury→palace

Features Applied:

- Location filter: city = "Paris"
- Star rating filter: 5-star (luxury)
- Synonym matching: luxury→palace

Query Enhancements: Location filtering, star rating extraction, synonym matching, fuzzy typo detection, attribute boosting

Embedding Models - Quantitative Comparison

Metric	Node2Vec	Feature-Based	Winner
Embedding Dimension	128	12	Feature (compact)
Index Build Time	∼2s	∼0.5s	Feature
Query Latency (similar)	2-4ms	2-10ms	Node2Vec
Query Latency (NL search)	2-4ms	12-36ms	Node2Vec
NL Query Support	Keyword only	Full semantic	Feature
Location Filtering	No	Yes	Feature
Ranking Queries	No	Yes	Feature
Synonym/Fuzzy Match	No	Yes	Feature

Table: Performance and Feature Comparison

Recommendation:

- **Node2Vec**: Best for "find similar hotels" based on traveller patterns
- **Feature-Based**: Best for natural language queries with filters and ranking
- **Unified Interface**: HotelEmbeddingRetriever allows runtime model switching

Unified Retriever Interface

Black-Box Design Pattern: Single interface abstracts both embedding models

Simple Function API:

```
from embeddings_retreiver import
    search_hotels, set_model_type

# Default:  feature model
results = search_hotels(
    "luxury hotel in Paris")

# Switch to node2vec
set_model_type('node2vec')
results = search_hotels("beach hotel")
```

Class API (more control):

```
retriever = HotelEmbeddingRetriever(
    driver, model_type='feature')

# Same methods for both models
retriever.search_by_query(query)
retriever.find_similar_hotels(name)

# Runtime model switching
retriever.model_type = 'node2vec'
```

Auto-initialization: Loads existing FAISS index or runs setup if needed

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

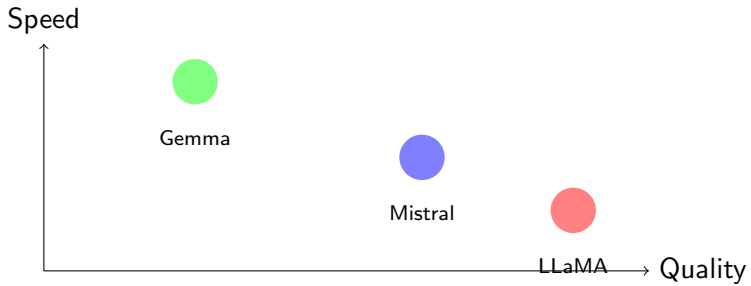
- Fastest response
- Concise answers
- Sometimes incomplete
- Good for simple queries

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:

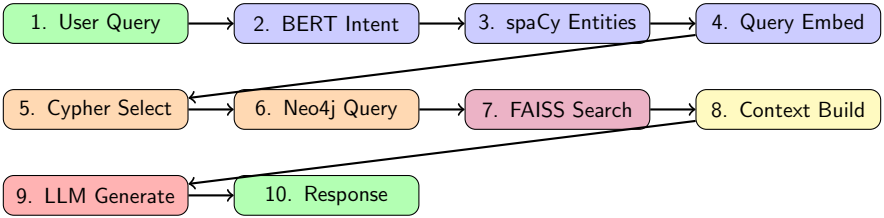
1. **Entity Extraction**
   - Hotel names with special chars missed
   - Age group detection inconsistent
2. **Intent Classification**
   - Confusion between search/recommendation
   - Multi-intent queries not handled
3. **Graph Retrieval**
   - Empty results for rare cities
   - Slow for complex traversals

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?