Hotel Recommendation Chatbot

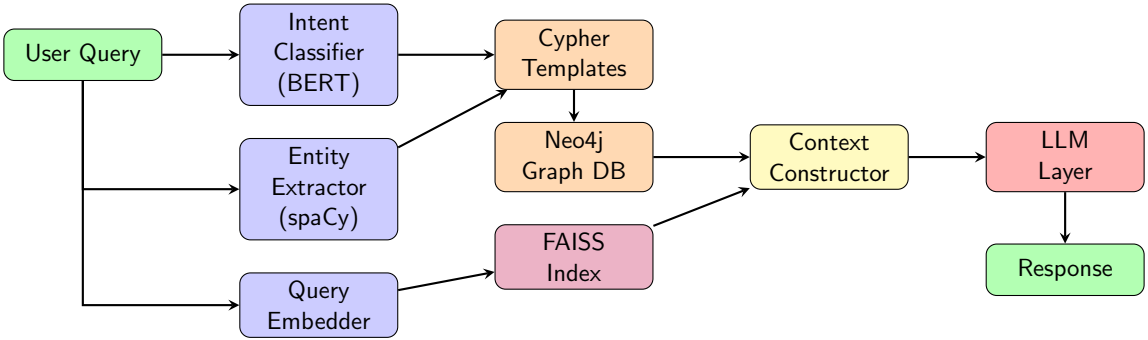
J.A.R.V.I.S.

Team 42

Omar Fouad Jana Elewainy Yara Tantawy Ziad Elgendy

December 16, 2025

System Architecture Overview



Task: Hotel Recommender: J.A.R.V.I.S.

Intent Classification - Fine Tuning a BERT Model

Model: `bert-base-uncased`
Dataset: Custom labeled dataset (200 samples)

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
9. rating_filter
10. general_question

Classification Examples:

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

Entity Extraction

Approach: spaCy NER + keyword lookups

Entity Types Extracted:

- **Hotels** - FAC, ORG labels + lookup
- **Cities/Countries** - GPE label
- **Traveller Types** - Keyword lookup
- **Demographics** - Lookup + DATE
- **Ratings** - Numeric extraction

Keyword Mappings:

- solo, alone → "Solo"
- business → "Business"
- cleanliness → cleanliness_base
- senior → age "55+"

Extraction Examples:

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

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

`''Hotels with cleanliness above 9''`

- cleanliness_base: 9.0

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

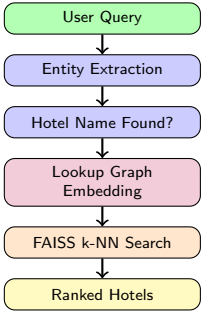
- hotels: ["The Azure Tower", "Marina Bay"]

`''Hotels recommended for seniors''`

- demographics: ["55+"]

Input Embedding

Query-to-Embedding Flow:



Key Insight:

- Graph embeddings encode structure, not text
- Query → Hotel → Embedding → FAISS
- Pre-computed embeddings stored in Neo4j
- Cosine similarity via FAISS IndexFlatIP

Embedding Models:

- Node2Vec (128-dim) - Random walks
- FastRP (128-dim) - Random projection

Function:

`search_by_query(query_text, top_k=5)`

Error Analysis & Limitations

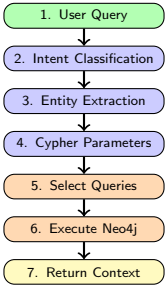
Intent Classification:

- Multi-intent queries not supported ("Find hotels and compare them")
- Limited training data (200 generated samples)
- Out-of-domain queries classified with false confidence
- No intent hierarchy or fallback mechanism

Entity Extraction:

- Hotel names with special characters missed (L'Ã‰toile)
- Relies on predefined keyword lists (not adaptive)
- spaCy NER misses domain-specific entities
- No coreference resolution ("that hotel")

Baseline Retrieval Pipeline



Key Function: `process_query_for_baseline(user_query, driver)`
Flow: Query → Intent → Entities → Params → Cypher → Neo4j → Context

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"

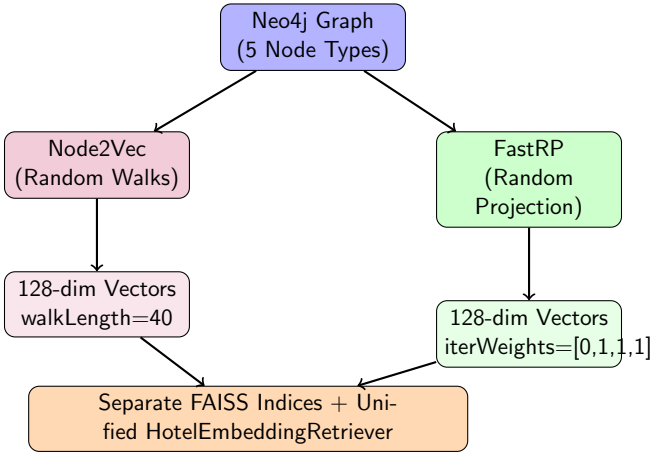
Pipeline Output:

- 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 Node Embedding Approach



Embedding Models Comparison

Property	Node2Vec	FastRP
Algorithm	Random Walks	Random Projection
Dimension	128	128
GDS Function	gds.node2vec.write()	gds.fastRP.write()
Walk Length	40	–
Iterations	10	–
Iter. Weights	–	[0.0, 1.0, 1.0, 1.0]
Computation Time	∼2-3s	∼0.2s
Storage	Neo4j + FAISS	Neo4j + FAISS
Similarity	Cosine (via FAISS)	Cosine (via FAISS)

Node2Vec Strengths:

- Captures higher-order patterns
- Better structural similarity
- Flexible p,q parameters
- More expressive

FastRP Strengths:

- 10x faster (∼0.2s vs ∼2s)
- Memory efficient
- Good for large graphs
- Simpler hyperparameters

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

Method: Random walks capture neighborhood structure and traveller patterns

FastRP Search:

Query Hotel: "Berlin Mitte Elite"

Similar Hotel	Score
The Kiwi Grand (Wellington)	0.712
Han River Oasis (Seoul)	0.698
Kremlin Suites (Moscow)	0.685

Method: Random projections capture immediate neighborhood relationships

Key Insight: Both capture graph structure, but Node2Vec explores deeper paths while FastRP focuses on immediate neighbors.

Embedding Comparison in UI

Side-by-Side Comparison:

The UI displays results from both embedding models simultaneously when a hotel is mentioned or found in context.

Example Query: "Tell me about The Azure Tower"

Node2Vec Results	
Colosseum Gardens	0.739
Tango Suites	0.710
The Royal Compass	0.693

FastRP Results	
Gaudi's Retreat	0.583
Marina Bay Zenith	0.567
The Golden Oasis	0.553

Why Different Results?

- **Node2Vec:** Explores deeper graph paths via random walks
- **FastRP:** Captures immediate neighborhood via projections
- Same hotel can have different "similar" hotels
- Validates both models work independently

UI Features:

- Automatic hotel extraction from query
- Falls back to first hotel in KG context
- Shows rank and similarity score
- Helps evaluate embedding quality

Selected Model: Sidebar dropdown chooses which embedding feeds into LLM context

Error Analysis & Improvement (Embeddings)

Error Analysis:

- Graph embeddings are **not text encoders**
- Cannot convert text queries to vectors
- Requires hotel name extraction first
- Search fails if no hotel found

Improvement Added:

- Fallback when no hotel name:
  - Keyword match on city/country
  - Use first hotel from KG context

Key Limitations:

1. Node embeddings + FAISS don't use Cypher → relationships only captured implicitly via node walks. Both **relationships** and **directionality** are lost.
2. Vector similarity finds fuzzy matches — for precise queries (numbers, names), we must apply custom filters first, then run hybrid retrieval.

LLM Context Construction

Process Flow:

1. **Intent Classification**
   - Classify the intent of user query to select relevant Cypher templates
2. **Entity Extraction**
   - Extract entities from user query to fill in Cypher template parameters
3. **Graph Retrieval**
   - Execute selected Cypher queries on Neo4j database
4. **Context Merge**
   - Combine all results to build the final context for LLM

Intent-to-Query Mapping

Intent	Cypher Queries Selected
hotel_recommendation	get_top_rated_hotels_in_city, get_top_rated_hotels_in_country
hotel_search	get_hotels_by_rating, get_hotels_in_city_by_rating, get_hotels_in_country_by_rating
hotel_info	get_hotel_info
review_query	get_hotel_reviews, get_hotel_review_count, get_latest_hotel_reviews, get_hotel_reviews_filtered
comparison	compare_two_hotels
traveller_preference	get_best_hotels_for_traveller_type, get_best_hotels_for_gender, get_best_hotels_for_age_group
location_query	get_hotels_in_city, get_hotels_in_country, get_cities_in_country
visa_query	get_countries_requiring_visa, get_hotels_accessible_without_visa
rating_filter	get_hotels_by_cleanliness_base, get_hotels_by_comfort_base, get_hotels_by_facilities_base
general_question	get_all_hotels

Total: 31 Cypher query templates mapped to 10 intents

Prompt Structure

Persona Definition:

"You are J.A.R.V.I.S., an advanced AI hotel concierge assistant. You speak with sophisticated eloquence and always address the user as 'sir' or 'madam'. Your responses are warm yet professional, helpful and conversational - never robotic or formulaic. NEVER start responses with phrases like 'Based on the data provided' or 'According to the information'. Instead, speak naturally as if you personally know these hotels and are offering genuine recommendations."

Key Instructions:

- Address the user respectfully as "sir" at least once
- Be conversational and natural, as if having a pleasant discussion
- Provide specific details from the data without mentioning "the data" or "the context"
- Use the Hotel information data available below (that was retrieved based on the query) as context/baseline information to help with the recommendations.
- If recommending hotels display ranked recommendations with explanations. Show why certain entities were recommended based on user preferences and KG data
- Keep responses concise but informative

Context Injection

Baseline Context Injection:

- Hotel information from KG retrieval (Cypher queries) is appended to the prompt

Embeddings Context:

- LLM is given a retriever that performs similarity search on embeddings
- Finds hotels similar to the one mentioned in query

Combined Approach: Both baseline (explicit KG data) and embedding (implicit similarity) contexts are merged to provide comprehensive information to the LLM.

LLM Comparison - Quantitative Results

Model	Latency (s)	Input Tok	Output Tok	Cost ($)	Sem. Acc.
Gemma-2-2B	0.51	6	31	0.000007	0.057
Mistral-7B	10.60	6	211	0.000128	0.037
LLaMA-3.1-8B	4.63	6	382	0.000307	0.252

Table: Performance Metrics (averaged across test queries)

Semantic Accuracy Calculation:

- **Method:** Cosine similarity (SentenceTransformer) between LLM response and reference embeddings
- **Test Query:** "Hotels recommended for seniors"
- **Reference:** "Canal House Grand in Amsterdam, high comfort/cleanliness score, fantastic location."

Cost: HuggingFace API pricing/1K tokens **Integration:** LangChain + HuggingFace Inference API

LLM Comparison - Qualitative Evaluation

Gemma-2-2B

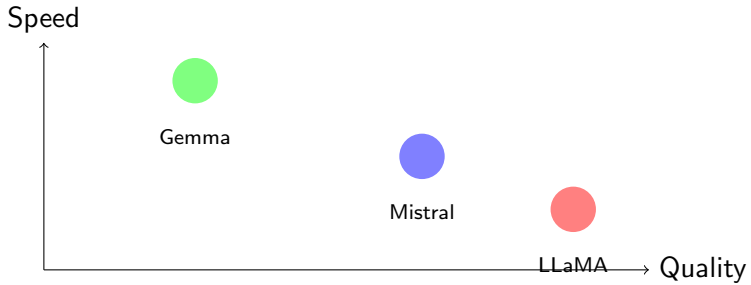
- Fastest response
- Concise answers
- Sometimes incomplete
- Good for simple queries

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 & Improvement (LLM)

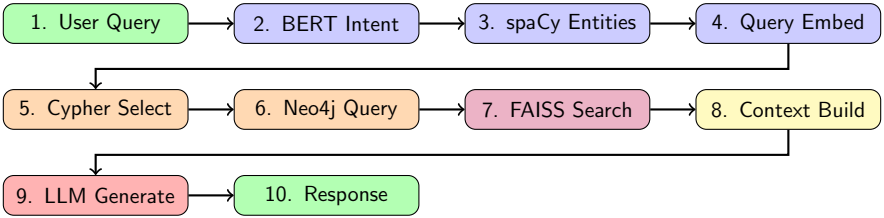
Errors Encountered & Fixed:

- **Robotic responses:** LLM started with "Based on the data..."
  - Fixed via persona prompt engineering
- **Hallucinations:** Invented hotel names not in KG
  - Fixed by injecting actual KG context
- **API timeouts:** HuggingFace rate limits
  - Added retry logic and error handling

Remaining Limitations:

- **Gemma-2-2B:** Often gives incomplete answers due to small model size
- **Low semantic accuracy:** Models struggle to match reference answers exactly
- **Context length:** Large KG results may exceed token limits
- **No memory:** Each query is independent, no conversation history

Pipeline Recap for Demo



Demo Features:

- Switch between embedding models (Node2Vec / FastRP)
- Switch between LLMs (Gemma / Mistral / LLaMA)
- Side-by-side embedding comparison
- Streamlit UI with J.A.R.V.I.S. persona

Demo Queries

Test Queries for Live Demo (Pre-loaded in UI):

1. **Recommendation:** "Recommend me a good hotel in Tokyo"
2. **Search:** "Find hotels in Paris"
3. **Hotel Info:** "Tell me about The Azure Tower"
4. **Reviews:** "Show me reviews for The Golden Oasis"
5. **Comparison:** "Compare The Azure Tower and L'Etoile Palace"
6. **Traveller:** "Best hotels for business travelers"
7. **Rating Filter:** "Hotels with cleanliness rating above 9"
8. **Demographics:** "Hotels recommended for seniors"

[LIVE DEMO]

Thank You!

Questions?