AI HYBRID TRAVEL ASSISTANT – VIETNAM TOURISM

Project Name:

AI Hybrid Travel Assistant for Vietnam Tourism Recommendations

Technologies Used:

Python, OpenAI GPT-4o-mini, Pinecone Vector Database, Neo4j Graph Database, RAG Architecture

Project Duration:

[Add your timeline]

Role:

Machine Learning Engineer

PROJECT OVERVIEW

This project implements an intelligent **Retrieval-Augmented Generation (RAG)** system that combines vector search, graph databases, and large language models to provide contextually rich travel recommendations for Vietnam. The system integrates three powerful technologies to create a **hybrid AI travel assistant** capable of answering complex travel queries with semantic understanding and relationship-aware context.

KEY FEATURES

Hybrid Retrieval Architecture

- Semantic vector search using Pinecone for similarity-based entity retrieval
- Graph-based relationship traversal using Neo4j for contextual enrichment
- LLM-powered response generation using OpenAI GPT-4o-mini for natural language answers

Comprehensive Dataset Coverage

- 350+ travel entities across 10 major Vietnamese destinations: Hanoi, Ha Long Bay, Hoi An, Da Nang, Nha Trang, Ho Chi Minh City
- Includes attractions, hotels, activities with metadata (tags, descriptions, best visiting times)
- 10+ relationship types: Located_In, Connected_To, Available_In, etc.

Technical Implementation

- 1536-dimensional embeddings using OpenAI text-embedding-3-small model
- Cosine similarity search with TOP_K=5 retrieval

- Batch processing with 32-item chunks for efficient data upload
- Interactive CLI interface for real-time query processing
- Graph visualization capabilities with HTML rendering

SYSTEM ARCHITECTURE

Core Components

Vector Database (Pinecone)

- Stores semantic embeddings of travel entities
- Enables fast similarity search using cosine distance metric
- Serverless deployment on AWS us-east-1

Graph Database (Neo4j)

- Models relationships between cities, attractions, hotels, and activities
- Implements graph constraints and MERGE operations for data integrity
- Supports multi-hop relationship traversal

Language Model (OpenAI)

- GPT-4o-mini for response generation
- Temperature = 0.2 for focused, factual responses
- Max tokens = 600 for concise recommendations

IMPLEMENTATION WORKFLOW

Phase 1: Setup and Configuration

- Configured API credentials for Neo4j, OpenAI, and Pinecone
- Prepared vietnam_travel_dataset.json with structured travel information
- Set up development environment with required Python libraries

Phase 2: Data Ingestion

Step 1: Vector Database Upload (pinecone_upload.py)

- Generated embeddings by combining entity name, description, and tags
- Created serverless Pinecone index with 1536 dimensions
- Uploaded vectors in batches of 32
- Stored metadata: id, name, type, description, tags

Step 2: Graph Database Loading (load_to_neo4j.py)

- Created Neo4j nodes with dual labels (specific type + Entity)

- Established uniqueness constraints on entity IDs
- Built directed relationships from connections array
- Optimized property storage by excluding nested objects

Step 3: Graph Visualization (visualize_graph.py)

- Generated interactive HTML visualization of travel network
- Enabled exploration of entity relationships and connections

Phase 3: Query Processing

Step 1: User Query Input

- Interactive CLI interface accepts natural language travel questions
- Example: "What are the best beach destinations in Vietnam?"

Step 2: Semantic Search

- Query embedded using text-embedding-3-small model
- Pinecone performs cosine similarity search
- Returns TOP_K=5 most relevant travel entities

Step 3: Graph Enrichment

- Execute Neo4j Cypher queries for retrieved entities
- Fetch neighboring nodes and relationships up to configurable depth
- Collect connected hotels, activities, and attractions

Step 4: Context Combination

- Merge vector search results with graph relationships
- Build enriched context containing both semantic matches and connections

Step 5: LLM Response Generation

- Construct prompt with system instructions for travel assistant role
- Include vector results as "semantically relevant nodes"
- Add graph data as "connected entities"
- GPT-4o-mini generates comprehensive, citation-backed response

Step 6: Final Output

- Display AI-generated recommendations
- Include node ID citations for traceability
- Provide actionable suggestions (accommodations, itineraries, best visit times)

TECHNICAL SKILLS DEMONSTRATED

- **Machine Learning:** Embedding generation, semantic similarity search, vector databases
- **NLP:** RAG architecture, prompt engineering, context window management
- **Database Management:** Neo4j graph modeling, Cypher query optimization, relationship design
- **API Integration:** OpenAI API, Pinecone serverless API, Neo4j driver implementation
- **Python Programming:** OOP, batch processing, error handling, modular architecture
- **Data Engineering:** ETL pipelines, JSON data processing, metadata management

---

PROJECT OUTCOMES

Performance Metrics

- Processes complex multi-entity travel queries
- Retrieves contextually relevant recommendations via hybrid search
- Generates natural language responses with proper citations
- Handles 350+ entities with sub-second query response times

Evaluation Results (7 Dimensions, 100 Points)

Dimension	**Points**	**Notes**
Functionality	20	End-to-end execution without errors
Debugging Skills	15	Resolved OpenAI v2 & Pinecone SDK issues
Code Quality	15	Modular structure with clear separation of concerns
Prompt Engineering	15	High-quality, relevant answer generation
Neo4j Query Design	10	Efficient graph context retrieval
Innovation	20	Advanced features and optimization
Documentation	5	Clear architectural explanation

Key Achievements

- Built production-ready RAG system combining vector and graph search
- Implemented efficient batch processing for large-scale data ingestion
- Designed prompt engineering for citation-backed responses
- Created interactive visualization for graph exploration
- Demonstrated expertise in multi-database integration

TECHNOLOGIES AND TOOLS

Programming Languages: Python

Machine Learning Frameworks: OpenAI API (GPT-4o-mini, text-embedding-3-small)

Databases:

- Pinecone (Serverless Vector Database)
- Neo4j (Cloud Graph Database)

Python Libraries:

- openai – LLM & embedding API client
- pinecone-client – Vector database operations
- neo4j – Graph database driver
- tqdm – Progress tracking
- json – Data parsing and processing

Development Tools: VS Code, Git, Virtual Environment

Cloud Services: AWS (Pinecone deployment), Neo4j AuraDB