Kiến Trúc AI/RAG - Tài Liệu Bảo Vệ Luận Văn

Holidate Hotel Booking System - AI Chatbot với RAG

Mục Lục Nhanh (Table of Contents)

BẢN TÓM TẮT "BỎ TÚI" (Executive Summary)

(Phần này dùng để trả lời nhanh trong 30 giây đầu tiên)

- **RAG Architecture:** Kết hợp LLM (GPT-4o) với Knowledge Base (Pinecone) để tránh hallucination và cung cấp dữ liệu real-time.
- **Decoupled ETL Pipeline:** S3 làm Data Lake, n8n xử lý độc lập → Không ảnh hưởng Backend performance.
- **Metadata Filtering:** Pinecone cho phép filter trước khi search (city, price, star rating) → Tăng accuracy, giảm cost.
- **YAML Frontmatter Parsing:** Code Node extract 25 trường metadata từ Markdown → Enable precise filtering.
- **Text Splitting:** Recursive Character Text Splitter xử lý documents dài → Chunks nhỏ hơn, retrieval chính xác hơn.

1. Kiến Trúc RAG (Retrieval-Augmented Generation)

1.1. Vấn đề & Giải pháp

Vấn đề:

- GPT-4o không biết thông tin cụ thể về hotels tại Việt Nam (giá, chính sách, tiện ích)
- Hallucination: LLM có thể bịa đặt thông tin không có thật
- Training data cũ (đến tháng 4/2024) → không có dữ liệu real-time
- Gửi toàn bộ Knowledge Base vào mỗi request → tốn rất nhiều token

Giải pháp:

- **RAG Pattern:** Kết hợp LLM với Knowledge Base
  - User query → Convert sang vector embedding
  - Search trong Vector DB → Retrieve top-k documents liên quan
  - Đưa context + query vào LLM → Generate answer dựa trên dữ liệu thực tế
- **Context Window Optimization:** Chỉ retrieve top-k documents → Tiết kiệm 90-95% cost

1.2. Luồng xử lý (Logic Flow)

```
User Query: "Khách sạn nào ở Đà Nẵng có bể bơi?"
    │
    ▼
Convert Query → Vector Embedding (text-embedding-3-large)
    │
    ▼
Search Pinecone Vector DB
    ├─ Vector similarity search
    ├─ Metadata filtering (city_name = "thanh-pho-da-nang")
    └─ Top-k retrieval (k = 5)
    │
    ▼
Retrieved Documents (Context)
    │
    ▼
GPT-4o + Context
    ├─ System Prompt: "Chỉ dùng thông tin từ context, không bịa đặt"
    ├─ User Query + Retrieved Documents
    └─ Generate Answer
    │
    ▼
Final Answer: "Dạ anh/chị, em tìm thấy Grand Mercure Danang có bể bơi..."
```

1.3. Key Code Snippet (Minh chứng)

Logic này được implement trong n8n workflow, xem chi tiết ở phần 4.

Mẹo Bảo Vệ (Defense Tip)

> "Hệ thống sử dụng RAG (Retrieval-Augmented Generation) để giải quyết vấn đề hallucination của LLM. Thay vì để GPT-4o tự trả lời dựa trên training data cũ, hệ thống retrieve thông tin từ Knowledge Base được cập nhật real-time, sau đó đưa context này vào LLM để generate câu trả lời chính xác. Điều này đảm bảo chatbot chỉ trả lời dựa trên dữ liệu thực tế, và nếu không có thông tin, chatbot sẽ thừa nhận thay vì bịa đặt."

---

2. Luồng Xử Lý Dữ Liệu (ETL Pipeline - Training Workflow)

Workflow file: `holidate_training_ai_agent_n8n_with_pinecone.json`

2.1. Vấn đề & Giải pháp

Vấn đề:

- Backend cần upload Knowledge Base nhưng không muốn bị block bởi heavy AI processing
- Cần extract metadata từ Markdown để enable filtering
- Documents quá dài → cần split thành chunks

Giải pháp:

- **Decoupled Architecture:** S3 làm Data Lake → Backend upload file và quên, n8n xử lý độc lập
- **YAML Frontmatter Parsing:** Code Node extract 25 trường metadata → Lưu vào Pinecone để filter
- **Text Splitting:** Recursive Character Text Splitter → Chia documents thành chunks nhỏ hơn

2.2. Luồng xử lý (Logic Flow)

```
Manual Trigger
    │
    ▼
1. Get KB data from S3 (List Objects)
   └─ HTTP Request: ListObjectsV2 API
    │
    ▼
2. Convert XML to JSON
   └─ Parse S3 XML response
    │
    ▼
3. Split (Array → Individual Items)
   └─ Mỗi file xử lý riêng
    │
    ▼
4. Download files from S3
   └─ Binary data (base64 encoded)
    │
    ▼
5. Convert Markdown to Text (Code Node) - CRITICAL
   ├─ Extract YAML Frontmatter → 25 trường metadata
   └─ Clean text (remove HTML, markdown formatting, duplicates)
    │
    ▼
6. Default Data Loader
   └─ Convert to LangChain Document format
    │
    ▼
7. Recursive Character Text Splitter
   ├─ Chunk size: configurable
   ├─ Chunk overlap: 100 characters
   └─ Mode: markdown (split theo headers, paragraphs)
    │
    ▼
8. Embeddings OpenAI
   ├─ Model: text-embedding-3-large
   └─ Dimensions: 3072
    │
```

```
    ▼
9. Pinecone Vector Store (Upsert)
   ├─ ID: doc_id (từ metadata)
   ├─ Vector: embedding (3072 dims)
   └─ Metadata: 25 fields (để filter sau này)
```

2.3. Key Code Snippet (Minh chứng)

Code Node - Extract YAML Frontmatter:

```javascript
// Trích xuất YAML Frontmatter (25 trường)
const yamlMatch = rawContent.match(/^---\s*([\s\S]*?)\s*---/);
const yamlContent = yamlMatch ? yamlMatch[1] : '';

// Parse metadata
metadata.doc_id = parseYamlValue(/doc_id:\s*"([^"]+)"/);
metadata.hotel_name = parseYamlValue(/hotel_name:\s*"([^"]+)"/);
metadata.city_name = parseYamlValue(/city_name:\s*"([^"]+)"/);
metadata.star_rating = parseInt(parseYamlValue(/star_rating:\s*(\d+)/), 10);
metadata.price = parseInt(parseYamlValue(/base_price:\s*(\d+)/), 10);
metadata.amenities = parseYamlArraySimple('amenity_tags');
// ... 20 trường khác

// Clean text
text = text
  .replace(/^---[\s\S]*?---\s*/, '')  // Remove YAML Frontmatter
  .replace(/<[^>]*>/g, '')             // Remove HTML tags
  .replace(/&#10;/g, '\n')             // Decode HTML entities
  .replace(/\*\*([^*]+)\*\*/g, '$1')  // Remove Markdown formatting
  // ... more cleaning steps
```

Pinecone Upsert:

```javascript
// Pinecone Vector Store node configuration
{
  "operation": "upsert",
  "id": "={{ $json.metadata.doc_id }}",
  "vector": "={{ $json.embedding }}",
  "metadata": {
    "doc_type": "={{ $json.metadata.doc_type }}",
    "hotel_id": "={{ $json.metadata.hotel_id }}",
    "city_name": "={{ $json.metadata.city_name }}",
    "star_rating": "={{ $json.metadata.star_rating }}",
    "price": "={{ $json.metadata.price }}",
    "amenities": "={{ $json.metadata.amenities }}"
    // ... 20 trường khác
  }
}
```

Mẹo Bảo Vệ (Defense Tip)

> "Hệ thống sử dụng Decoupled Architecture với S3 làm Data Lake. Backend chỉ cần upload Markdown files lên S3 và không cần chờ AI processing, đảm bảo Backend performance không bị ảnh hưởng. n8n workflow xử lý độc lập: extract metadata từ YAML Frontmatter (25 trường), clean text, split thành chunks, generate embeddings, và upsert vào Pinecone. Code Node là critical component - nó parse YAML Frontmatter để extract metadata như city_name, star_rating, price, amenities, cho phép Pinecone filter chính xác trước khi search."

---

3. Cơ Sở Dữ Liệu Vector (Pinecone)

3.1. Vấn đề & Giải pháp

Vấn đề:

- Cần lưu trữ và tìm kiếm semantic meaning của text
- Vector similarity search có thể trả về documents không liên quan về business logic
- Cần filter theo metadata (city, price, star rating) trước khi search

Giải pháp:

- **Pinecone Vector Database:** Serverless, low latency (<100ms), hỗ trợ metadata filtering
- **Metadata Filtering:** Filter bằng metadata TRƯỚC khi search → Tăng accuracy, giảm cost
- **3072 Dimensions:** Sử dụng `text-embedding-3-large` → Higher accuracy cho Vietnamese content

3.2. Luồng xử lý (Logic Flow)

```
User Query: "Khách sạn 5 sao ở Đà Nẵng có bể bơi?"
    │
    ▼
Convert Query → Embedding (3072 dimensions)
    │
    ▼
Pinecone Query với Metadata Filter
    ├─ Vector: query embedding
    ├─ Filter: {
    │     city_name: "thanh-pho-da-nang",
    │     star_rating: { $gte: 5 },
    │     amenities: { $in: ["swimming_pool"] }
    │   }
    └─ Top-k: 5
    │
    ▼
Retrieved Documents (chỉ hotels ở Đà Nẵng, 5 sao, có bể bơi)
    └─ Similarity scores + Metadata
```

3.3. Key Code Snippet (Minh chứng)

```
// Metadata Filtering Example
const filter = {
  doc_type: "hotel_profile",
  city_name: "thanh-pho-da-nang",
  star_rating: { $gte: 5 },
  amenities: { $in: ["swimming_pool", "pool"] }
};

// Pinecone Query
const results = await pinecone.query({
  vector: queryEmbedding,  // 3072 dimensions
  filter: filter,           // Metadata filter
  topK: 5
});

// Pinecone Record Structure
{
  "id": "hotel-123-chunk-1",
  "values": [0.123, -0.456, 0.789, ...],  // 3072 floats
  "metadata": {
    "doc_type": "hotel_profile",
    "hotel_name": "Grand Mercure Danang",
    "city_name": "thanh-pho-da-nang",
    "star_rating": 5,
    "price": 2500000,
    "amenities": ["swimming_pool", "spa", "gym"]
  }
}
```

Mẹo Bảo Vệ (Defense Tip)

> "Hệ thống chọn Pinecone vì 3 lý do chính: (1) Serverless architecture - không cần quản lý infrastructure, tự scale, (2) Low latency - dưới 100ms cho mỗi query, phù hợp cho chatbot real-time, và (3) Metadata Filtering - tính năng quan trọng nhất. Thay vì chỉ dùng vector similarity (có thể trả về hotels không liên quan), hệ thống filter bằng metadata trước khi search. Ví dụ: user hỏi 'Khách sạn 5 sao ở Đà Nẵng có bể bơi?' → hệ thống chỉ search trong subset hotels ở Đà Nẵng, 5 sao, có bể bơi. Điều này tăng accuracy và giảm cost đáng kể."

---

4. Luồng Chatbot (Inference Workflow)

Workflow file: `holidate_chatbot_ai_agent_n8n_with_pinecone.json`

4.1. Vấn đề & Giải pháp

Vấn đề:

- Cần chatbot tự động quyết định khi nào cần search Knowledge Base
- Cần giữ context conversation (user hỏi follow-up questions)
- Cần trả lời chính xác, không bịa đặt

Giải pháp:

- **ReAct Agent (GPT-4o):** Tự động reasoning và quyết định gọi tool
- **Vector Store Tool:** Agent tự động gọi khi cần thông tin
- **WindowBufferMemory:** Giữ lịch sử conversation để hiểu context

4.2. Luồng xử lý (Logic Flow)

```
1. User Question (Webhook)
   └─ POST request: { "chatInput": "...", "sessionId": "..." }
    │
    ▼
2. AI Agent (GPT-4o) - Reasoning
   ├─ Analyze question: Intent, entities, requirements
   ├─ Decide: Cần thông tin từ Knowledge Base
   └─ Call Vector Store Tool
    │
    ▼
3. Vector Store Tool
   ├─ Convert query → embedding (text-embedding-3-large)
   ├─ Search Pinecone với:
   │   ├─ Query vector
   │   ├─ Metadata filter (optional)
   │   └─ Top-k: 5
   └─ Retrieve documents
    │
    ▼
4. Agent nhận context từ tool
   ├─ Document 1: "Grand Mercure Danang có bể bơi..."
   ├─ Document 2: "Pullman Danang có bể bơi..."
   └─ ...
    │
    ▼
5. Agent Generate Answer
   ├─ System Prompt: "Chỉ dùng thông tin từ context"
   ├─ User Query + Retrieved Documents
   ├─ Memory: Previous conversation (nếu có)
   └─ Generate answer (Markdown format)
    │
    ▼
6. Response (JSON)
   └─ { "output": "Dạ anh/chị, em tìm thấy..." }
```

4.3. Key Code Snippet (Minh chứng)

System Prompt (AI Agent):

```
You are a professional and friendly AI ChatBot assistant of the Holidate Hotel
Booking System.
```

```
Mandatory rules:
1. Always use the information lookup tool before answering.
   Absolutely DO NOT fabricate information (Hallucination).
2. Honesty: If the information is not in the data found, be honest and say:
   "I currently do not have detailed information about this part in the system..."
3. Quote: When quoting, always include a warning:
   "This price is for reference only and may change depending on the time."
4. Writing style: Polite, friendly, use natural Vietnamese,
   address "em" and call customers "anh/chi".
5. Format: Use Markdown (bold, bullet points) to present room information.
```

Vector Store Tool Configuration:

```
{
  "mode": "retrieve-as-tool",
  "toolDescription": "Use this tool to retrieve information about hotels in
Vietnam, including room details, amenities, pricing policies, and locations.
Always call this tool when the user asks about booking accommodation or hotel
recommendations.",
  "pineconeIndex": "holidate-training-ai-agent-n8n-index"
}
```

Memory Configuration:

```
{
  "sessionIdType": "customKey",
  "sessionKey": "={{ $json.sessionId }}"
}
```

Mẹo Bảo Vệ (Defense Tip)

> "Hệ thống sử dụng ReAct Agent pattern với GPT-4o. Agent tự động reasoning để phân tích câu hỏi và quyết định khi nào cần gọi Vector Store Tool để tìm thông tin. System prompt được thiết kế nghiêm ngặt: bắt buộc phải gọi tool trước khi trả lời, không được bịa đặt, và phải thừa nhận nếu không có thông tin. WindowBufferMemory giữ lịch sử conversation để agent hiểu context - ví dụ: user hỏi 'Khách sạn ở Đà Nẵng' rồi hỏi tiếp 'Giá bao nhiêu?' → agent hiểu 'giá' là giá của khách sạn Đà Nẵng vừa nói."

---

5. FAQ - Trả Lời Câu Hỏi Phản Biện

Q1: Tại sao không gọi Backend API trực tiếp từ n8n thay vì dùng S3?

Vấn đề:

- Backend API có thể chậm nếu phải query database phức tạp
- n8n workflow sẽ bị block → timeout risk
- Nếu gọi API trực tiếp → n8n phụ thuộc vào Backend availability

Giải pháp:

- **S3 Data Lake Pattern:** Backend upload file → Fire-and-forget
- **Decoupling:** n8n xử lý độc lập → Không block Backend
- **Reliability:** S3 có durability 99.999999999% → Có thể retry nếu workflow fail

Defense Tip:

> "Chúng em sử dụng S3 làm Data Lake thay vì gọi Backend API trực tiếp vì 3 lý do: (1) Latency - S3 download nhanh và ổn định hơn API có thể query database phức tạp, (2) Decoupling - n8n không phụ thuộc vào Backend availability, nếu Backend down thì n8n vẫn có thể xử lý files đã upload, và (3) Timeouts - Backend có thể xử lý lâu (generate Knowledge Base cho 100 hotels), HTTP request có timeout limit, còn S3 + async processing thì không bị timeout. Pattern này gọi là Event-Driven Architecture."

Q2: Làm thế nào xử lý dữ liệu cũ (outdated data)?

Vấn đề:

- Hotel data thay đổi (giá mới, chính sách mới) → Cần update Knowledge Base
- Không muốn có duplicate data trong Pinecone

Giải pháp:

- **Idempotent Upsert Pattern:** Sử dụng `doc_id` làm unique identifier
- **Same ID, Update Vector:** Nếu ID đã tồn tại → Update vector và metadata
- **Eventual Consistency:** Dữ liệu được update trong vài phút

Defense Tip:

> "Hệ thống sử dụng Idempotent Upsert Pattern. Khi hotel data thay đổi, Backend generate Markdown mới và upload lên S3 với cùng file path (overwrite). n8n workflow trigger, download file mới, và upsert vào Pinecone với cùng `doc_id` từ metadata. Pinecone sẽ tự động update vector và metadata nếu ID đã tồn tại, hoặc insert mới nếu chưa có. Điều này đảm bảo dữ liệu cũ tự động bị thay thế, không cần manual cleanup, và đảm bảo eventual consistency."

Q3: Điều gì xảy ra nếu text quá dài (exceed token limit)?

Vấn đề:

- Documents có thể rất dài (hàng nghìn ký tự)
- Vector DB và LLM có giới hạn về độ dài text

Giải pháp:

- **Recursive Character Text Splitter:** Chia document thành chunks
- **Chunk Overlap:** 100 characters để không mất context ở ranh giới
- **Mode: markdown:** Split theo cấu trúc Markdown (headers, paragraphs) → Giữ semantic meaning

Defense Tip:

> "Hệ thống sử dụng Recursive Character Text Splitter trong training workflow. Document lớn được chia thành chunks nhỏ hơn (ví dụ: 1000 characters mỗi chunk), với overlap 100 characters để không mất thông tin ở ranh giới. Mỗi chunk được embed riêng và lưu vào Pinecone với ID như `{doc_id}-chunk-1`, `{doc_id}-chunk-2`. Khi search, Pinecone trả về top-k chunks (có thể từ cùng 1 document), và agent combine các chunks để generate answer đầy đủ. Mode `markdown` đảm bảo split theo cấu trúc semantic (headers, paragraphs) thay vì cắt ngẫu nhiên."

Q4: Làm sao ngăn chặn việc spam chatbot gây tốn chi phí OpenAI?

Vấn đề:

- User có thể spam chatbot → Tốn rất nhiều tiền OpenAI API
- Cần bảo vệ hệ thống khỏi abuse

Giải pháp:

- **Rate Limiting:** API Gateway giới hạn 10 requests/phút/IP
- **User Quota:** Chỉ user đã login mới được chat full tính năng, user guest bị giới hạn số lượt
- **Caching:** Cache các câu trả lời phổ biến để không gọi OpenAI nhiều lần

Defense Tip:

> "Hệ thống áp dụng 2 lớp bảo vệ: (1) **Rate Limiting** ở API Gateway (giới hạn 10 requests/phút/IP). (2) **User Quota** trong Business Logic: Chỉ user đã login mới được chat full tính năng, user guest bị giới hạn số lượt. Ngoài ra, cache các câu trả lời phổ biến để không gọi OpenAI nhiều lần."

Q5: Có gửi thông tin nhạy cảm của khách (SĐT, Email) sang OpenAI không?

Vấn đề:

- User có thể hỏi về thông tin cá nhân trong conversation
- Cần đảm bảo PII (Personally Identifiable Information) không bị gửi sang OpenAI

Giải pháp:

- **PII Scrubbing:** Trước khi gửi prompt sang OpenAI, hệ thống mask hoặc replace các pattern nhạy cảm
- **Data Retention:** OpenAI API được cấu hình `retention: 0` (không lưu data để train) theo chính sách Enterprise Privacy

Defense Tip:

> "Trước khi gửi prompt sang OpenAI, hệ thống có bước **PII Scrubbing** (làm sạch dữ liệu). Các pattern như SĐT, Email sẽ được mask (ví dụ: `09***`) hoặc replace bằng placeholder. OpenAI API cũng được cấu hình `retention: 0` (không lưu data để train) theo chính sách Enterprise Privacy."

Phụ Lục (Appendix)

A.1. n8n Workflow Configuration (Training)

Node: Get KB data from S3

```
{
  "url": "https://s3.ap-southeast-1.amazonaws.com/holidate-storage/?list-type=2&prefix=knowledge_base/vietnam/thu-do-ha-noi",
  "authentication": "predefinedCredentialType",
  "nodeCredentialType": "aws"
}
```

Node: Download files from S3

```
{
  "bucketName": "holidate-storage",
  "fileKey": "={{ $json.Key }}"
}
```

Node: Embeddings OpenAI

```
{
  "model": "text-embedding-3-large",
  "options": {
    "dimensions": 3072
  }
}
```

Node: Recursive Character Text Splitter

```
{
  "chunkOverlap": 100,
  "options": {
    "splitCode": "markdown"
  }
}
```

A.2. n8n Workflow Configuration (Chatbot)

Node: Webhook

```
{
  "httpMethod": "POST",
  "path": "9c4bdf4a-4fbc-485b-b3bc-e09647d8d450",
  "responseMode": "responseNode"
}
```

Node: AI Agent

```
{
  "promptType": "define",
  "text": "={{ $json.chatInput }}",
  "options": {
    "systemMessage": "You are a professional and friendly AI ChatBot assistant..."
  }
}
```

Node: Pinecone Vector Store (Tool)

```
{
  "mode": "retrieve-as-tool",
  "toolDescription": "Use this tool to retrieve information about hotels in Vietnam...",
  "pineconeIndex": "holidate-training-ai-agent-n8n-index"
}
```

Node: Simple Memory

```
{
  "sessionIdType": "customKey",
  "sessionKey": "={{ $json.sessionId }}"
}
```

A.3. Metadata Fields (25 trường)

1. `doc_id`, `doc_type`, `hotel_id`, `hotel_name`
2. `city_name`, `district_name`, `address`
3. `star_rating`, `review_score`, `review_count`
4. `price` (base_price hoặc reference_min_price)
5. `amenities` (combine amenity_tags + room_amenity_tags)
6. `breakfast_included`, `wifi_available`, `smoking_allowed`
7. `room_name`, `max_adults`, `max_children`
8. `view_type`, `bed_type`, `area_sqm`
9. `to_beach_meters`, `room_type`, `has_balcony`

Tổng Kết

Hệ thống AI/RAG của Holidate được thiết kế với các đặc điểm chính:

1. **RAG Architecture:** Kết hợp LLM (GPT-4o) với Knowledge Base (Pinecone) để tránh hallucination
2. **Decoupled ETL Pipeline:** S3 làm Data Lake, n8n xử lý độc lập → Không ảnh hưởng Backend performance
3. **Metadata Filtering:** Pinecone cho phép filter trước khi search → Tăng accuracy, giảm cost
4. **Intelligent Agent:** GPT-4o với tool calling → Tự động quyết định khi nào cần search
5. **Text Splitting:** Xử lý documents dài bằng Recursive Character Text Splitter

Công nghệ Stack:

- **LLM:** GPT-4o (OpenAI)
- **Embeddings:** text-embedding-3-large (3072 dimensions)
- **Vector DB:** Pinecone (Serverless)
- **Orchestration:** n8n (Workflow automation)
- **Storage:** AWS S3 (Data Lake)
- **Format:** Markdown với YAML Frontmatter

Tất cả components đều có error handling, logging, và monitoring để đảm bảo reliability và maintainability.