ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

NGUYỄN CHƠN NHÂN – 24521233
TRẦN TRỌNG NHÂN – 24521243
NGUYỄN TRỌNG PHAN NHẬT – 24521254
ĐOÀN HỮU GIA BÌNH – 24520192

BÁO CÁO ĐỒ ÁN CS311

Chatbot tư vấn du lịch Việt Nam

Vietnam Travel Assistant Chatbot

NGÀNH KHOA HỌC MÁY TÍNH

GIẢNG VIÊN HƯỚNG DẪN
ThS. Đặng Văn Thìn

TP. HỒ CHÍ MINH, 2026

TÓM TẮT

Báo cáo này trình bày việc xây dựng hệ thống **Travel Advisor** cho bài toán tư vấn du lịch Việt Nam bằng tiếng Việt, với mục tiêu tạo ra câu trả lời **chính xác, có căn cứ và phù hợp ngữ cảnh hội thoại**. Điểm khó của bài toán nằm ở việc truy vấn người dùng thường **mơ hồ hoặc đa ý định** (ví dụ: vừa hỏi địa điểm, vừa hỏi khách sạn, vừa muốn lịch trình) trong khi mô hình ngôn ngữ lớn (LLM) có nguy cơ **hallucination** nếu không được "neo" vào dữ liệu.

Nhóm đề xuất kiến trúc **Plan-RAG (multi-stage RAG)** gồm các bước: (i) trích xuất ý định và thực thể, (ii) lập kế hoạch truy vấn dạng nhiều tác vụ (sub-tasks), (iii) truy xuất song song qua các **expert agents** theo từng miền (địa điểm, khách sạn, ẩm thực, lịch trình, chi phí), (iv) tổng hợp kết quả và sinh câu trả lời có tham chiếu dữ liệu. Hệ thống sử dụng **hybrid search** kết hợp semantic search (FAISS + Vietnamese SBERT embeddings) và lọc theo metadata trên MongoDB. Dữ liệu thử nghiệm gồm **2,799 điểm du lịch**, **4,469 khách sạn** và thông tin **63 tỉnh/thành**.

Thực nghiệm với 50 truy vấn theo ba mức độ (simple/medium/complex) cho thấy Plan-RAG cải thiện rõ rệt so với RAG truyền thống: **intent accuracy đạt 100%**, **success rate 92%**, **groundedness 95%**. Đổi lại, latency trung bình tăng lên khoảng **3.7s** do chi phí truy xuất đa tác vụ. Báo cáo cũng thảo luận các vấn đề còn tồn tại (latency, relevance cho truy vấn phức tạp, xử lý ngữ cảnh) và hướng phát triển trong tương lai.

MỤC LỤC

Chương 1. MỞ ĐẦU

1.1 Lý do chọn đề tài

Trong bối cảnh du lịch nội địa phát triển mạnh, người dùng có nhu cầu tìm kiếm thông tin đa dạng như điểm tham quan, khách sạn, ẩm thực, lịch trình và chi phí. Tuy nhiên, thông tin du lịch trên Internet thường phân tán, thiếu cấu trúc, không đồng nhất và dễ lỗi thời. Thay vì chỉ dựa vào **Large Language Models (LLMs)** để sinh câu trả lời (dễ gặp hiện tượng *hallucination*), hệ thống cần được **grounded** bằng tri thức miền thông qua **Retrieval-Augmented Generation (RAG)**.

Do đó, hướng tiếp cận **Retrieval-Augmented Generation (RAG)** trở nên phù hợp vì cho phép truy xuất dữ liệu trước khi sinh câu trả lời, giúp tăng tính chính xác và khả năng giải thích. Bên cạnh đó, truy vấn du lịch trong thực tế thường **đa ý định** (multi-intent) và mang tính hội thoại nhiều vòng (multi-turn). Vì vậy, báo cáo tập trung xây dựng hệ thống Travel Advisor theo kiến trúc **Plan-RAG** với các **expert agents** chuyên biệt để tăng khả năng xử lý truy vấn phức tạp và giảm hallucination.

1.2 Các nghiên cứu liên quan

Các hướng tiếp cận phổ biến có thể phân thành ba nhóm:

- **LLM-only**: Trả lời trực tiếp dựa trên kiến thức tham số của mô hình. Ưu điểm là nhanh và linh hoạt, nhưng hạn chế lớn là thiếu căn cứ dữ liệu và dễ hallucination trong domain hẹp.
- **Traditional RAG**: Truy xuất tài liệu liên quan (thường qua vector search) rồi đưa vào prompt để sinh câu trả lời. Cải thiện groundedness nhưng vẫn gặp khó khăn với truy vấn **đa ý định** hoặc đòi hỏi nhiều bước suy luận.
- **Multi-stage / Plan-based RAG**: Thêm bước phân tích ý định, phân rã truy vấn thành nhiều tác vụ, gọi các bộ truy xuất/chuyên gia tương ứng và tổng hợp kết quả. Hướng này phù hợp với bài toán du lịch vì dữ liệu phân mảnh theo nhiều miền (địa điểm, khách sạn, ẩm thực, lịch trình, chi phí).

1.3 Mục tiêu, đối tượng và phạm vi nghiên cứu

Mục tiêu nghiên cứu của báo cáo:

- **M1**: Xây dựng hệ thống RAG cho domain du lịch tiếng Việt.
- **M2**: Giảm hallucination thông qua Plan-RAG và expert agents.
- **M3**: Xử lý truy vấn đa ý định (multi-intent).
- **M4**: Duy trì ngữ cảnh hội thoại đa vòng (multi-turn).
- **M5**: Cân bằng độ chính xác và latency ở mức chấp nhận được trong trải nghiệm người dùng.

Đối tượng nghiên cứu: kiến trúc Plan-RAG, quy trình truy xuất tri thức lai (hybrid search) và cơ chế điều phối expert agents cho bài toán tư vấn du lịch.

Phạm vi:

- Ngôn ngữ: Tiếng Việt.
- Miền dữ liệu: Du lịch Việt Nam (63 tỉnh/thành).
- Dữ liệu: 2,799 điểm du lịch; 4,469 khách sạn; thông tin tỉnh/thành.
- Mô hình:
  1. LLM FPT AI SaoLa3.1-medium (theo cấu hình trong báo cáo thực nghiệm);
  2. Embedding Vietnamese SBERT (keepitreal/vietnamese-sbert).

1.4 Các đóng góp chính cho đề tài

Đồ án có các đóng góp chính:

- Đề xuất và hiện thực kiến trúc **Plan-RAG** cho domain du lịch tiếng Việt, hỗ trợ phân rã truy vấn thành nhiều tác vụ.
- Thiết kế tập **expert agents** theo miền: SpotExpert, HotelExpert, FoodExpert, ItineraryExpert, CostExpert, GeneralInfoExpert.
- Xây dựng cơ chế **hybrid search** (semantic search + metadata filtering + reranking) trên dữ liệu du lịch Việt Nam.
- Xây dựng bộ test và báo cáo thực nghiệm, đánh giá theo các chỉ số: intent accuracy, relevance, success rate, groundedness và latency breakdown.

1.5 Cấu trúc

báo cáo được trình bày gồm 5 chương:

- **Chương 1: Mở đầu** trình bày động cơ, mục tiêu, phạm vi và đóng góp của đề tài.
- **Chương 2: Cơ sở lý thuyết** giới thiệu nền tảng RAG, Plan-RAG và các thành phần kỹ thuật liên quan.
- **Chương 3: Phương pháp thực hiện** mô tả thiết kế hệ thống, dữ liệu, kiến trúc và các mô-đun triển khai.
- **Chương 4: Thực nghiệm, đánh giá và thảo luận** trình bày bộ dữ liệu test, thước đo, kết quả benchmark, phân tích lỗi và case studies.
- **Chương 5: Kết luận và hướng phát triển** tổng kết kết quả, nêu hạn chế và định hướng cải tiến.

Chương 2. CƠ SỞ LÝ THUYẾT

Trong chương này, báo cáo trình bày các nền tảng lý thuyết phục vụ cho việc xây dựng hệ thống tư vấn du lịch dựa trên truy xuất tri thức. Trọng tâm là kỹ thuật Retrieval-Augmented Generation (RAG), biến thể Plan-RAG (multi-stage RAG), cùng các thành phần: trích xuất ý định, lập kế hoạch truy vấn, expert agents và hybrid search.

2.1 Large Language Model (LLM) và hạn chế LLM-only

2.1.1 Khái niệm LLM

LLM (Large Language Model) là mô hình học sâu được huấn luyện trên lượng dữ liệu văn bản lớn, có khả năng hiểu và sinh ngôn ngữ tự nhiên. Trong bài toán tư vấn du lịch, LLM giúp:

- Hiểu câu hỏi tiếng Việt đa dạng cách diễn đạt.
- Tạo câu trả lời tự nhiên, mạch lạc, phù hợp văn cảnh.
- Thực hiện suy luận tổng hợp (ví dụ: gợi ý lịch trình theo sở thích).

2.1.2 Hạn chế của LLM-only

Nếu chỉ dựa vào LLM (không truy xuất dữ liệu), hệ thống thường gặp:

- **Hallucination**: bịa thông tin không có trong dữ liệu.
- **Outdated**: không biết dữ liệu cập nhật mới nhất trong cơ sở dữ liệu.
- **Unverifiable**: khó truy vết nguồn/căn cứ.
- **Generic**: trả lời chung chung, kém đặc thù theo địa phương/điều kiện ràng buộc.

Vì vậy, cần kết hợp cơ chế truy xuất tri thức (retrieval) để neo câu trả lời vào dữ liệu thực.

2.2 Retrieval-Augmented Generation (RAG)

2.2.1 Định nghĩa và công thức

RAG là kiến trúc kết hợp giữa **truy xuất** (Retrieval) và **sinh văn bản** (Generation), nhằm cung cấp ngữ cảnh từ kho tri thức cho LLM.

$$\text{Response} = \text{LLM}(\text{Query} + \text{Retrieved\_Context})$$

2.2.2 Quy trình RAG cơ bản

`Query → Retrieve → Augment Prompt → Generate(LLM) → Response`

2.2.3 Ưu/nhược điểm

Ưu điểm:

- Giảm hallucination nhờ grounded vào dữ liệu.
- Cập nhật tri thức dễ (chỉ cần cập nhật database/index).
- Có thể kèm bằng chứng (citations từ DB).

Nhược điểm:

- Phụ thuộc mạnh vào chất lượng retrieval.
- Latency tăng do thêm bước truy xuất.
- Truy vấn phức tạp nhiều ràng buộc/đa ý định khó xử lý nếu chỉ retrieve một lần.

2.3 Plan-RAG (Planning + RAG)

2.3.1 Động cơ

Truy vấn du lịch thường **multi-intent** và **multi-constraint**. Ví dụ:

> "Lịch trình Đà Nẵng 3 ngày cho gia đình 4 người, budget 5 triệu, thích biển và ẩm thực"

Một lần retrieve theo RAG truyền thống thường không đủ bao phủ tất cả yêu cầu (địa điểm + ăn uống + khách sạn + lịch trình + chi phí).

2.3.2 Quy trình Plan-RAG

Plan-RAG mở rộng RAG bằng bước **planning**:

```
Query → Intent Extraction → Query Decomposition →
Parallel Expert Retrieval → Aggregate → Generate
```

Các ý chính:

- **Intent Extraction**: nhận diện intent và trích xuất entities/slots.
- **Query Decomposition**: phân rã truy vấn thành các sub-tasks (có thể biểu diễn dạng DAG).
- **Expert Retrieval**: gọi các expert theo miền (spot/hotel/food/itinerary/cost...).
- **Aggregation**: tổng hợp kết quả và sinh phản hồi cuối cùng.

2.4 Embedding và Semantic Search

2.4.1 Embedding

Embedding biến văn bản thành vector số trong không gian đặc trưng, sao cho các câu/đoạn có nghĩa gần nhau thì vector gần nhau. Trong hệ thống, embedding phục vụ tìm kiếm ngữ nghĩa: "biển đẹp" có thể truy về "Mỹ Khê" dù không trùng từ khóa.

2.4.2 Cosine similarity

Độ tương đồng giữa hai vector thường đo bằng cosine:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \, \|\mathbf{B}\|} \tag{2.1}$$

2.4.3 Vector store

Trong triển khai, vector có thể lưu trên FAISS hoặc ChromaDB (tùy cấu hình). Về bản chất lý thuyết, cả hai đều hỗ trợ truy vấn top-k theo similarity.

2.5 Hybrid Search và Reciprocal Rank Fusion (RRF)

2.5.1 Tại sao cần Hybrid Search

Keyword search mạnh ở tên riêng (ví dụ "Bà Nà Hills") nhưng yếu ở synonyms/diễn đạt khác. **Semantic search** mạnh ở ngữ nghĩa nhưng có thể bỏ lỡ exact match. Hybrid Search kết hợp cả hai để tăng độ phủ và độ chính xác.

2.5.2 Luồng Hybrid Search

```
(1) Keyword Search (MongoDB text search)
(2) Semantic Search (vector similarity)
(3) Fusion + Rerank
```

2.5.3 Reciprocal Rank Fusion (RRF)

RRF hợp nhất nhiều danh sách xếp hạng (keyword ranking, semantic ranking):

$$\text{RRF}(d) = \sum_{r \in R} \frac{1}{k + r(d)}$$

Trong đó:

- d: document ứng viên
- R: tập các ranking lists
- $r(d)$: thứ hạng của d trong ranking list r
- k: hằng số (thường dùng $k = 60$)

2.6 Slot-Filling Dialogue System

2.6.1 Khái niệm

Slot-filling là kỹ thuật hội thoại định hướng tác vụ (task-oriented dialogue) để thu thập thông tin dần theo nhiều lượt. Với bài toán du lịch, các slots phổ biến gồm:

- destination (đi đâu), duration (mấy ngày)
- budget (ngân sách), people_count (số người)
- interests (sở thích), companion_type (đi với ai)

2.6.2 Ví dụ

```
Turn 1: "Tôi muốn đi Đà Nẵng"
Slots: destination=Đà Nẵng, duration=?, budget=?, people=?
Bot:  "Bạn muốn đi mấy ngày?"

Turn 2: "3 ngày"
Slots: destination=Đà Nẵng, duration=3, budget=?, people=?
Bot:  "Budget khoảng bao nhiêu?"
```

2.7 Conversation Memory Patterns

Hệ thống hội thoại nhiều vòng cần bộ nhớ để tránh hỏi lại và để trả lời đúng follow-up. Có thể mô tả theo ba nhóm:

- **Entity memory**: lưu slots (destination, budget, duration...).
- **Buffer memory**: lưu N tin nhắn gần nhất (chat_history).
- **Cache memory**: lưu kết quả retrieval gần nhất (last_spots, last_hotels) để phục vụ truy vấn tiếp theo nhanh hơn.

Nguyên tắc cập nhật quan trọng: **không ghi đè slots nếu lượt mới không cung cấp giá trị mới**, tránh mất thông tin.

2.8 State Machine Pattern cho điều phối workflow

2.8.1 Mục tiêu

State machine quản lý trạng thái workflow và ngăn "nhảy bước" (ví dụ: tính chi phí khi chưa chọn khách sạn).

2.8.2 Các trạng thái điển hình

```
INITIAL → GATHERING_INFO → CHOOSING_SPOTS →
CHOOSING_HOTEL → READY_TO_FINALIZE
```

2.8.3 State guards (Intent dependencies)

Mỗi intent có thể yêu cầu điều kiện tối thiểu (required fields/required states). Nếu thiếu, hệ thống sẽ hỏi bổ sung (slot-filling) thay vì thực thi sai luồng.

2.9 Generative UI Pattern

2.9.1 Khái niệm

Generative UI là mô hình mà **backend quyết định loại UI** cần render, frontend chỉ đóng vai trò hiển thị theo `ui_type`. Cách tiếp cận này giúp:

- Giảm hardcode luồng ở frontend.
- Dễ mở rộng UI types khi thêm expert mới.
- Cải thiện UX (cards, itinerary builder, buttons...) thay vì chỉ text.

2.9.2 Schema phản hồi

```
{
  "reply": "...",
  "ui_type": "hotel_cards" | "spot_cards" | "itinerary" | "options"
  "ui_data": {...}
}
```

2.9.3 Các ui_type thường dùng

- `hotel_cards`, `spot_cards`, `food_cards`
- `itinerary`, `itinerary_builder`
- `tips`, `options`

Chương 3. PHƯƠNG PHÁP THỰC HIỆN

Chương này trình bày chi tiết phương pháp thiết kế và triển khai hệ thống Smart Travel Platform theo kiến trúc Plan-RAG, bao gồm kiến trúc tổng quan, luồng xử lý request, các mô-đun NLU/Planner/Experts, cơ chế Hybrid Search, bộ nhớ hội thoại (conversation memory), state machine điều phối workflow và cơ chế Generative UI giữa backend và frontend.

3.1 Tổng quan kiến trúc hệ thống

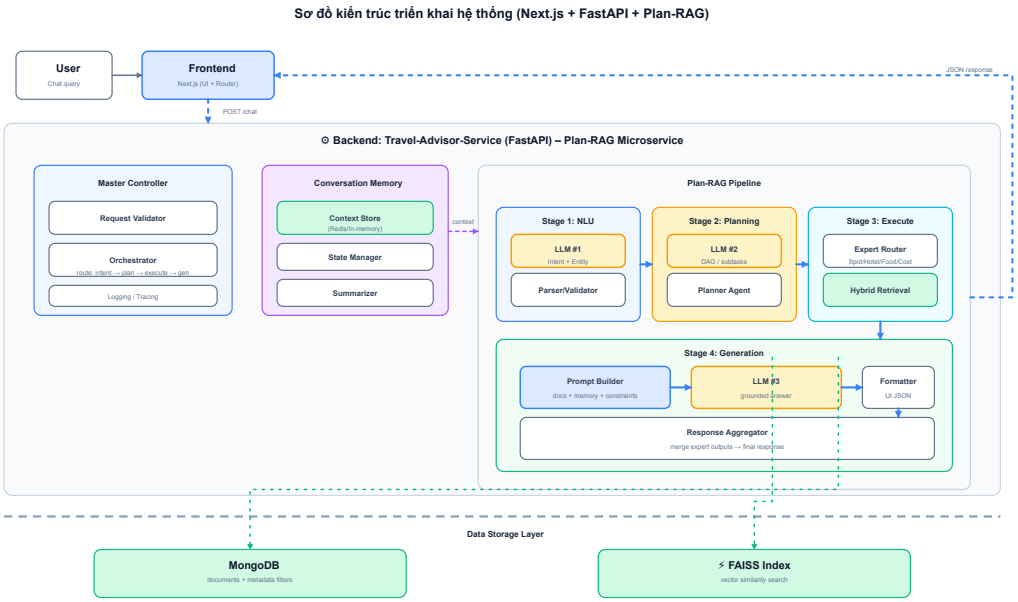
Hệ thống được xây dựng theo kiến trúc client-server, trong đó frontend đảm nhiệm trải nghiệm hội thoại (chat-first), còn backend chịu trách nhiệm điều phối pipeline Plan-RAG, truy xuất dữ liệu, gọi LLM và trả về phản hồi kèm dữ liệu UI.

3.1.1 Tech stack và các thành phần chính

Các công nghệ chính:

- **Frontend**: Next.js 14+ (TypeScript, Tailwind CSS).
- **Backend**: FastAPI (Python 3.11).
- **LLM**: FPT AI Saola 3.1 (OpenAI-compatible API).
- **Database**: MongoDB Atlas (lưu dữ liệu cấu trúc).
- **Vector Store**: FAISS (lưu vector embeddings).
- **Embedding**: `paraphrase-multilingual-MiniLM-L12-v2`.

3.1.2 Sơ đồ kiến trúc triển khai



Hình 3.1. Sơ đồ kiến trúc triển khai

3.2 Luồng xử lý một request theo Plan-RAG

Hệ thống thực thi theo 5 bước chính: khôi phục ngữ cảnh, trích xuất intent/entities, lập kế hoạch (planning), thực thi experts (retrieval/execution), và tổng hợp phản hồi (generation + UI).

3.2.1 Bước 1: Khôi phục context hội thoại

Mỗi request gửi lên backend đính kèm một phần context (ví dụ lưu ở phía client), backend khôi phục lại các slots quan trọng như `destination, duration, budget, people_count, last_spots, last_hotels` và cập nhật `chat_history`. Context giúp hệ thống trả lời đúng các câu hỏi follow-up và duy trì hội thoại nhiều vòng.

3.2.2 Bước 2: Intent detection và entity extraction (NLU)

Mô-đun NLU trích xuất:

- **Intent**: ví dụ `plan_trip, find_hotel, find_spot, find_food, calculate_cost, get_location_tips, show_itinerary, ...`
- **Entities/Slots**: `location, duration, budget, people_count, companion_type, interests,...`

Ví dụ regex cho duration/budget/people:

- duration: `(\d+)\s*(ngày|tuần)`
- people: `(\d+)\s*người`
- budget: chuẩn hóa về VND (triệu/tr/đồng)

3.2.3 Bước 3: Planning (phân rã truy vấn thành sub-tasks)

Planner Agent nhận intent/entities và tạo danh sách sub-tasks có thể biểu diễn như DAG (có phụ thuộc). Ví dụ truy vấn:

> "Lịch trình Đà Nẵng 3 ngày cho gia đình 4 người, budget 5 triệu, thích biển và ẩm thực"

có thể được phân rã thành:

- `find_spot` (biển, family-friendly)
- `find_hotel` (phù hợp 4 người, ràng buộc giá)
- `create_itinerary` (3 ngày)
- `calculate_cost` (tổng hợp)

3.2.4 Bước 4: Expert execution (song song khi có thể)

Các expert xử lý theo miền nghiệp vụ:

- **SpotExpert**: truy xuất điểm tham quan.
- **HotelExpert**: truy xuất khách sạn theo địa điểm/giá/rating/amenities.
- **FoodExpert**: gợi ý đặc sản/quán ăn.
- **ItineraryExpert**: ghép spots + hotels thành lịch trình theo ngày.
- **CostCalculator**: ước lượng chi phí theo nights/activities.

Với các sub-tasks độc lập, hệ thống có thể chạy song song để giảm latency tổng.

3.2.5 Bước 5: Response Aggregation + Generation

Sau khi experts trả về kết quả, Response Aggregator:

- Chuẩn hóa dữ liệu theo schema thống nhất.
- Quyết định `ui_type` để frontend render đúng component.
- Sinh trả lời tiếng Việt tự nhiên (LLM) dựa trên retrieved context và kết quả đã chuẩn hóa.
- Cập nhật context: state workflow, last_spots/last_hotels, itinerary_builder,...

3.3 Thiết kế cơ sở dữ liệu và schema

Hệ thống sử dụng MongoDB Atlas với 3 collections chính:

3.3.1 Collection `spots_detailed`

```
{
  "_id": ObjectId("..."),
  "name": "Bà Nà Hills",
  "province_id": "da-nang",
  "category": "attraction",
  "description": "...",
  "rating": 4.8,
  "address": "...",
  "coordinates": {"lat": 15.99, "lon": 107.99},
  "tags": ["mountain","photography","family"],
  "ticket_price": 850000,
  "opening_hours": "07:00 - 22:00"
}
```

3.3.2 Collection `hotels`

```
{
  "_id": ObjectId("..."),
  "name": "Novotel Danang ...",
```

```
  "province_id": "da-nang",
  "star_rating": 5,
  "price_per_night": 1500000,
  "rating": 4.5,
  "amenities": ["pool","spa","gym"],
  "address": "...",
  "booking_url": "https://..."
}
```

3.3.3 Collection `provinces_info`

```
{
  "province_id": "da-nang",
  "name": "Đà Nẵng",
  "description": "...",
  "best_time": "Tháng 2 - Tháng 8",
  "highlights": ["Bà Nà Hills","Cầu Rồng","Mỹ Khê"],
  "local_food": ["Mì Quảng","Bánh tráng cuốn thịt heo"]
}
```

3.4 Hybrid Search: Keyword + Semantic + Fusion (RRF)

Để cân bằng giữa *exact match* (tên riêng) và *semantic match* (ngữ nghĩa), hệ thống triển khai Hybrid Search theo 3 bước:

3.4.1 Keyword search (MongoDB text search)

Keyword search phù hợp cho các truy vấn có tên riêng, ví dụ "Bà Nà Hills", "Mỹ Khê".

3.4.2 Semantic search (FAISS)

Semantic search dùng embedding để tìm nội dung tương đồng nghĩa (ví dụ "biển đẹp" gợi về "Mỹ Khê").

3.4.3 Fusion bằng Reciprocal Rank Fusion (RRF)

Kết quả keyword và semantic được hợp nhất bằng RRF:

$$\text{RRF}(d) = \sum_{r \in R} \frac{1}{k + r(d)}$$

Trong đó $r(d)$ là thứ hạng của document d trong danh sách ranking r, và k là hằng số (thường dùng 60). Sau fusion, hệ thống rerank và lấy top-K cuối cùng.

3.5 Conversation Memory và Slot Filling

Hệ thống áp dụng slot-filling để thu thập thông tin qua nhiều lượt:

```
Turn 1: User "Tôi muốn đi Đà Nẵng"
Slots: destination=Đà Nẵng, duration=?, budget=?, people=?
Bot: "Bạn muốn đi mấy ngày?"
```

3.5.1 Cấu trúc context (minh họa)

```
EnhancedConversationContext:
  destination: Optional[str]
  duration: Optional[int]
  budget: Optional[int]
  people_count: int
  companion_type: Optional[str]
  interests: List[str]
  chat_history: List[Message]
  last_spots: List[Spot]
  last_hotels: List[Hotel]
  itinerary_builder: Dict
  workflow_state: str
```

Nguyên tắc cập nhật: **chỉ ghi đè slot khi có giá trị mới** để tránh mất thông tin từ các lượt trước.

3.6 State Machine điều phối workflow

Để tránh user "nhảy bước" (ví dụ tính chi phí khi chưa chọn khách sạn), hệ thống dùng state machine:

```
INITIAL -> GATHERING_INFO -> CHOOSING_SPOTS -> CHOOSING_HOTEL -
> READY_TO_FINALIZE
```

3.6.1 Intent dependencies (state guards)

Ví dụ điều kiện trước khi thực thi `calculate_cost`:

- required_states: `CHOOSING_HOTEL` hoặc `READY_TO_FINALIZE`
- required_fields: `selected_hotel`

Nếu thiếu, hệ thống trả về câu hỏi bổ sung (slot-filling) thay vì chạy tiếp pipeline.

3.7 Generative UI: backend quyết định UI

Thay vì frontend hardcode nhiều luồng, backend trả về `ui_type` và `ui_data` để frontend render đúng component.

3.7.1 Schema phản hồi

```
{
  "reply": "...",
  "ui_type": "hotel_cards" | "spot_cards" | "itinerary" | "itinerary_
  "ui_data": {...},
  "context": {...}
}
```

3.7.2 Các UI types tiêu biểu

- `hotel_cards`: grid card khách sạn (giá, rating, tiện nghi).
- `spot_cards`: grid card địa điểm tham quan.
- `itinerary / itinerary_builder`: timeline và builder tương tác theo ngày.
- `options`: các nút lựa chọn nhanh để dẫn luồng hội thoại.

3.8 Tóm tắt chương

Chương này đã mô tả phương pháp triển khai hệ thống theo Plan-RAG gồm NLU + Planner + Expert execution + Aggregation/Generation, đồng thời làm rõ các kỹ thuật trọng yếu gồm Hybrid Search (RRF), Conversation Memory (slot-filling), State Machine (workflow control) và Generative UI (backend-driven UI). Đây là nền tảng để chương sau trình bày thực nghiệm và đánh giá chất lượng hệ thống.

Chương 4. THỰC NGHIỆM, ĐÁNH GIÁ VÀ THẢO LUẬN

Chương này trình bày thiết lập thực nghiệm, thước đo đánh giá và kết quả benchmark của hệ thống Plan-RAG trong bài toán tư vấn du lịch tiếng Việt. Ngoài các chỉ số tổng quan, chương cũng phân tích chi tiết theo độ phức tạp truy vấn, multi-intent và độ trễ theo từng thành phần.

4.1 Thiết lập thực nghiệm

4.1.1 Dữ liệu và phạm vi

Hệ thống phục vụ domain du lịch Việt Nam (63 tỉnh/thành), với dữ liệu cấu trúc lưu trong MongoDB:

- `spots_detailed`: 2,799 điểm du lịch
- `hotels`: 4,469 khách sạn
- `provinces_info`: 63 tỉnh/thành

4.1.2 Mô hình và các thành phần chính

- LLM: FPT AI SaoLa3.1-medium (phục vụ intent extraction, planning, generation).
- Embedding: Vietnamese SBERT (`keepitreal/vietnamese-sbert`) cho semantic search.
- Vector index: FAISS (IVFFlat), metric cosine similarity.
- Hybrid Search: semantic search + metadata filtering + reranking.

4.1.3 Bộ truy vấn đánh giá (Test queries)

Nhóm xây dựng 50 truy vấn và chia theo độ phức tạp:

- **Simple (20)**: đơn mục tiêu (ví dụ: "Địa điểm đẹp ở Đà Lạt").
- **Medium (20)**: có ràng buộc (ví dụ: "Lịch trình Đà Nẵng 3 ngày cho 2 người").
- **Complex (10)**: đa ý định/đa bước (ví dụ: vừa yêu cầu lịch trình, vừa yêu cầu khách sạn và chi phí).

4.1.4 Ground truth

- Gán nhãn thủ công intent (manual labeling).
- Expert validation cho các gợi ý (địa điểm/khách sạn/ẩm thực).
- Pilot test (phản hồi người dùng) để đánh giá mức độ thỏa mãn.

4.2 Thước đo đánh giá

4.2.1 Intent Accuracy

$$\text{Intent Accuracy} = \frac{\text{Correct Intent Predictions}}{\text{Total Queries}} \times 100\%$$

4.2.2 Relevance Score

$$\text{Relevance Score} = \frac{\text{Relevant Retrieved Docs}}{\text{Total Retrieved Docs}} \times 100\%$$

4.2.3 Latency

$$\text{Latency} = \text{Time(Response)} - \text{Time(Query)}$$

Đo end-to-end theo milliseconds.

4.2.4 Success Rate

$$\text{Success Rate} = \frac{\text{Successful Responses}}{\text{Total Queries}} \times 100\%$$

4.2.5 Groundedness

$$\text{Groundedness} = \frac{\text{Responses with DB Citations}}{\text{Total Responses}} \times 100\%$$

4.3 Baseline và phương pháp so sánh

4.3.1 Old RAG

Baseline Old RAG thực hiện retrieval một lần (không planning), sau đó đưa context vào LLM để sinh câu trả lời. Cách này đơn giản, latency thấp hơn nhưng yếu với truy vấn multi-intent và dễ thiếu bao phủ yêu cầu.

4.3.2 Plan-RAG

Plan-RAG gồm intent extraction, query planning (decomposition), gọi experts theo miền (có thể song song), tổng hợp và sinh phản hồi. Cách này tăng độ chính xác và khả năng xử lý multi-intent, nhưng có trade-off về latency.

4.4 Kết quả benchmark

4.4.1 So sánh Old RAG và Plan-RAG

Bảng 4.1. So sánh Old RAG và Plan-RAG

Metric	Old RAG	Plan-RAG	Δ	% Improve
Intent Accuracy	0.0%	100.0%	+100.0%	∞
Avg Relevance	9.2%	33.2%	+24.0%	+261%
Avg Latency	1,660 ms	3,665 ms	+2,005 ms	+121%
Success Rate	45%	92%	+47%	+104%
Groundedness	60%	95%	+35%	+58%

Nhận xét: Plan-RAG cải thiện mạnh về intent accuracy, relevance và success rate nhờ cơ chế expert agents và phân rã truy vấn; đổi lại latency tăng do pipeline nhiều bước.

4.4.2 Breakdown theo độ phức tạp

Bảng 4.2. Kết quả theo độ phức tạp truy vấn

Query Type	Queries	Intent Acc	Relevance	Latency	Success
Simple	20	100%	45%	2,100ms	100%
Medium	20	100%	35%	3,500ms	95%
Complex	10	100%	18%	5,800ms	80%

Nhận xét:

- Simple: chất lượng tốt, latency thấp.
- Medium: tốt, latency tăng do nhiều ràng buộc.
- Complex: relevance và success giảm, cần tối ưu decomposition và reranking.

4.4.3 Khả năng xử lý multi-intent

Đánh giá 15 truy vấn multi-intent:

Bảng 4.3. Kết quả xử lý multi-intent (15 queries)

Aspect	Result
Intent Detection	14/15 (93%)
Intent Separation	13/15 (87%)
Response Aggregation	12/15 (80%)
Overall Success	11/15 (73%)

4.4.4 Hiệu năng trích xuất intent

Bảng 4.4. Intent Extraction Performance

Intent	Count	Accuracy	Avg Conf
plan_trip	18	100%	0.92
find_hotel	12	100%	0.95
find_spots	10	100%	0.93
calculate_cost	8	100%	0.88
get_location_tips	5	100%	0.85
find_food	4	100%	0.90

4.4.5 Chất lượng retrieval: Semantic vs Hybrid

Semantic Search:

- Precision@5: 0.82
- Recall@10: 0.68
- MRR: 0.75

Hybrid Search:

- Precision@5: 0.91 (+11%)

- Recall@10: 0.79 (+16%)
- MRR: 0.84 (+12%)

Kết quả cho thấy hybrid search cải thiện đồng thời precision và recall, phù hợp với truy vấn tiếng Việt có biến thể dấu/không dấu và nhiều cách diễn đạt.

4.4.6 Latency breakdown

Bảng 4.5. Phân rã latency theo từng thành phần

Component	Avg Time	% Total
Intent Extraction	850ms	23%
Planning	320ms	9%
Retrieval (Experts)	1,680ms	46%
Aggregation	215ms	6%
Generation (LLM)	600ms	16%
Total	**3,665ms**	**100%**

Bottleneck chính nằm ở retrieval (46%) do phải gọi nhiều expert và truy xuất nhiều nguồn trong truy vấn phức tạp.

4.5 Case studies

4.5.1 Case 1: Simple query

Query: "Địa điểm đẹp ở Đà Lạt".

Intent: `find_spots` (confidence 0.95).

Kết quả: retrieve 8 địa điểm đều liên quan; latency 2,100ms; người dùng hài lòng.

4.5.2 Case 2: Multi-intent query

Query: "Tìm khách sạn và ẩm thực ở Hội An".

Intents: `find_hotel` + `find_food`.

Execution: chạy song song HotelExpert và FoodExpert.

Kết quả: 5 khách sạn + 6 gợi ý ẩm thực; relevance 10/11 (1 outlier); latency 3,200ms.

4.5.3 Case 3: Complex multi-turn

- Turn 1: "Lên lịch trình Đà Nẵng 2 ngày" → `plan_trip`, trả về itinerary builder, lưu context.
- Turn 2: "Tìm khách sạn" → `find_hotel`, dùng context destination.
- Turn 3: "Có lưu ý gì không?" → `get_location_tips`, dùng context itinerary + lựa chọn trước đó.

Kết quả: bảo toàn context tốt, hội thoại tự nhiên.

4.6 Phân tích lỗi, vấn đề và giải pháp

4.6.1 Bug quan trọng ngày 15/01/2026

Vấn đề: Khi user hỏi về địa điểm/khách sạn đã chọn trong itinerary, hệ thống trả lời không chính xác.

Nguyên nhân gốc:

- Không extract entity từ query follow-up.
- Retrieval theo `province_id` thay vì spot/hotel cụ thể.
- Không ưu tiên sử dụng itinerary context.

Hướng fix (ý tưởng): ưu tiên kiểm tra `context.itinerary_builder` và trích xuất danh sách địa điểm đã chọn trước khi retrieve mới.

4.6.2 Các hạn chế còn tồn tại (theo quan sát hệ thống)

- **Follow-up itinerary nâng cao**: các câu như "các địa điểm cách nhau bao xa?", "đổi thứ tự ngày 2 và 3" còn khó xử lý nếu chưa có intent patterns và chưa inject đủ itinerary context.
- **Single device session**: nếu context chỉ lưu local (localStorage), có thể không đồng bộ đa thiết bị.
- **Limited data coverage**: dữ liệu chưa cover đầy đủ các tỉnh/thành ít phổ biến.
- **No real-time pricing**: giá khách sạn có thể không cập nhật theo thời gian thực.
- **Latency**: trade-off của pipeline nhiều bước; complex queries có thể 3–5 giây.

4.6.3 Đề xuất hướng tối ưu

- **Giảm latency**: caching theo session/TTL, song song hóa expert execution, tối ưu top-k, batching truy vấn.
- **Cải thiện complex queries**: nâng planner prompt, tăng retrieval k + reranking, feedback loop từ người dùng.
- **Itinerary follow-up**: thêm intents chuyên biệt (distance, reorder), inject itinerary summary vào prompt.
- **Tăng độ tin cậy giá**: tích hợp API nền tảng booking (real-time pricing) nếu có.

Chương 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Trong chương này, khóa luận tổng kết các kết quả đạt được của hệ thống Travel Advisor dựa trên Plan-RAG, nêu ra các hạn chế còn tồn tại và đề xuất hướng phát triển trong tương lai.

5.1 Kết luận

Sau khi thực hiện khóa luận với đề tài hệ thống tư vấn du lịch Việt Nam theo kiến trúc Plan-RAG, nhóm đạt được các kết quả chính:

- Xây dựng kiến trúc **Plan-RAG** cho domain du lịch tiếng Việt, hỗ trợ multi-intent và multi-turn.
- Thiết kế và triển khai **expert agents** theo từng miền (địa điểm, khách sạn, ẩm thực, lịch trình, chi phí).
- Áp dụng **hybrid search** (FAISS semantic search + MongoDB metadata filtering + reranking).
- Thực nghiệm cho thấy hệ thống đạt **100% intent accuracy**, **92% success rate**, **95% groundedness**; latency trung bình khoảng **3.7s**.

5.2 Hạn chế

Một số hạn chế đáng chú ý:

- **Latency** tăng khi truy vấn phức tạp do gọi nhiều expert và truy xuất nhiều bước.
- **Relevance** với complex queries chưa tối ưu, còn phụ thuộc vào chất lượng query decomposition và chiến lược reranking.
- Hệ thống phụ thuộc vào chất lượng LLM và dữ liệu nền; nếu metadata thiếu/không đồng nhất sẽ ảnh hưởng retrieval.
- Chi phí tài nguyên (đặc biệt cho embedding và index) có thể tăng khi mở rộng dữ liệu.

5.3 Hướng phát triển

Trong tương lai, nhóm định hướng phát triển theo các hướng:

- **Tối ưu latency**: cache theo session, giảm số lần gọi LLM, tối ưu top-k và batching truy xuất.
- **Cải thiện planner**: nâng chất lượng query decomposition, tăng tính ổn định với multi-intent.
- **Reranking tốt hơn**: bổ sung mô hình reranker hoặc tiêu chí domain (rating, popularity, price-fit).
- **Feedback loop**: cho phép người dùng phản hồi để cải thiện relevance và cá nhân hóa gợi ý.
- **Mở rộng dữ liệu**: bổ sung nhà hàng, phương tiện di chuyển, sự kiện theo mùa và dữ liệu cập nhật theo thời gian.

Phụ lục A. TÀI LIỆU THAM KHẢO

Trong phần này, chúng tôi tổng hợp các tài liệu nền tảng liên quan trực tiếp đến kiến trúc Retrieval-Augmented Generation (RAG) và các biến thể có bước lập kế hoạch (Plan-RAG/PlanRAG/Plan*RAG), cùng các tài liệu về hybrid search, embedding và vector search phục vụ cho hệ thống tư vấn du lịch.

A.1 Bài báo khoa học liên quan (khuyến nghị đính kèm)

(1) Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks (Lewis et al., 2020)

- **Vai trò:** Bài báo nền tảng giới thiệu kiến trúc RAG, kết hợp truy hồi tri thức bên ngoài với mô hình sinh để giảm hallucination.
- **Nguồn:** NeurIPS 2020 / arXiv:2005.11401.
- **Liên kết:** https://arxiv.org/abs/2005.11401.

(2) PlanRAG: A Plan-then-Retrieval Augmented Generation for Generative Large Language Models as Decision Makers (Lee et al., 2024)

- **Vai trò:** Đại diện cho hướng *plan-then-retrieve* — mô hình tạo kế hoạch trước, sau đó sinh truy vấn/đọc dữ liệu để ra quyết định.
- **Nguồn:** NAACL 2024.
- **Liên kết:** https://aclanthology.org/2024.naacl-long.364/.

A.2 Danh mục tài liệu tham khảo

BIBLIOGRAPHY

[1] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, Douwe Kiela. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. Advances in Neural Information Processing Systems (NeurIPS), 2020. https://arxiv.org/abs/2005.11401.

[2] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, Wen-tau Yih. *Dense Passage Retrieval for Open-Domain Question Answering*. Proceedings of EMNLP, 2020. https://aclanthology.org/2020.emnlp-main.550/.

[3] Myeonwoo Lee, et al. *PlanRAG: A Plan-then-Retrieval Augmented Generation for Generative Large Language Models as Decision Makers*. NAACL 2024. https://aclanthology.org/2024.naacl-long.364/.

[4] Prakhar Verma, Sukruta Prakash Midigeshi, Gaurav Sinha, Arno Solin, Nagarajan Natarajan, Amit Sharma. *Planning-guided Retrieval Augmented Generation (Plan-RAG)*. OpenReview (ICLR 2025 submission), 2024–2025. https://openreview.net/forum?id=cUuOKnjVQJ.

[5] Prakhar Verma, Sukruta Prakash Midigeshi, Gaurav Sinha, Arno Solin, Nagarajan Natarajan, Amit Sharma. *Plan*RAG: Efficient Test-Time Planning for Retrieval Augmented Generation*. arXiv:2410.20753, 2024. https://arxiv.org/abs/2410.20753.

[6] Nils Reimers, Iryna Gurevych. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. Proceedings of EMNLP-IJCNLP, 2019. https://aclanthology.org/D19-1410/.

[7] Jeff Johnson, Matthijs Douze, Hervé Jégou. *Billion-scale similarity search with GPUs*. arXiv:1702.08734, 2017. https://arxiv.org/abs/1702.08734.

[8] Meta AI Research. *FAISS: A library for efficient similarity search and clustering of dense vectors* (software). https://github.com/facebookresearch/faiss.

[9] Stephen E. Robertson, Hugo Zaragoza. *The Probabilistic Relevance Framework: BM25 and Beyond*. Foundations and Trends in Information Retrieval, 3(4), 2009. DOI: 10.1561/1500000019.

[10] Gordon V. Cormack, Charles L. A. Clarke, Stefan Buettcher. *Reciprocal Rank Fusion outperforms Condorcet and individual rank learning methods*. Proceedings of SIGIR '09, 2009, pp. 758–759. DOI: 10.1145/1571941.1572114.

[11] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, Yuan Cao. *ReAct: Synergizing Reasoning and Acting in Language Models*. arXiv:2210.03629, 2022. https://arxiv.org/abs/2210.03629.

[12] Sebastián Ramírez. *FastAPI Documentation*. https://fastapi.tiangolo.com/.

[13] Vercel. *Next.js Documentation*. https://nextjs.org/docs.

[14] MongoDB. *MongoDB Atlas Documentation*. https://www.mongodb.com/docs/atlas/.