
AI Travel Planner
Design your travel
INFO5002 Python Final Project
Yan Zhao
Seattle-April 15,2025


Part 1. Project Idea

Part 2. Tech Architecture

Part 3. Key Implementation

Part 4. Challenges & Solutions

Part 5. Continuous Improvement

# Melody's Travel Story

**1. Challenge**
Two weeks of exhausting planning, multiple browser tabs, fragmented information, worried that decisions weren't optimal

2. Turning Point
Discovery of the AI Travel Planner, only requires inputting budget, time and preferences to receive personalized recommendations

3. Experience
Smart planning, non-traditional route recommendations, real-time cost visualization, and the flexible adjustments during the journey

**4. Result**
"AI not only saved her time, but also helped her discover experiences she didn't even know she wanted to have"

"Planning has ***transformed*** from the most **painful** part of travel ***into*** **part of the travel experience** itself."

# Part 1

## Part one

Project Idea

# AI Travel Planner

## Pain Points Analysis

Traditional travel planning is time-consuming, labor-intensive, and suffers from fragmented information.

## Solution

- All-in-one AI Travel Planning
- Automated Research + Smart Itinerary Generation (AI Agent)
- Real-Time Cost Estimation & Visualization

Webpage.docx

*Your travel*

# AI Travel Planner ✈️

Plan your next adventure with AI Travel Planner by researching and planning a personalized itinerary on autopilot using llama

Where do you want to go?

Bali

How many days do you want to travel for?

5 − +

How many persons for this trip?

1 − +

How much is the estimated travel expense (US dollar)?

4000 − +

Generate Itinerary

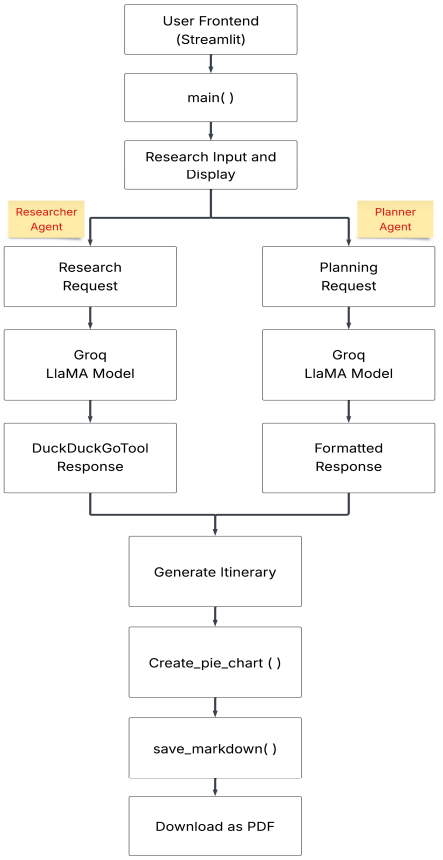

# Part two

## Technical Architecture

# System Architecture Diagram

Lucidchart

[IINK TO LUCIDCHART]




**Our system is built on modern components:**

System Architecture Diagram shows the complete flow from user frontend to research and planning agents. The key is the collaboration between the researcher agent and planner agent.

# Python Tech Stack

- ✓ Core framework: Streamlit for user interface
- ✓ AI integration: Using Groq API and llama model for natural language processing
- ✓ Agent design: Two specialized AI agents (Researcher and Planner)
- ✓ Data structures: Pydantic models for data validation and formatting
- ✓ Data processing: NumPy for numerical operations in cost calculations and chart data
- ✓ Visualization: Matplotlib for expense analysis charts
- ✓ Output formats: Markdown and PDF generation capabilities

# Major Python Web Framework Comparison

| Feature | Streamlit | Flask | Django | FastAPI |
|---|---|---|---|---|
| Design Philosophy | Data visualization focused | Micro-framework, flexible customization | Large, comprehensive full-stack framework | API-first, asynchronous performance |
| State Management | Stateful, manages session state | Optional state, supports sessions | Stateful, complete session system | Stateless, RESTful design |
| Frontend-Backend Relationship | Integrated, automatic frontend rendering | Supports integrated/separated approach | Traditionally integrated | Frontend-backend separation |
| Performance Characteristics | Moderate, interaction-focused | Lightweight, moderate performance | Feature-rich but heavier | High performance, async concurrency |
| Learning Curve | Very gentle, simple to learn | Relatively gentle, highly flexible | Steepest, comprehensive concepts | Relatively gentle, modern approach |
| Best Use Cases | Data science apps/dashboards | Prototypes/small-medium apps | Complex enterprise applications | High-performance APIs/microservices |
| Documentation Generation | Doesn't emphasize API docs | Requires additional extensions | Requires third-party tools | Built-in OpenAPI documentation |
| Data Interaction | Primarily returns interactive UI | Flexible, supports multiple formats | Primarily returns HTML pages | Primarily returns JSON data |

**Before Streamlit:**

Data scientists struggled to share their work, requiring HTML, CSS, JavaScript expertise and web frameworks, or collaboration with front-end developers to build interactive applications, diverting focus from their core analytical work.

**After Streamlit:**

Data scientists can transform complex analyses and machine learning models into interactive web applications using only Python code, dramatically reducing development time and allowing them to concentrate on solving data problems.

```python
# Import the Streamlit library
import streamlit as st

# Create a title for the application
st.title('Streamlit Demo App')

# Display descriptive text on the page
st.write('This is a simple Streamlit application demo')

# Add a header
st.header('Common Streamlit Components')

# Create a sidebar with a slider
st.sidebar.header('Control Panel')
user_input = st.sidebar.slider('Select a value', 0, 100, 50)

# Display the selected value
st.write(f'Selected value: {user_input}')

# Create a two-column layout
col1, col2 = st.columns(2)
```

# Part 3

## Part three

Key Implementation

# Dual-Agent Workflow & Prompt Engineering

1. Researcher Agent

- Automated Query Generation
  - Analyzes user inputs → Generates 3-5 targeted search queries
- Multi-Source Data Aggregation
  - Consolidates activities/hotels/cost data
  - Outputs structured summaries

2. Planner Agent

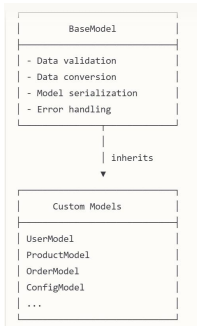
- Type Enforcement
  - Validates costs as integers via Pydantic
- Itinerary Optimization
  - Balances activities by duration/budget

## "Prompt Engineering"

Prompt engineering is **the foundation** of the entire agent design

- Structured Output Control
- Error Prevention

# Structured Output Engine---Pydantic




All Pydantic models inherit from the BaseModel base class

```python
class Output(BaseModel):
    destination: str = Field(
        ...,
        description="The main destination of the trip."
    )
    duration: int = Field(
        ...,
        description="The number of days for the trip."
    )
    cost: Cost = Field(
        ...,
        description="The cost details of the trip."
    )
    itinerary: Itinerary = Field(
        ...,
        description="The detailed itinerary of the trip"
    )
```

Field metadata: Provides additional information such as descriptions, examples, constraints, etc.

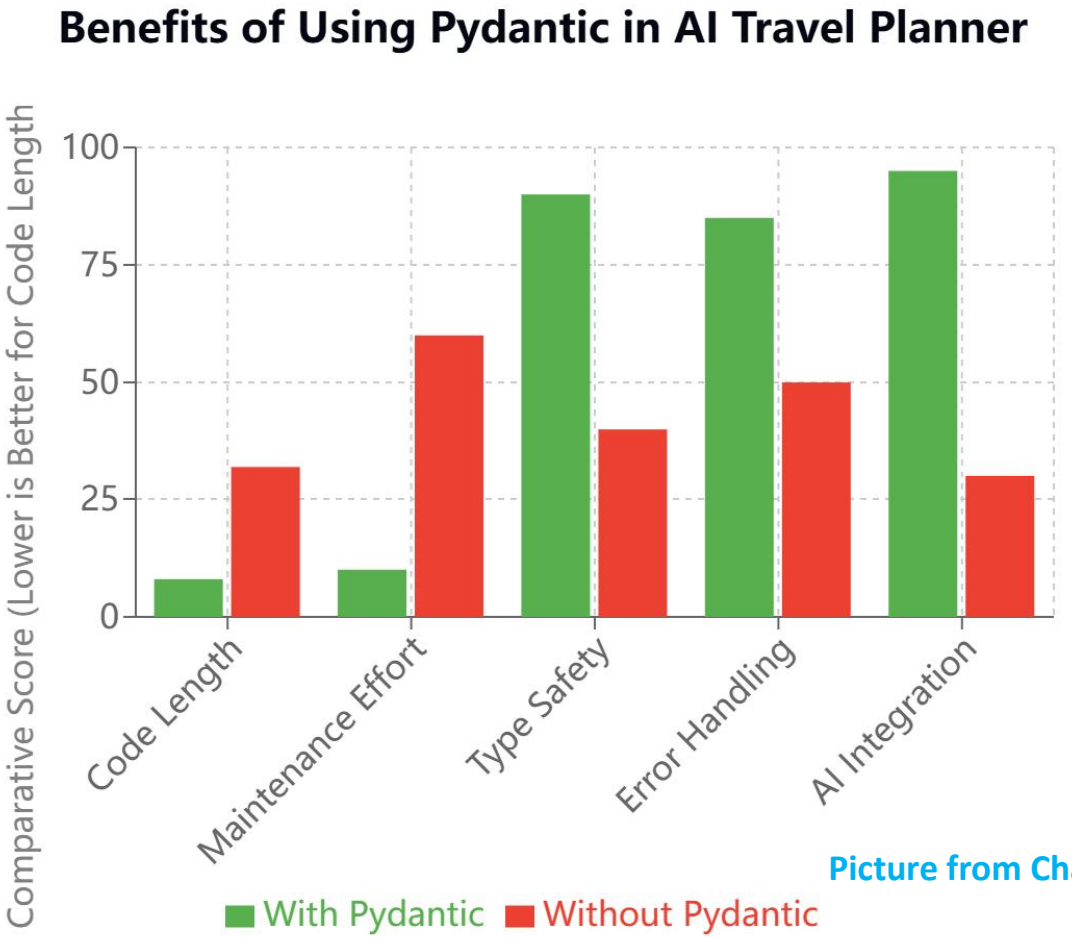
Nested Model Definitions.Field types can be other Pydantic models

### Integration with Agent Framework

```python
planner = Agent(
   # ...other configurations...
   response_model=Output, # Here the framework uses a Pydantic model
)
# Then the framework internally processes the response and applies Pydantic validation
response = planner.run(planner_prompt, stream=False)
```

- Agent Framework:  Agno (internally integrates Pydantic functionality)
- response_model:  Agent Framework Parameter
- Output:  Pydantic Defined Class

*This tells the framework to use a Pydantic model for **validating and structuring AI responses**.*




Picture from ChatGpt

## Compares the benefits of using Pydantic vs not using it.

The benefits of using Pydantic are visually apparent: significant advantages in code length, maintenance effort, type safety, error handling, and AI integration.

| Feature | `agno` | LangChain |
|---|---|---|
| **Data Validation** | Integrates Pydantic directly with `response_model=Output` parameter | Uses output parsers and structured output schemas with more explicit parsing |
| **Agent Implementation** | Clean separation of roles (`TravelResearcher` and `ItineraryPlanner`) with dedicated instructions | Uses agent chains with ReAct framework or custom agent implementations |
| **Tool Integration** | Simple tool integration as seen with `DuckDuckGoTools()` | Has an extensive ecosystem of tool integrations and connections |
| **Model Support** | Demonstrated with Groq API for LLama-3.3-70b | Supports various models including OpenAI, Anthropic, HuggingFace, etc. |
| **Response Handling** | Automatic parsing to Pydantic objects: `if isinstance(response.content, Output)` | Often requires manual parsing setup or separate output parsers |
| **Streaming Support** | Simple streaming support with `stream=False` parameter | More complex streaming API with callbacks |
| **Code Structure** | Cleaner, more declarative approach to defining agents and behaviors | More complex, flexible, but often requires more boilerplate code |
| **Output Generation** | Direct integration with visualization and document generation in your code | Requires additional steps to connect to visualization libraries |
| **Documentation** | Less widely known and possibly less documentation | Extensive documentation and community resources |
| **Ecosystem Size** | Appears more streamlined and focused | Larger ecosystem with more integrations and community contributions |
| **Learning Curve** | Appears simpler from your implementation | Can be steeper due to more components and patterns |

# Agno

# vs

# LangChain

Regarding framework selection, I compared Agno and LangChain:

- Agno is more concise in data validation, agent implementation, response handling, and code structure
- LangChain has a broader ecosystem but comes with increased complexity

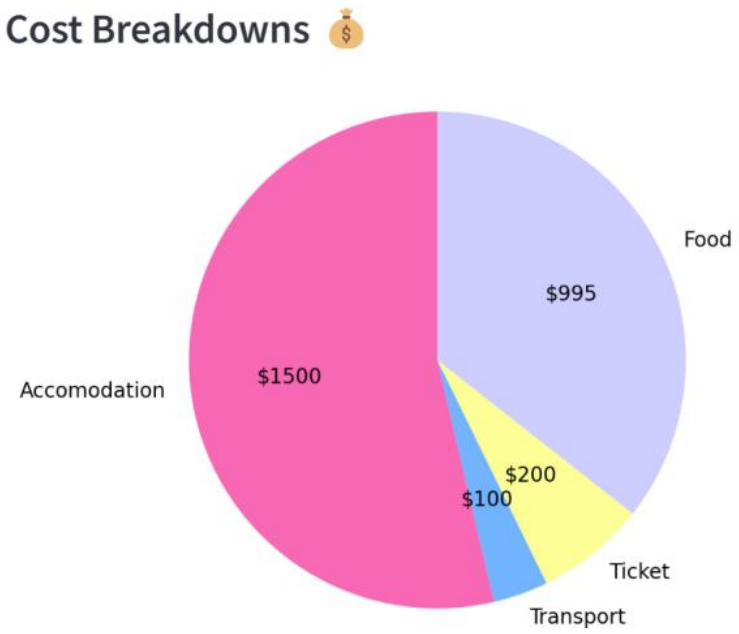
# Cost Visualization

Cost Visualization is one of users' favorite features.

*Pie Chart Generated with Matplotlib*

Display the cost distribution across categories:

- Accommodation
- Transportation
- Tickets
- Food & Beverage




## Colorblind Accessibility

**Visualization Standards**:

| Category | Color Code | Accessibility |
|---|---|---|
| Accommodation | `#4C72B0` | WCAG 2.1 AA Compliant |
| Transportation | `#55A868` | Colorblind Friendly |
| Tickets | `#CCB974` | High Contrast |
| Dining | `#8172B2` | Avoids Red-Green Spectrum |

Color scheme certified for international accessibility standards
Avoiding red-green spectrum, ensuring high contrast

# Part 4

## Part four

Challenges & Solutions

# Challenges & Solutions

**API Rate Limiting Challenges**

I chose Groq. See Next Page

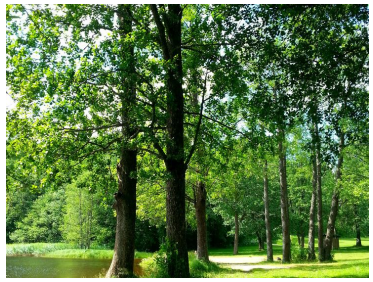
**Streamlit Dynamic Interactivity Challenge**

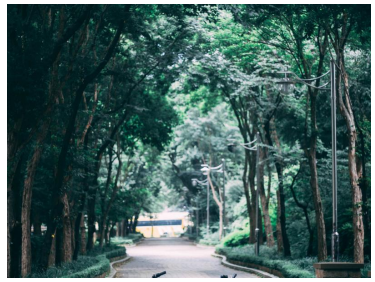
```
with st.spinner("Researching travel information..."):
with st.spinner("Creating your itinerary..."):
```

**A long wait without any feedback.Ultimately solved the problem by separating spinner displays, adding intermediate result presentations**

**Structured Output Formatting Challenges**

**Pydantic**





**PDF Generation Cross-Platform Compatibility**

**Debugging and debugging**

1. **OpenAI (GPT models):** Paid service with usage-based pricing. No significant free tier beyond limited trial credits.
2. **Anthropic (Claude):** Primarily paid service with consumption-based pricing. Limited free access through Claude web interface.
3. **Google (Gemini):** Offers both paid tiers and a free tier with usage limits and reduced capabilities.
4. **Groq:** Has a free tier with generous limits (as you're using in your code), plus paid options for higher usage.
5. **Hugging Face:** Offers both free and paid options. Many open-source models can be run locally for free if you have the hardware, or you can use their Inference API which has both free (with limitations) and paid tiers.

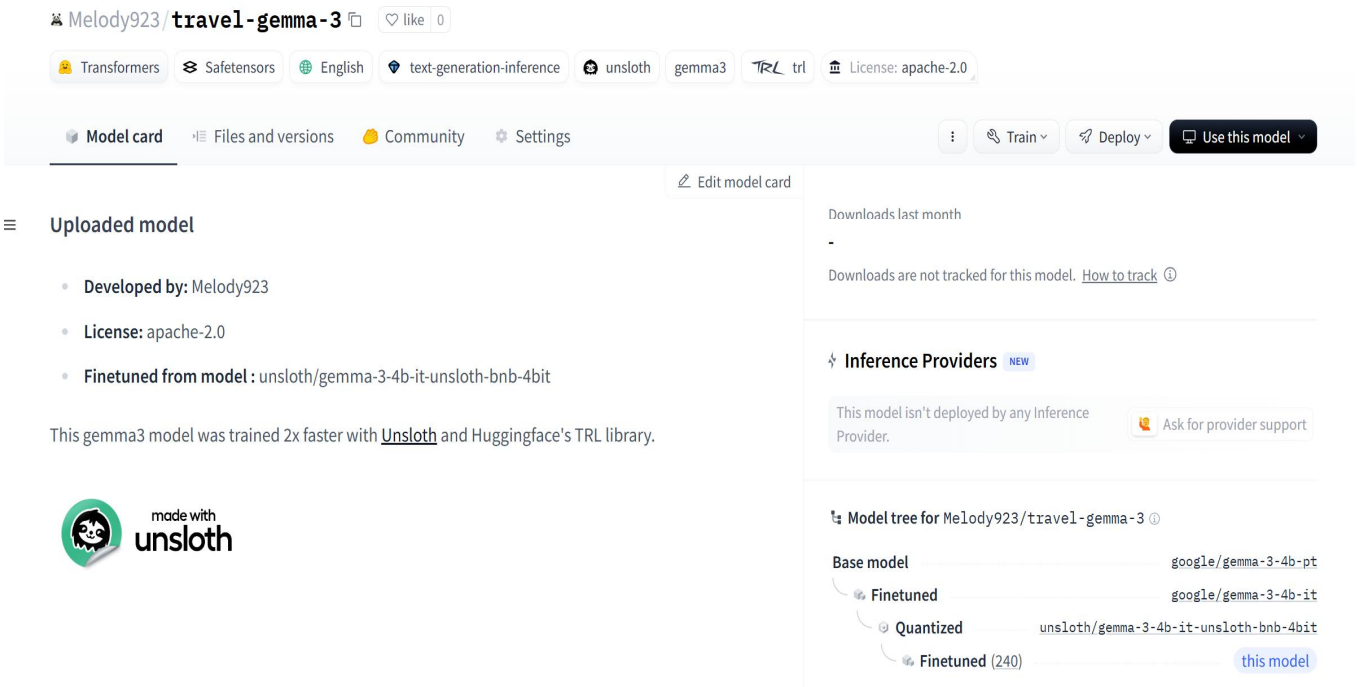
**Among these, Groq and Hugging Face currently offer the most generous free options for developers.**


# Part five

## Continuous Improvement

# Implementation of Fine-Tuned LLM

## Fine-Tuned LLM Uploaded to HuggingFace




[Google Coloab for fine-tuning model]

Base Model: Gemma-3-27B-IT
Dataset from Huggingface

| Rank | Model | Elo | 95% CI | Open | Type | #params/#activated |
|---|---|---|---|---|---|---|
| 1 | Grok-3-Preview-02-24 | 1412 | +8/-10 | - | - | - |
| 1 | GPT-4.5-Preview | 1411 | +11/-11 | - | - | - |
| 3 | Gemini-2.0-Flash-Thinking-Exp-01-21 | 1384 | +6/-5 | - | - | - |
| 3 | Gemini-2.0-Pro-Exp-02-05 | 1380 | +5/-6 | - | - | - |
| 3 | ChatGPT-4o-latest (2025-01-29) | 1377 | +5/-4 | - | - | - |
| 6 | DeepSeek-R1 | 1363 | +8/-6 | yes | MoE | 671B/37B |
| 6 | Gemini-2.0-Flash-001 | 1357 | +6/-5 | - | - | - |
| 8 | o1-2024-12-17 | 1352 | +4/-6 | - | - | - |
| 9 | **Gemma-3-27B-IT** | **1338** | **+8/-9** | **yes** | **Dense** | **27B** |
| 9 | Qwen2.5-Max | 1336 | +7/-5 | - | - | - |
| 9 | o1-preview | 1335 | +4/-3 | - | - | - |
| 9 | o3-mini-high | 1329 | +8/-6 | - | - | - |
| 13 | DeepSeek-V3 | 1318 | +8/-6 | yes | MoE | 671B/37B |
| 14 | GLM-4-Plus-0111 | 1311 | +8/-8 | - | - | - |
| 14 | Qwen-Plus-0125 | 1310 | +7/-5 | - | - | - |
| 14 | Claude 3.7 Sonnet | 1309 | +9/-11 | - | - | - |
| 14 | Gemini-2.0-Flash-Lite | 1308 | +5/-5 | - | - | - |

# Deploying the Fine-Tuned Model

## There are challenges in using the fine-tuned model

1. **Deploying fine-tuned models requires hardware accelerators (GPU/TPU), or the model must be quantized or converted to GGUF/ONNX formats for efficient CPU execution.**

2. **I attempted to quantize the fine-tuned model and deploy it using VLLM on a VM with a single L4 GPU, but encountered a known bug affecting quantized models in the current VLLM version.**

3. **To deploy using OLLAMA or llama.cpp, I would need to convert the model weights to their specific supported formats, which is the direction I need to explore next.**

Continuous Learning...


Thank you
Your travel
Yan Zhao “Melody”
April 15, 2025