# AI Travel Planner: Technical Report

## Multi-Agent System for Intelligent Trip Planning

**Project:** AI Travel Planner

**Author:** Gunasai Vallu

**Date:** January 27, 2026

**Version:** 2.0

## Executive Summary

This report presents a comprehensive analysis of an AI-powered travel planning system built using a multi-agent architecture. The system leverages Large Language Models (LLMs), real-time API integrations, and specialized agents to automate the end-to-end trip planning process. The platform provides users with intelligent flight and hotel recommendations, dynamic itinerary generation, and budget-optimized travel solutions.

**Key Features:**

- Multi-agent coordination for specialized decision-making
- Real-time flight and hotel data retrieval via SerpAPI
- AI-powered reasoning and trade-off analysis
- Dynamic itinerary generation with contextual recommendations
- Budget tracking and currency conversion (INR)
- RESTful API with Streamlit user interface

## Table of Contents



## 1. System Architecture and Agent Roles

### 1.1 High-Level Architecture

The system implements a **layered multi-agent architecture** using LangGraph for orchestration and LangChain for LLM integration. The architecture consists of four primary layers:

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■
■ Presentation Layer ■
■ (Streamlit UI + FastAPI Endpoints) ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■
■
■■■■■■■■■■■■■■■■■■■■■■■■▼■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■
■ Agent Orchestration Layer ■
■ (LangGraph State Machine + Multi-Agent Router) ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■
■
■■■■■■■■■■■■■■■■■■■■■■■■▼■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■
■ Specialized Agents Layer ■
■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■
■ ■ Flight ■ ■ Hotel ■ ■ Reasoning■ ■ Place ■ ■
■ ■ Agent ■ ■ Agent ■ ■ Agent ■ ■ Agent ■ ■
■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■
■
■■■■■■■■■■■■■■■■■■■■■■■■▼■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■
■ Tools & Data Layer ■
■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■
■ ■ SerpAPI ■ ■ Google ■ ■ Weather ■ ■ Currency ■ ■
■ ■ Flights ■ ■ Places ■ ■ API ■ ■ API ■ ■
■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■■■■■■■■■■■■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■
```

## 1.2 Core Components

### 1.2.1 Agent Orchestrator (`agentic_workflow.py`)

The **GraphBuilder** class implements the state machine that coordinates all agents:

**Key Responsibilities:**

- Initializes and binds tools to the LLM
- Manages conversation state and message history
- Routes execution between agent node and tool nodes
- Implements circuit breaker pattern (max 10 tool calls)
- Detects completion based on response patterns

**State Management:**

```
class AgentState(TypedDict):
messages: Annotated[List[AnyMessage], operator.add]
```

```
tool_calls_count: int # Tracks tool invocations
```

**Workflow Logic:**

1. **Entry Point:** Agent receives user request with trip parameters

2. **Decision Node:** Agent decides whether to call tools or generate response

3. **Tool Execution:** External APIs are called via tool nodes

4. **Iteration:** Results feed back to agent for next decision

5. **Termination:** Agent generates final markdown response when:

- 10 tool calls reached (circuit breaker)
- Complete itinerary detected in response
- No more tool calls needed

## 1.2.2 Specialized Agents

##### **Flight Agent** (`flight_agent.py`)

Implements sophisticated flight evaluation using weighted scoring:

**Evaluation Criteria:**

- **Price (50% weight):** Primary cost optimization factor
- **Duration (30% weight):** Total travel time minimization
- **Layovers (20% weight):** Preference for direct flights

**Algorithm:**

```
score = (norm_price × 0.50) + (norm_duration × 0.30) + (norm_layovers × 0.20)
```

**Normalization:**

- Prices and durations normalized to [0, 1] range
- Layovers: 0.0 (direct), 0.5 (1 stop), 1.0 (2+ stops)
- Lower score = better option

**Output:**

- Sorted list of flights by composite score
- Top recommendation tagged with "AI Recommended" and "Best Value"
- Human-readable justification for each recommendation

##### **Hotel Agent** (`hotel_agent.py`)

Analyzes accommodation options with focus on value optimization:

**Evaluation Criteria:**

- User ratings (primary factor)
- Price per night (affordability)
- Location proximity to key destinations
- Available amenities

**Selection Logic:**

```
best = max(hotels, key=lambda h: (h["rating"] or 0, -h["price_per_night"]))
```

**Categorization:**

- **Budget:** < ■5,000/night
- **Moderate:** ■5,000 - ■15,000/night
- **Luxury:** > ■15,000/night

##### **Reasoning Agent** (`reasoning_agent.py`)

Uses LLM to provide transparent explanations:

**Responsibilities:**

- Analyzes trade-offs between flight options (budget vs. premium)
- Compares hotel choices (location vs. price)
- Justifies final recommendations based on user preferences
- Explains opportunity costs of different choices

**Example Output:**

```
Budget vs Premium:
- Budget option saves ■12,000 but has 1 layover (adds 3 hours)
- Premium option offers direct flight, saving time and stress
- Recommendation: Budget flight - savings justify minor inconvenience
```

##### **Place Search Agent** (`place_search_tool.py`)

Discovers local attractions, restaurants, and activities:

**Data Sources:**

- Primary: Google Places API
- Fallback: Tavily Search API

**Search Functions:**

- `search_attractions()`: Top tourist spots
- `search_restaurants()`: Dining recommendations
- `search_activities()`: Experience-based activities
- `search_transportation()`: Local transit options

## 1.3 System Prompt Engineering

The system uses a comprehensive prompt (`prompt.py`) that defines:

**Workflow Phases:**

1. **Phase 1 - Logistics:** Flight + Hotel + Weather searches

2. **Phase 2 - Content Discovery:** Attractions + Restaurants + Activities

3. **Phase 3 - Synthesis:** Stop calling tools, generate markdown response

**Stop Rules:**

- CRITICAL: After 6-8 tool calls, generate final response
- Prevents infinite tool-calling loops
- Ensures token budget compliance

**Output Format:**

- Structured markdown with sections:
- Flight options (categorized)
- Hotel options (categorized)
- Weather forecast
- Reasoning agent analysis

- Day-by-day itinerary
- Budget breakdown
- Essential travel info

### 1.4 Model Configuration

The system supports multiple LLM providers via `model_loader.py`:

**Supported Models:**

- **Groq:** Meta-Llama-4-Scout-17B (128K context, primary choice)
- **OpenAI:** GPT-4o
- **Anthropic:** Claude 3.5 Sonnet

**Anti-Hallucination Settings:**

- Temperature: 0.1 (very low for factual accuracy)
- Top-p: 0.85 (focused sampling)
- Max tokens: 8,000 for responses

## 2. Data Sources and API Integrations

### 2.1 SerpAPI Integration

**Primary Data Provider:** SerpAPI (Google Flights & Google Hotels)

#### 2.1.1 Flight Search (`flight_serpapi_tool.py`)

**API Endpoint:** `google_flights` engine

**Request Parameters:**

```
{
"engine": "google_flights",
"departure_id": origin_iata_code,
"arrival_id": destination_iata_code,
"outbound_date": "YYYY-MM-DD",
"currency": "INR",
"gl": "in", # Geographic location (India)
"hl": "en", # Language
"type": "2" # One-way (1=Round-trip, 2=One-way)
}
```

**Response Processing:**

1. Parse `best_flights` and `other_flights` arrays

2. Extract multi-leg flight information

3. Calculate total duration from all legs

4. Identify layover cities and count stops

5. Format times to 12-hour format

6. Apply Flight Agent scoring algorithm

**Data Extracted per Flight:**

- Airline name and flight number
- Departure/arrival times and airports
- Total duration (converted from minutes to hours/minutes)
- Number of layovers
- Route path (origin → stops → destination)
- Carbon emissions
- Price in INR

**Fallback Mechanism:**

- If no flights found for requested date, retry 30 days ahead
- Ensures user always receives results

### 2.1.2 Hotel Search (`hotel_serpapi_tool.py`)

**API Endpoint:** `google_hotels` engine

**Request Parameters:**

```
{
"engine": "google_hotels",
"q": f"hotels in {location}",
"check_in_date": "YYYY-MM-DD",
"check_out_date": "YYYY-MM-DD",
"adults": "2",
"currency": "INR",
"sort_by": 8 # Lowest price
}
```

**Response Processing:**

1. Filter hotels with ratings ≥ 4.0/5.0

2. Parse price strings (handle ■ and $ symbols)

3. Extract location from GPS coordinates or vicinity

4. Limit amenities to top 3 (token optimization)

5. Calculate total cost (price × nights)

6. Categorize into Budget/Moderate/Luxury

**Data Extracted per Hotel:**

- Hotel name and type
- Overall rating and review count
- Price per night and total cost
- Location address (truncated to 75 chars)
- Top amenities (max 3)
- Check-in/check-out times

**Quota Management:**

- Returns up to 10 hotels per category
- Prevents token overflow while maintaining variety

## 2.2 Weather API Integration

**Provider:** OpenWeatherMap API

**Tool:** `get_weather_forecast()` in `weather_info_tool.py`

**Functionality:**

- Fetches 5-day forecast at 3-hour intervals
- Aggregates data by date to show daily high/low
- Aligns forecast with travel dates when possible
- Converts temperatures to Celsius

**Request:**

```
url =
f"http://api.openweathermap.org/data/2.5/forecast?q={city}&units;=metric"
```

**Processing:**

1. Group forecasts by date

2. Calculate daily high/low temperatures

3. Identify most common weather condition

4. Format output as readable forecast

**Output Example:**

```
■■ 5-Day Weather Forecast for Delhi:
Mon, 27 Jan: High 18.5°C / Low 8.2°C, Clear Sky
Tue, 28 Jan: High 19.1°C / Low 9.0°C, Partly Cloudy
...
```

## 2.3 Google Places API Integration

**Tool:** `place_search_tool.py` wrapping Google Places and Tavily

**Search Types:**

1. **Attractions:** Top tourist destinations

2. **Restaurants:** Dining options with cuisine types

3. **Activities:** Experience-based recommendations

4. **Transportation:** Local transit modes

**Fallback Strategy:**

- Primary: Google Places API (higher accuracy)
- Fallback: Tavily Search (broader coverage)

**Error Handling:**

```
try:
result = google_places_search.search_attractions(place)
except Exception as e:
result = tavily_search.search_attractions(place) # Fallback
```

## 2.4 Currency Conversion

**Tool:** `currency_conversion_tool.py`

**Provider:** ExchangeRate-API

**Functionality:**

- Real-time currency conversion
- Supports 150+ currencies
- Used for budget calculations

## 2.5 Utility Tools

**Expense Calculator:**

- `calculate_total_expense()`: Sums trip costs
- `estimate_total_hotel_cost()`: Price × nights
- `calculate_daily_expense_budget()`: Total ÷ days

**Arithmetic Operations:**

- Basic math tools for LLM to calculate budgets
- Prevents hallucinated calculations

# 3. Reasoning and Recommendation Logic

## 3.1 Multi-Agent Decision Framework

The system implements a **hierarchical decision-making process** where specialized agents handle domain-specific evaluations, and a reasoning agent synthesizes recommendations.

### 3.1.1 Flight Selection Algorithm

**Step 1: Data Normalization**

```
norm_price = (price - min_price) / (max_price - min_price)
norm_duration = (duration - min_duration) / (max_duration - min_duration)
norm_layovers = min(layovers * 0.5, 1.0)
```

**Step 2: Weighted Scoring**

```
score = (norm_price × 0.50) +
(norm_duration × 0.30) +
(norm_layovers × 0.20)
```

**Step 3: Ranking**

- Sort flights by ascending score (lower = better)
- Tag top option with "AI Recommended"

**Step 4: Justification Generation**

```
if norm_price == 0.0: reasons.append("Lowest Price")
elif norm_price <= 0.2: reasons.append("Great Value")

if norm_time == 0.0: reasons.append("Fastest Route")
elif norm_time <= 0.2: reasons.append("Quick Flight")
```

```
if layovers == 0: reasons.append("Non-stop")
```

**Example:**

```
Flight: IndiGo 2153 - ■8,450
Score: 0.12
Tags: ["AI Recommended", "Best Value"]
Reason: "Great Value, Quick Flight, Non-stop"
```

### 3.1.2 Hotel Selection Algorithm

**Evaluation Formula:**

```
best_hotel = max(hotels, key=lambda h: (h["rating"], -h["price"]))
```

**Logic:**

1. Prioritize highest-rated hotels (4.0+ stars)

2. Among equal ratings, prefer lower price

3. Consider location proximity (manual inspection)

**Categorization:**

- **Budget:** Price < ■5,000/night
- Target: Budget-conscious travelers
- Trade-off: May be farther from city center
- **Moderate:** ■5,000 - ■15,000/night
- Target: Mid-range travelers
- Balance: Location + amenities + price
- **Luxury:** Price > ■15,000/night
- Target: Premium experience seekers
- Benefits: Central location, full amenities

### 3.1.3 Reasoning Agent Logic

The Reasoning Agent uses the LLM to generate natural language explanations:

**Prompt Structure:**

```
prompt = f"""
Explain why the following flight and hotel were selected.

Flight: {flight_details}
Hotel: {hotel_details}

Clearly explain trade-offs and benefits.
"""
```

**Output Components:**

1. **Flight Trade-offs:**

- Budget vs. Premium comparison
- Time vs. money analysis

- Comfort vs. savings explanation

2. **Hotel Trade-offs:**
   - Location vs. Price analysis
   - Amenities vs. Budget explanation
   - Ratings vs. Cost comparison
3. **Final Recommendation:**
   - Synthesized choice based on user preferences
   - Clear reasoning for each selection
   - Total cost breakdown

**Example Reasoning:**

```
Flight Trade-offs:
Budget vs Premium:
- Budget option (SpiceJet ■6,200) saves ■4,800 but has 1 layover
- Premium option (Vistara ■11,000) is non-stop, saves 2 hours
- Recommendation: Budget flight - the savings of ■4,800 justify
the 2-hour layover for a 5-day trip.

Hotel Trade-offs:
Location vs Price:
- Budget hotel (■3,500/night) is 8km from center, basic amenities
- Moderate hotel (■8,000/night) is in city center, includes breakfast
- Recommendation: Moderate hotel - central location saves ■2,000
in daily transport and adds convenience.
```

## 3.2 Itinerary Generation Logic

**Dynamic Planning Algorithm:**

**Step 1: Time Allocation**

- Morning: 9 AM - 12 PM (3 hours)
- Afternoon: 12 PM - 5 PM (5 hours, includes lunch)
- Evening: 5 PM - 9 PM (4 hours, includes dinner)

**Step 2: Activity Matching**

- Match activities to trip vibe (Relaxed, Adventure, Cultural, etc.)
- Use real place names from `search_attractions()` results
- Rotate activity types to avoid monotony

**Step 3: Cost Estimation**

- Attractions: ■200-■1,000 per entry
- Meals: ■500-■1,500 per person
- Transport: ■100-■500 per trip

**Step 4: Vibe-Based Customization**

- **Relaxed:** Cafes, beaches, parks
- **Adventure:** Hiking, water sports, zip-lining
- **Cultural:** Museums, temples, historical sites
- **Nightlife:** Bars, clubs, night markets
- **Family:** Zoos, theme parks, child-friendly spots

## 3.3 Budget Calculation Engine

**Comprehensive Cost Breakdown:**

```
Total Trip Cost =
(Flight Price × Travelers) +
(Hotel Price × Nights) +
(Food Cost × Days × Travelers) +
(Local Transport × Days) +
(Attractions × Count) +
(Contingency × 10%)
```

**Food Estimates:**

- Budget: ■800/person/day
- Moderate: ■1,500/person/day
- Luxury: ■3,000/person/day

**Transport Estimates:**

- Metro/Bus: ■100/day
- Auto/Taxi: ■500/day
- Car Rental: ■2,000/day

## 3.4 Quality Assurance Mechanisms

**1. Token Budget Management:**

- Limit tool calls to 10 maximum
- Truncate long addresses to 75 characters
- Use compact JSON (remove whitespace)
- Limit hotels to 10 per category

**2. Data Validation:**

- Date validation (prevent past dates)
- Price parsing (handle multiple currency symbols)
- Rating filters (≥ 4.0 stars only)
- Deduplication (prevent repeated entries)

**3. Error Handling:**

- Try-catch blocks for all API calls
- Fallback dates for unavailable flights
- Fallback search engines (Tavily after Google Places)
- Graceful degradation when APIs fail

# 4. Limitations and Potential Improvements

## 4.1 Current Limitations

### 4.1.1 Data Coverage Limitations

**Flight Data:**

- **Limitation:** Dependent on SerpAPI which scrapes Google Flights
- May not include all budget airlines
- Limited to routes Google Flights covers
- Real-time pricing may have delays (5-15 minutes)

- **Impact:** Users might miss cheaper options from airline-specific deals

**Hotel Data:**

- **Limitation:** Google Hotels may not include:
- Airbnb or vacation rentals
- Smaller guesthouses and hostels
- Properties not listed on booking platforms
- **Impact:** Limited options for budget-conscious or alternative accommodation seekers

**Places Data:**

- **Limitation:** Relies on Google Places and Tavily
- May miss newly opened attractions
- Limited to places with online presence
- May not include local hidden gems
- **Impact:** Itineraries may feel generic for frequent travelers

### 4.1.2 Reasoning Limitations

**LLM Hallucination Risk:**

- **Issue:** Despite low temperature (0.1), LLM may still:
- Invent place names if search results are sparse
- Provide outdated cultural information
- Make assumptions about user preferences
- **Mitigation:** System prompt enforces "use REAL names from tools"

**Trade-off Analysis Depth:**

- **Issue:** Current reasoning is price-focused
- Doesn't account for personal preferences (vegetarian, accessibility)
- Limited context on traveler experience level
- No consideration of travel insurance or visa requirements

### 4.1.3 Scalability Limitations

**Token Budget Constraints:**

- **Issue:** Groq free tier has token limits
- Long trips (>7 days) may hit context window
- Multiple travelers increase complexity
- Detailed itineraries consume significant tokens

**API Rate Limits:**

- SerpAPI: 100 searches/month on free tier
- Google Places: 1,000 requests/month
- OpenWeatherMap: 1,000 calls/day
- **Impact:** System may become unavailable under high traffic

### 4.1.4 User Experience Limitations

**No Booking Integration:**

- System provides recommendations but doesn't allow direct booking
- Users must manually copy details to booking sites
- No price tracking or alerts

**Limited Personalization:**

- No user profile storage
- No learning from past trips

• No saved preferences across sessions

**No Multi-City Support:**

• Current architecture supports only A→B trips
• Cannot plan A→B→C→A routes
• No support for road trips with multiple stops

## 4.2 Potential Improvements

### 4.2.1 Enhanced Data Integration

**Recommendation 1: Expand Data Sources**

**Proposed Additions:**

```
# Add Skyscanner API for more flight options
from skyscanner import FlightsSearch
flights_1 = serpapi_search()
flights_2 = skyscanner_search()
combined_flights = merge_and_deduplicate(flights_1, flights_2)

# Add Booking.com API for more hotels
from booking import HotelSearch
hotels_1 = google_hotels_search()
hotels_2 = booking_com_search()
combined_hotels = merge_and_rank(hotels_1, hotels_2)
```

**Benefits:**

• 2-3x more flight options
• Better coverage of budget accommodations
• More competitive pricing

**Recommendation 2: Add User Reviews and Sentiment Analysis**

```
# Scrape TripAdvisor reviews
from langchain_community.tools import TripAdvisorReviewsTool

reviews = get_reviews(hotel_name)
sentiment = analyze_sentiment(reviews) # Positive/Neutral/Negative

# Adjust rating based on recent feedback
adjusted_rating = base_rating + sentiment_modifier
```

**Benefits:**

• More nuanced hotel recommendations
• Detect recent quality changes
• Warn about deteriorating properties

**Recommendation 3: Real-Time Price Tracking**

```
# Implement price monitoring
class PriceTracker:
def track_flight(self, route, date):
current_price = get_price(route, date)
historical_avg = get_avg_price(route, lookback=30)
```

```
if current_price < historical_avg * 0.85:
return {"status": "GREAT_DEAL", "savings": savings}
elif current_price > historical_avg * 1.15:
return {"status": "WAIT", "suggestion": "Price likely to drop"}
```

**Benefits:**

- Help users decide when to book
- Alert on price drops
- Prevent overpaying during peak seasons

### 4.2.2 Advanced Reasoning Capabilities

**Recommendation 4: Persona-Based Recommendations**

**Implementation:**

```
class TravelerPersona:
def __init__(self, type: str):
self.type = type # "budget", "luxury", "family", "solo", "adventure"
self.preferences = self._load_preferences()

def _load_preferences(self):
if self.type == "family":
return {
"hotel_priorities": ["kid_friendly", "pool", "breakfast"],
"activities": ["zoos", "parks", "museums"],
"avoid": ["nightclubs", "bars", "extreme_sports"]
}
# ... other personas
```

**Prompt Engineering:**

```
system_prompt = f"""
You are planning for a {persona.type} traveler.
Priorities: {persona.preferences}
When recommending hotels, PRIORITIZE: {persona.hotel_priorities}
When planning activities, FOCUS ON: {persona.activities}
AVOID: {persona.avoid}
"""
```

**Benefits:**

- More relevant recommendations
- Better user satisfaction
- Reduced need for manual filtering

**Recommendation 5: Multi-Criteria Decision Analysis (MCDA)**

**Current:** Simple weighted scoring

**Proposed:** AHP (Analytic Hierarchy Process)

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np
```

```
class AdvancedFlightRanker:
def rank(self, flights, user_priorities):
"""
user_priorities = {
'price': 0.4,
'duration': 0.3,
'airline_quality': 0.15,
'departure_time': 0.10,
'carbon_footprint': 0.05
}
"""
scores = []
for flight in flights:
score = sum(
normalize(flight[criterion]) * weight
for criterion, weight in user_priorities.items()
)
scores.append(score)

return sorted(zip(flights, scores), key=lambda x: x[1])
```

**Benefits:**

- More granular user control
- Better handling of complex preferences
- Transparency in decision-making

**Recommendation 6: Conversational Refinement**

**Current:** One-shot planning

**Proposed:** Multi-turn conversation

```
class IterativeRefinement:
def refine_plan(self, initial_plan, user_feedback):
"""
User: "This itinerary has too much walking"
System: Adjusts to include more rest time, reduces daily activities

User: "I want more cultural experiences"
System: Swaps adventure activities for museums and heritage sites
"""
feedback_analysis = analyze_feedback(user_feedback)

if "too much walking" in feedback_analysis:
new_plan = reduce_walking_distance(initial_plan)
new_plan = add_rest_breaks(new_plan)

return new_plan
```

**Benefits:**

- Higher user satisfaction
- Personalized adjustments

• Learning user preferences

### 4.2.3 Scalability Enhancements

**Recommendation 7: Caching Layer**

```
import redis
from functools import lru_cache

class CachedFlightSearch:
def __init__(self):
self.redis = redis.Redis(host='localhost', port=6379)

def search(self, origin, dest, date):
cache_key = f"flights:{origin}:{dest}:{date}"

# Check cache (valid for 1 hour)
cached = self.redis.get(cache_key)
if cached:
return json.loads(cached)

# API call
result = serpapi.search(...)
self.redis.setex(cache_key, 3600, json.dumps(result))

return result
```

**Benefits:**

• Reduce API costs (SerpAPI charges per search)
• Faster responses (avoid repeated API calls)
• Better handling of concurrent users

**Recommendation 8: Asynchronous Processing**

```
import asyncio
from concurrent.futures import ThreadPoolExecutor

async def parallel_search(origin, dest, date):
with ThreadPoolExecutor() as executor:
# Run searches concurrently
flight_future = executor.submit(search_flights, origin, dest, date)
hotel_future = executor.submit(search_hotels, dest, date, date+3)
weather_future = executor.submit(get_weather, dest)
places_future = executor.submit(search_attractions, dest)

# Wait for all to complete
flights = flight_future.result()
hotels = hotel_future.result()
weather = weather_future.result()
places = places_future.result()

return {
"flights": flights,
"hotels": hotels,
```

```
"weather": weather,
"places": places
}
```

**Benefits:**

- 3-4x faster response time
- Better user experience
- Handle more concurrent users

**Recommendation 9: Model Optimization**

**Current:** Groq Llama-4-Scout-17B (128K context)

**Proposed:** Hybrid approach

```
class HybridModelStrategy:
def __init__(self):
self.fast_model = ChatGroq(model="llama-3.1-8b-instant") # Quick tasks
self.smart_model = ChatGroq(model="llama-4-scout-17b") # Complex reasoning

def route_request(self, task_complexity):
if task_complexity == "simple":
return self.fast_model # E.g., format conversion
else:
return self.smart_model # E.g., trade-off analysis
```

**Benefits:**

- Reduce costs (use cheaper model when possible)
- Faster responses for simple tasks
- Reserve powerful model for complex reasoning

### 4.2.4 User Experience Enhancements

**Recommendation 10: Direct Booking Integration**

```
# Integrate with Booking.com Affiliate API
class BookingIntegration:
def generate_booking_link(self, hotel_id, checkin, checkout):
affiliate_id = os.getenv("BOOKING_AFFILIATE_ID")
return f"https://www.booking.com/hotel/{hotel_id}?" \
f"checkin={checkin}&checkout;={checkout}&aid;={affiliate_id}"

def track_conversion(self, booking_id):
# Earn commission on completed bookings
pass
```

**Benefits:**

- Monetization opportunity (affiliate commissions)
- Seamless user experience
- Direct price comparison

**Recommendation 11: User Profiles and Memory**

```
from langchain.memory import ConversationBufferMemory

class UserProfile:
def __init__(self, user_id):
self.user_id = user_id
self.preferences = self._load_preferences()
self.past_trips = self._load_history()

def update_from_trip(self, trip_data):
# Learn from user choices
if trip_data["selected_budget_hotel"]:
self.preferences["budget_conscious"] += 0.1

if trip_data["selected_adventure_activities"]:
self.preferences["adventure_seeker"] += 0.1

def personalize_search(self, query):
# Adjust search based on past behavior
if self.preferences.get("prefers_direct_flights"):
query["filter_layovers"] = True

return query
```

**Benefits:**

- Faster planning for repeat users
- Better recommendations over time
- Reduced cognitive load

**Recommendation 12: Mobile App Development**

**Current:** Web-only (Streamlit)

**Proposed:** React Native mobile app

```
// React Native component
import { TravelPlannerAPI } from './api';

const TripPlanner = () => {
const [location, setLocation] = useState(null);

useEffect(() => {
// Get user's current location
navigator.geolocation.getCurrentPosition((pos) => {
setLocation({
lat: pos.coords.latitude,
lng: pos.coords.longitude
});
});
}, []);

const generatePlan = async () => {
const plan = await TravelPlannerAPI.planTrip({
origin: location, // Auto-fill from GPS
destination: destination,
```

```
dates: selectedDates
});

return plan;
};

return (
<View>
<Button title="Plan My Trip" onPress={generatePlan} />
</View>
);
};
```

**Benefits:**

- Better mobile experience (70% of travel searches are mobile)
- Push notifications for price drops
- Offline access to saved itineraries

### 4.2.5 Advanced Features

**Recommendation 13: Multi-City Route Optimization**

**Use Case:** User wants to visit Delhi → Jaipur → Agra → Delhi

**Implementation:**

```
from scipy.optimize import linear_sum_assignment
import numpy as np

class MultiCityPlanner:
def optimize_route(self, cities, start_city):
"""
Uses Traveling Salesman Problem (TSP) algorithm
to find optimal visiting order
"""
# Build distance matrix
n = len(cities)
distances = np.zeros((n, n))

for i, city_a in enumerate(cities):
for j, city_b in enumerate(cities):
if i != j:
distances[i][j] = self.get_distance(city_a, city_b)

# Solve TSP
optimal_order = self.solve_tsp(distances, start_city)

# Generate itinerary for each leg
itinerary = []
for i in range(len(optimal_order) - 1):
leg = self.plan_leg(
origin=optimal_order[i],
destination=optimal_order[i+1]
```

```
)
itinerary.append(leg)

return itinerary
```

**Benefits:**

- Minimize total travel time
- Reduce backtracking
- Optimize for user preferences (scenic vs. fast)

**Recommendation 14: Group Trip Coordination**

```
class GroupTripPlanner:
def coordinate_group(self, participants):
"""
Handle multiple travelers with different:
- Origin cities
- Budget constraints
- Preferences
"""
# Find common destination
destination = self.find_central_destination(participants)

# Find flights that arrive around same time
synchronized_flights = self.sync_arrivals(participants, destination)

# Book accommodations with group discounts
group_hotel = self.find_group_hotel(
travelers=len(participants),
budget=avg([p.budget for p in participants])
)

# Merge preferences
consensus_itinerary = self.merge_preferences(
[p.preferences for p in participants]
)

return {
"flights": synchronized_flights,
"hotel": group_hotel,
"itinerary": consensus_itinerary
}
```

**Benefits:**

- Coordinate complex group logistics
- Handle conflicting preferences democratically
- Group discount opportunities

**Recommendation 15: Visa and Documentation Assistant**

```
class VisaAssistant:
def check_requirements(self, origin_country, dest_country):
"""
```

```
Check visa requirements and provide guidance
"""
# API call to visa requirement database
requirements = visa_api.check(origin_country, dest_country)

if requirements["visa_required"]:
return {
"visa_type": requirements["type"],
"processing_time": requirements["processing_days"],
"cost": requirements["fee"],
"documents_needed": requirements["documents"],
"application_link": requirements["application_url"],
"warning": f"Apply at least {requirements['processing_days']} days before
travel"
}
else:
return {
"visa_required": False,
"passport_validity": "Ensure passport valid for 6 months"
}
```

**Benefits:**

- Prevent travel disruptions
- Proactive documentation reminders
- Better trip preparation

## 4.3 Implementation Roadmap

**Phase 1 (Months 1-2): Data Quality**

- Add Skyscanner and Booking.com APIs
- Implement caching layer
- Add price tracking

**Phase 2 (Months 3-4): Advanced Reasoning**

- Implement persona-based recommendations
- Add conversational refinement
- Multi-criteria decision analysis

**Phase 3 (Months 5-6): Scalability**

- Async processing
- Hybrid model strategy
- Database for user profiles

**Phase 4 (Months 7-9): UX Enhancements**

- Mobile app development
- Booking integration
- User profile system

**Phase 5 (Months 10-12): Advanced Features**

- Multi-city route optimization
- Group trip coordination
- Visa assistance

# Appendix: System Diagrams

## A1. System Architecture Diagram

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■
■ USER INTERFACE ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■
■ ■
■ ■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■ ■
■ ■ Streamlit UI ■ ■ FastAPI REST ■ ■
■ ■ (Port 8501) ■■■■■■■■■■■■■■■■ (Port 8001) ■ ■
■ ■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■ ■
■ ■ ■ ■
■ ■ HTTP POST /plan-trip ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■
■ AGENT ORCHESTRATION ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■
■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ LangGraph State Machine ■ ■
■ ■ ■ ■
■ ■ ■■■■■■■■■■■ ■■■■■■■■■■■ ■■■■■■■■■■■ ■ ■
■ ■ ■ Agent ■■■■■■■■■■ Tools ■■■■■■■■ Agent ■ ■ ■
■ ■ ■ Node ■ ■ Node ■ ■ Node ■ ■ ■
■ ■ ■■■■■■■■■■■ ■■■■■■■■■■■ ■■■■■■■■■■■ ■ ■
■ ■ ■ ■ ■ ■ ■
■ ■ ■ ■ ■ ■ ■
■ ■ ▼ ▼ ▼ ■ ■
■ ■ Max 10 Tool Calls Execute APIs Generate ■ ■
■ ■ Circuit Breaker Return Results Final Plan ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■
■ SPECIALIZED AGENTS ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■
```

```
■ ■
■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■
■ ■Flight Agent ■ ■Hotel Agent ■ ■Reasoning ■ ■
■ ■ ■ ■ ■ ■Agent ■ ■
■ ■• Scores by: ■ ■• Filters: ■ ■ ■ ■
■ ■ - Price(50%)■ ■ - Rating≥4.0■ ■• Explains ■ ■
■ ■ - Time(30%) ■ ■ - Budget ■ ■ trade-offs ■ ■
■ ■ - Stops(20%)■ ■ - Location ■ ■• Justifies ■ ■
■ ■ ■ ■ ■ ■ choices ■ ■
■ ■• Returns ■ ■• Categorizes ■ ■• Uses LLM ■ ■
■ ■ ranked list ■ ■ 3 tiers ■ ■ reasoning ■ ■
■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■
■ ■ ■ ■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■
■ ■ ■
▼ ▼ ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■
■ TOOLS LAYER ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■
■ ■
■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■
■ ■search_ ■ ■search_ ■ ■get_weather_ ■ ■
■ ■flights() ■ ■hotels() ■ ■forecast() ■ ■
■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■
■ ■ ■ ■ ■
■ ■■■■■■■■▼■■■■■■■■ ■■■■■■■■▼■■■■■■■■ ■■■■■■■■▼■■■■■■■■ ■
■ ■search_ ■ ■search_ ■ ■convert_ ■ ■
■ ■attractions()■ ■restaurants()■ ■currency() ■ ■
■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■
■ ■ ■ ■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■
■ ■ ■
▼ ▼ ▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■
■ EXTERNAL APIs ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■
■ ■
■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■
■ ■ SerpAPI ■ ■Google Places ■ ■OpenWeatherMap■ ■
■ ■ ■ ■ API ■ ■ API ■ ■
■ ■• Flights ■ ■ ■ ■ ■ ■
■ ■• Hotels ■ ■• Attractions ■ ■• 5-day ■ ■
■ ■ ■ ■• Restaurants ■ ■ forecast ■ ■
■ ■INR pricing ■ ■• Activities ■ ■• Celsius ■ ■
■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■ ■
■ ■
```

```
■ ■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■ ■
■ ■ Tavily ■ ■ExchangeRate ■ ■
■ ■ Search ■ ■ API ■ ■
■ ■ ■ ■ ■ ■
■ ■• Fallback ■ ■• Currency ■ ■
■ ■ search ■ ■ conversion ■ ■
■ ■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■ ■
■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■
```

## A2. Request Flow Diagram

```
■■■■■■■■■■■■■■
■ USER ■
■■■■■■■■■■■■■■
■
■ 1. Enter trip details
■ (origin, destination, dates, budget, vibe)
▼
■■■■■■■■■■■■■■■■■■■■■■■■
■ Streamlit UI ■
■■■■■■■■■■■■■■■■■■■■■■■■
■
■ 2. HTTP POST /plan-trip
■ {from_city, destination, start_date, ...}
▼
■■■■■■■■■■■■■■■■■■■■■■■■■
■ FastAPI App ■
■■■■■■■■■■■■■■■■■■■■■■■■■
■
■ 3. Create HumanMessage with comprehensive prompt
■ Including: workflow phases, stop rules, output format
▼
■■■■■■■■■■■■■■■■■■■■■■■■■
■ GraphBuilder ■
■ (State Machine) ■
■■■■■■■■■■■■■■■■■■■■■■■■■
■
■ 4. Initialize state:
■ {messages: [...], tool_calls_count: 0}
▼
■■■■■■■■■■■■■■■■■■■■■■■■■
■ Agent Node ■
■■■■■■■■■■■■■■■■■■■■■■■■■
■
■ 5. LLM analyzes request
■ Decision: Call tools or respond?
▼
■■■■ Tool Call? ■■■ YES ■■
■ ■
■ ▼
```

```
■ ■■■■■■■■■■■■■■■■■■■■■
■ ■ Tool Node ■
■ ■■■■■■■■■■■■■■■■■■■■■
■ ■ • search_flights ■
■ ■ • search_hotels ■
■ ■ • get_weather ■
■ ■ • search_places ■
■ ■■■■■■■■■■■■■■■■■■■■■
■ ■
■ ■ 6. Execute API calls
■ ■ (SerpAPI, Google, etc.)
■ ▼
■ ■■■■■■■■■■■■■■■■■■■■■
■ ■ External APIs ■
■ ■ Return Results ■
■ ■■■■■■■■■■■■■■■■■■■■■
■ ■
■ ■ 7. Append results to messages
■ ■ Increment tool_calls_count
■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■
■
■ 8. Loop back to Agent Node
■ (Repeat 5-7 until stop condition)
■
■ Stop Conditions:
■ • tool_calls_count >= 10
■ • Complete itinerary detected
■ • No more tool calls needed
■
▼
■■■■■■■■■■■■■■■■■■■■■
■ Agent Node ■
■ (Final Pass) ■
■■■■■■■■■■■■■■■■■■■■■
■
■ 9. Generate complete markdown response
■ • Flight options (Budget/Moderate/Premium)
■ • Hotel options (categorized)
■ • Weather forecast
■ • Reasoning analysis
■ • Day-by-day itinerary
■ • Budget breakdown
▼
■■■■■■■■■■■■■■■■■■■■■
■ FastAPI ■
■ Return Response ■
■■■■■■■■■■■■■■■■■■■■■
■
■ 10. JSON response with markdown
▼
■■■■■■■■■■■■■■■■■■■■■
```

```
■ Streamlit UI ■
■ Display Tabs: ■
■ • Overview ■
■ • Flights ■
■ • Hotels ■
■ • Itinerary ■
■ • Budget ■
■■■■■■■■■■■■■■■■■■■■■
■
■ 11. Rendered trip plan
▼
■■■■■■■■■■■■
■ USER ■
■ Downloads■
■ MD/PDF ■
■■■■■■■■■■■■
```

## A3. Flight Agent Decision Flow

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ search_flights() Tool Called ■
■ Input: origin, destination, date ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ SerpAPI: google_flights Engine ■
■ Request Parameters: ■
■ • departure_id: IATA code ■
■ • arrival_id: IATA code ■
■ • outbound_date: YYYY-MM-DD ■
■ • currency: INR ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Parse Response ■
■ • best_flights[] ■
■ • other_flights[] ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Extract Flight Data (per flight): ■
■ • Airline, Flight Number ■
■ • Departure/Arrival Times & Airports ■
■ • Total Duration (minutes) ■
■ • Layover Count ■
■ • Price (INR) ■
■ • Carbon Emissions ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
```

```
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Flight Agent Evaluation ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
■■■■ Step 1: Normalize
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■
■ ■ norm_price = ■
■ ■ (price - min) / ■
■ ■ (max - min) ■
■ ■ ■
■ ■ norm_duration = ■
■ ■ (dur - min) / ■
■ ■ (max - min) ■
■ ■ ■
■ ■ norm_layovers = ■
■ ■ min(stops*0.5, 1.0) ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■
■
■■■■ Step 2: Calculate Score
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■
■ ■ score = ■
■ ■ (n_price × 0.50) + ■
■ ■ (n_dur × 0.30) + ■
■ ■ (n_lay × 0.20) ■
■ ■ ■
■ ■ Lower score = Better ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■
■
■■■■ Step 3: Generate Reason
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■
■ ■ IF n_price == 0: ■
■ ■ "Lowest Price" ■
■ ■ IF n_dur == 0: ■
■ ■ "Fastest Route" ■
■ ■ IF layovers == 0: ■
■ ■ "Non-stop" ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Sort Flights by Score (Ascending) ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Tag Top Flight: ■
■ • "AI Recommended" ■
■ • "Best Value" ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
▼
```

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Categorize All Flights: ■
■ • Budget: Bottom 1/3 by price ■
■ • Moderate: Middle 1/3 ■
■ • Premium: Top 1/3 ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■ Return JSON: ■
■ { ■
■ "route": "DEL → BOM", ■
■ "flights": [ ■
■ { ■
■ "Airline": "IndiGo", ■
■ "Price": 6200, ■
■ "Score": 0.12, ■
■ "Tags": ["AI Recommended"], ■
■ "Category": "Budget", ■
■ "Reason": "Great Value, ■
■ Non-stop" ■
■ }, ■
■ ... ■
■ ], ■
■ "count": 9, ■
■ "currency": "INR" ■
■ } ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

## A4. Data Flow Diagram

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■
■ USER INPUT ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■
■ origin: "Dubai" ■
■ destination: "Delhi" ■
■ start_date: "2026-02-15" ■
■ days: 5 ■
■ travelers: 2 ■
■ budget: "Moderate" ■
■ vibe: "Cultural" ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■
■ PHASE 1: LOGISTICS ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■
```

```
■ ■
■ ■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ search_flights ■ ■ search_hotels ■ ■
■ ■ ■ ■ ■ ■
■ ■ Input: ■ ■ Input: ■ ■
■ ■ • origin: DXB ■ ■ • location: DEL ■ ■
■ ■ • dest: DEL ■ ■ • checkin: 02-15 ■ ■
■ ■ • date: 02-15 ■ ■ • checkout: 02-20■ ■
■ ■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ ■ ■
■ ▼ ▼ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ SerpAPI (Google Flights/Hotels) ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ ■ ■
■ ▼ ▼ ■
■ ■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ Flights JSON ■ ■ Hotels JSON ■ ■
■ ■ • 9 options ■ ■ • 30 options ■ ■
■ ■ • INR prices ■ ■ • Categorized ■ ■
■ ■ • Flight Agent ■ ■ • Rated 4.0+ ■ ■
■ ■ scored ■ ■ ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■ ■
■ ■
■ ■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ get_weather ■ ■
■ ■ ■ ■
■ ■ Input: ■ ■
■ ■ • city: Delhi ■ ■
■ ■ • date: 02-15 ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ ■
■ ▼ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ OpenWeatherMap API ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ ■
■ ▼ ■
■ ■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ Weather Data ■ ■
■ ■ • 5-day forecast ■ ■
■ ■ • High/Low temps ■ ■
■ ■ • Conditions ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■
■ PHASE 2: CONTENT DISCOVERY ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

```
■■■■■
■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■search_attractions■ ■search_restaurants■ ■
■ ■ ■ ■ ■ ■
■ ■Input: Delhi ■ ■Input: Delhi ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ ■ ■
■ ▼ ▼ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ Google Places API (Primary) ■ ■
■ ■ Tavily Search (Fallback) ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ ■ ■
■ ▼ ▼ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ Attractions List ■ ■ Restaurants List ■ ■
■ ■ • Red Fort ■ ■ • Karim's ■ ■
■ ■ • India Gate ■ ■ • Bukhara ■ ■
■ ■ • Qutub Minar ■ ■ • Indian Accent ■ ■
■ ■ • Lotus Temple ■ ■ • Saravana Bhavan■ ■
■ ■ • Humayun's Tomb ■ ■ • Moti Mahal ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■search_activities ■ ■
■ ■ ■ ■
■ ■Input: Delhi ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ ■
■ ▼ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ Activities List ■ ■
■ ■ • Food tours ■ ■
■ ■ • Heritage walks ■ ■
■ ■ • Shopping ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■
■ PHASE 3: REASONING & SYNTHESIS ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■
■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ LLM (Llama-4-Scout-17B) ■ ■
■ ■ ■ ■
■ ■ Input Context: ■ ■
■ ■ • All flight data ■ ■
■ ■ • All hotel data ■ ■
```

■ ■ • Weather forecast ■ ■
■ ■ • Attractions list ■ ■
■ ■ • Restaurants list ■ ■
■ ■ • Activities list ■ ■
■ ■ • User preferences (budget, vibe) ■ ■
■ ■ ■ ■
■ ■ Processing: ■ ■
■ ■ 1. Analyze flight trade-offs ■ ■
■ ■ 2. Compare hotel options ■ ■
■ ■ 3. Match activities to "Cultural" vibe ■ ■
■ ■ 4. Generate 5-day itinerary ■ ■
■ ■ 5. Calculate comprehensive budget ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ ■
■ ▼ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■ ■ FINAL MARKDOWN RESPONSE ■ ■
■ ■ ■ ■
■ ■ # ✈■ 5-Day Trip: Dubai → Delhi ■ ■
■ ■ ■ ■
■ ■ ## ■ Flight Options ■ ■
■ ■ ### Budget Flights ■ ■
■ ■ **Air India AI915** - ■18,500 ■ ■
■ ■ ■ Departs: 09:15 AM from DXB ■ ■
■ ■ ■ Arrives: 02:30 PM at DEL ■ ■
■ ■ ■■ Duration: 3h 15m ■ ■
■ ■ ■ Non-stop ■ ■
■ ■ ■ ■
■ ■ **Flight Agent Recommendation:** ■ ■
■ ■ ■ Best Value: Air India AI915 ■ ■
■ ■ Reason: Non-stop, Great timing, Budget-friendly■ ■
■ ■ ■ ■
■ ■ ## ■ Hotels in Delhi ■ ■
■ ■ ### Moderate Hotels ■ ■
■ ■ **The Leela Ambience** ■4.5/5 ■ ■
■ ■ ■ ■8,500/night × 5 nights = ■42,500 ■ ■
■ ■ ■ Connaught Place (City Center) ■ ■
■ ■ ■ Pool, Spa, Free Breakfast ■ ■
■ ■ ■ ■
■ ■ ## ■ Reasoning Agent Analysis ■ ■
■ ■ For a Moderate Cultural trip, I recommend: ■ ■
■ ■ ✈■ Flight: Air India AI915 (■18,500) ■ ■
■ ■ Direct flight, no layover hassle ■ ■
■ ■ ■ Hotel: The Leela Ambience (■8,500/night) ■ ■
■ ■ Central location saves transport costs ■ ■
■ ■ ■ Total Core Cost: ■79,500 ■ ■
■ ■ ■ ■
■ ■ ## ■ DETAILED DAY-BY-DAY ITINERARY ■ ■
■ ■ **Day 1: Arrival & Mughal Heritage** ■ ■
■ ■ * Morning: Red Fort visit (■600) ■ ■
■ ■ * Afternoon: Lunch at Karim's (■800/person) ■ ■
■ ■ * Evening: India Gate & Rashtrapati Bhavan ■ ■

```
■ ■ ■ ■
■ ■ **Day 2: Spiritual Delhi** ■ ■
■ ■ * Morning: Lotus Temple (Free entry) ■ ■
■ ■ * Afternoon: Akshardham Temple (■250) ■ ■
■ ■ * Evening: Dinner at Indian Accent (■2,500) ■ ■
■ ■ ■ ■
■ ■ ... (Days 3-5 continue) ■ ■
■ ■ ■ ■
■ ■ ## ■ Budget Breakdown ■ ■
■ ■ | Flights | ■18,500 × 2 = ■37,000 ■ ■
■ ■ | Hotels | ■8,500 × 5 = ■42,500 ■ ■
■ ■ | Food | ■1,500 × 5 × 2 = ■15,000 ■ ■
■ ■ | Transport | ■500 × 5 = ■2,500 ■ ■
■ ■ | Attractions | ■3,000 ■ ■
■ ■ | Contingency | ■10,000 ■ ■
■ ■ |--------------|-------------------------------| ■ ■
■ ■ | **TOTAL** | **■110,000** ■ ■
■ ■ | **Per Person**| **■55,000** ■ ■
■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■
■
▼
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■
■ USER INTERFACE ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■
■ Streamlit renders 5 tabs: ■
■ 1. Overview - Full markdown ■
■ 2. Flights - Filtered flight section ■
■ 3. Hotels - Filtered hotel section ■
■ 4. Itinerary - Day-by-day expandable ■
■ 5. Budget - Financial breakdown ■
■ ■
■ Download options: ■
■ • Markdown (.md file) ■
■ • PDF (via ReportLab) ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■
```

## A5. Tool Execution Timeline

```
Time Agent Action Tool Called API Response Time
■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■
0:00 User submits trip request - -
0:01 GraphBuilder initializes - -
0:02 Agent analyzes prompt - -
0:03 → Decision: Call flight tool search_flights() -
0:04 ↓ SerpAPI call - 2.5s
0:06 ↓ Flight Agent processes - 0.3s
```

```
0:07 ← Return 9 categorized flights - -
0:08 Agent receives flight data - -
0:09 → Decision: Call hotel tool search_hotels() -
0:10 ↓ SerpAPI call - 3.1s
0:13 ↓ Filter & categorize - 0.4s
0:13 ← Return 30 hotels (3 tiers) - -
0:14 Agent receives hotel data - -
0:15 → Decision: Call weather get_weather_forecast() -
0:16 ↓ OpenWeatherMap API - 1.2s
0:17 ↓ Parse & format - 0.1s
0:17 ← Return 5-day forecast - -
0:18 Agent receives weather - -
0:19 → Decision: Call attractions search_attractions() -
0:20 ↓ Google Places API - 1.8s
0:22 ← Return top 10 attractions - -
0:23 Agent receives attractions - -
0:24 → Decision: Call restaurants search_restaurants() -
0:25 ↓ Google Places API - 1.5s
0:26 ← Return restaurant list - -
0:27 Agent receives restaurants - -
0:28 → Decision: Call activities search_activities() -
0:29 ↓ Google Places API - 1.3s
0:30 ← Return activity options - -
0:31 Agent receives activities - -
0:32 Tool calls count: 6 - -
0:33 → Decision: Generate response - -
0:34 ↓ LLM synthesis (8K tokens) - 12.5s
0:46 ← Complete markdown itinerary - -
0:47 FastAPI returns JSON - -
0:48 Streamlit renders tabs - -
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■■■■■■■■■■■■■■■■■■■■■
TOTAL TIME: 48 seconds
```

**Performance Breakdown:**

- API Calls: 11.4s (24%)
- LLM Processing: 12.5s (26%)
- Data Processing: 0.8s (2%)
- Network/Overhead: 23.3s (48%)

**Optimization Opportunities:**

- Parallel API calls: Could reduce to ~3s (currently sequential)
- Caching: Repeated searches save 100% of API time
- Async processing: 40-50% faster overall

## Conclusion

The AI Travel Planner demonstrates a robust multi-agent architecture capable of automating complex trip planning workflows. By leveraging specialized agents for flight evaluation, hotel analysis, and contextual reasoning, the system delivers intelligent, transparent recommendations.

**Key Achievements:**

- • ■ Functional multi-agent coordination via LangGraph
- • ■ Real-time data integration from multiple sources
- • ■ Transparent reasoning and justification for recommendations
- • ■ Dynamic itinerary generation with real place names
- • ■ Comprehensive budget tracking in INR
- • ■ User-friendly interface with downloadable outputs

**Areas for Growth:**

- Enhanced data coverage through additional APIs
- Advanced reasoning with persona-based recommendations
- Scalability improvements via caching and async processing
- Direct booking integration for seamless user experience
- Mobile app development for broader accessibility

The system provides a strong foundation for future enhancements, with clear pathways to production-ready deployment through the outlined improvement roadmap.

**Report Prepared By:** Gunasai Vallu

**Contact:** vallugunasai05@gmail.com

**Project Repository:** AI-TRAVEL-PLANNER v2.0

**Date:** January 27, 2026