WanderWise: A Multimodal AI System for Personalized Trip Planning Using Mood, Language, and Visual Search

1. Problem Statement

Tourists visiting Egypt often concentrate on just a few popular landmarks (e.g., Pyramids, Luxor Temple), while many culturally rich locations remain unexplored. This is due to:

- Lack of personalized travel planning tools
- Overreliance on user review counts rather than quality
- No support for mood, crowd level, or visual exploration
- Difficulty in building full-day or multi-day plans across Stay, Do, and Eat experiences

2. Solution Overview

We propose a modular AI-powered recommendation system with three interconnected modes, each targeting a unique user interaction pathway:

2.1 Mood-Based Recommendation (One-Day)

- User selects weights across six curated moods: *Family, Sports, Adventure, History, Entertainment, Art*
- Returns same-day personalized activities, food, and hotels
- Avoids overcrowded locations using predicted crowd scores

2.2 Prompt-Based RAG Trip Planner (Multi-Day)

- User provides natural language prompt (e.g., "5-day historical trip to Aswan with family")
- System parses preferences, retrieves best candidates, and generates structured, multi-day itinerary with GPT-4

2.3 Visual Landmark Recommendation (Image + Text)

- User uploads an image of a monument and specifies a location (e.g., Luxor)
- System returns 3 visually similar places from that city using CLIP embeddings and cosine similarity

3. Dataset Overview

Our system relies on **three primary datasets**, each serving a distinct module in the pipeline: mood-based recommendation, prompt-driven itinerary generation, and image-based visual retrieval.

3.1 Tourism & Activity Dataset (WanderWise_with_CrowdScores.xlsx)

Data Collection

- Scraped from 4 heterogeneous online platforms:
    - **TripAdvisor**
    - **Expedia**
    - **GetYourGuide**
    - **Top Rated Online**
- Each source provided different structures (some focused on reviews, others on duration/pricing)
- Unified into a **core tourism dataset** containing attractions, events, restaurants, and experiences

Preprocessing Steps:

- **Column Unification**: Removed platform-specific fields (e.g., booking URLs), retained common ones
- **Standardization**:
    - Capitalization normalization: "luxor" → "Luxor"
    - Price cleaning: removed ranges, converted to float
    - Duration filled using median per category
- **Duplicate Removal**: Used title + location as composite keys
- **Description Cleaning**:
    - Removed HTML, emojis, extra spaces
    - Trimmed promotional content (e.g., "Book now!")
- **Text Enhancement with GPT**:

  - GPT models rephrased vague/low-quality descriptions
  - Extracted potential tags (e.g., “family-friendly”)

Mood Annotation:

- Manually curated mood categories: Family, Sports, Adventure, History, Entertainment, Art
- Each activity assigned a mood vector: [0.8, 0.0, 0.6, 1.0, 0.1, 0.9] Example for a historical art museum
- Tagging was semi-automated using GPT for inference and human-in-the-loop corrections

Crowd Scores:

Each activity includes:

```
{
 "8-10": 0.31,
 "10-12": 0.71,
 "12-14": 0.85,
 ...
}
```

- Scores were:
  - Estimated from review timestamps
  - Adjusted with popularity index
  - Normalized (0–1)
- Used to **avoid suggesting crowded locations at peak times**

Purpose:

- **Feeds the Mood-Based Recommender** and **RAG-Based Do/Eat modules**
- Embeddings are computed using all-mpnet-base-v2 for semantic retrieval
- Mood tags used for filtering and cosine similarity calculations

3.2 Hotels Dataset (Hotels-16-3.xlsx)

Why Separate?

Hotel data had:

- Different schema (rooms, amenities, star ratings)
- Structured numerical data vs. text-heavy activities

Preprocessing:

- **Missing Price**: Imputed using city-level median
- **Amenities**: Standardized and grouped (e.g., “WiFi” vs. “Free Wifi”)
- **Text Enrichment**: Combined = Description + " | Amenities: " + Amenities + " | Near: " + Nearby_Landmarks
- **Embedding**: Combined field embedded with all-mpnet-base-v2 for semantic search

Hidden Gem Boosting:

Hotels with:

- Rating ≥ 4.7 and
- Number of reviews < 10
  are **boosted** in retrieval score to increase visibility of **less-explored but high-potential places**.

Purpose:

- Used exclusively by the Stay module (both mood-based and RAG planner)
- Stored in Stay_Embeddings ChromaDB collection

3.3 Image Dataset (Egypt Landmarks – Kaggle)

Description:

- Sourced from Kaggle dataset: “Egypt Landmarks”
- Contains over 1,000 images across dozens of Egyptian cities
- Original metadata was minimal; only file name and image

Restructuring: To support **city-level filtering** in image retrieval, the folder structure was manually reorganized into:

```
temples/
 └── luxor/
     └── luxor_temple/
         └── img1.jpg
         └── img2.jpg
 └── aswan/
     └── philae/
         └── img1.jpg
```

Purpose:

- Powers **Image-Based Recommender**
- When a user uploads an image and selects a city:
  - Their image is embedded
  - Cosine similarity is computed between their embedding and all embeddings from the selected city
  - Top 3 matches returned

4. Technical Implementation Details

4.1 Mood-Based Recommender

Goal: Provide quick 1-day trip suggestions based on user mood

How it works:

- Each Do activity is annotated with six mood scores (0 to 1)
- When a user selects their mood weights (e.g., Family = 0.9, Art = 0.6), we compute the cosine similarity between the mood input vector and each activity's mood vector
- Top N matches are filtered by location and time slot (using crowd scores)

Filtering Logic:

- Activities with similarity ≥ 0.6
- Time slots with crowd score ≤ 0.5
- Eat and Stay recommendations are also filtered by location, rating, and mood-related tags

4.2 RAG-Based Prompt-to-Itinerary System

Goal: Convert a natural language user prompt into a structured, multi-day, cost-aware travel plan

Components:

- Prompt Interpreter (gpt-3.5-turbo):
  Parses user prompt into structured JSON containing:
  - location, budget, duration, stay_description, do_description
- Semantic Search Engine:
  - All entries in Stay, Do, and Eat datasets are embedded using all-mpnet-base-v2 (SentenceTransformers)
  - Stored in ChromaDB vector databases for fast cosine similarity search
  - User's parsed preferences are embedded and used to retrieve matching records
- **Itinerary Generator (gpt-4-turbo):**
  - Injects retrieved records and user constraints into a carefully designed prompt
  - LLM builds a structured day-by-day plan:
    - Selects 1 hotel
    - Assigns 2–4 activities/day (based on time and crowd)
    - Suggests 3 meals/day (Eat module)
    - Respects budget and avoids duplication

Example Prompt Logic:

Given:

- Budget: 40000

- Duration: 5 days

- Location: Dahab

- Hotel options: [retrieved]

- Activities: [retrieved]

- Meals: [retrieved]

Generate: 5-day plan with 3 meals/day, up to 12 hrs/day, 1 hotel, unique items, stay within budget.

4.3 Visual Landmark Search using CLIP

Goal: Retrieve visually similar landmarks within a specified city using an image query

Steps:

1. Load and cache all images from temples/ folder
2. Encode each image using:
3. Normalize embeddings to unit vectors
4. Store alongside metadata (place name, city, folder path)
5. When a user uploads a query image and selects a city:
    - Encode the image using CLIP
    - Compare against all embeddings from that city using cosine similarity
    - Return top 3 most visually similar results

Enhancement: The system ensures location filtering, meaning similar results are contextually relevant to the city specified.

System Architecture Overview

The system is built as a modular pipeline with shared resources:

- **Shared Embedding Store:**
  ChromaDB holds vector embeddings for all tourism records, split into collections:
  - Stay_Embeddings, Do_Embeddings, Eat_Embeddings
- **LLM Components:**
  - gpt-3.5-turbo: Query parser
  - gpt-4-turbo: Itinerary planner
- **Retrieval Engines:**
  - Cosine similarity for mood and image systems
  - Filtered vector search via SentenceTransformers
- **Frontend Layer:**
  - Built with Gradio
  - Input modes: Mood sliders, Text prompt, Image + City
  - Outputs: Day-wise plan, costs, best times, images

5. Application Demo & User Interaction

The system is deployed using a Gradio frontend, which provides:

Mode	Input	Output
Mood-Based	Mood sliders	1-day itinerary (Stay, Do, Eat)
RAG-Based	Prompt (text)	Multi-day itinerary
Visual Search	Image + City	Top 3 visually similar landmarks

Each result includes:

- Name, location, rating, description, category
- Estimated best time to visit (crowd-based)
- Budget breakdown per day
- Hotel and food options
- Highlighting of hidden gems

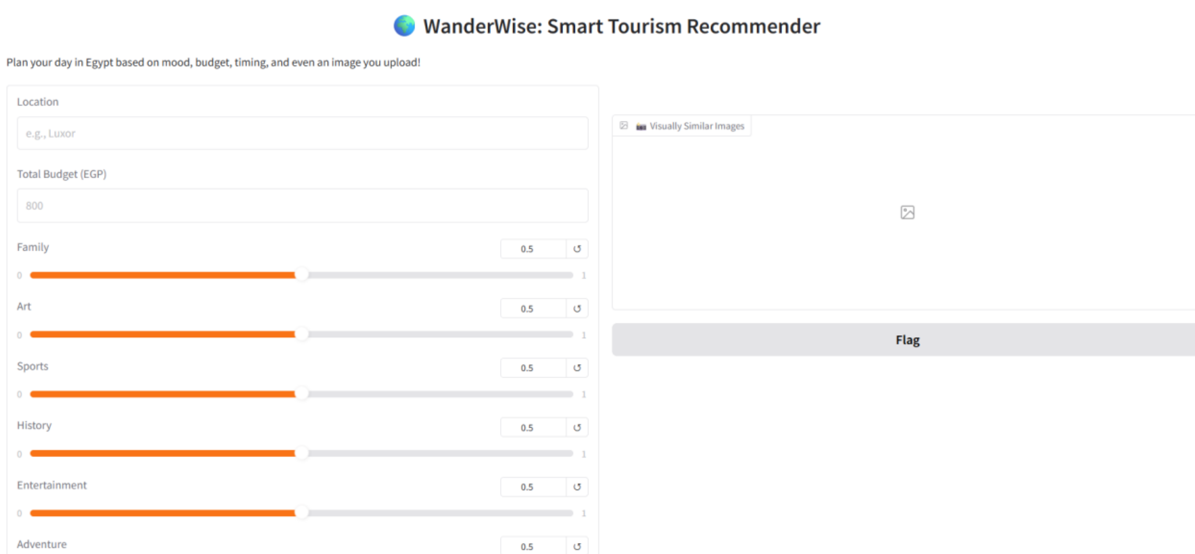

Fig1. System Interface

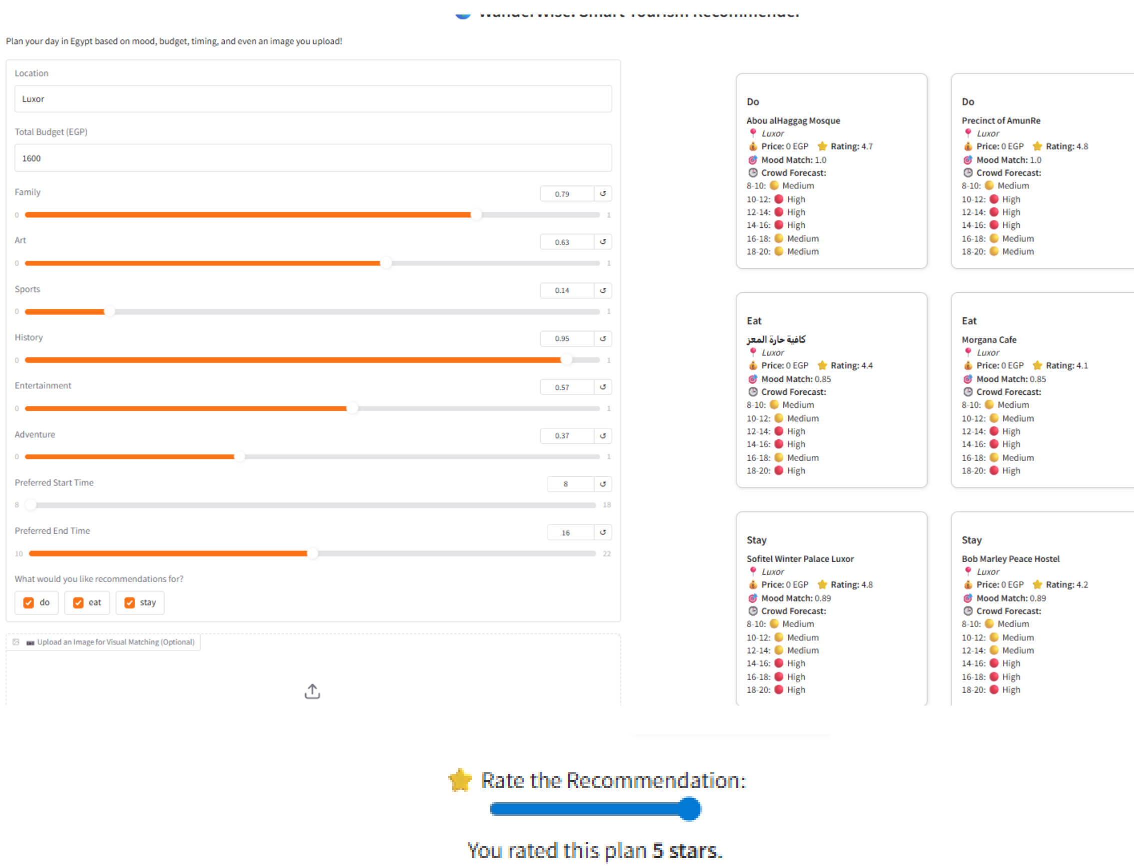

Fig2. User Input and System Output

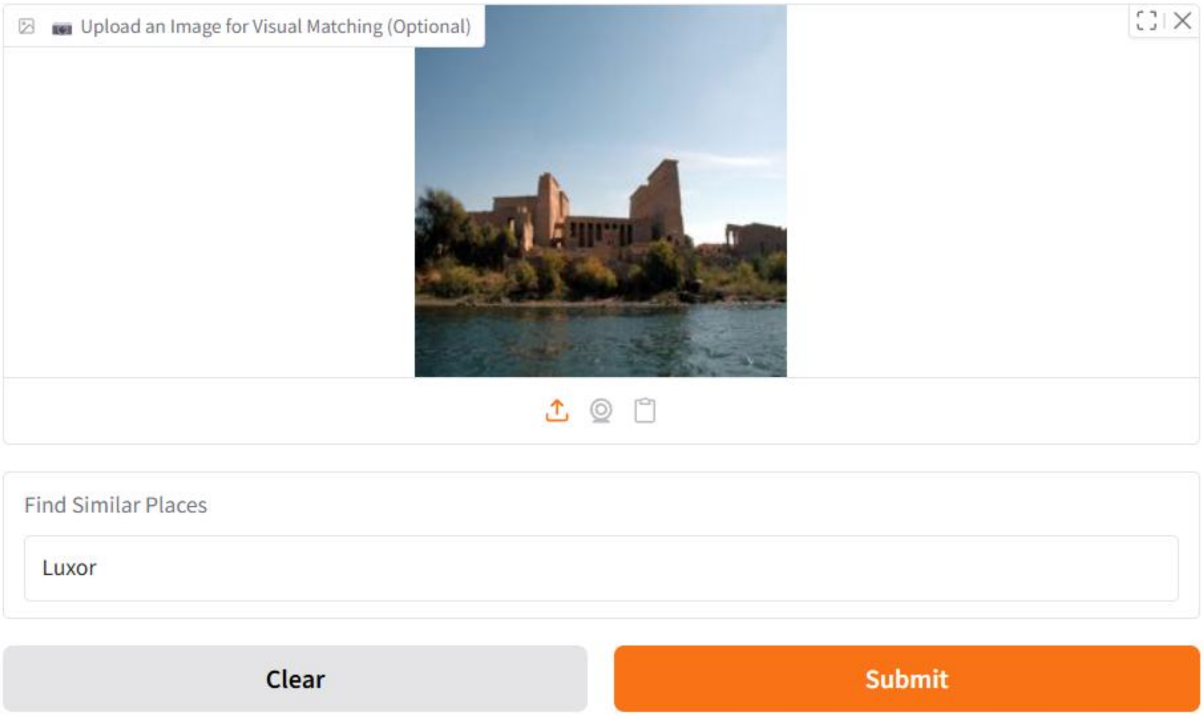

Fig3. Upload Image and add the location

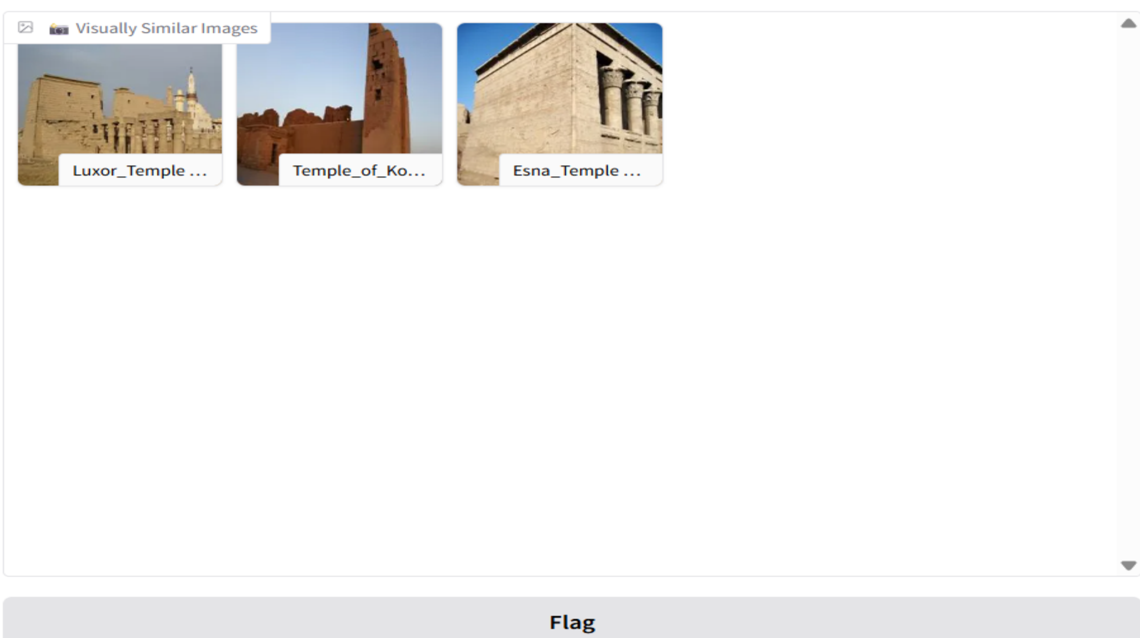

Fig4. Image Recommendation Output

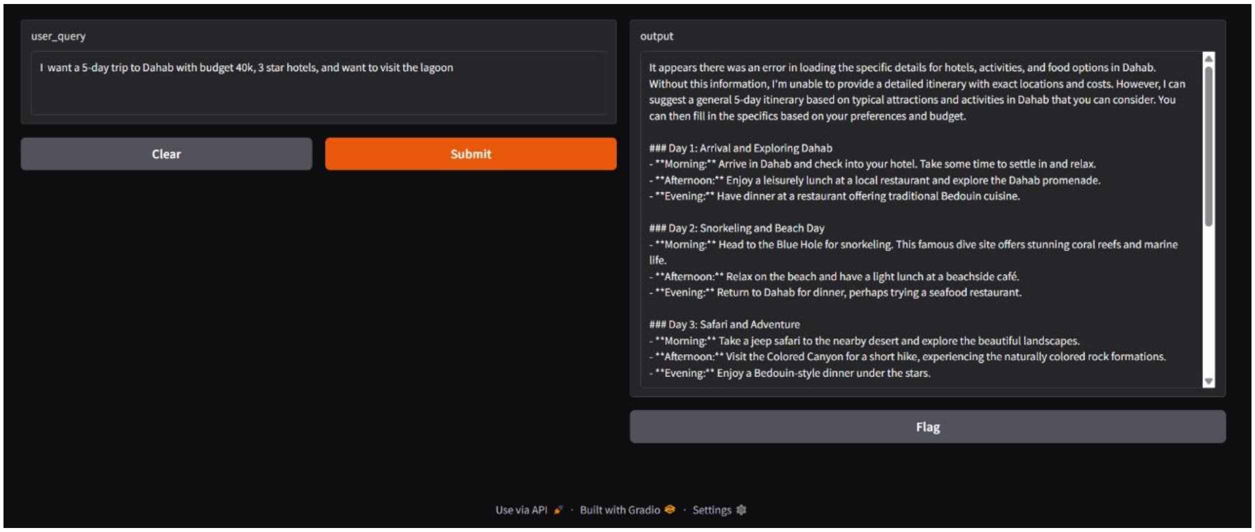

Fig5. RAG system Output

7. Required Libraries

To run this project locally, install the following:

pip install transformers torch chromadb pandas gradio

pip install openai==0.28

These cover:

- **transformers**: For CLIP and GPT API interaction
- **torch**: Backbone for CLIP and deep model inference
- **chromadb**: Lightweight vector store for semantic embeddings
- **pandas**: Dataset manipulation and cleaning
- **gradio**: Frontend interface for app deployment
- **openai==0.28**: LLM-based query rewriting and itinerary generation