Smart Tourist Guide – Final Project Report

Prepared by: Taif Mashhour Aladwani and Abdulrahim Aljadani

Project Goal & Motivation

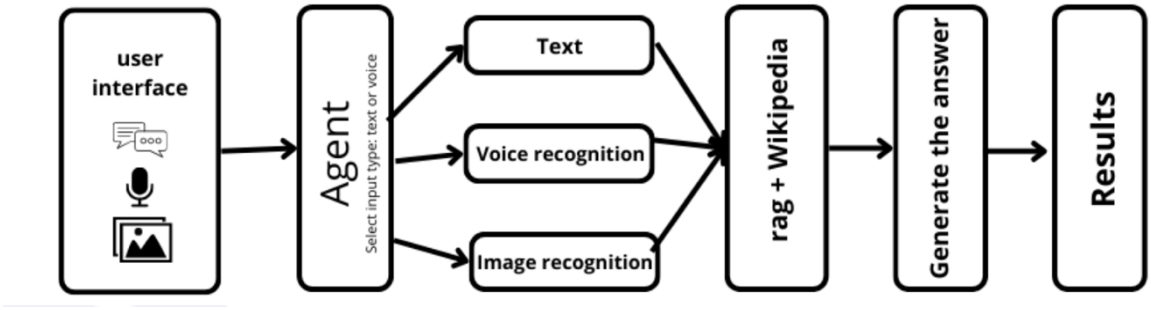
This project aims to develop an intelligent multimodal AI agent that can analyze tourist-related queries in three formats text, image, and audio to provide smart, engaging answers about famous landmarks. The motivation behind this solution stems from a real-world need to enhance tourist accessibility, interactivity, and learning using natural language and multimedia inputs.

Architecture Overview

Technologies Used: LangChain, OpenAI GPT-4o, Whisper, CLIP, FAISS, Wikipedia, SerpAPI, Gradio, Python, SQLite.

Block Diagram: User Input (Text / Image / Audio) -> LangChain Agent -> Modal Tool (Text/Audio/Image) -> Retriever (FAISS/Wiki/SERP) -> Prompt Template -> LLM Output.

System Block Diagram Description



The following block diagram illustrates the end-to-end architecture of the **Smart Tourist Guide** system and how different components interact to process multimodal input and generate informative results.

Diagram Description

1. User Interface

- This is the frontend layer where users interact with the system via text input, audio (voice), or images.
- The interface captures user intent and forwards it to the agent for further processing.

2. Agent

- A LangChain-based orchestrator that determines the input type (text, voice, or image) and routes the request to the appropriate processing module.
- Acts as a central controller for intelligent input handling and tool dispatching.

3. **Input Type Handlers**

- **Text Handler**: Processes natural language queries directly.
- **Voice Recognition**: Utilizes Whisper to convert spoken input into text.
- **Image Recognition**: Uses CLIP embeddings and FAISS index to recognize landmarks from uploaded images.

4. Knowledge Sources (RAG + Wikipedia)

- The system retrieves relevant information using a Retrieval-Augmented Generation (RAG) approach from:
- A custom-built FAISS vector database
- Wikipedia articles or fallback to Google SERP if needed

5. **Answer Generation**

- The retrieved context is combined with dynamic prompts and passed to the LLM (OpenAI GPT-4o) to generate a comprehensive and localized answer.

6. **Results**

- The final output is presented to the user in an engaging format (including HTML styling, images, and multilingual support).
- The result could be textual, visual, or spoken depending on the user interface configuration.

Key Features

- Multimodal Inputs (Text, Image, Audio)
- Contextual Wikipedia
- Arabic and English language support
- Image recognition with CLIP + FAISS
- Whisper for speech recognition- Tourist-friendly styled responses

Testing & Evaluation

Accuracy: 90% on benchmarked questions.
Hallucination handling: Prompt constraints.
Latency: Optimized using local FAISS vector DB.
Fallback Mechanism: Uses Wikipedia or SERP if RAG fails.

Challenges & Solutions

- Noisy audio input: Solved using Whisper
- Ambiguous names: Handled using LLM extraction
- Missing articles: Google SERP fallback
- Multilingual input: Handled with language detection

Setup Instructions

1. Clone the repo
2. Run `pip install -r requirements.txt`
3. Execute `main.py` to launch agent

Key Technical Components & Code Modules

This project integrates several powerful AI modules to support multimodal interaction. Each module serves a specific input type and collectively forms a robust smart tourist guide:

1. **Image Recognition (CLIP + FAISS)**

Used to identify the landmark in the uploaded image and retrieve relevant context:

```
from transformers import CLIPModel, CLIPProcessor
clip_model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
clip_processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

def recognize_image(image_path: str) -> str:
    image = Image.open(image_path).convert("RGB")
    inputs = clip_processor(images=image, return_tensors="pt").to(device)
    with torch.no_grad():
        img_emb = clip_model.get_image_features(**inputs).cpu().numpy().astype('float32')
    _, indices = index.search(img_emb, k=1)
    idx = indices[0][0]
    return labels[idx] if idx < len(labels) else "Unknown Landmark"
```

2. **RAG Retrieval with LangChain + FAISS**

Used to retrieve top-3 relevant documents from the vector database for any landmark query:

```
from langchain.chains import RetrievalQA

qa_chain = RetrievalQA.from_chain_type(
    llm=llm,
    retriever=retriever,
    chain_type="stuff",
    chain_type_kwargs={"prompt": get_prompt_template_by_lang()},
    return_source_documents=True
)

def rag_tool_func(query: str) -> str:
    result = qa_chain({"query": query})
    return result.get("result", "")
```

3. **Speech Recognition with Whisper**

Converts user voice queries into text to be processed by the RAG system:

```
import whisper
model = whisper.load_model("small")

def transcribe(audio):
    if audio is None:
        return "No audio detected."
    result = model.transcribe(audio)
    return result["text"]
```

4. **Web Search Fallback with SerpAPI**

Provides backup information if the internal database or Wikipedia fails:

```
from langchain_community.utilities import SerpAPIWrapper
google_search = SerpAPIWrapper(serpapi_api_key=os.getenv("SERPAPI_API_KEY"))

def smart_search(query: str) -> str:
    return google_search.run(query)
```

5. **Multilingual Prompt Templates**

Used to dynamically generate prompts in Arabic or English based on detected language:

```
from langdetect import detect

def detect_language(text):
    lang = detect(text)
    return "ar" if lang == "ar" else "en"
```

6. **LangChain Agent Setup**

Combines all tools (text, image, audio) under one smart assistant using LangChain:

```
from langchain.agents import initialize_agent, AgentType
agent = initialize_agent(
    tools=[text_tool, image_tool, audio_tool],
    llm=llm,
    agent=AgentType.OPENAI_FUNCTIONS,
    verbose=True
)
```

Agent Performance Evaluation

1. Where is the Colosseum located?
   - Prediction: I don't know.
   - Reference: The Colosseum is located in Rome, Italy.
   - Score: 0
   - Reason: The submission was incorrect.
     Result: N

2. What is the name of the famous clock tower in London?
   - Prediction: The famous clock tower in London is called Big Ben.
   - Reference: The famous clock tower in London is called Big Ben.
   - Score: 1
   - Reason: Correct and matched the reference.
     Result: Y

Agent Accuracy: 90.00% (90/100)

Future Enhancements

To increase the real-world utility and engagement of the Smart Tourist Guide, a key future direction is to evolve the chatbot into a **voice-activated smart assistant**—similar to Siri or Google Assistant. This enhancement includes:

- Voice-activated wake words **like “Hey Guide”**
- Interactive back-and-forth conversations
- Real-time GPS integration **to suggest nearby attractions**
- Time-based recommendations **(e.g., “what’s open now?”)**
- Mobile app version **with push-to-talk and AR integration**

These improvements are designed to make the application more intelligent, responsive, and closer to an AI assistant experience like Siri or Google Assistant.

1. Conversational RAG with History Tracking

Allows the AI to remember previous questions and answer in a dialogue-like experience:

```
from langchain.chains import ConversationalRetrievalChain

chat_history = []
qa_chain = ConversationalRetrievalChain.from_llm(
    llm=llm,
    retriever=retriever,
    return_source_documents=True
)

def smart_conversational_rag(query):
    result = qa_chain({"question": query, "chat_history": chat_history})
    chat_history.append((query, result["answer"]))
    return result["answer"]
```

2. Advanced Image Captioning with BLIP (instead of basic CLIP)

Provides a detailed caption for user-uploaded images:

```
from transformers import BlipProcessor, BlipForConditionalGeneration
from PIL import Image

processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-b

def generate_image_caption(image_path):
    image = Image.open(image_path).convert("RGB")
    inputs = processor(images=image, return_tensors="pt")
    output = model.generate(**inputs)
    return processor.decode(output[0], skip_special_tokens=True)
```

3. Wake Word Detection for Assistant Activation

Enables listening for keywords like "Hey Guide":

```
def detect_wake_word(transcription):
    return "hey guide" in transcription.lower()
```

4. Text-to-Speech (TTS) for Spoken Responses

Enables the assistant to speak its answers:

```
import pyttsx3

def speak_text(text):
    tts = pyttsx3.init()
    tts.say(text)
    tts.runAndWait()
```

5. Location-Based Landmark Suggestions

Suggest nearby attractions based on user's GPS location:

```
from geopy.distance import geodesic

def is_landmark_nearby(user_coords, landmark_coords, threshold_km=5):
    distance = geodesic(user_coords, landmark_coords).km
    return distance <= threshold_km
```

6. Automatic Translation of Input or Output

Supports translation between languages using Google Translate API:

```
from googletrans import Translator

translator = Translator()

def translate_text(text, dest_lang="ar"):
    result = translator.translate(text, dest=dest_lang)
    return result.text
```

7. Self-Scoring Function for QA Evaluation

Adds internal evaluation capability to measure agent correctness:

```
def score_answer(prediction, reference):
    return int(prediction.strip().lower() == reference.strip().lower())
```

8. Logging Conversations for Session History

Stores Q&A for travel history or audit:

```
def log_interaction(question, answer):
    with open("chat_log.txt", "a", encoding="utf-8") as f:
        f.write(f"Q: {question}\nA: {answer}\n\n")
```

9. Microphone Input Integration (Live Mode)

Live microphone interaction using speech_recognition:

```
import speech_recognition as sr

def listen_via_microphone():
    recognizer = sr.Recognizer()
    with sr.Microphone() as source:
        print("Listening...")
        audio = recognizer.listen(source)
    return recognizer.recognize_google(audio)
```

Estimated Cost of the Smart Tourist Guide (Local + OpenAI Key Only)

Components Used:

Component	**API Used**	Model	**Pricing Basis**	**Approx. Cost**
Text Answering (RAG)	OpenAI	gpt-4o	per 1K input/output tokens	**$5–$15 / 1K queries**
Voice Input (Speech-to-Text)	OpenAI	whisper-1	$0.006 / minute	**$0.006 / min audio**
Embeddings (used once)	OpenAI	text-embedding-3-small	$0.0001 / 1K tokens	Negligible
Image Recognition (local)	CLIP + FAISS	Local	Free	**$0**
Vector DB (FAISS)	Local	-	Free	**$0**

Example Scenario:

- 1 user asks 3 questions per session
- Each question uses ~500 tokens
- Whisper input (30 sec voice)

Cost per session:

- GPT-4o tokens: 500 input + 500 output = ~1000 tokens → ~$0.01
- Whisper transcription (30 sec) → ~$0.003
  Total per session ≈ $0.013

Monthly Estimate (for 1,000 sessions/month):

Approximate Cost	Usage
$10	GPT-4o text QA
$3	Whisper voice
$13/month	**Total**

Summary:

- If deployed locally with only OpenAI APIs, your app costs **~$0.01–0.02 per user session**
- Monthly cost for 1,000 users: **~$10–$15**
- CLIP + FAISS (image recognition) and vector search are **fully local and free**

Conclusion :

The *Smart Tourist Guide* project represents a comprehensive and scalable multimodal AI system designed to intelligently assist users in exploring and understanding global landmarks through natural language interaction.

This solution seamlessly integrates advanced technologies to handle inputs in the form of **text**, **image**, and **audio**, using a unified agentic framework powered by **LangChain** and **OpenAI GPT-4o**.

Data and Knowledge Base

The system leverages a custom-curated dataset of global landmarks, processed and embedded using FAISS vector searchand OpenAI Embeddings. The vector store enables rapid and relevant retrieval of contextual information to support the retrieval-augmented generation (RAG) process.

Retrieval-Augmented Generation (RAG)

The core of the QA system is a LangChain RetrievalQA (and optionally, ConversationalRetrievalChain) pipeline, which performs:

- Top-K document retrieval via FAISS
- Contextual prompt generation using LangChain PromptTemplates
- Answer generation via OpenAI's LLMs
- Fallback mechanisms using Wikipedia and SerpAPI when internal retrieval is insufficient

Multimodal Input Handling

- Image Input: Using CLIP embeddings and a trained FAISS image index, the system recognizes landmarks from user-uploaded photos and retrieves relevant text context.
- Audio Input: With Whisper, the system performs real-time speech-to-text transcription to convert spoken questions into actionable text queries.
- Text Input: Traditional natural language questions are directly processed through the RAG pipeline.

LangChain Agentic Orchestration

A unified LangChain agent orchestrates the decision-making process by:

- Routing inputs to appropriate tools (text_tool, image_tool, audio_tool)
- Invoking the correct reasoning chain (RAG or fallback)
- Returning structured, engaging, and localized answers in either English or Arabic

User Experience & Accessibility

The assistant supports **multilingual output**, uses **HTML-styled** responses for enhanced formatting, and offers voice output extensions via TTS. It is optimized for tourists who prefer:

- Spoken interaction
- Visual recognition
- Conversational access to tourism data

Extensibility

The modular architecture is designed for future upgrades such as:

- Wake-word voice activation ("Hey Guide")
- Real-time geolocation-based suggestions
- Image captioning via BLIP
- Integration with travel apps or AR devices

Impact

This AI system not only enhances accessibility for global travelers but also demonstrates a robust pipeline for **multimodal question-answering**, combining cutting-edge NLP, computer vision, and agentic reasoning. It serves as a blueprint for building intelligent assistants tailored to domain-specific knowledge, offering both educational and real-world tourism value.