**Project Status**

**Phase 1 – MVP-1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Status** | **Comments** | **Cursor cost of implementing** |
| Phase 1 | Complete | * API Integration * Parameters tracking **Accuracy** (Factual correctness)   + **Response Time** (Measure latency)   + **Creativity** (Subjective but testable)   + **Logical Reasoning** (Test complex questions)   + **Tone & Style** (Formal, casual, technical, etc.) | $1.04 (26 requests) ($21.08) |
| Phase 2 | Complete | * Data analysis of **how different settings impact responses** | $1.28 (x requests) ($22.36) |
| Phase 3 |  | * **Advanced Capabilities – Tool Calling & Function Execution** * **Functions implemented**    + Weather   + Calculator   + Currency converter   + Wikipedia   + Auto amazon shopping (mock/simulation function) | $1.56 (x requests) ($23.92) |
| Phase 4 |  |  |  |
| Phase 5 |  |  |  |
| Phase 6 |  |  |  |
| Phase 7 | Complete | UI to implement Phase 1, 2, 3  To be enhanced based on Phases 2-6 |  |

**Phase 2 – Enhancements / Backlog**

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Status** | **Comments** | **Cursor cost of implementing** |
| Phase 1 |  | * Deepseek not available by the provider, need to add later |  |
| Phase 2 |  | * Add feedback loop to the model!?? * Need to add/update Model Tier comparison * Top P: not visible in the advanced parameter test * Cretaing a process where prompt is analysed against various models and then model is choosen based on which is most cost effective |  |
| Phase 3 |  | * Issue with anthropic * Amazon function not returning values |  |
| Phase 4 |  |  |  |
| Phase 5 |  |  |  |
| Phase 6 |  |  |  |
| Phase 7 |  |  |  |

**Project Plan: Hands-on Exploration of AI Models**

This project aims to **compare, analyze, and demonstrate** the capabilities of various AI models through a structured, hands-on approach. The goal is to **understand their strengths, weaknesses, and ideal use cases** by implementing experiments rather than just reading about them.

### ****Goal:****

* Understand AI model differences by testing real-world **use cases**.
* Compare **capabilities, limitations, response time, accuracy, reasoning, creativity**, etc.
* Implement API-based evaluations in **Python** for objective measurement.

## **🟢 Phase 1: API Integration & Baseline Comparisons**

**📌 Objective:** Set up API access and conduct **basic response analysis** across multiple AI models.

✅ **Tasks:**

1. **Integrate APIs** for:
   * ✅ OpenAI (GPT-4o, GPT-4o-mini)  
     ✅ Anthropic (Claude-3 Opus, Sonnet)  
     ✅ Google (Gemini 1.5 Pro, Flash)  
     ✅ Meta (LLaMA-3 via Together.ai/OLLAMA)  
     ✅ DeepSeek (DeepSeek-V2, DeepSeek Coder)
2. Implement **a Python script** to send identical prompts to all models.
3. Log and compare responses based on:
   * **Accuracy** (Factual correctness)
   * **Response Time** (Measure latency)
   * **Creativity** (Subjective but testable)
   * **Logical Reasoning** (Test complex questions)
   * **Tone & Style** (Formal, casual, technical, etc.)

📌 **Use Cases to Test:**

* **General Knowledge:** “Explain quantum mechanics in simple terms.”
* **Opinion-Based:** “What is the best way to learn programming?”
* **Logic & Math:** “What is the next number in the series: 2, 6, 12, 20, ?”
* **Coding Ability:** “Write a Python function to sort a list of dictionaries by a key.”
* **Writing Style:** “Write a short story about a time-traveling scientist.”

🛠 **Deliverable:**

* Python script with API integration
* CSV file containing model responses & metadata (latency, tokens used, etc.)

## **📊 Summary of Key Comparisons**

| **Feature** | **GPT-4o** | **Claude-3** | **Gemini 1.5** | **LLaMA-3** | **DeepSeek-V2** |
| --- | --- | --- | --- | --- | --- |
| **Reasoning** | 🔥🔥🔥🔥🔥 | 🔥🔥🔥🔥 | 🔥🔥🔥 | 🔥🔥 | 🔥🔥🔥 |
| **Speed** | ⚡⚡⚡⚡ | ⚡⚡⚡ | ⚡⚡⚡⚡⚡ | ⚡⚡ | ⚡⚡⚡ |
| **Cost-Efficiency** | 💰💰 | 💰💰💰 | 💰💰💰 | 💰💰💰💰 | 💰💰💰 |
| **Tool Calling** | ✅ | ✅ | ❌ | ❌ | ❌ |
| **Coding Ability** | ✅ | ✅ | ✅ | ✅ | ✅ |
| **Creativity** | 🎭🎭🎭🎭🎭 | 🎭🎭🎭🎭 | 🎭🎭🎭 | 🎭🎭 | 🎭🎭🎭 |

## **🟢 Phase 2: Parameter Tuning & Response Control**

**📌 Objective:** Understand how different **API parameters** affect output.

✅ **Tasks:**

1. Experiment with **temperature**:
   * **temperature=0 (deterministic)** → Minimal creativity
   * **temperature=1 (creative)** → High variance in responses
2. Test **max\_tokens**:
   * Measure how each model **truncates responses**
3. Test **response formatting (json\_schema)**:
   * Extract structured output (e.g., convert answers into JSON objects)
4. Measure **response length variation** under different parameters.

📌 **Use Cases to Test:**

* **Factual Accuracy:** Ask a **factual question** at **temperature=0** (Should be consistent).
* **Creative Writing:** "Write a poem about the future of AI." at **temperature=1**.
* **Structured Data Output:** Ask all models to return a JSON response:

json

CopyEdit

{

"question": "Capital of France?",

"answer": "Paris",

"source": "Wikipedia"

}

* **Short vs. Long Responses:** Compare short summaries vs. detailed explanations:
  + “Summarize the plot of Inception in 20 words.”
  + “Summarize the plot of Inception in 200 words.”

🛠 **Deliverable:**

* Python script for **parameter testing**
* Data analysis of **how different settings impact responses**

## **🟢 Phase 3: Advanced Capabilities – Tool Calling & Function Execution**

**📌 Objective:** Test **function calling/tool usage** to see how models can interact with external systems.

✅ **Tasks:**

1. Implement **function calling (tool calling) API** for OpenAI & Claude & Deepseek.
2. Simulate **real-world scenarios where the model needs to take action**, e.g.:
   * Calling an **API** for live weather
   * Fetching **stock prices**
   * Executing **a mathematical calculation**
3. Compare **accuracy, reasoning, and ability to pick the correct tool.**

📌 **Use Cases to Test:**

* **Weather API Call:**
  + “What’s the weather in London?” → Model should call an API instead of hallucinating an answer.
* **Stock Price Lookup:**
  + “Get me today’s price for Apple stock.”
* **Math Execution:**
  + Instead of relying on text-based calculations, use a function like:

python

CopyEdit

def calculate\_area(radius):

return 3.14 \* radius \* radius

* **E-commerce Order Handling:**
  + Simulate a chatbot that processes orders via tool calling:

json

CopyEdit

{

"role": "assistant",

"content": [

{

"type": "tool\_call",

"name": "place\_order",

"input": {

"product\_id": "1234",

"quantity": 2,

"shipping\_address": "221B Baker Street, London"

}

}

]

}

🛠 **Deliverable:**

* Python script for **tool calling experiments**
* Logs of **when models decide to use a tool vs. when they don't**

## **🟢 Phase 4: Performance, Cost & Latency Optimization**

**📌 Objective:** Test **response times, caching, batch processing**, and **cost efficiency**.

✅ **Tasks:**

1. Compare **streaming vs. standard responses**:
   * **Does streaming improve latency?**
2. Test **Anthropic’s explicit caching** (cache\_control).
3. Implement **batch processing** for cost optimization:
   * Run multiple requests **in parallel** and measure cost savings.

📌 **Use Cases to Test:**

* **Latency Test:** Measure response time across models for:
  + Simple questions (low computation)
  + Complex reasoning tasks (high computation)
* **Cost Analysis:** Compare **batch API** vs. individual requests:
  + Run **100 API calls** in parallel and compare per-token cost.
* **Streaming Response:** Test difference in **UX and latency** for real-time chatbot use.

🛠 **Deliverable:**

* Python script measuring **latency, cost, and caching effects**

## **🟢 Phase 5: Running Open-Source Models Locally**

**📌 Objective:** Compare **API models vs. local models (LLaMA-3, DeepSeek, Mistral).**

✅ **Tasks:**

1. Install **OLLAMA** and run LLaMA-3 locally.
2. Compare **DeepSeek-V2, DeepSeek Coder, and Mistral** with API-based models.
3. Test **if local models can match OpenAI/Gemini/Claude.**

📌 **Use Cases to Test:**

* **Basic Chatbot Responses:** Compare LLaMA-3, deepseek vs. GPT-4o in **general conversation**.
* **Coding Ability:** See if **open-source models** can write **Python scripts as effectively**.
* **Creative Writing:** Can locally run models **generate high-quality stories?**

🛠 **Deliverable:**

* Report comparing **local models vs. API models.**

## **🔹 Phase 6: Final Evaluation & Findings**

📌 **Objective:** Summarize **which model is best for different use cases.**

✅ **Tasks:**

1. Compile **strengths & weaknesses** of each model.
2. Identify **best use cases** for different models.
3. Create a **comparison table & visualization** summarizing insights.

🛠 **Deliverable:**

* **Comparison dashboard** summarizing performance metrics.
* **Final report & presentation** highlighting key takeaways.

**🔹 Phase 7: Building a Basic UI for Model Comparison**

📌 **Objective:** Create a **simple web-based UI**  where a user enters a query, and the system:

✅ **Runs the query against all models** (GPT-4o, Claude-3, Gemini 1.5, LLaMA-3, DeepSeek-V2, etc.)  
✅ **Displays responses from each model**   
✅ **Evaluates & suggests the best model** based on predefined scoring metrics

**✅ Tasks**

**1️ UI Design & Implementation**

* Create a **basic web UI** (**HTML, CSS, JS**).
* Add a **text input box** where users enter a query.
* Include a **“Run Test” button** that triggers all API calls.

**2️ Backend: Multi-Model Execution**

* On button click, the UI should:  
  ✅ Send the **same query** to all integrated models.  
  ✅ Fetch **responses, latency, and token usage**.  
  ✅ Display responses side-by-side in a structured table.

📌 **Example Output Layout in UI**

| **Model** | **Response** | **Latency** | **Tokens Used** | **Accuracy Score** | **Creativity Score** |
| --- | --- | --- | --- | --- | --- |
| **GPT-4o** | *Response Text...* | 1.2s | 60 | ✅ 95% | 🎭 80% |
| **Claude-3** | *Response Text...* | 1.5s | 55 | ✅ 90% | 🎭 85% |
| **Gemini 1.5** | *Response Text...* | 1.0s | 50 | ✅ 85% | 🎭 75% |
| **LLaMA-3** | *Response Text...* | 2.0s | 70 | ✅ 80% | 🎭 70% |
| **DeepSeek-V2** | *Response Text...* | 1.8s | 65 | ✅ 88% | 🎭 78% |

**3️ Automated Model Scoring & Ranking**

* Define a **scoring system** based on:  
  ✅ **Response Accuracy** (Factual correctness)  
  ✅ **Latency** (Faster responses get higher scores)  
  ✅ **Creativity Score** (For writing or story-based tasks)  
  ✅ **Token Efficiency** (Lower token usage is better)  
  ✅ **Tool-Use Capability** (If function calling was needed)

📌 **Example Scoring Formula**

python

CopyEdit

def evaluate\_model(response, latency, token\_usage, accuracy, creativity):

score = (accuracy \* 0.4) + ((1/latency) \* 0.2) + ((100-token\_usage) \* 0.1) + (creativity \* 0.3)

return round(score, 2)

* Rank **models based on score** and highlight the **best** model for that query.

**4️ Final Output: Model Recommendation**

* After evaluating all models, the UI should display:  
  ✅ **Best Model for this Query** (E.g., "GPT-4o performed best.")  
  ✅ **Why this Model Won** (E.g., "Best accuracy, fastest response, most creative.")  
  ✅ **Alternative Options** (E.g., "Claude-3 is a good second choice.")

📌 **Example UI Output**

yaml

CopyEdit

🔹 Best Model: GPT-4o

✅ Accuracy: 95%

⚡ Speed: 1.2s

🎭 Creativity: 80%

Alternative: Claude-3 (More creative but slightly slower)

**🛠 Deliverables**

✅ A **working web UI** where users can enter **any query** and compare models.  
✅ **Real-time scoring system** to **automatically suggest the best model.**  
✅ **Downloadable comparison logs** (CSV format).

**🔹 Final Project Flow -Phase 7**

**1️ User enters a prompt** → Clicks "Run Test"  
**2️ System queries all models** → Fetches responses  
**3️ Backend scores each model** → Ranks best performers  
**4️ UI displays responses, latency, token use, and best model suggestion**

## **🔹 Final Deliverables**

✅ **Automated Python framework** to test all major models.  
✅ **Comparison dashboard** UI summarizing:

* Speed, cost, accuracy, reasoning, tool use.  
  ✅ **Final report & presentation** summarizing key takeaways.

**🎯 Outcome: What You’ll Gain**

🚀 **Deep understanding** of how different AI models behave.  
🚀 **Hands-on experience** in **APIs, optimization, caching, and tool calling.**  
🚀 **Practical knowledge** for choosing the right model for the right use case.

🚀 **Hands-on experience with AI model comparison in real-time.**  
🚀 **Automated, data-driven decision-making for AI selection.**  
🚀 **A working prototype that can be expanded into a real-world application.**

Hi, can you step by step in as detail as possible, consider me beginner and provide step by step guide how to get API keys for these models for integration; this is phase 1 of our learning project ✅ OpenAI (GPT-4o, GPT-4o-mini) ✅ Anthropic (Claude-3 Opus, Sonnet) ✅ Google (Gemini 1.5 Pro, Flash) ✅ Meta (LLaMA-3 via Together.ai/OLLAMA) ✅ DeepSeek (DeepSeek-V2, DeepSeek Coder)

**ChatGPT said:**

**Phase 1: Step-by-Step Guide to Getting API Keys for AI Model Integration**

Since you’re a beginner, I’ll provide a detailed, step-by-step guide for **each provider** (OpenAI, Anthropic, Google, Meta, and DeepSeek) to get **API keys** for model integration.

**✅ Step 1: Get API Key for OpenAI (GPT-4o, GPT-4o-mini)**

💡 **Website:** <https://platform.openai.com/signup>

**1️⃣ Create an OpenAI Account**

1. Go to [**OpenAI Platform**](https://platform.openai.com/signup).
2. Sign up using **Google, Microsoft, or an email/password**.
3. **Verify your email** (if using email signup).
4. OpenAI may ask for **phone number verification** for security.

**2️⃣ Get API Key**

1. After logging in, go to [**API Keys**](https://platform.openai.com/api-keys).
2. Click **"Create a new secret key"**.
3. **Copy the API key** immediately!  
   🚨 **Important:** You **won't** be able to see the key again later.

**3️⃣ Store API Key Securely**

* **DO NOT** share or expose this key publicly.
* Save it in a **.env file** or a **secure password manager**.

**✅ Step 2: Get API Key for Anthropic (Claude-3 Opus, Sonnet)**

💡 **Website:** https://console.anthropic.com

**1️⃣ Create an Anthropic Account**

1. Go to **Anthropic Console**.
2. Click **Sign Up** (or Login if you already have an account).
3. Sign up using **Google or an email/password**.
4. **Verify your email** (check inbox/spam folder).

**2️⃣ Get API Key**

1. Go to **API Keys Page**.
2. Click **"Create API Key"**.
3. **Copy the API Key** immediately (you won’t see it again).

**3️⃣ Store API Key Securely**

* Save it **in a secure place** (like .env file).
* Example .env file entry:

env

CopyEdit

ANTHROPIC\_API\_KEY="your-api-key-here"

**✅ Step 3: Get API Key for Google Gemini (Gemini 1.5 Pro, Flash)**

💡 **Website:** <https://aistudio.google.com>

**1️⃣ Enable Google AI API**

1. Go to [**Google AI Studio**](https://aistudio.google.com).
2. Click **Sign In** and use your **Google account**.
3. Accept **Terms & Conditions** if prompted.

**2️⃣ Get API Key**

1. Go to **Gemini API Console**.
2. Click **"Create API Key"**.
3. **Copy the API key** immediately (you won't see it again).

**3️⃣ Enable Gemini API Access**

1. Go to **Google Cloud Console**.
2. Create a new project **(if not done already)**.
3. Go to **"APIs & Services" → "Enable APIs"**.
4. Search for **"Gemini API"** → Click **Enable**.

🚨 **Important:** Google may require **billing setup** (but provides free-tier credits).

**✅ Step 4: Get API Key for Meta (LLaMA-3 via Together.ai or OLLAMA)**

**Option 1: Use Hosted API via Together.ai**  
💡 **Website:** <https://www.together.ai>

**1️⃣ Create a Together AI Account**

1. Go to **Together AI Signup**.
2. Sign up using **Google, GitHub, or email/password**.
3. Verify your email.

**2️⃣ Get API Key**

1. Go to **API Keys Page**.
2. Click **"Create API Key"**.
3. **Copy the API Key** for use.

**Option 2: Run LLaMA-3 Locally via OLLAMA**  
💡 **Website:** <https://ollama.com>

**1️⃣ Install OLLAMA**

1. Download and install OLLAMA:
   * **Windows**: https://ollama.com/download
   * **Mac/Linux**: Run in terminal:

sh

CopyEdit

curl -fsSL https://ollama.com/install.sh | sh

**2️⃣ Run LLaMA-3 Locally**

1. Open terminal and run:

sh

CopyEdit

ollama pull llama3

1. Once downloaded, start a model:

sh

CopyEdit

ollama run llama3

💡 **No API key needed for OLLAMA (since it runs locally).**

**✅ Step 5: Get API Key for DeepSeek (DeepSeek-V2, DeepSeek Coder)**

💡 **Website:** <https://deepseek.com>

**1️⃣ Create a DeepSeek Account**

1. Go to [**DeepSeek AI**](https://deepseek.com).
2. Click **Sign Up** and create an account.
3. Verify your email.

**2️⃣ Get API Key**

1. Go to **DeepSeek API Keys**.
2. Click **"Create API Key"**.
3. **Copy the API Key** immediately.

**💡 Storing API Keys Securely**

💡 **Use a .env file** to store API keys safely.  
🔒 **Never expose API keys in public repositories!**

**Example .env File**

env

CopyEdit

OPENAI\_API\_KEY="your-openai-api-key"

ANTHROPIC\_API\_KEY="your-anthropic-api-key"

GEMINI\_API\_KEY="your-google-api-key"

TOGETHER\_AI\_API\_KEY="your-together-ai-key"

DEEPSEEK\_API\_KEY="your-deepseek-api-key"

for this project can you create a requiremnets.txt so that we have all dependecies mentioned there and help me create a virtual environment. i have got all the keys for all these models, please help me create a .evn file so that i can add then there and then create a simple python script to go a call action so that we can test if the API keys are working or not; we will incrementatlly add all test cases so create a folder with test cases and add the API checking test case to start with

#!/bin/bash

# Create a new virtual environment

python3 -m venv venv

# Activate virtual environment

source venv/bin/activate

# Install requirements

pip install -r requirements.txt

Run the API tests:

pytest tests/test\_api\_connections.py -v

ai-models-exploration/

├── .env

├── requirements.txt

├── setup\_env.sh

├── src/

│ ├── \_\_init\_\_.py

│ └── evaluation\_metrics.py

├── tests/

│ ├── \_\_init\_\_.py

│ └── test\_api\_connections.py

└── README.md

Post all APIs working

Create the necessary directories:

mkdir -p tests test\_results

Install additional requirements:

pip install pytest-asyncio pandas matplotlib

First update your packages:

pip install -r requirements.txt –upgrade

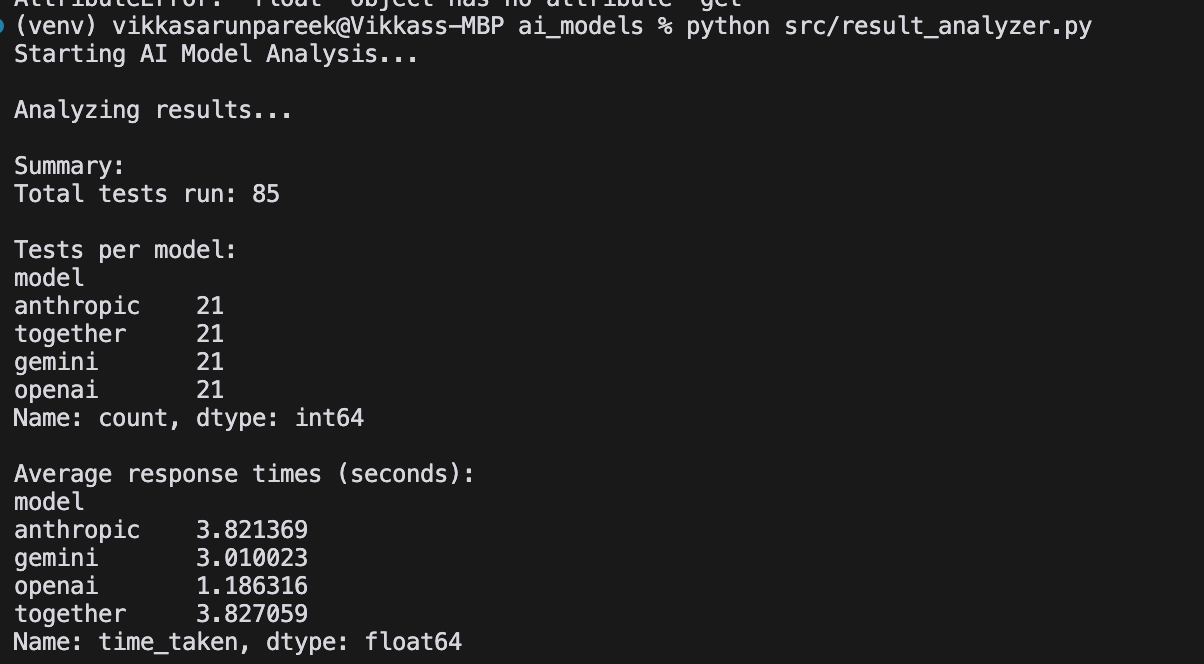
Then run the tests:

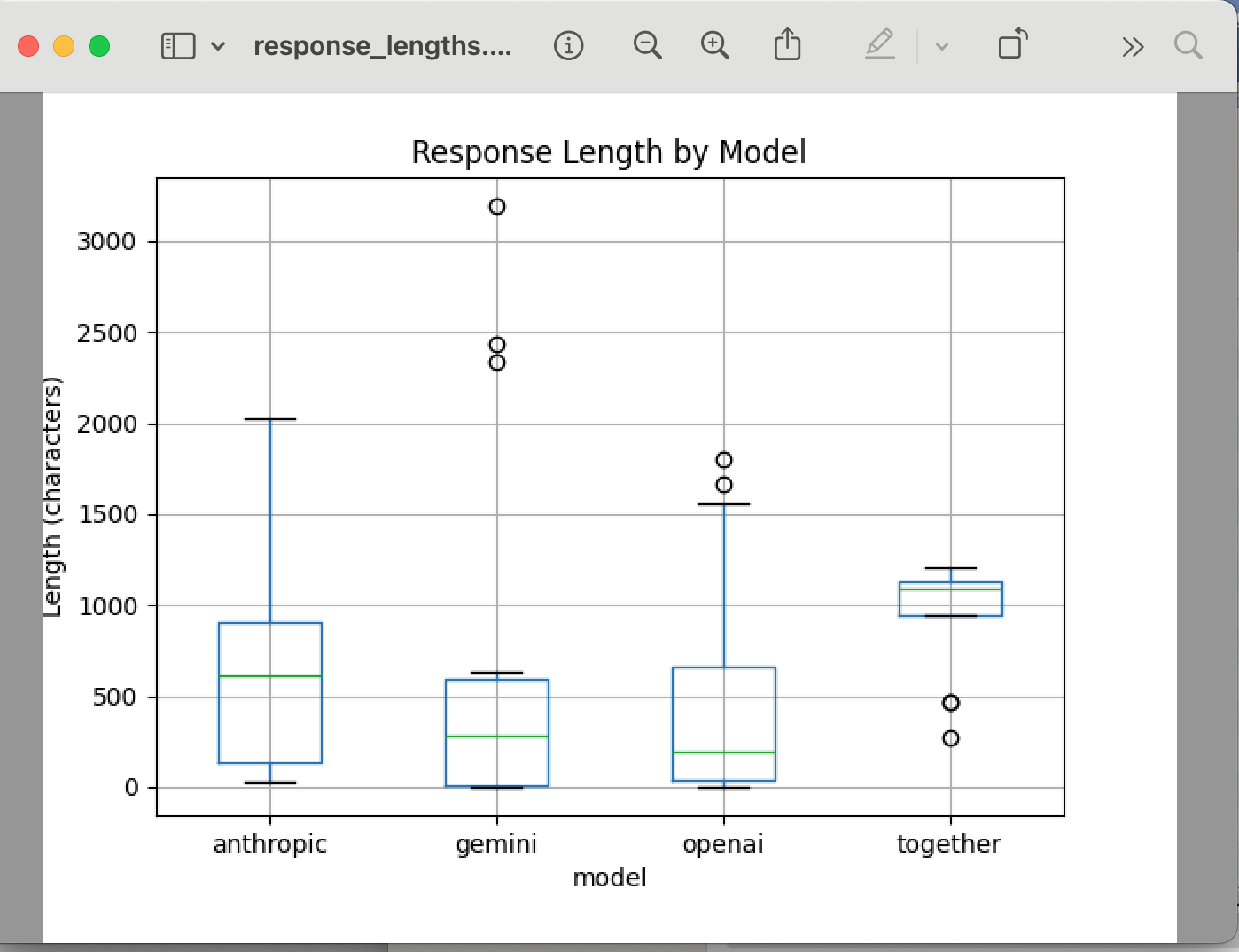
pytest tests/test\_model\_comparison.py -v

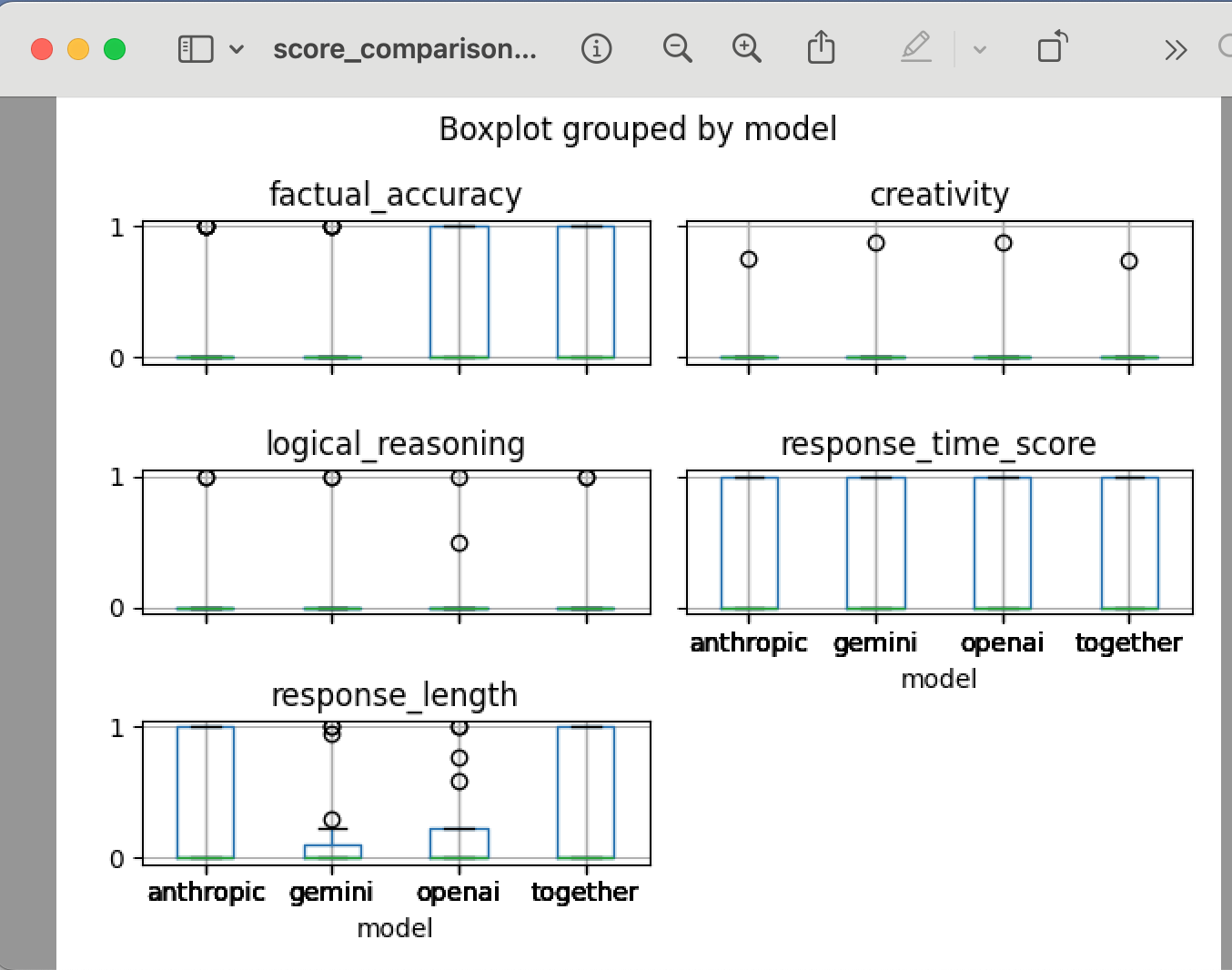
The tests should now run without the async/await errors. The test results will be saved in the test\_results directory, which you can then analyze using the result\_analyzer.py script.After the tests complete, you can analyze the results:

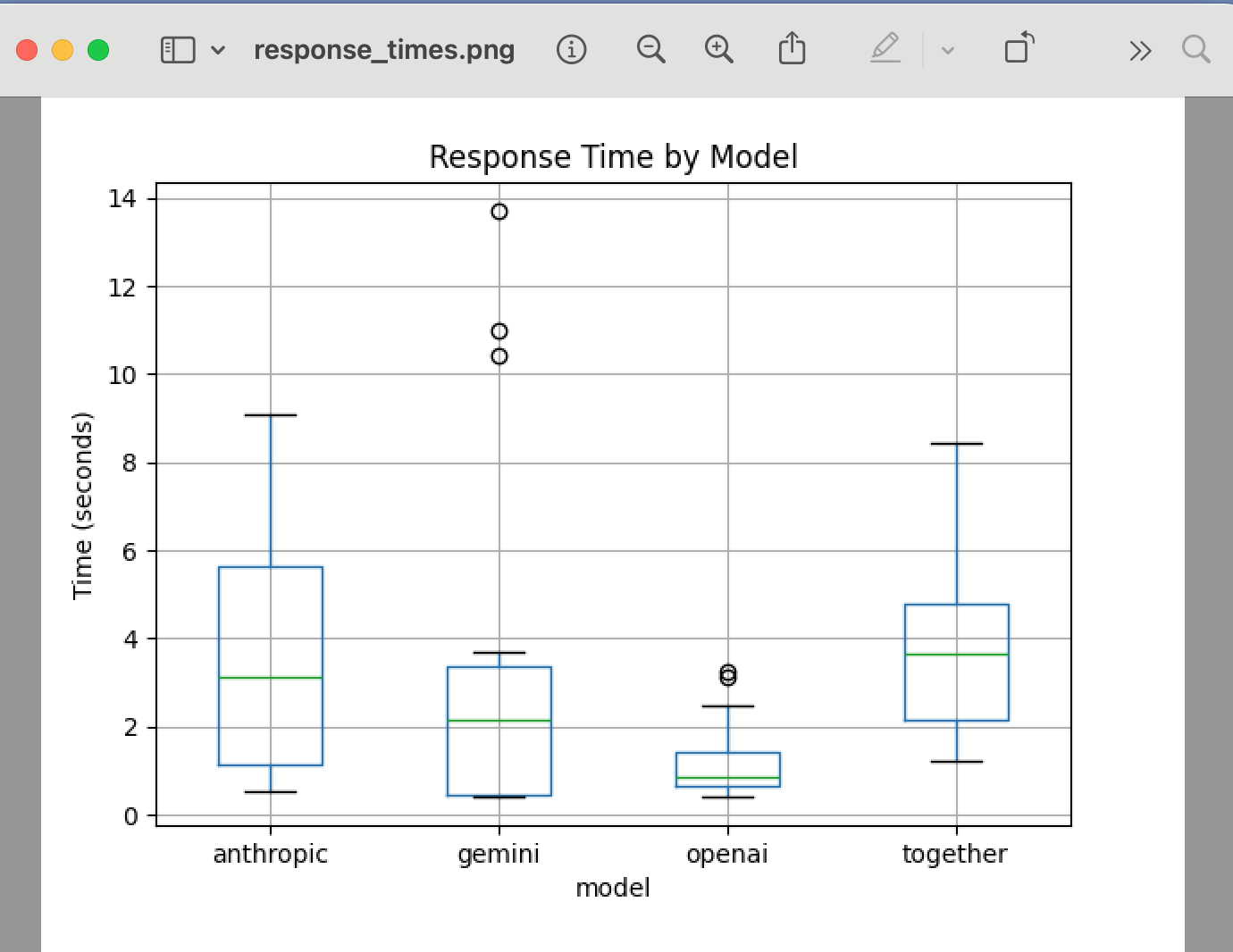
Output (cursor script)











**Next Step – UI Creation in HTML, CSS and JS. / Flask**

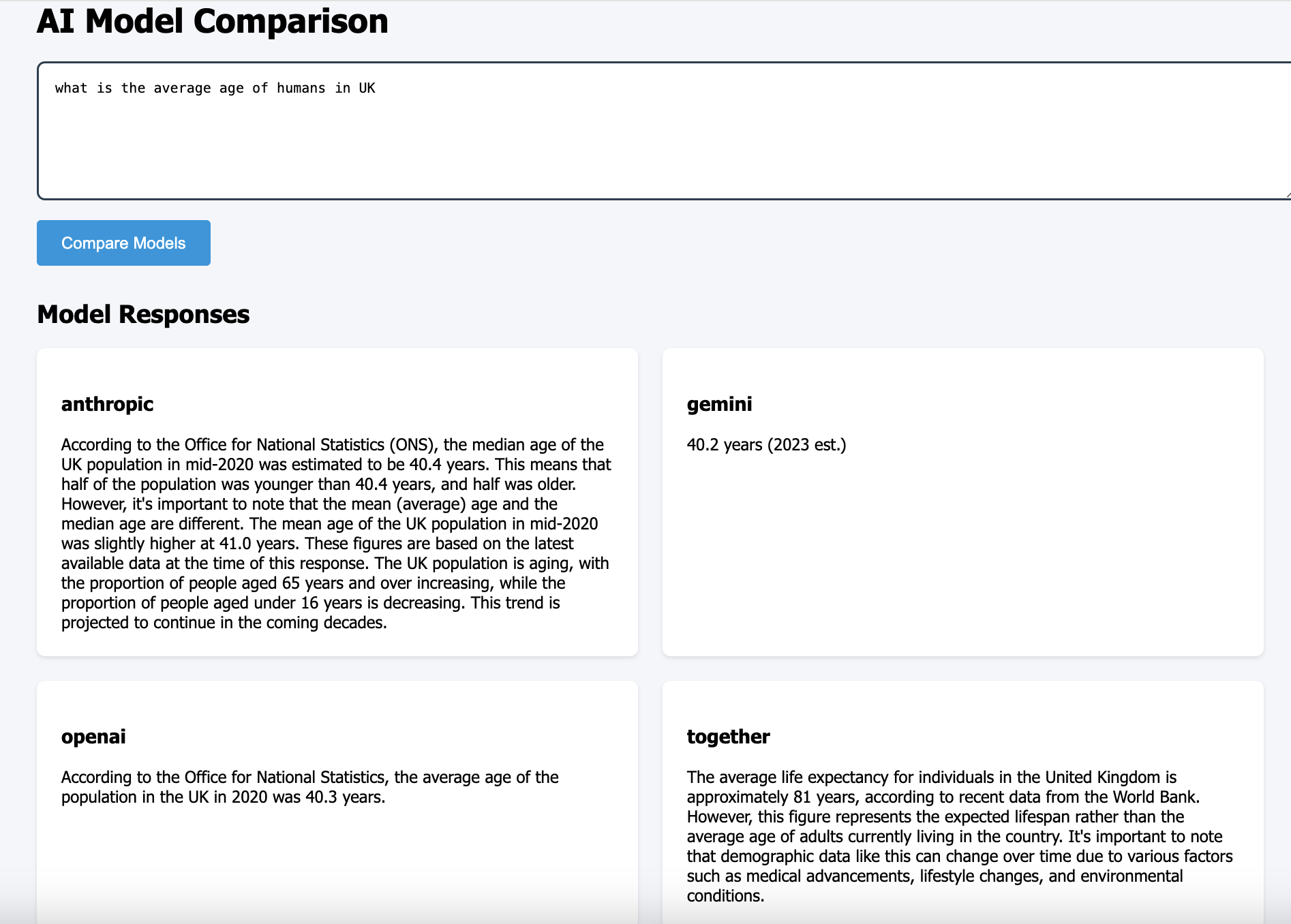
To run the application:

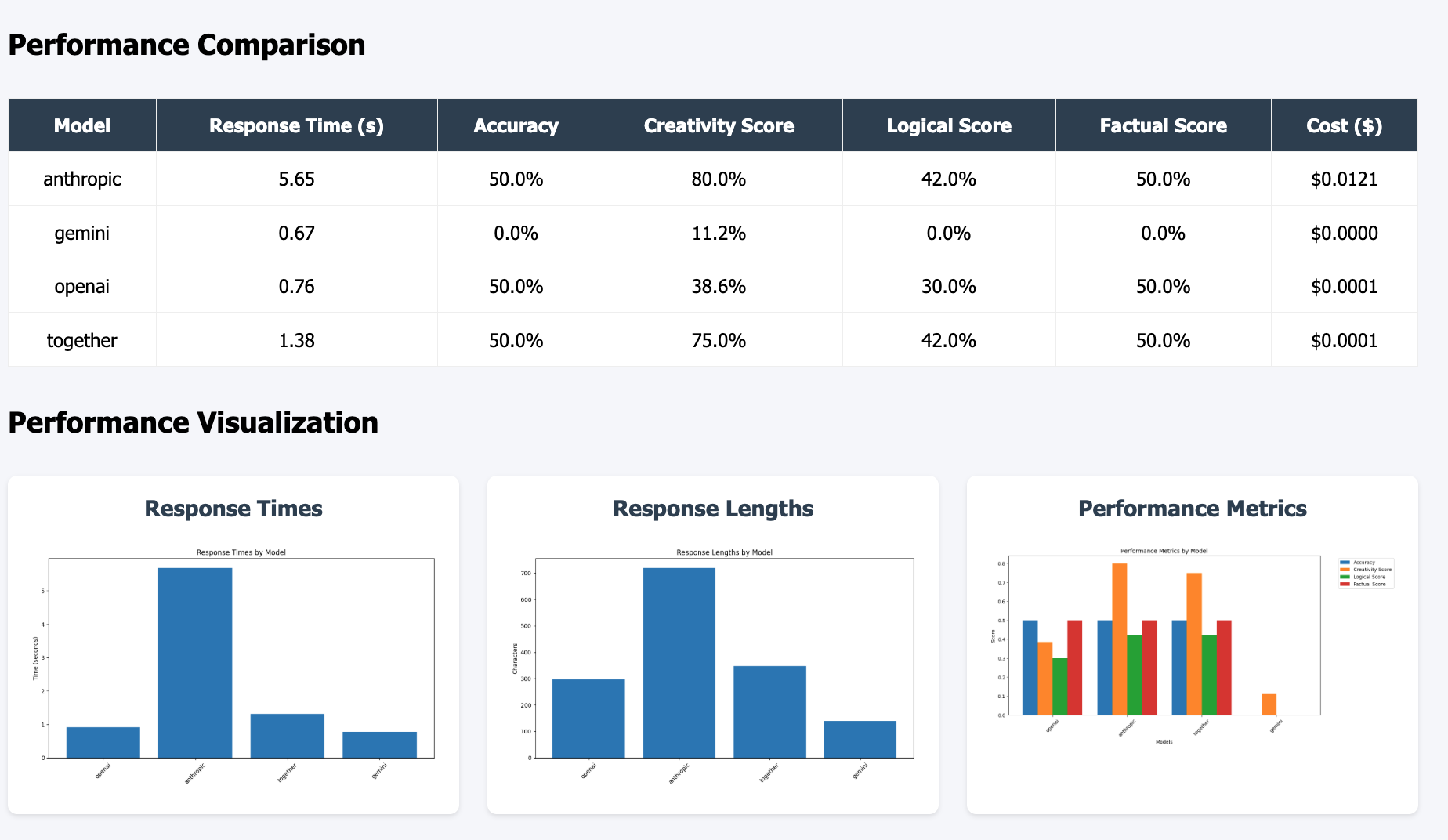
Install Flask:

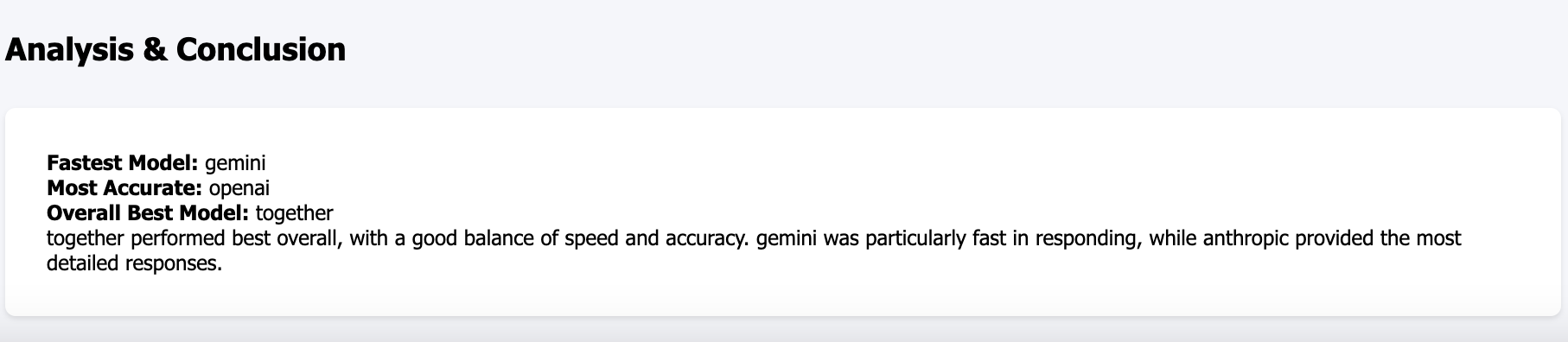
pip install flask

Run the server:

python app.py







**Understanding Phase 2: Parameter Tuning & Response Control (Open AI)**

**📌 Why is this Phase Important?**

This phase is **critical** because it allows us to **fine-tune API parameters** and understand how different settings affect the model’s behavior. It helps us determine:

* **Consistency vs. Creativity:** Adjusting temperature impacts whether responses are **deterministic (fixed output)** or **highly varied (creative)**.
* **Control Over Output Length:** max\_tokens limits the response length, useful for both **concise answers** and **detailed explanations**.
* **Structured Responses:** Using json\_schema (or similar formatting controls) ensures models **output structured data** instead of free text.
* **Comparing Models Fairly:** This phase **standardizes settings across all providers**, ensuring we are evaluating models on a **level playing field**.

**📌 What Should We Do in This Phase?**

We need to conduct systematic **experiments** by adjusting key parameters for different models and **analyzing their impact**. This involves:

**1️⃣ Experiment with temperature**

* **temperature=0 (Deterministic, Factual)** → Answers should be **fixed and consistent**.
* **temperature=1 (Creative, Unpredictable)** → Responses will **vary widely**, ideal for storytelling or poetry.

**2️⃣ Experiment with max\_tokens**

* **Ensure models do not cut off responses unexpectedly.**
* **Compare short vs. long responses** (e.g., summarizing a movie in 20 vs. 200 words).

**3️⃣ Test Response Formatting (json\_schema)**

* **Ensure structured JSON output is correctly formatted.**
* Example:

json

CopyEdit

{

"question": "Capital of France?",

"answer": "Paris",

"source": "Wikipedia"

}

* Some providers support explicit JSON output control (response\_format="json"), while others require prompt engineering.

**4️⃣ Measure Response Length Variation**

* **How much does length vary with different temperature settings?**
* **Do different models truncate long responses?**
* **How well do they maintain coherence in longer outputs?**

**📌 Do We Need to Change Any Settings in Model Provider Portals?**

✅ **No, we can do everything via code** by passing these parameters dynamically in API calls. However:

* Some providers **allow overriding certain defaults via the UI (e.g., OpenAI & Anthropic allow setting default temperature limits)**.
* We will primarily **handle it programmatically**, ensuring consistency across tests.

**📌 Adding Top 3 Models per Provider in Comparison**

This is a **great idea** because different AI providers **offer multiple models** for various use cases. Here’s **why they have different models**:

**Why Do Model Providers Have Multiple Models?**

1. **Optimized for Different Tasks**
   * **Faster, cheaper models for quick responses** (e.g., gpt-3.5-turbo, claude-3-haiku).
   * **Advanced models for complex reasoning** (e.g., gpt-4-turbo, claude-3-opus).
2. **Trade-off Between Cost, Speed, and Intelligence**
   * Some models **generate faster responses but with lower accuracy**.
   * Others are **slower but more logical**.
3. **Model Size and Training Differences**
   * **Small models (lightweight, fast)**: Good for simple tasks, **low cost**.
   * **Large models (complex, better reasoning)**: Good for analysis, **high cost**.

**How to Integrate the Top 3 Models per Provider?**

* **Run comparative tests** for:
  + OpenAI: gpt-4-turbo, gpt-3.5-turbo, davinci-003
  + Anthropic: claude-3-opus, claude-3-sonnet, claude-3-haiku
  + Google Gemini: gemini-1.5-pro, gemini-1.5-flash, gemini-1
  + Together AI: Llama-3-70B, Mixtral-8x7B, Command-R

Let me break down Phase 2 and explain its importance: (Cursor AI)

### Why Phase 2 is Important

 **Parameter Control**: Different parameters can dramatically change model outputs:

 temperature=0 makes responses more deterministic/factual

 temperature=1 makes responses more creative/varied

 This affects reliability vs creativity tradeoff

 **Cost Efficiency**:

 max\_tokens controls response length

 Proper truncation saves costs

 Helps prevent unnecessary long responses

 **Output Consistency**:

 JSON schema ensures structured outputs

 Makes responses more parseable/usable in applications

 Important for system integration

### Implementation Options

 **Via Provider Portals vs Code**:

 Most settings can be controlled via code

 No need to change provider portal settings

 Exception: Some models might need endpoint creation (like Together.ai)

 **Testing Multiple Models per Provider**:

OpenAI:

- GPT-4 (most capable)

- GPT-3.5-Turbo (balanced)

- GPT-3.5-Turbo-16k (longer context)

Anthropic:

- Claude-3 Opus (most powerful)

- Claude-3 Sonnet (balanced)

- Claude-3 Haiku (fastest)

Together.ai:

- Mixtral-8x7B

- LLaMA-2-70B

- CodeLlama-34B

Google:

- Gemini Ultra

- Gemini Pro

- Gemini Nano

### Why Different Models?

 **Resource Optimization**:

 Smaller models = faster & cheaper

 Larger models = more capable but expensive

 Helps balance cost vs performance

 **Use Case Specialization**:

 Some models better at code

 Others better at creative writing

 Some optimized for factual responses

 **Context Length Differences**:

 Different token limits

 Affects ability to handle long inputs

 Price varies with context length

### Suggested Testing Approach:

 **Parameter Testing**:

test\_parameters = {

'temperature': [0, 0.3, 0.7, 1],

'max\_tokens': [100, 500, 1000],

'top\_p': [0.1, 0.5, 1.0]

}

output\_schemas = {

'json': {'format': 'json', 'fields': ['answer', 'confidence']},

'structured': {'format': 'yaml', 'fields': ['summary', 'details']},

'free\_text': {'format': 'text'}

}

test\_cases = [

{'type': 'factual', 'prompt': 'Capital of France?'},

{'type': 'creative', 'prompt': 'Write a poem about AI'},

{'type': 'structured', 'prompt': 'Analyze this JSON data'}

]

**Understanding Phase 3: Advanced Capabilities – Tool Calling & Function Execution (Open AI)**

**📌 Why is This Phase Important?**

This phase **tests how well AI models interact with external systems** by calling functions or APIs instead of just providing text-based responses.

Here’s why it matters:

* **Prevents hallucination** – AI models often generate made-up information. Using real-time APIs helps ensure accuracy.
* **Enhances model usability** – Instead of just answering questions, AI can **take action**, like fetching real-world data.
* **Tests reasoning skills** – AI must decide **when** to call a function vs. when to generate a response on its own.

**📌 What Do We Need to Do?**

We will **implement and evaluate function calling (tool calling) capabilities** for models that support it, including: ✅ **OpenAI (GPT-4-turbo)**  
✅ **Anthropic (Claude-3)**

We will **simulate real-world use cases** where AI needs to call external functions instead of just generating text.

**📌 Tasks & Key Scenarios to Test**

**1️⃣ Implement Function Calling for OpenAI, Claude, and DeepSeek**

* OpenAI supports function calling via tool\_calls
* Claude uses tool\_use

**2️⃣ Simulate Real-World Scenarios**

We will test **four major scenarios** where models should call functions instead of guessing:

1. **Weather API Call**  
   **Example Prompt:** *“What’s the weather in London?”*  
   **Expected Behavior:**
   * Instead of guessing, the AI should call a real weather API and return an accurate result.
2. **Stock Price Lookup**  
   **Example Prompt:** *“Get me today’s price for Apple stock.”*  
   **Expected Behavior:**
   * The model should **detect that stock prices require live data** and call an external API instead of making up a number.
3. **Mathematical Execution**  
   **Example Prompt:** *“What is the area of a circle with radius 7?”*  
   **Expected Behavior:**
   * Instead of estimating, the model should **call a Python function** like:

python

CopyEdit

def calculate\_area(radius):

return 3.14 \* radius \* radius

* + This ensures **precision in mathematical calculations**.

1. **E-commerce Order Processing (Chatbot Simulation)**  
   **Example Prompt:** *“Order 2 units of product 1234 and deliver to 221B Baker Street, London.”*  
   **Expected Behavior:**
   * The model **should generate a structured tool call** to handle order processing:

json

CopyEdit

{

"role": "assistant",

"content": [

{

"type": "tool\_call",

"name": "place\_order",

"input": {

"product\_id": "1234",

"quantity": 2,

"shipping\_address": "221B Baker Street, London"

}

}

]

}

* + This mimics **real-world chatbot functionality for online stores**.

**📌 How Will We Evaluate Performance?**

1. **Does the model correctly decide when to call a function vs. generating a response?**
2. **How accurate are the API calls?** (e.g., does it actually get the right weather?)
3. **How well does the model reason about tool usage?**
   * Example: If an API fails, does the model retry or provide a fallback response?

**📌 Deliverables**

✅ **can we add another tab in the UI to implement / test this, by doing this we will not touch existing working functioniily**   
✅ **messages on the UI showing when models decide to use a tool vs. when they don’t.**  
✅ **Comparative report analyzing how well different models handle function execution.**

Let me break down Phase 3 and propose an implementation approach:**Understanding Phase 3: (Cursor AI)**

 This phase tests AI models' ability to interact with external systems through function calling

 Key aspects:

 Preventing hallucination by using real APIs

 Testing models' ability to decide when to use functions vs give text responses

 Testing reasoning skills in real-world scenarios**Implementation Approach:**

 **UI Changes:**

New Tab: "Function Testing" with:

- Function Registry section (where we define available functions)

- Test Scenario Builder

- Results Display Area

Backend Architecture:

a. Function Registry System:

- Function definitions with parameters

- Input/output schemas

- Available tools/APIs

b. Generic Testing Framework:

- Dynamic function registration

- Scenario builder

- Response analyzer

 **Making it Generic:**Instead of hardcoding scenarios, we could:

a. Function Definition Interface:

- Name & Description

- Input parameters

- Output format

- Authentication needs

- Usage examples

b. Scenario Builder:

- Select available functions

- Define test cases

- Set expected outcomes

- Configure evaluation metrics

Example Use Cases:

- Weather API calls

- Calculator functions

- Data lookup operations

- External database queries

- File operations

**Key Considerations:**

 Security: How to safely execute external functions

 Error Handling: Managing API failures

 Cost Management: Tracking API usage

 Evaluation: How to measure function calling accuracyWould you like me to:

 Start with the UI tab and basic framework

 Focus on making it generic first

 Begin with a specific implementation and then generalize?Also, should we prioritize:

 Ease of adding new functions

 Security of function execution

 Flexibility of testing scenarios

 Real-time evaluation

Issues:

* + Almost all API Keys require payment
  + When coding in cursor use command @codebase to ensure same models are referenced across scripts
  + Cursor cleaned up files before running code and removed all the files

Learnings

* + Use multiple providers to debug error as cursor uses too much credits and makes too much errors
  + AI **followed a different pattern to get the next number in the sequence, so if you want a particular algorithm to execute then give that same of the pattern as well**
  + **Keep limits of answers/output flexible (I gave output length 50 and it failed as it was less, so updated it to 25)**
  + Define, redefine and reenforce requirements effectively
  + OpenAI is better in understanding the context and derive code for the same, in some cases its better than Cursor AI