ChatGPT

AI Chatbot Agents with LangChain, FastAPI & Streamlit

This repository contains a small but extensible framework for building conversational agents on top of the **LangChain** ecosystem. It combines a **FastAPI** backend with a **Streamlit** front-end and supports multiple large-language-model providers (Groq and OpenAI) as well as an optional web search tool via **Tavily**. The goal is to make it easy to experiment with different models, prompts and search strategies while providing a simple user interface and a RESTful API.

Key features

- **Multiple providers** – choose between Groq's models (e.g. *llama-3.3-70b-versatile*, *mixtral-8x7b-32768*) and OpenAI's models (e.g. *gpt-4o-mini*).
- **Pluggable search** – optionally enable web search using the Tavily Search API to answer questions that require up-to-date information.
- **Configurable system prompt** – tailor the agent's behaviour by supplying your own system prompt.
- **REST API** – the backend exposes a `/chat` endpoint that accepts a JSON request and returns the agent's reply.
- **Streamlit UI** – a lightweight web interface for interacting with the agent without writing any code.

Repository structure

File	Purpose
`ai_agents.py`	Contains the function that builds a LangChain agent and invokes it. It selects the appropriate LLM (Groq or OpenAI) based on the request and adds the Tavily search tool when enabled.
`Backend.py`	A FastAPI application that exposes a `/chat` endpoint. It validates requests, constructs the agent via `ai_agents.get_response_from_ai_agent` and returns the generated response.
`frontend.py`	A Streamlit application that lets users choose a provider, select a model, write a system prompt and ask questions. It sends the request to the FastAPI backend and displays the agent's answer.

Prerequisites

This project requires **Python 3.9** or higher. You will also need API keys for the LLM providers and the search tool:

- `GROQ_API_KEY` – your Groq API key (required when using Groq models).
- `OPENAI_API_KEY` – your OpenAI API key (required when using OpenAI models).
- `TAVILY_API_KEY` – your Tavily API key (only needed when the search checkbox is enabled).

Create a `.env` file in the project root and set these variables if you intend to use those services. The agent function will automatically load environment variables using `dotenv`.

Install the dependencies via `pip`:

```
python3 -m venv .venv && source .venv/bin/activate
pip install -U pip
pip install fastapi uvicorn streamlit langchain langchain-groq langchain-
openai langchain-tavily pydantic requests python-dotenv
```

> **Note:** the exact package names and versions may vary. Refer to the import statements in the source files for guidance.

Running the backend

The FastAPI application listens for chat requests on port `9999`. Start it with:

```
python Backend.py
# or with uvicorn for auto-reload during development:
uvicorn Backend:app --host 127.0.0.1 --port 9999 --reload
```

Once running, you can interact with the `/chat` endpoint programmatically. Here is a minimal `curl` example:

```
curl -X POST http://127.0.0.1:9999/chat \
  -H "Content-Type: application/json" \
  -d '{
    "model_name": "llama-3.3-70b-versatile",
    "model_provider": "Groq",
    "system_prompt": "Act as a helpful assistant.",
    "messages": ["What is the capital of India?"],
    "allow_search": false
  }'
```

If the `model_name` is not in the allowed list (`llama3-70b-8192`, `mixtral-8x7b-32768`, `llama-3.3-70b-versatile`, `gpt-4o-mini`), the API will return an error.

Running the frontend

The Streamlit UI provides a simple way to compose requests. Launch it with:

```
streamlit run frontend.py
```

By default Streamlit runs on port `8501` and will open a browser window. From the interface you can:

- Write a **system prompt** to shape the assistant's personality.
- Select a **provider** (Groq or OpenAI) and then choose from the available models.
- Toggle the **Allow Web Search** checkbox to include the Tavily search tool.
- Enter your query and press **Ask Agent!** to send the request to the backend. The response will be displayed below the form.

Make sure the backend is running on `http://127.0.0.1:9999` (or adjust the `API_URL` in `frontend.py` if you choose a different host/port).

Environment variables

The application uses the `python-dotenv` package to load API keys from a `.env` file. A typical `.env` might look like this:

```
GROQ_API_KEY=your-groq-key
OPENAI_API_KEY=your-openai-key
TAVILY_API_KEY=your-tavily-key
```

Without valid keys, the calls to the corresponding services will fail. Keep your keys secret and never commit them to version control.

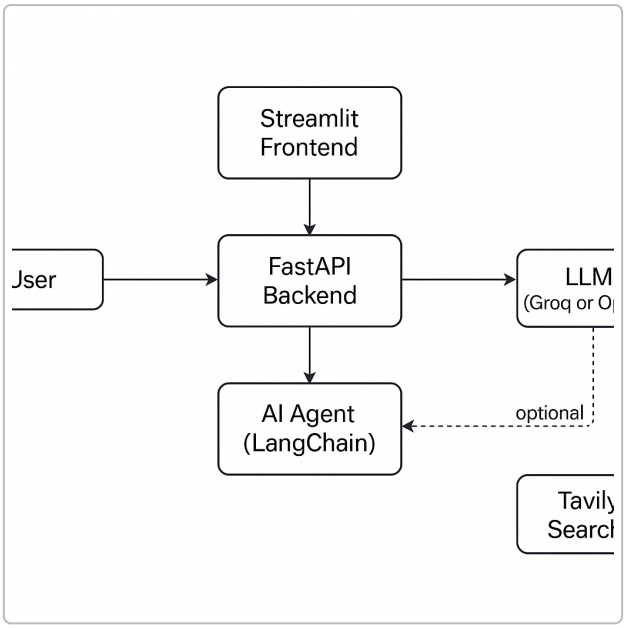
Extending the project

The modular design makes it easy to add additional providers or tools:

- **Additional LLMs** – add the provider's Python client and include the model name in `ALLOWED_MODEL_NAMES`. Update `get_response_from_ai_agent` to handle the new provider.
- **Custom tools** – create a new LangChain tool and add it to the list of tools when `allow_search` (or another flag) is enabled.
- **Advanced prompts** – implement your own prompting strategies or prompt templates in the frontend to guide the agent.

Architecture overview

The high-level flow is illustrated below. A user interacts with the Streamlit frontend, which sends a JSON payload to the FastAPI backend. The backend constructs an AI agent using LangChain. Depending on the selected provider and settings, the agent will call Groq or OpenAI models and may leverage the Tavily search tool to augment the response.



License

This project is provided without a specific license. Please adapt and use it at your own discretion.

Acknowledgements

This repository builds upon the great work of the LangChain community and uses open-source tools such as Streamlit, FastAPI and Tavily. Thank you to the developers of these libraries for making conversational AI accessible.
