AI Chatbot With Web-Search

Project Overview:

The project includes implementing an AI Chatbot that uses LLMs (Large Language Models) and real time web search features. The project consists of three main systems, a locally hosted LLM (Llama 3.2 1B Instruct), a web interface (Streamlit) and a search system (Serper dev tool).

Component 1: Locally Hosted LLM Using FastAPI

The model which is used for local hosting is Llama 3.2 1B Instruct.

Reason for selection:

- Lightweight yet capable for conversations
- Large context window suitable for long chat history and search results
- Good performance yet less resource consumption

The LLM is hosted using FastAPI and OpenAI like endpoints with streaming token feature

Endpoint: POST /v1/chat/completions

Example Input Schema:

```
{
  "model": "llama-3.2-1b-instruct",
  "messages": [
    {"role": "user", "content": "Your question here"}
  ],
  "use_search": false
}
```

The input Schema for the endpoint was designed like OpenAI Chat Completions with an additional use_search parameter which indicates the user forcing web search.

Streaming Implementation:

The streaming was achieved using few key components

1. TextIteratorStreamer: Streaming utility function provided by Hugging Face
2. Threading: Separated token generation from response streaming to prevent blocks

   Reference of Streaming and Threading: Link

3. JSON Chunking: Sends the chunks in OpenAI streaming JSON format
4. FastAPI: to send streaming content via HTTP
   a. Host: 127.0.0.1
   b. Port: 8000

Generation Parameters:

- max_new_tokens: 4000 (sufficient for detailed responses)
- temperature: 0.7 (balanced creativity/consistency)
- top_p: 0.95 (nucleus sampling for quality)
- repetition_penalty: 1.1 (reduces repetitive output)

Example Output Schema:

```
{
    "choices": [{
        "delta": {"content": content},
        "index": 0,
        "finish_reason": None
    }],
    "object": "chat.completion.chunk"
}
```

Streaming content can be accessed using a POST request to the endpoint.

Component 2: Search Web System Using Serper Dev Tool

This system has two-tier approach to making the decision if a search is needed or not.

Tier 1: Keyword based search

Triggers a search query immediately if any keywords match the search query. A few keywords based on time, news, real-time data, recent events etc are predefined.

- Time-sensitive terms: "latest", "recent", "current", "today", "2024", "2025"
- News-related: "news", "update", "trending"

Tier 2: LLM powered decision making

The keyword-based search is backed by an LLM powered decision making system which makes search decisions based on specified criterion in the prompt.

```
decision_prompt = f"""You are an AI assistant that determines whether a user query requires current web search or
can be answered with existing knowledge.
Analyze this user query and respond with ONLY "YES" or "NO":

- Answer "YES" if the query asks for:
* Current events, recent news, or breaking updates
* Real-time information (stock prices, weather, sports scores)
* Recent developments or latest information about any topic
* Information that changes frequently or is time-sensitive
* Specific facts that might have changed recently
* Current status of ongoing situations

User Query: "{query}"
Decision (YES/NO only):"""
```

(disclaimer: prompt created using AI and edited for strict reasoning)

The user also has the ability to force search from the frontend which triggers the web search without analysing the user query.

Search Integration Workflow

1. **Query Analysis**: check for keywords in the user query
2. **Web Search Execution**: Query Serper API for relevant results
3. **Context Injection**: Add search results as system context
4. **Response Generation**: LLM generates response using both its knowledge and search context

5. **Source Citation**: Automatically append citations to the response

Component 3: Chat Interface Using Streamlit

The chat user interface for this application is created using Streamlit. The UI has a build in streaming support for the api endpoint which can be accessed using st.write_stream.

User Experience Features

- **Visual Search Indicator**: Shows "Searching the web..." when search is triggered
- **Conversation Memory**: sends last 5 conversations to api endpoint to maintain context
- **Force Search Option**: Sidebar toggle to force searching

Design Decisions

1. **Llama 3.2 1B Instruct as LLM**: light weight and high context window
2. **Threading for Non-Blocking during Streaming**: used for separating token generation from response streaming to prevent blocks
3. **Context Window Management**: only last five conversations are passed to LLM to prevent context overflow while keeping conversation context
4. **Search Result Integration**: provided first five results from the Serper Dev Tool as context
5. **Citation**: user is provided with citation of source when web search answers are generated

Possible Enhancements:

1. Use semantic search from chat history for getting context from past conversations instead of passing last five chats
2. Use the peopleAlsoAsk section in json from serper api for question suggestion to user (now only using organic section)
3. Enable voice input and output feature