Learning Log: Local LLM Chat App Development

Project Name: Local LLM Chat App (Ollama + FastAPI + Streamlit)

1. Topics Learned

A. Core Technologies

Topic Key Learnings

Ollama - Running local LLMs (Mistral, Llama2)

FastAPI - Building REST APIs in python

Streamlit - Creating interactive UIs

WebSockets - Streaming LLM responses via SSE (Server-Sent Events)

B. Advanced Concepts

Asynchronous Programming: async/await for handling concurrent LLM requests.

Model Management: Switching between different Ollama models dynamically.

State Management: Using Streamlit’s session\_state for chat history persistence.

2. Issues Faced & Solutions

A. Backend Challenges

Issue Solution

Ollama not responding Verified ollama serve was running in background

SSE streaming failures Used StreamingResponse with proper text/event-stream headers

CORS errors Added CORSMiddleware to FastAPI with allow\_origins=["\*"]

B. Frontend Challenges

Issue Solution

Chat freezing during streams Implemented timeout (30s) and error handling in requests.post()

No response rendering Fixed JSON parsing of SSE chunks (data: {...} format)

State reset on rerun Used st.session\_state to persist messages

C. Integration Challenges

Issue Solution

Port conflicts Assigned unique ports (Chat: 8000/8501, Summarization: 8001/8502)

Model loading delays Added a loading spinner with st.spinner()

3. Key Code Snippets

A. Backend (FastAPI)

python

# Streaming endpoint

@app.post("/chat")

async def chat(request: ChatRequest):

def generate():

for chunk in ollama.chat(model=request.model, messages=[...], stream=True):

yield f"data: {json.dumps(chunk['message'])}\n\n"

return StreamingResponse(generate(), media\_type="text/event-stream")

B. Frontend (Streamlit)

python

# Handling streaming responses

for line in response.iter\_lines():

if line:

decoded = line.decode('utf-8')

if decoded.startswith('data:'):

data = json.loads(decoded[5:])

st.write(data['content'])

4. Lessons Learned

Async > Sync: Async FastAPI endpoints improved throughput for LLM streaming.

Debugging Tools:

Backend: FastAPI’s /docs endpoint for API testing.

Frontend: Browser’s Network tab to inspect SSE streams.

Modular Design: Separating llm/ module made it reusable for summarization.

# Local LLM Chat App

This is a real-time chat application using Streamlit, FastAPI, and Ollama's local LLMs.

## Project Structure

- `backend/`: The FastAPI backend that serves the LLM.

- `frontend/`: The Streamlit frontend for the user interface.

- `llm/`: Contains the LLM logic (in this case, handled by the `ollama` library).

## Setup and Usage

### Prerequisites

- Python 3.7+

- Ollama installed and running with the `mistral` model.

### Backend Setup

1. Navigate to the `backend` directory:

```bash

cd backend

```

2. Install the required packages:

```bash

pip install -r requirements.txt

```

3. Run the FastAPI server:

```bash

uvicorn main:app --reload

```

### Frontend Setup

1. Navigate to the `frontend` directory:

```bash

cd frontend

```

2. Install the required packages:

```bash

pip install -r requirements.txt

```

3. Run the Streamlit app:

```bash

streamlit run app.py

```

Now, open your browser to the Streamlit URL (usually `http://localhost:8501`) to use the chat application.