Issues Faced

Problem 1:

Cross-Origin Resource Sharing (CORS) Error

Issue:

When calling FastAPI endpoints from Streamlit (frontend → backend), browsers may block the request due to **CORS restrictions**.

Example error in browser console:

Access to fetch at 'http://localhost:8000/chat' from origin 'http://localhost:8501' has been blocked by CORS policy

Solution:
Enable CORS in FastAPI:

```
from fastapi.middleware.cors import CORSMiddleware

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

Problem 2:

File Upload Parsing Error

Issue:

extract_text(file) may throw errors if the uploaded file is not properly passed to the backend.

UploadFile in FastAPI gives a file-like object, but uploaded_file.getvalue() in Streamlit gives raw bytes.

Solution:
Convert file correctly before sending:

```
files = {"file": ("document.pdf", uploaded_file.getvalue(), "application/pdf")}
response = requests.post("http://localhost:8000/summarize", files=files)
```

Problem 3:

Missing Dependency Errors

Issue:

Running the app may fail with errors like:

ModuleNotFoundError: No module named 'ollama_client'
ModuleNotFoundError: No module named 'doc_parser'

Solution:
Ensure you create ollama_client.py and doc_parser.py with proper functions (get_chat_response, get_summary, extract_text).

Install required libraries:

```
pip install fastapi uvicorn streamlit pydantic requests pypdf2
```

Problem 4:

Uvicorn Server Not Running

Issue:
Streamlit frontend won't work unless FastAPI backend is running at http://localhost:8000.

Solution:
Start backend first:

```
uvicorn main:app --reload --port 8000
```

Then run Streamlit frontend:

```
streamlit run app.py
```

Problem 5:

Large PDF Processing Crash

Issue:
Uploading large PDFs may cause memory errors or timeouts in extract_text.

Solution:

Use chunk-based PDF parsing (PyPDF2, pdfplumber, or langchain.text_splitter).
Apply summarization **incrementally per chunk**.

```
from langchain.text_splitter import RecursiveCharacterTextSplitter
splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=200)
chunks = splitter.split_text(content)
```

Problem 6:

Inconsistent LLM Responses

Issue:
get_summary or get_chat_response may return empty/irrelevant outputs depending on model or prompt.

Solution:

Add **prompt engineering** for clarity.

Example for summarization:

```
prompt = f"Summarize the following document in bullet points:\n\n{content}"
summary = get_summary(prompt)
```

Problem 7:

Streamlit Blocking UI

Issue:
When backend response takes time (large docs or long chats), Streamlit UI looks frozen.

Solution:
Use Streamlit's spinner and success indicators:

```
 with st.spinner("Processing..."):
   response = requests.post(...)
st.success("Done!")
```