YouTube RAG Chatbot – Conversational AI Over Any YouTube Video

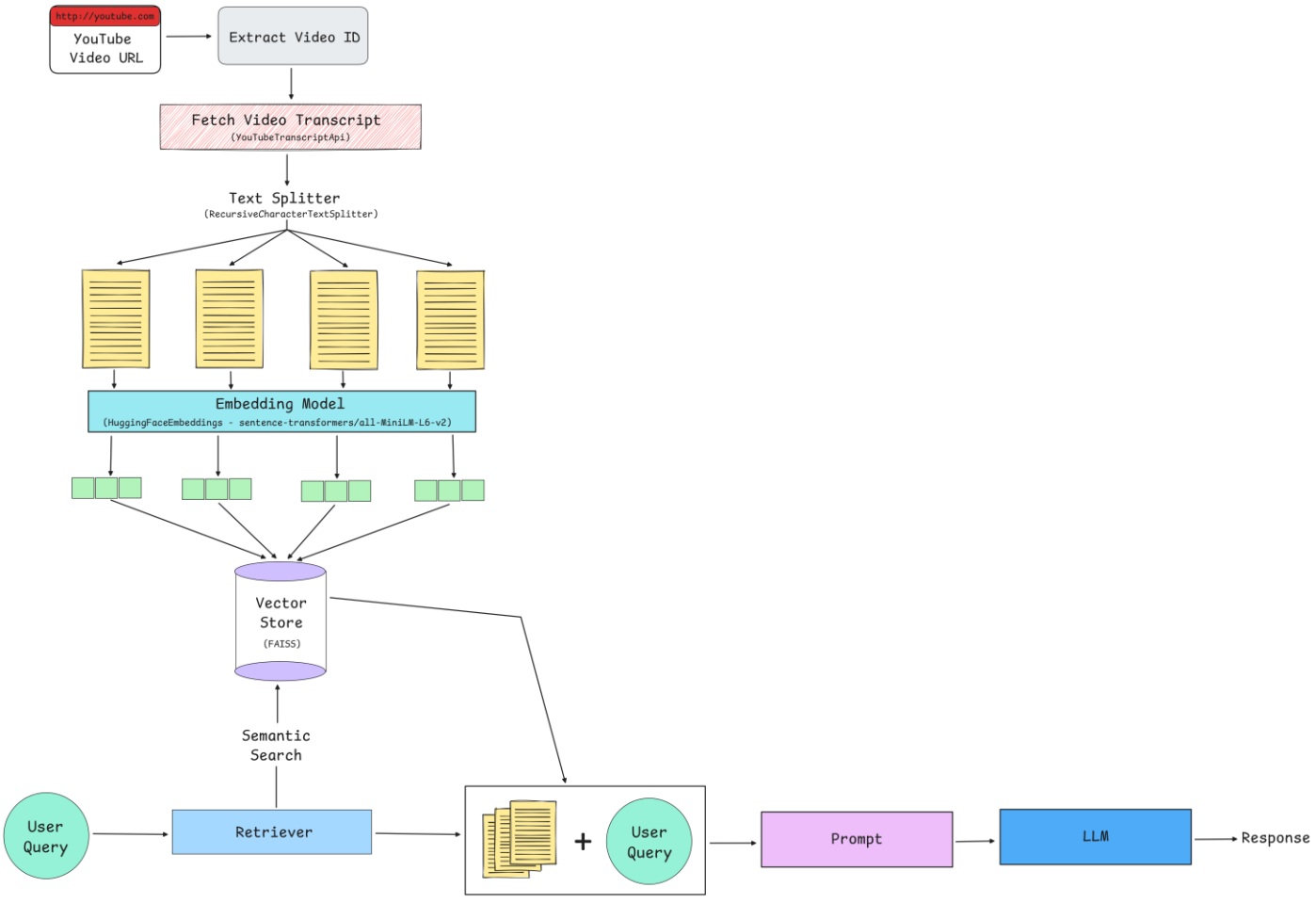
1. Objective:

The goal of this project is to build a fully working **Retrieval-Augmented Generation (RAG) chatbot** that allows users to have an intelligent conversation with any YouTube video. Users can enter a YouTube URL, after which the system extracts the transcript, chunks it, embeds it, performs vector similarity search, and uses an LLM to answer questions strictly based on the video's content.

2. Tools / Frameworks Used

- **Python**
- **Streamlit** for the interactive web UI
- **YouTubeTranscriptApi** for transcript extraction
- **HuggingFaceEmbeddings**
    - Model: *sentence-transformers/all-MiniLM-L6-v2*
- **FAISS Vector Store** for similarity search
- **LangChain** components:
    - *RecursiveCharacterTextSplitter*
    - *PromptTemplate*
    - *RunnableParallel, RunnablePassthrough*
    - *StrOutputParser*
- **Google Gemini-2.5-flash** via *ChatGoogleGenerativeAI* LLM
- **dotenv** for managing API keys

3. Approach / Workflow Summary



4. Key Implementation Steps

Below are the major steps implemented in your actual code:

a. Loading Embeddings & LLM

```
embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
```

```
llm = ChatGoogleGenerativeAI(model="gemini-2.5-flash")
```

These are cached using `st.cache_resource` for efficient repeated use.

b. Chunking the Transcript

Chunk size: **1000 characters**
Chunk overlap: **100 characters**

This ensures semantic continuity.

```
splitter = RecursiveCharacterTextSplitter(chunk_size=1000, chunk_overlap=100)
```

```
chunks = splitter.create_documents([transcript])
```

c. Creating FAISS Vector Store

```
vectorstore = FAISS.from_documents(chunks, embeddings)
```

d. Setting Up the RAG Chain

Retriever returns top-k (k=5) relevant chunks.

Prompt enforces grounded answers only:

"You are a helpful YouTube RAG assistant.
Answer ONLY from the given transcript content.
If transcript is insufficient, say you don't know."

The final RAG chain:

```
rag_chain = (
    RunnableParallel(context=retriever | format_docs, question=RunnablePassthrough())
    | prompt
    | llm
    | StrOutputParser()
)
```

5. Results & Observations

What Worked Well

- **Accurate retrieval:** MiniLM embeddings + FAISS returned the most relevant transcript chunks.
- **Fast query response:** Gemini-2.5-flash provides low latency for conversational answers.
- **Grounded answers:** The prompt prevents hallucination by restricting answers strictly to transcript content.
- **Robust transcript fetching:** Proxy configuration solved regional transcript access issues.
- **User-friendly interface:** Streamlit UI is simple and intuitive.

Outputs Consistently Included:

- Correct factual recall from the video
- Timestamp-independent content understanding
- Multi-turn conversation with memory

6. Learnings & Future Improvements

Learnings

- RAG pipelines dramatically boost factual accuracy for domain-specific questions.
- Chunking strategy & embedding model choice strongly influence retrieval quality.
- Prompt engineering significantly reduces hallucinations.
- Managing transcripts from YouTube requires handling network issues, missing captions, and proxies.
- Building complete end-to-end systems is more about **integration logic** than pure ML.

Future Improvements

- Add multi-video memory (chat across several videos).
- Include a **summarization option** for long videos.
- Add semantic search + timeline answer linking (e.g., mention timestamps).
- Replace FAISS with scalable vector DB (Pinecone / Weaviate) for large-scale usage.
- Improve UI (e.g., history download, dark mode, video section highlighting).
- Add fallback for videos without transcripts using Whisper transcription.

7. Code & Demo Link

GitHub Repository: https://github.com/rahul5r/YouTubeChatbot

The complete pipeline is implemented in *app.py.*