Chatbot Documentation

I developed a chatbot that can understand podcast transcripts, identify speakers correctly, maintain chat memory, and link back to the original sources. Below, I explain the key methods and features used in building this system.

Key Methods & Features

Vector Database (FAISS) for Retrieval

I used FAISS to store podcast transcripts as embeddings for efficient retrieval.

How Chunks Are Created:

- Transcripts are split by section topics to keep related content together.
- Each chunk contains speaker names, timestamps, YouTube links, and speech content to ensure responses have full context.
- Chunk size is optimized (around 1000 tokens with 200-token overlap) to balance detail and efficiency.

Semantic Search:

- When a user asks a question, the chatbot converts it to an embedding and searches for the most relevant transcript segments.
- This ensures that only meaningful responses are returned.

Speaker Attribution

I designed the system to always include speaker names in responses.

Data Structure I Built:
Each transcript entry contains:

- Speaker information
- Section topics
- Timestamps
- YouTube links
- Speech content

Dialogue Formatting:
When creating chunks, I made sure to preserve speaker details in the following format:

```

```
dialogue = f"**{speaker}**: {speech} on section {current_section}(specific video part : {youtube_link} on {video})
```

This ensures the chatbot correctly attributes each response.

**Memory Management**

I implemented session-based memory so that the chatbot remembers past conversations.

How Memory Works:

- Session Tracking: Each user conversation has a unique session ID to keep track of context.
- Conversation Buffer: I used ConversationBufferMemory to store past messages.
- Context Window Management: The chatbot retrieves only the last three exchanges to prevent overflowing the context window.

Smart Context Handling:

- Instead of blindly searching the vector database again, I used Gemini AI to decide whether to:
    - Retrieve from vector search (if the user asks a new question)
    - Use chat history (if the question is a follow-up based on recent messages)

This reduces unnecessary searches and makes responses more efficient.

**Source Attribution**

Each response links back to the original podcast source to ensure credibility.

How I Handled Source Information:

- Each transcript chunk includes:
    - Podcast title
    - YouTube link
    - Timestamp and it's link

This way, users can verify responses directly from the source.
```

Challenges & Solutions

Challenge 1: Handling Long Transcripts

Problem: Podcast transcripts are long and exceed the LLM's context window.
Solution:

- Chunking strategy respects section boundaries (no mid-sentence cuts).
- FAISS vector database ensures fast and relevant searches.
- Relevance-based retrieval only fetches the most important transcript sections.

Challenge 2: Mixing Conversations from Different Podcasts

Problem: Some queries require information from multiple podcasts, but mixing them could create confusion.
Solution:

- Implemented query-based vector store selection so that the chatbot searches only the most relevant podcasts.
- If a question is broad (e.g., "Who talks about AGI?"), the chatbot retrieves from multiple vector stores and merges responses.