# Chat History Feature - Design Document

## 1. Introduction

This document details the implementation of the chat history feature for the existing chatbot system. The chat history feature enhances context awareness by maintaining a record of past user interactions and utilizing them to generate standalone queries for improved response accuracy. This ensures that the chatbot maintains coherence across multiple exchanges.

## 2. System Overview

### 2.1 Objective

The chat history feature aims to:  
- Maintain short-term and long-term memory of conversations.  
- Retrieve past messages to provide context-aware responses.  
- Convert user queries into standalone questions for improved LLM comprehension.  
- Integrate past responses into the chatbot’s decision-making process.

### 2.2 Workflow

The chat history feature operates as follows:  
1. The user inputs a query.  
2. The system retrieves past interactions from memory (`chat\_context`) and FAISS.  
3. The query is rewritten into a standalone question.  
4. The chatbot retrieves relevant context from previous interactions.  
5. The chatbot generates a response based on retrieved context.  
6. The chatbot stores the new interaction for future reference.  
7. The process repeats to ensure continuity across multiple interactions.

## 3. Key Modules

### 3.1 Context Retrieval

Extracts historical conversation data from `chat\_context` and retrieves relevant information from FAISS to maintain continuity in chatbot responses.

def retrieve\_context\_from\_variable():  
 """Extracts recent conversation history from `chat\_context` variable."""  
 return "\n".join([f"User: {entry['user']}\nAssistant: {entry['assistant']}" for entry in chat\_context])

### 3.2 Standalone Question Generation

Ensures user queries are rewritten into fully self-contained questions, improving response accuracy by converting partial or follow-up queries into complete standalone inputs.

def generate\_standalone\_question(chat\_history, user\_input):  
 """Rewrites user input into a standalone question"""  
 standalone\_question\_prompt = f"""  
 You are an AI that reformulates user questions into standalone queries.  
 Given the conversation history and user input, rewrite it as a fully independent question.  
  
 ### Chat History:  
 {chat\_history}  
  
 ### User Input:  
 {user\_input}  
  
 ### Standalone Question:  
 """  
 rephrased\_question = ""  
 for chunk in local\_llm.stream(standalone\_question\_prompt):  
 rephrased\_question += chunk  
   
 return rephrased\_question.strip()

### 3.3 FAISS Vector Search

Uses FAISS for efficient semantic similarity search, ensuring that the chatbot retrieves the most relevant past conversations to use as context. The system leverages HNSW indexing to speed up search operations.

def query\_vector\_store(concept):  
 """Retrieve relevant documents from FAISS using optimized HNSW indexing."""  
 docs = vector\_store.similarity\_search(concept, k=10)  
 return "\n".join([doc.page\_content for doc in docs]) if docs else "No relevant context found."

### 3.4 Memory Management

The chatbot maintains a memory buffer that keeps track of past interactions, allowing it to recall recent exchanges. Older messages may be pruned to optimize performance.

def manage\_memory():  
 """Manages in-memory chat history"""  
 all\_memory = memory.load\_memory\_variables({}).get("chat\_history", [])  
 memory\_chunks = []  
 for message in all\_memory:  
 memory\_chunks.extend(tokenize\_and\_chunk(message.content))  
 return "\n".join(memory\_chunks[-5:]) # Keep last 5 chunks

### 3.5 Tokenization and Chunking

Tokenization and chunking ensure that large conversation history is split into manageable segments to avoid exceeding LLM input limits. Uses Hugging Face’s tokenizer to divide large inputs efficiently.

from transformers import AutoTokenizer  
  
# Initialize Tokenizer  
tokenizer = AutoTokenizer.from\_pretrained("gpt-2")  
  
def tokenize\_and\_chunk(text, max\_tokens=768, overlap=200):  
 """Tokenizes text and splits it into smaller overlapping chunks."""  
 tokens = tokenizer.encode(text)  
 chunks = [tokens[i:i + max\_tokens] for i in range(0, len(tokens), max\_tokens - overlap)]  
 return [tokenizer.decode(chunk) for chunk in chunks]

## 4. System Constraints & Future Enhancements

### 4.1 Constraints

- Requires GPT-4 API access for standalone question rewriting.  
- FAISS indexing may need further optimization for larger datasets.  
- Assumes consistent formatting of `chat\_context`.

### 4.2 Future Enhancements

1. \*\*Adaptive Memory Retention\*\* - Dynamically adjusts the number of stored interactions.  
2. \*\*User Intent Prediction\*\* - Recognizes query intent to refine responses.  
3. \*\*Extended Long-Term Memory\*\* - Allows retrieval of older interactions beyond FAISS limits.  
4. \*\*Dynamic Context Weighting\*\* - Assigns priority to critical past responses.

## 5. Conclusion

The chat history feature significantly enhances chatbot capabilities by enabling it to recall and incorporate past conversations into responses. By retrieving prior exchanges and rewriting user queries into standalone questions, the chatbot ensures that its answers remain contextually accurate and coherent over time.

## 6. References

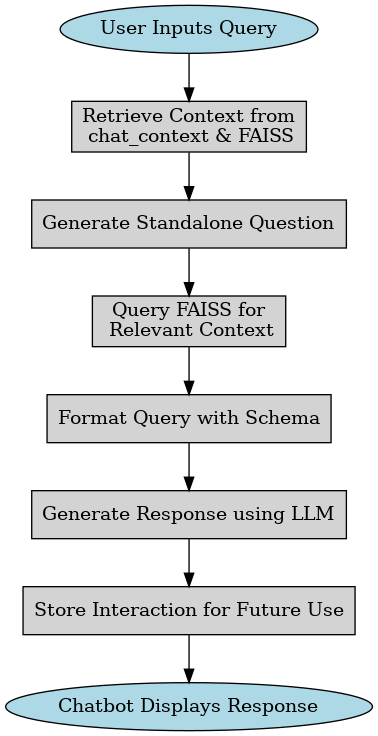
1. Johnson, J., & Goyal, A. (2019). "FAISS: Facebook AI Similarity Search."

2. Vaswani, A., et al. (2017). "Attention Is All You Need." NeurIPS.

3. LangChain Documentation - https://docs.langchain.com

## 7. Workflow Flowchart

The following flowchart illustrates the process of handling user queries, retrieving past context, generating standalone questions, and processing the final response.



## 7. Editable Workflow Flowchart

The flowchart below illustrates the chatbot's workflow, including query processing, context retrieval, and response generation. This version is an editable SmartArt diagram for modification within Microsoft Word.

|  |
| --- |
| User Inputs Query |
| Retrieve Context from chat\_context & FAISS |
| Generate Standalone Question |
| Query FAISS for Relevant Context |
| Format Query with Schema |
| Generate Response using LLM |
| Store Interaction for Future Use |
| Chatbot Displays Response |