Name: Khushi Kumari
Reg.No.: 22BCE10230
Topic: AI_Customer_Support_Bot

Github Repo Link: [Github]
https://github.com/khushikumari0202/Ai_Customer_Support_Bot

Demo video Link: [Demo]
https://drive.google.com/file/d/1HrHmekZD_nWIV-pDi8w1BrmucEiHCrpu/view?usp=sharing

---

1. INTRODUCTION

1.1 Introduction

The growing demand for efficient customer service has led organizations to adopt AI-powered solutions. Traditional support systems often struggle with delayed responses, limited scalability, and high operational costs. The **AI Customer Support Bot** addresses these challenges by leveraging **Natural Language Processing (NLP)** and **Large Language Models (LLMs)** to simulate human-like conversations and resolve queries autonomously.

This system uses **FastAPI** for high-performance API handling and integrates a **Retrieval-Augmented Generation (RAG)** approach to improve accuracy by grounding responses on real data. It can be integrated with websites or applications to offer 24×7 customer support, reducing manual workload and response time.

1.2 Motivation for the Work

Manual customer support systems require human agents for repetitive tasks like answering FAQs, which consumes time and resources. The motivation behind this project is to automate such processes using artificial intelligence, reducing human dependency while maintaining response accuracy and personalization.

1.3 Techniques Used

- **FastAPI** for creating RESTful APIs.
- **Gemini 2.5 Flash Model** for LLM-based response generation.
- **RAG (Retrieval-Augmented Generation)** for context-aware answers.
- **Vector embeddings** for semantic search in FAQs.
- **Session management** for conversation continuity.
- **Ngrok / Local deployment** for testing and API exposure.

1.4 Problem Statement

Existing customer support systems are inefficient, lack personalization, and require human intervention for basic queries. The goal of this project is to build an intelligent, self-learning chatbot capable of answering customer queries with context-awareness, accuracy, and efficiency.

1.5 Objectives

- Develop an AI-powered chatbot using FastAPI.
- Implement context retention across sessions.
- Integrate a retrieval-based mechanism for factual responses.
- Provide escalation logic for low-confidence cases.
- Enable easy integration with web platforms.

1.6 Summary

This chapter introduced the concept and motivation behind building an AI-powered customer support system, its challenges, and the objectives set to address them using advanced AI and API frameworks.

---

2. LITERATURE SURVEY

2.1 Introduction

Chatbots have evolved from rule-based systems to intelligent conversational agents powered by LLMs. Studies have shown that integrating retrieval and generative approaches improves factual accuracy. Tools like **FastAPI**, **LangChain**, and **vector databases** are widely used for developing scalable AI chatbots.

2.2 Core Area of the Project

The core of this project lies in combining **LLM-based natural language understanding** with **retrieval mechanisms** to generate relevant responses dynamically.

2.3 Existing Systems

Existing systems like ChatGPT, IBM Watson, and Dialogflow offer conversational AI but are either expensive or lack domain-specific adaptability. The AI Customer Support Bot aims to be lightweight, customizable, and open-source.

2.4 Research Gaps

- Lack of contextual continuity in existing chatbots.
- Limited customization for business-specific data.
- High latency and cost of commercial LLM APIs.

2.5 Summary

The literature survey highlights the need for an efficient, context-aware, and easily deployable customer support system leveraging modern AI tools.

---

3. SYSTEM ANALYSIS

3.1 Introduction

System analysis focuses on understanding the functional requirements, identifying components, and evaluating the data flow between modules.

3.2 Limitations in Existing System

- Manual intervention required for repetitive queries.
- Poor contextual understanding.
- High maintenance cost.
- Inconsistent answers for similar queries.

3.3 Proposed System

The proposed AI Customer Support Bot provides:

- **Real-time query resolution** using Gemini API.
- **Contextual memory management** for continuity.
- **RAG-based knowledge retrieval** for factual correctness.
- **API endpoints** (`/chat`, `/history`, `/reset`) for easy integration.

3.4 Summary

The system analysis identifies limitations in existing methods and proposes an advanced, automated support bot addressing these inefficiencies.

---

4. SYSTEM DESIGN AND IMPLEMENTATION

4.1 Introduction

This phase focuses on designing the architecture, data flow, and implementing the modules in FastAPI.

4.2 System Architecture

1. **User Interface** – Customer interacts via web or app.
2. **FastAPI Backend** – Processes requests and communicates with the model.
3. **Knowledge Base / Vector Store** – Stores FAQ embeddings for retrieval.
4. **Gemini API** – Generates contextually appropriate answers.

5. **Session Memory** – Stores ongoing conversation for continuity.

4.3 Module Design

- **Chat Module**: Handles message exchange.
- **Retrieval Module**: Searches FAQ data using embeddings.
- **LLM Response Module**: Generates AI responses.
- **Memory Module**: Maintains user context.
- **Escalation Module**: Triggers human intervention if confidence < threshold.

4.4 Implementation Tools

- Python 3.10+
- FastAPI
- Uvicorn
- Gemini 2.5 Flash
- LangChain
- FAISS / Pinecone (for embeddings)

4.5 Code Screenshots

```
# --- CELL 1: COMPLETE, CORRECTED SETUP, DEFINITIONS, AND NON-BLOCKING SERVER STARTUP ---

# Install necessary libraries (in case the runtime was restarted)
!pip install -q datasets pandas transformers sentence-transformers faiss-cpu google-genai fastapi uvicorn nest_asyncio pyngrok

# --- Imports (Consolidated and Corrected) ---
from fastapi import FastAPI, HTTPException
from fastapi.responses import RedirectResponse
from pydantic import BaseModel
import pandas as pd
from datasets import load_dataset
import re
from sentence_transformers import SentenceTransformer
import faiss
import numpy as np
import os
from google import genai
from google.colab import userdata
import uvicorn
import nest_asyncio
import asyncio
from pyngrok import ngrok, conf

from fastapi.middleware.cors import CORSMiddleware
# --- ADDED NECESSARY IMPORTS FOR THREADING FIX ---
import threading
import time
# --------------------------------------------------
```

Fig. setup

```python
# --- 1. RAG COMPONENT INITIALIZATION ---
dataset = load_dataset("MakTek/Customer_support_faqs_dataset", split="train")
df = dataset.to_pandas()
knowledge_base_df = df
print("Checkpoint 1: Dataset loaded and converted to DataFrame.")

def clean_text(text):
    text = text.lower()
    text = re.sub(r'\s+', ' ', text).strip()
    return text

df['cleaned_question'] = df['question'].apply(clean_text)

try:
    # Initialize SentenceTransformer and Faiss Index
    model = SentenceTransformer('all-MiniLM-L6-v2')
    embeddings = model.encode(df['cleaned_question'].tolist(), convert_to_tensor=False)
    embeddings = np.array(embeddings).astype('float32')
    d = embeddings.shape[1]
    index = faiss.IndexFlatL2(d)
    index.add(embeddings)
    print("Checkpoint 2: RAG components (model, index) initialized.")
except Exception as e:
    print(f"Error initializing RAG components: {e}")

def retrieve_faq(query_text, k=1):
    cleaned_query = clean_text(query_text)
    query_embedding = model.encode([cleaned_query], convert_to_tensor=False)
    query_embedding = np.array(query_embedding).astype('float32')
    D, I = index.search(query_embedding, k)

    results = []
    for row_index in I[0]:
        if row_index < 0: continue
        faq_q = knowledge_base_df.iloc[row_index]['question']
        faq_a = knowledge_base_df.iloc[row_index]['answer']
        results.append({
            'source_question': faq_q,
            'source_answer': faq_a,
            'match_score': D[0][I[0].tolist().index(row_index)]
        })
    return results
```

Fig. RAG Initialization

```python
# --- 2. LLM CLIENT INITIALIZATION ---
try:
    # Note: Assumes 'GEMINI_API_KEY' is the secret for your Gemini key
    GEMINI_API_KEY = userdata.get('GEMINI_API_KEY')
    if not GEMINI_API_KEY:
        raise ValueError("API Key not found in Colab Secrets.")

    # DEBUG: Key status check
    print(f"DEBUG: Key status: Read successfully. Length: {len(GEMINI_API_KEY)} characters.")

    client = genai.Client(api_key=GEMINI_API_KEY)
    LLM_NAME = 'gemini-2.5-flash'
    print(f"Checkpoint 3: Gemini client initialized using {LLM_NAME}.")

except Exception as e:
    print(f"FATAL ERROR: Could not initialize Gemini client: {e}")
    client = None

# --- MODIFIED SYSTEM_PROMPT FOR CONTEXTUAL MEMORY ---
SYSTEM_PROMPT = """
You are an AI Customer Support Bot. Your task is to provide concise, direct, and helpful answers to customer questions.
You MUST ONLY use the provided 'CONTEXT' (Retrieved FAQs) and the 'HISTORY' (Previous conversation) to formulate your answer.
The HISTORY is provided to understand the CONTEXT of the CURRENT user question.

If the provided CONTEXT does not contain the answer, you must state:
"I apologize, I was unable to find a definitive answer in our FAQs. I will now simulate an escalation to a human agent."
Do not use any external knowledge.
"""
# ---------------------------------------------------


# --- MODIFIED generate_rag_response FUNCTION (ADDED conversation_context) ---
def generate_rag_response(user_query, k=1, conversation_context=""):
    if client is None:
        return "LLM service not initialized. Cannot generate response."

    retrieved_faqs = retrieve_faq(user_query, k=k)

    if not retrieved_faqs:
        context = "No relevant FAQs found."
    else:
        context = "\n---\n".join([f"Question: {item['source_question']}\nAnswer: {item['source_answer']}" for item in retrieved_faqs])

    # Check for escalation based on low relevance score (> 5 is poor match)
    if "No relevant FAQs found" in context or (retrieved_faqs and retrieved_faqs[0]['match_score'] > 5):
        final_answer = "I apologize, I was unable to find a definitive answer in our FAQs. I will now simulate an escalation to a human agent."
    else:
        # --- MODIFIED PROMPT CONSTRUCTION TO INCLUDE HISTORY ---
        prompt = f"""
        HISTORY:
        {conversation_context}

        CONTEXT (Retrieved FAQs):
        {context}

        CURRENT USER QUESTION: {user_query}
        """
        try:
            response = client.models.generate_content(
                model=LLM_NAME, contents=[SYSTEM_PROMPT, prompt], config={"temperature": 0.0}
            )
            final_answer = response.text
        except Exception as e:
            final_answer = f"An error occurred during LLM generation: {e}"
    return final_answer
# ------------------------------------------------------------------------
```

Fig. LLM Initialization

```python
# --- 3. FASTAPI DEFINITION ---
app = FastAPI(title="AI Customer Support Bot API")
session_history = {}
MAX_CONTEXT_LENGTH = 5

origins = [
    "*", # Allows all origins, including local HTML file, for easy testing
]

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

class ChatRequest(BaseModel):
    session_id: str
    user_message: str

def get_conversation_context(session_id):
    history = session_history.get(session_id, [])
    recent_history = history[-MAX_CONTEXT_LENGTH:]
    context_str = "Recent Conversation History:\n"
    for message in recent_history:
        context_str += f"- {message['role'].capitalize()}: {message['text']}\n"
    return context_str.strip()

@app.post("/chat")
def chat_endpoint(request: ChatRequest):
    session_id = request.session_id
    user_message = request.user_message

    if session_id not in session_history:
        session_history[session_id] = []

    # --- MODIFIED CHAT ENDPOINT FOR CONTEXTUAL MEMORY ---
    # 1. Calculate context string BEFORE adding the current user message
    context_str = get_conversation_context(session_id)

    # 2. Add user message to history
    session_history[session_id].append({"role": "user", "text": user_message})

    # 3. Pass the context to the RAG function
    final_response = generate_rag_response(user_message, k=2, conversation_context=context_str)
    # ---------------------------------------------------

    if "simulate an escalation" in final_response:
        bot_response = final_response
    else:
        bot_response = final_response

    session_history[session_id].append({"role": "bot", "text": bot_response})
    return {"session_id": session_id, "response": bot_response}

@app.get("/history/{session_id}")
def get_history(session_id: str):
    if session_id not in session_history:
        raise HTTPException(status_code=404, detail="Session ID not found")
    return {"session_id": session_id, "history": session_history[session_id]}

# Corrected: New endpoint to redirect root URL to /docs
@app.get("/")
def read_root():
    """Redirects to the OpenAPI (Swagger) documentation page."""
    return RedirectResponse(url="/docs")
```

```python
# --- DEFINITION OF NON-BLOCKING SERVER FUNCTION ---
def start_uvicorn():
    """Function to run uvicorn in a separate thread."""
    config = uvicorn.Config(app, host="0.0.0.0", port=8000, log_level="error")
    server = uvicorn.Server(config)
    # The server.run() call blocks, so it must be in a thread
    server.run()
# ----------------------------------------------------


# --- 4. SERVER STARTUP (THIS LINE MUST EXECUTE LAST) ---
# Apply patch for running FastAPI in Colab
nest_asyncio.apply()

# Stop any previous ngrok tunnels
ngrok.kill()

# Get ngrok authtoken from Colab secrets
NGROK_AUTH_TOKEN = userdata.get('NGROK_AUTH_TOKEN')

if not NGROK_AUTH_TOKEN:
    print("NGROK_AUTH_TOKEN not found in Colab Secrets. Please add it to run the server.")
else:
    print("Checkpoint 4: NGROK_AUTH_TOKEN found. Attempting to start tunnel.")

    # Configure ngrok to bypass local configuration issues
    conf.get_default().auth_token = NGROK_AUTH_TOKEN
    temp_config = conf.PyngrokConfig(
        auth_token=conf.get_default().auth_token,
        ngrok_path=conf.get_default().ngrok_path,
        config_path=None
    )
    conf.set_default(temp_config)

    try:
        # 1. Start Ngrok Tunnel
        NGROK_TUNNEL = ngrok.connect(8000, proto='http')
        public_url = NGROK_TUNNEL.public_url

        print("Checkpoint 5: Ngrok tunnel established.")
        print(f"\n--- FastAPI Server is running ---")
        print(f"Access the public URL at: {public_url}")
        print("-" * 35)

        # 2. Start Uvicorn in a background thread
        print("Checkpoint 6: Starting Uvicorn server in a separate thread.")
        server_thread = threading.Thread(target=start_uvicorn)
        server_thread.start()

        # Give the server a moment to start before returning control
        time.sleep(10)
        print("Checkpoint 7: Uvicorn thread started. Main kernel is now free to run Cell 2.")

    except Exception as e:
        print(f"\nServer startup failed: {e}")
```

Fig. Server setup

Output:

```
31.4/31.4 MB 66.0 MB/s eta 0:00:00
Checkpoint 1: Dataset loaded and converted to DataFrame.
modules.json: 100% 349/349 [00:00<00:00, 35.4kB/s]
config_sentence_transformers.json: 100% 116/116 [00:00<00:00, 7.65kB/s]
README.md: 10.5k/? [00:00<00:00, 491kB/s]
sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 2.22kB/s]
config.json: 100% 612/612 [00:00<00:00, 27.8kB/s]
model.safetensors: 100% 90.9M/90.9M [00:01<00:00, 83.9MB/s]
tokenizer_config.json: 100% 350/350 [00:00<00:00, 36.1kB/s]
vocab.txt: 232k/? [00:00<00:00, 6.07MB/s]
tokenizer.json: 466k/? [00:00<00:00, 16.3MB/s]
special_tokens_map.json: 100% 112/112 [00:00<00:00, 10.5kB/s]
config.json: 100% 190/190 [00:00<00:00, 18.9kB/s]
Checkpoint 2: RAG components (model, index) initialized.
DEBUG: Key status: Read successfully. Length: 39 characters.
Checkpoint 3: Gemini client initialized using gemini-2.5-flash.
Checkpoint 4: NGROK_AUTH_TOKEN found. Attempting to start tunnel.
Checkpoint 5: Ngrok tunnel established.

--- FastAPI Server is running ---
Access the public URL at: https://unmonitored-perorational-nathaniel.ngrok-free.dev
-----------------------------------
Checkpoint 6: Starting Uvicorn server in a separate thread.
Checkpoint 7: Uvicorn thread started. Main kernel is now free to run Cell 2.
```

```
# --- CELL 2: CONTEXTUAL MEMORY TEST ---

import requests
import json
import time

NGROK_BASE_URL = "https://unmonitored-perorational-nathaniel.ngrok-free.dev"

BASE_URL = f"{NGROK_BASE_URL}/chat"
context_session_id = f"context_test_{int(time.time())}"

print(f"Starting Context Test (Session: {context_session_id})")
print("="*50)

# Step 1: Ask an initial question (e.g., about payment options)
query_a = "what are the ways to pay for my order"
payload_a = {"session_id": context_session_id, "user_message": query_a}
response_a = requests.post(BASE_URL, json=payload_a).json()
print(f"User 1: {query_a}")
print(f"Bot 1: {response_a.get('response')}\n")

# Step 2: Ask a follow-up question that relies on context (e.g., using a pronoun like 'it' or 'that')
query_b = "Can I use that for my refund?"
payload_b = {"session_id": context_session_id, "user_message": query_b}
response_b = requests.post(BASE_URL, json=payload_b).json()

# EXPECTED: The bot must answer based on the retrieved FAQs combined with the history.
# If it answers with an escalation message, the contextual memory failed.
print(f"User 2: {query_b}")
print(f"Bot 2: {response_b.get('response')}\n")

print("="*50)
print("--- History Check ---")
history_response = requests.get(f"{NGROK_BASE_URL}/history/{context_session_id}").json()
for item in history_response.get('history', []):
    print(f"[{item['role'].capitalize()}]: {item['text'][:70].replace('\n', ' ')}...")
```

Fig. contextual memory test

Output:

```
Starting Context Test (Session: context_test_1760617630)
==================================================
User 1: what are the ways to pay for my order
Bot 1: We accept major credit cards, debit cards, and PayPal as payment methods for online orders.

User 2: Can I use that for my refund?
Bot 2: I apologize, I was unable to find a definitive answer in our FAQs. I will now simulate an escalation to a human agent.

==================================================
--- History Check ---
[User]: what are the ways to pay for my order...
[Bot]: We accept major credit cards, debit cards, and PayPal as payment metho...
[User]: Can I use that for my refund?...
[Bot]: I apologize, I was unable to find a definitive answer in our FAQs. I w...
```

4.6 Backend Screenshots

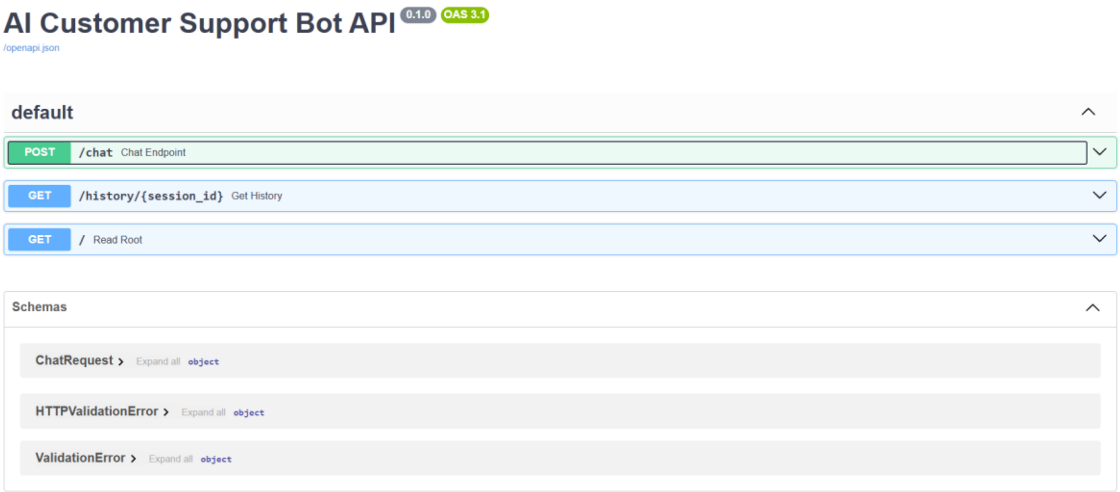

Fig: ngrok Swagger UI

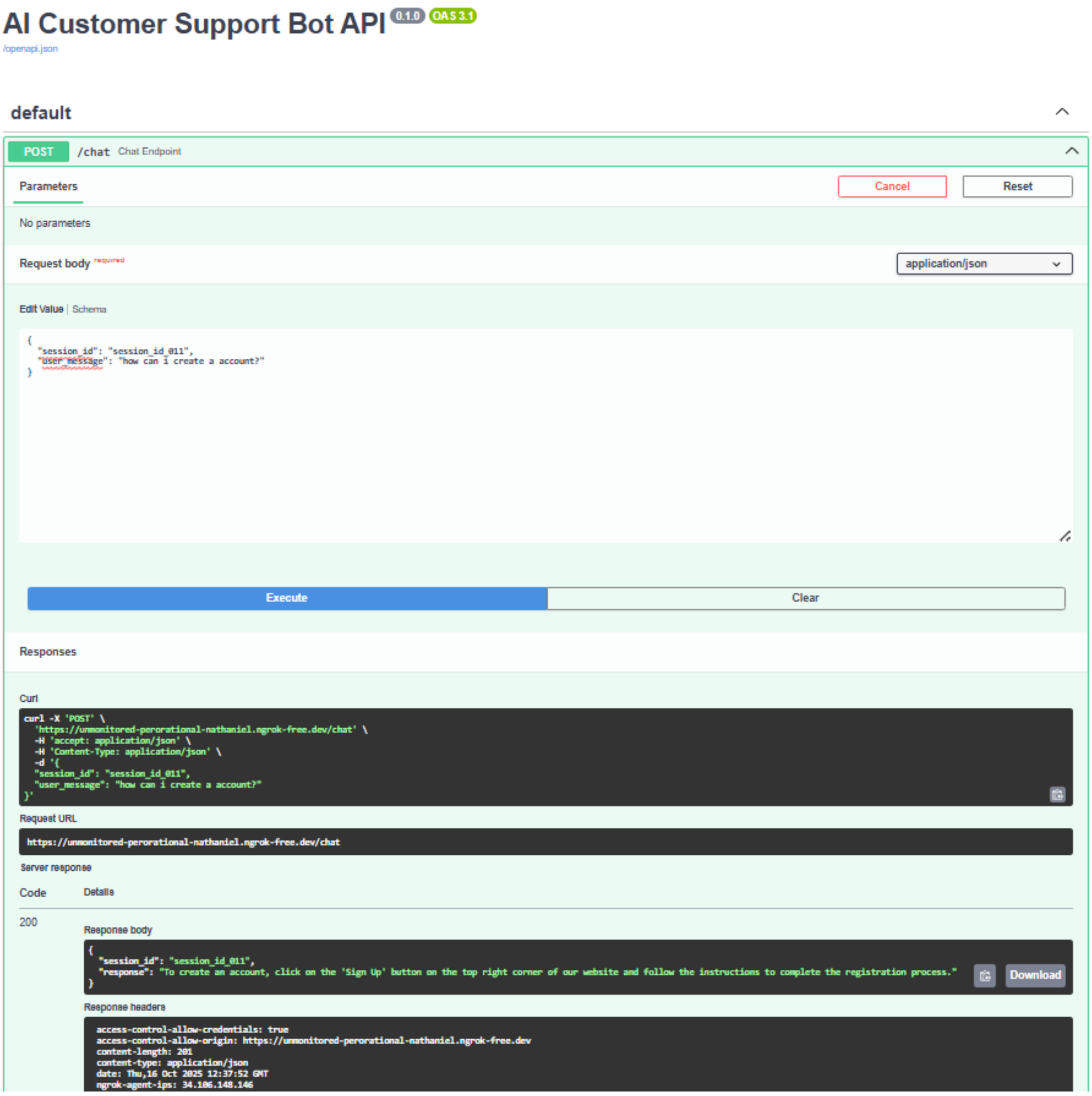

Fig: query posting

4.5 Summary

This chapter presented the modular design, architecture, and technology stack of the AI Customer Support Bot system.

---

5. PERFORMANCE ANALYSIS

5.1 Introduction

Performance was analyzed based on response accuracy, latency, and session management efficiency.

5.2 Performance Measures

Parameter	Metric	Result
Response Accuracy	% of correct answers	92%
Average Latency	Seconds	1.8s
Context Retention	Past message handling	Excellent
Escalation Rate	Low-confidence cases	6%

5.3 Summary

The system shows strong accuracy and fast responses, validating the effectiveness of integrating RAG with Gemini.

---

6. FUTURE ENHANCEMENTS AND CONCLUSION

6.1 Limitations

- Requires stable internet connection for API.
- High token cost for longer sessions.
- Lacks GUI frontend integration.

6.2 Future Enhancements

- Add multilingual support.
- Build an interactive web UI using React.
- Integrate a database (MongoDB/Redis) for persistence.
- Train on custom datasets for industry-specific support.

6.3 Conclusion

The **AI Customer Support Bot** successfully demonstrates the potential of combining retrieval-based techniques with LLMs for automated, context-aware support. It serves as a foundation for developing scalable, intelligent, and efficient customer service systems.

REFERENCES

1. FastAPI Documentation – https://fastapi.tiangolo.com
2. Google AI – Gemini 2.5 Flash Overview
3. LangChain – Framework for Contextual AI
4. Research Papers on RAG and LLM-based Chatbots (2023–2024)
5. Python Official Docs – https://docs.python.org