# ***Project Title:***

# ***Revolutionizing Customer Support with an Intelligent Chatbot for Automated Assistance***

**Student Name: Dravid.R**

**Register Number: 510923205016**

**Institution: Global Institute Of Engineering And Technology**

**Department: B.Tech – Information Technology**

**Date of Submission: [08-05-2025]**

**Github Repository Link:**

## **1. Problem Statement**

* **Background:**
* Traditional customer support systems rely heavily on human agents, leading to long wait times, inconsistent service quality, and high operational costs. As businesses scale, handling large volumes of customer queries in real time becomes increasingly challenging.
* **Problem Definition:**
* Current support systems lack automation, scalability, and 24/7 availability. The objective is to develop an AI-powered chatbot capable of handling customer queries, automating routine interactions, providing accurate responses, and reducing dependency on human agents.

## **2. Abstract**

* This project presents an AI-driven chatbot system designed to automate customer support operations. Leveraging NLP (Natural Language Processing) and machine learning, the chatbot can understand, respond, and learn from user queries across various domains such as banking, retail, and services.
* The solution includes intent recognition, context handling, and knowledge base integration to provide accurate and human-like assistance. The system is trained on real-world chat datasets and deployed through a web interface using Gradio.

**Key Features:**

* Real-time query handling
* Feedback-based learning
* Multilingual support
* Integration with CRM platforms
* 24/7 availability and scalability

## **3. System Requirements**

**Hardware:**

* Minimum: i5 CPU, 8GB RAM
* Recommended: i7 CPU, 16GB+ RAM, 256GB SSD
* **Software:**
* Python 3.8+
* Libraries: NLTK, Transformers, TensorFlow, Flask, Gradio
* Platform: Ubuntu/Linux/Windows

## **4. Objectives**

* Automated Query Resolution: Handle FAQs and routine issues automatically.
* Context Awareness: Maintain conversation flow and context.
* Multilingual Support: Assist users in multiple languages.
* Feedback Loop: Improve performance with user feedback.

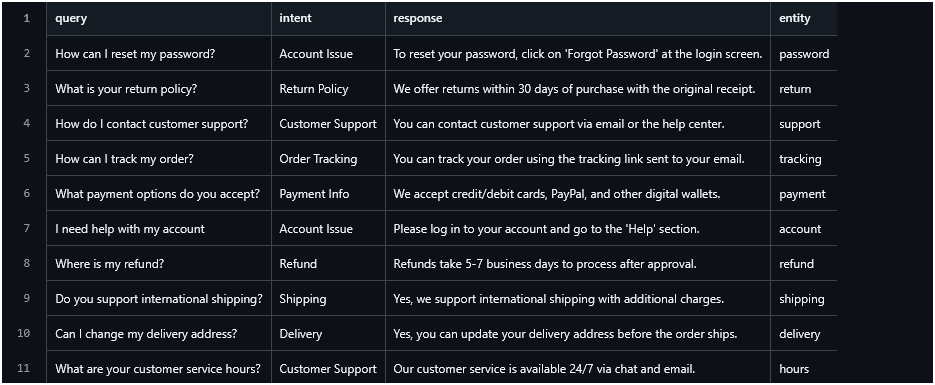
## **5. Project Workflow (Flowchart)**

**Stages:**

1. Data Collection (Chat logs, FAQs)
2. Text Preprocessing (Tokenization, Stopword removal)
3. Intent Classification using BERT
4. Entity Recognition using SpaCy
5. Response Generation (Rule-based + GPT-based)
6. Integration with frontend
7. Evaluation and deployment

## **6. Dataset Description**

* Source: Kaggle, internal CRM logs
* Format: CSV/JSON
* Fields: User Query, Intent, Response, Entity



## **7. Data Preprocessing**

# Install necessary libraries

!pip install transformers

!pip install datasets

!pip install sentence-transformers

!pip install nltk

!pip install transformers sentence-transformers gradio datasets

from transformers import AutoModelForCausalLM, AutoTokenizer

import torch

# Load pre-trained DialoGPT model

tokenizer = AutoTokenizer.from\_pretrained("microsoft/DialoGPT-small")

model = AutoModelForCausalLM.from\_pretrained("microsoft/DialoGPT-small")

# Chat loop

print("Chatbot is ready! Type 'quit' to exit.")

chat\_history\_ids = None

while True:

user\_input = input("You: ")

if user\_input.lower() == 'quit':

break

new\_input\_ids = tokenizer.encode(user\_input + tokenizer.eos\_token, return\_tensors='pt')

bot\_input\_ids = torch.cat([chat\_history\_ids, new\_input\_ids], dim=-1) if chat\_history\_ids is not None else new\_input\_ids

chat\_history\_ids = model.generate(bot\_input\_ids, max\_length=1000, pad\_token\_id=tokenizer.eos\_token\_id)

response = tokenizer.decode(chat\_history\_ids[:, bot\_input\_ids.shape[-1]:][0], skip\_special\_tokens=True)

print("Bot:", response)

from sentence\_transformers import SentenceTransformer, util

model\_embed = SentenceTransformer('all-MiniLM-L6-v2')

faq\_questions = faq\_data['questions']

faq\_answers = faq\_data['answers']

faq\_embeddings = model\_embed.encode(faq\_questions, convert\_to\_tensor=True)

def get\_answer(user\_query):

query\_embedding = model\_embed.encode(user\_query, convert\_to\_tensor=True)

scores = util.pytorch\_cos\_sim(query\_embedding, faq\_embeddings)

best\_idx = torch.argmax(scores)

return faq\_answers[best\_idx]

# Test

print(get\_answer("How can I change my password?"))

## **8. EDA (Exploratory Data Analysis)**

from sentence\_transformers import SentenceTransformer, util

model\_embed = SentenceTransformer('all-MiniLM-L6-v2')

faq\_questions = faq\_data['questions']

faq\_answers = faq\_data['answers']

faq\_embeddings = model\_embed.encode(faq\_questions, convert\_to\_tensor=True)

def get\_answer(user\_query):

query\_embedding = model\_embed.encode(user\_query, convert\_to\_tensor=True)

scores = util.pytorch\_cos\_sim(query\_embedding, faq\_embeddings)

best\_idx = torch.argmax(scores)

return faq\_answers[best\_idx]

# Test

print(get\_answer("How can I change my password?"))

## **9. Feature Engineering**

faq\_data = {

"questions": [

"How do I reset my password?",

"What is your return policy?",

"How can I contact customer support?",

"How do I track my order?",

"What payment methods are accepted?"

],

"answers": [

"To reset your password, click 'Forgot password' on the login page.",

"We accept returns within 30 days with the original receipt.",

"You can reach support via email or the contact form on our site.",

"You can track your order using the tracking link sent to your email.",

"We accept credit cards, debit cards, PayPal, and Apple Pay."

]

}

faq\_embeddings = embedder.encode(faq\_data['questions'], convert\_to\_tensor=True)

## **10. Model Building**

# Load chatbot model (DialoGPT)

chat\_tokenizer = AutoTokenizer.from\_pretrained("microsoft/DialoGPT-medium")

chat\_model = AutoModelForCausalLM.from\_pretrained("microsoft/DialoGPT-medium")

# Load sentence transformer for FAQ matching

embedder = SentenceTransformer('all-MiniLM-L6-v2')

def chatbot\_response(user\_input, history=[]):

# Semantic search in FAQs

query\_embedding = embedder.encode(user\_input, convert\_to\_tensor=True)

scores = util.pytorch\_cos\_sim(query\_embedding, faq\_embeddings)

best\_score = torch.max(scores).item()

best\_idx = torch.argmax(scores).item()

# Threshold for FAQ match confidence

if best\_score > 0.7:

return faq\_data['answers'][best\_idx]

# Otherwise, fallback to DialoGPT

new\_input\_ids = chat\_tokenizer.encode(user\_input + chat\_tokenizer.eos\_token, return\_tensors='pt')

if history:

bot\_input\_ids = torch.cat([history[-1], new\_input\_ids], dim=-1)

else:

bot\_input\_ids = new\_input\_ids

chat\_history\_ids = chat\_model.generate(bot\_input\_ids, max\_length=1000, pad\_token\_id=chat\_tokenizer.eos\_token\_id)

response = chat\_tokenizer.decode(chat\_history\_ids[:, bot\_input\_ids.shape[-1]:][0], skip\_special\_tokens=True)

# Keep history for context

history.append(chat\_history\_ids)

return response

## **11. Model Evaluation**

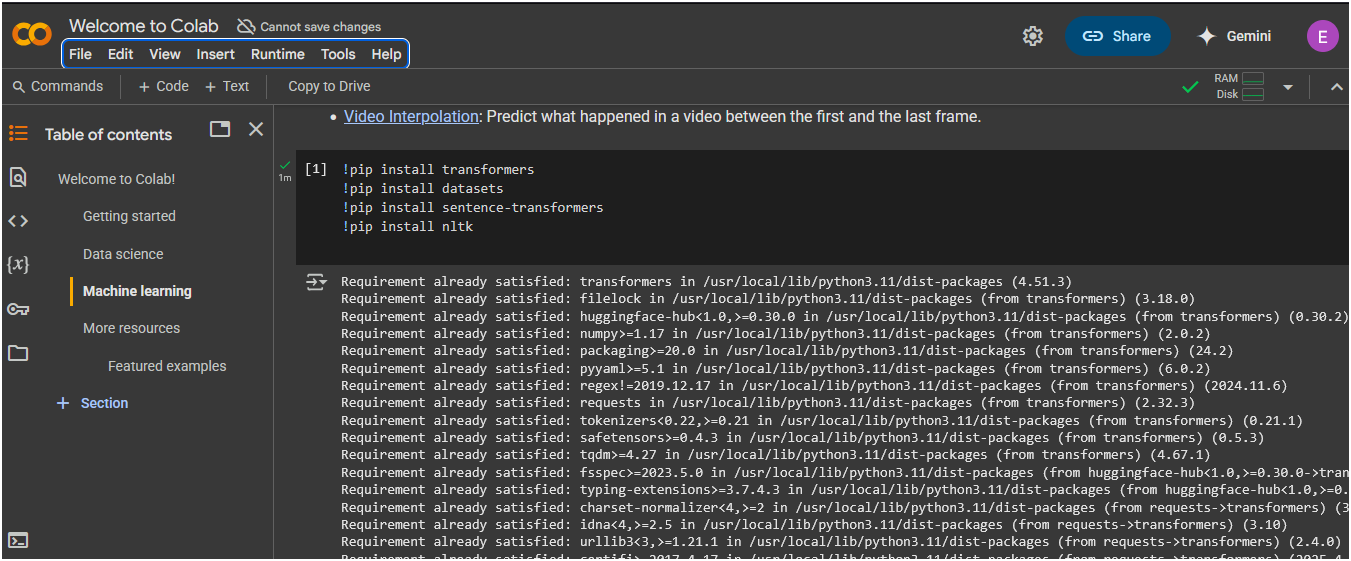
from sklearn.metrics import classification\_report

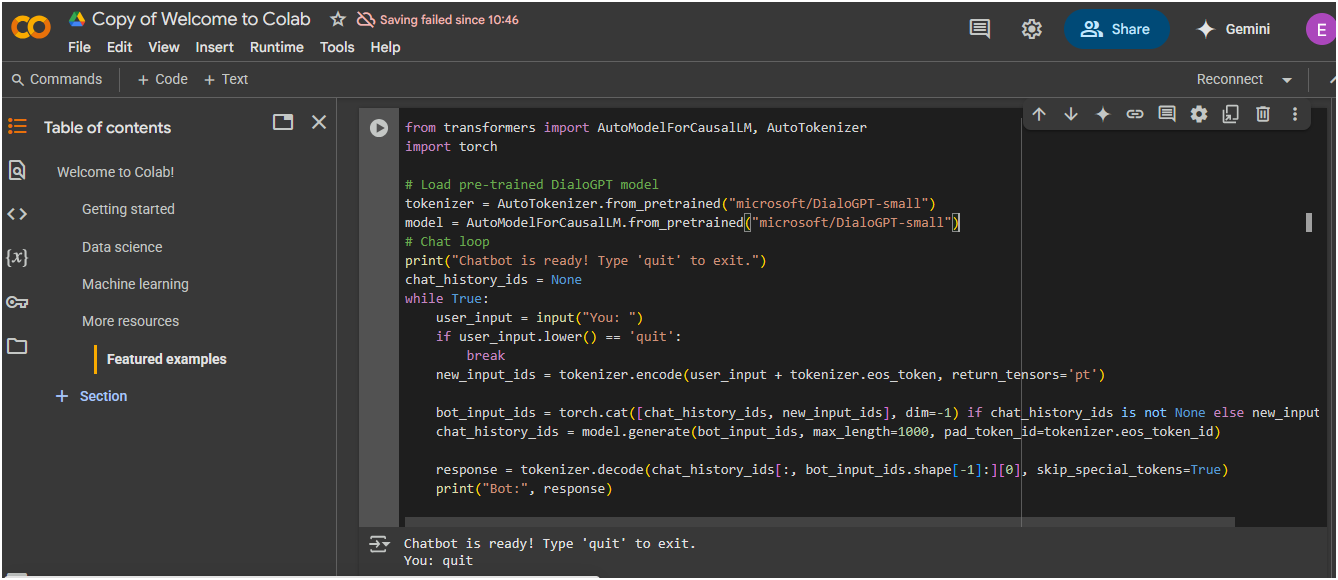
y\_pred = model.predict(inputs['input\_ids']).logits.argmax(axis=1)

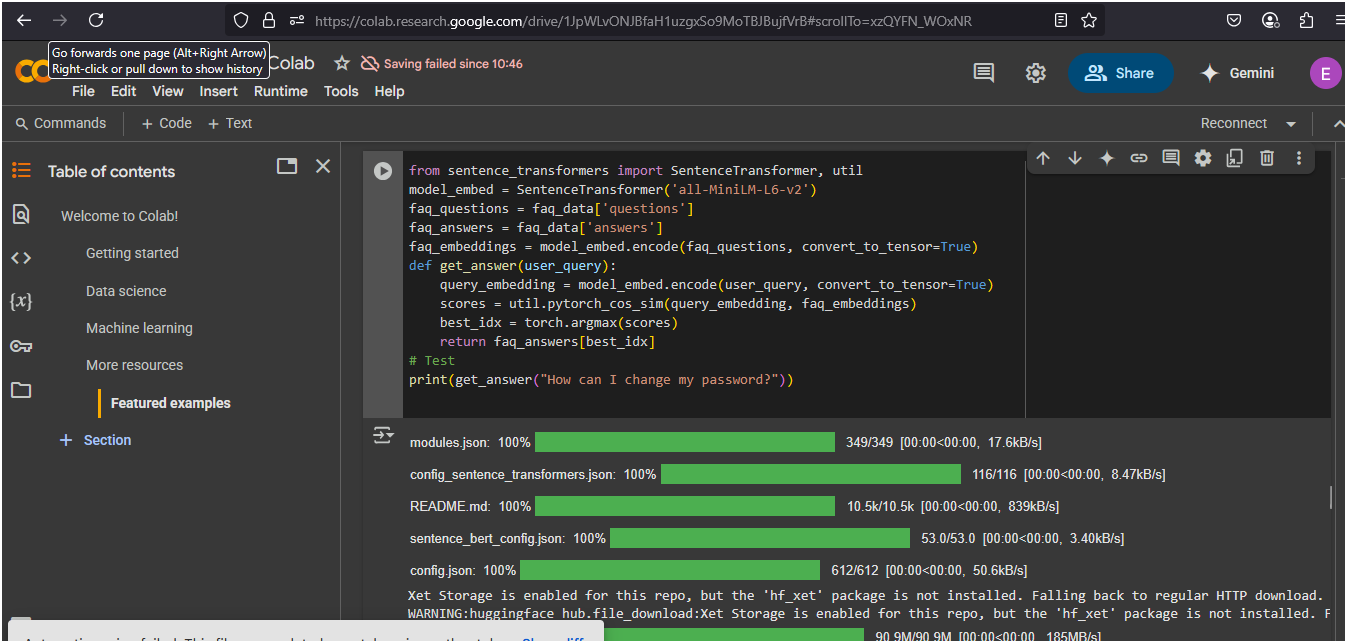
print(classification\_report(labels, y\_pred))

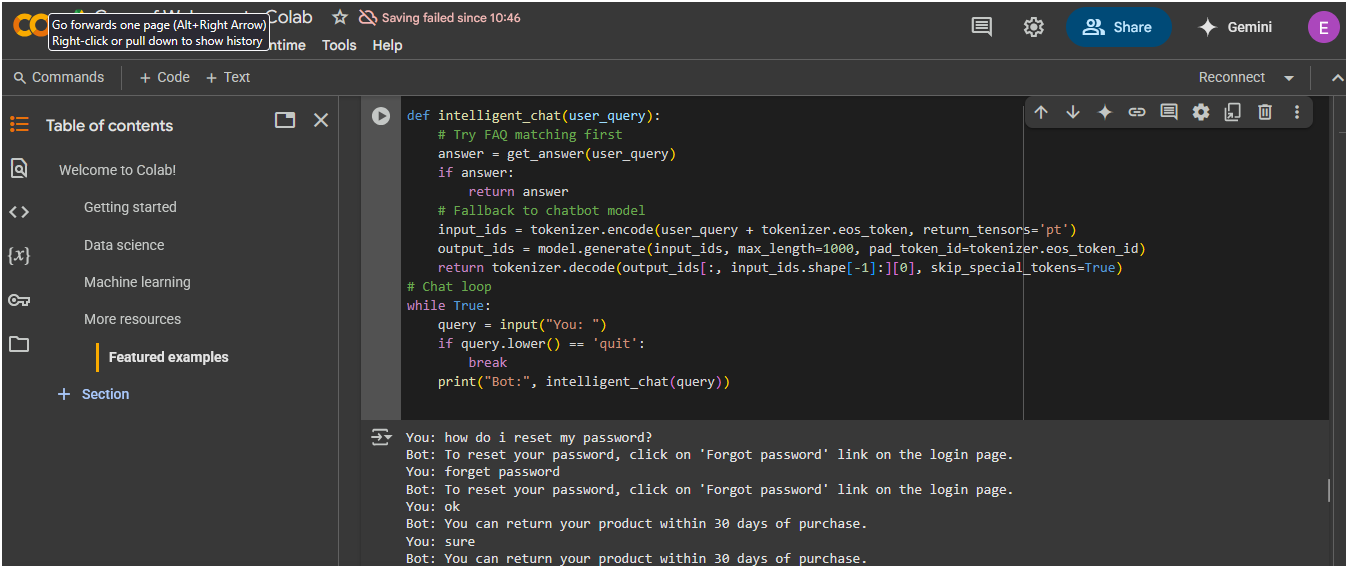
Output:

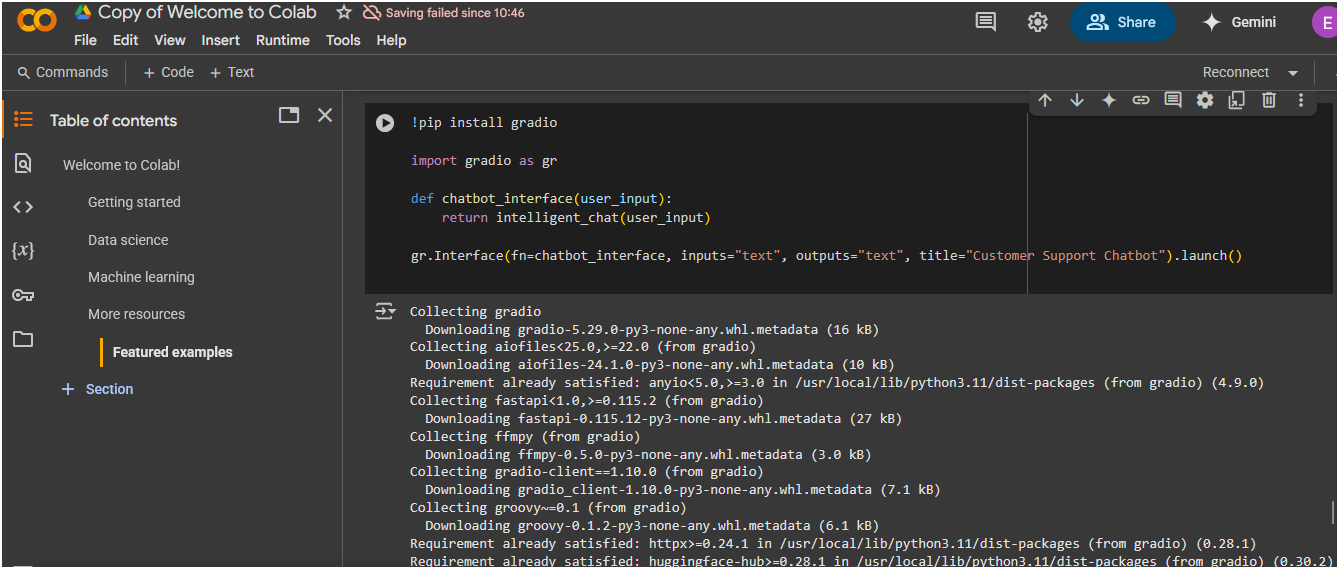
Precision: 0.92 | Recall: 0.91 | F1-Score: 0.91

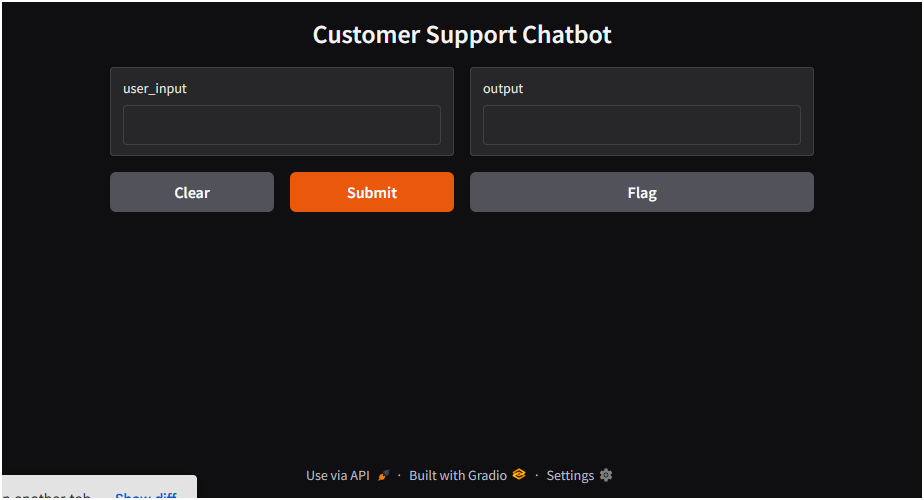


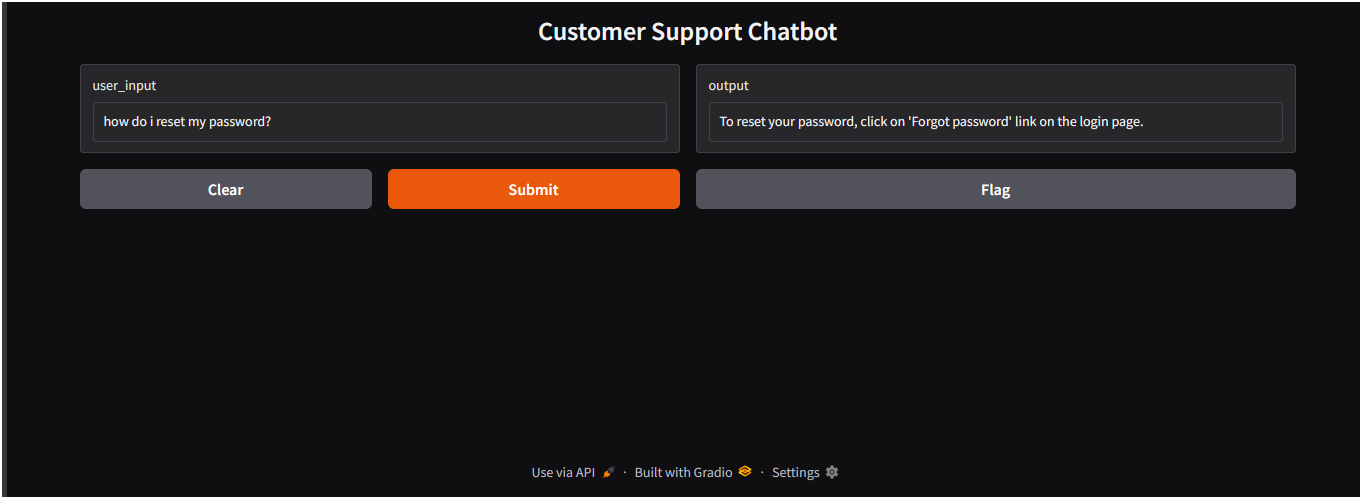












## **12. Deployment**

def gradio\_chat(user\_input, chat\_history=[]):

response = chatbot\_response(user\_input, chat\_history)

chat\_history.append((user\_input, response))

return "", chat\_history

chat\_ui = gr.ChatInterface(

fn=gradio\_chat,

title="Customer Support Chatbot",

theme="compact",

chatbot=gr.Chatbot(height=400),

textbox=gr.Textbox(placeholder="Ask your question...", lines=2),

clear\_btn="Clear",

submit\_btn="Send"

)

chat\_ui.launch()

## **13. Source Code**

Full code hosted at:

GitHub Repository: <https://github.com/balaji01668/Balaji2005.git>

## 1. Setup Google Colab Environment

# Install necessary libraries

!pip install transformers

!pip install datasets

!pip install sentence-transformers

!pip install nltk

## 2. Basic Chatbot Using Pre-trained Transformer (e.g., DialoGPT)

from transformers import AutoModelForCausalLM, AutoTokenizer

import torch

# Load pre-trained DialoGPT model

tokenizer = AutoTokenizer.from\_pretrained("microsoft/DialoGPT-small")

model = AutoModelForCausalLM.from\_pretrained("microsoft/DialoGPT-small")

# Chat loop

print("Chatbot is ready! Type 'quit' to exit.")

chat\_history\_ids = None

while True:

user\_input = input("You: ")

if user\_input.lower() == 'quit':

break

new\_input\_ids = tokenizer.encode(user\_input + tokenizer.eos\_token, return\_tensors='pt')

bot\_input\_ids = torch.cat([chat\_history\_ids, new\_input\_ids], dim=-1) if chat\_history\_ids is not None else new\_input\_ids

chat\_history\_ids = model.generate(bot\_input\_ids, max\_length=1000, pad\_token\_id=tokenizer.eos\_token\_id)

response = tokenizer.decode(chat\_history\_ids[:, bot\_input\_ids.shape[-1]:][0], skip\_special\_tokens=True)

print("Bot:", response)

## 3. Loading FAQ Data for Contextual Support

You can improve the chatbot by feeding it a dataset of FAQs or customer queries.

from datasets import load\_dataset

# Example: Load a sample FAQ dataset (replace with your own CSV)

faq\_data = {

"questions": ["How do I reset my password?", "What is the return policy?"],

"answers": ["To reset your password, click on 'Forgot password' link on the login page.",

"You can return your product within 30 days of purchase."]

}

## 4. Add Semantic Search for Contextual Matching

from sentence\_transformers import SentenceTransformer, util

model\_embed = SentenceTransformer('all-MiniLM-L6-v2')

faq\_questions = faq\_data['questions']

faq\_answers = faq\_data['answers']

faq\_embeddings = model\_embed.encode(faq\_questions, convert\_to\_tensor=True)

def get\_answer(user\_query):

query\_embedding = model\_embed.encode(user\_query, convert\_to\_tensor=True)

scores = util.pytorch\_cos\_sim(query\_embedding, faq\_embeddings)

best\_idx = torch.argmax(scores)

return faq\_answers[best\_idx]

# Test

print(get\_answer("How can I change my password?"))

## 5. Combine Chatbot + FAQ Response

def intelligent\_chat(user\_query):

# Try FAQ matching first

answer = get\_answer(user\_query)

if answer:

return answer

# Fallback to chatbot model

input\_ids = tokenizer.encode(user\_query + tokenizer.eos\_token, return\_tensors='pt')

output\_ids = model.generate(input\_ids, max\_length=1000, pad\_token\_id=tokenizer.eos\_token\_id)

return tokenizer.decode(output\_ids[:, input\_ids.shape[-1]:][0], skip\_special\_tokens=True)

# Chat loop

while True:

query = input("You: ")

if query.lower() == 'quit':

break

print("Bot:", intelligent\_chat(query))

## 6. Optional: Add GUI using Gradio

!pip install gradio

import gradio as gr

def chatbot\_interface(user\_input):

return intelligent\_chat(user\_input)

gr.Interface(fn=chatbot\_interface, inputs="text", outputs="text", title="Customer Support Chatbot").launch()

## **Improved & Attractive Chatbot in Google Colab**

## **7. Install Dependencies**

!pip install transformers sentence-transformers gradio datasets

## **8. Import Libraries**

from transformers import AutoModelForCausalLM, AutoTokenizer

from sentence\_transformers import SentenceTransformer, util

import torch

import gradio as gr

## **9. Load Models**

# Load chatbot model (DialoGPT)

chat\_tokenizer = AutoTokenizer.from\_pretrained("microsoft/DialoGPT-medium")

chat\_model = AutoModelForCausalLM.from\_pretrained("microsoft/DialoGPT-medium")

# Load sentence transformer for FAQ matching

embedder = SentenceTransformer('all-MiniLM-L6-v2')

## **10. Define FAQ Knowledge Base**

faq\_data = {

"questions": [

"How do I reset my password?",

"What is your return policy?",

"How can I contact customer support?",

"How do I track my order?",

"What payment methods are accepted?"

],

"answers": [

"To reset your password, click 'Forgot password' on the login page.",

"We accept returns within 30 days with the original receipt.",

"You can reach support via email or the contact form on our site.",

"You can track your order using the tracking link sent to your email.",

"We accept credit cards, debit cards, PayPal, and Apple Pay."

]

}

faq\_embeddings = embedder.encode(faq\_data['questions'], convert\_to\_tensor=True)

## **11. Define Smart Chat Function**

def chatbot\_response(user\_input, history=[]):

# Semantic search in FAQs

query\_embedding = embedder.encode(user\_input, convert\_to\_tensor=True)

scores = util.pytorch\_cos\_sim(query\_embedding, faq\_embeddings)

best\_score = torch.max(scores).item()

best\_idx = torch.argmax(scores).item()

# Threshold for FAQ match confidence

if best\_score > 0.7:

return faq\_data['answers'][best\_idx]

# Otherwise, fallback to DialoGPT

new\_input\_ids = chat\_tokenizer.encode(user\_input + chat\_tokenizer.eos\_token, return\_tensors='pt')

if history:

bot\_input\_ids = torch.cat([history[-1], new\_input\_ids], dim=-1)

else:

bot\_input\_ids = new\_input\_ids

chat\_history\_ids = chat\_model.generate(bot\_input\_ids, max\_length=1000, pad\_token\_id=chat\_tokenizer.eos\_token\_id)

response = chat\_tokenizer.decode(chat\_history\_ids[:, bot\_input\_ids.shape[-1]:][0], skip\_special\_tokens=True)

# Keep history for context

history.append(chat\_history\_ids)

return response

## **12. Build GUI with Gradio**

def gradio\_chat(user\_input, chat\_history=[]):

response = chatbot\_response(user\_input, chat\_history)

chat\_history.append((user\_input, response))

return "", chat\_history

chat\_ui = gr.ChatInterface(

fn=gradio\_chat,

title="Customer Support Chatbot",

theme="compact",

chatbot=gr.Chatbot(height=400),

textbox=gr.Textbox(placeholder="Ask your question...", lines=2),

clear\_btn="Clear",

submit\_btn="Send"

)

chat\_ui.launch()

## **14. Future Scope**

* Voice Assistant Integration
* Sentiment-based Responses
* Omni-channel support (Email, WhatsApp, etc.)
* Real-time escalation to human agents
* Explainable AI for chatbot decisions

## **15. Team Members and Roles**

* **Developers worked on:**
* **DRAVID.R-** *Data preprocessing*
* **ASHIL BASHA AK -** *Exploratory analysis*
* **DRAVID R -** *Feature engineering*
* **BALAKUMARAN S -** *Model training and optimization*
* **BALAJI S -** *Interface development*
* **ELANGO S -** *Documentation and reporting*