Project Title:

Revolutionizing Customer Support with an Intelligent Chatbot for Automated Assistance

Student Name: ASHIL BASHA AK

Register Number: 510923205006

Institution: Global Institute Of Engineering And Technology

Department: B.Tech – Information Technology

Date of Submission: [08-05-2025]

Github Repository Link: : https://github.com/Ashilbasha/phase-2.git

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)

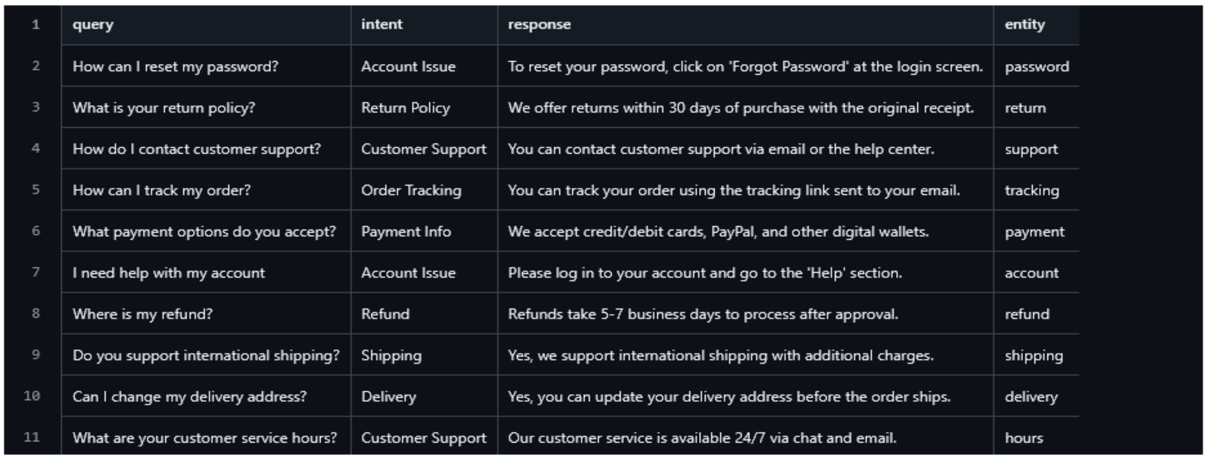
Stages:

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



query	intent	response	entity
How can I reset my password?	Account Issue	To reset your password, click on 'Forgot Password' at the login screen.	password
What is your return policy?	Return Policy	We offer returns within 30 days of purchase with the original receipt.	return
How do I contact customer support?	Customer Support	You can contact customer support via email or the help center.	support
How can I track my order?	Order Tracking	You can track your order using the tracking link sent to your email.	tracking
What payment options do you accept?	Payment Info	We accept credit/debit cards, PayPal, and other digital wallets.	payment
I need help with my account	Account Issue	Please log in to your account and go to the 'Help' section.	account
Where is my refund?	Refund	Refunds take 5-7 business days to process after approval.	refund
Do you support international shipping?	Shipping	Yes, we support international shipping with additional charges.	shipping
Can I change my delivery address?	Delivery	Yes, you can update your delivery address before the order ships.	delivery
What are your customer service hours?	Customer Support	Our customer service is available 24/7 via chat and email.	hours

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
```

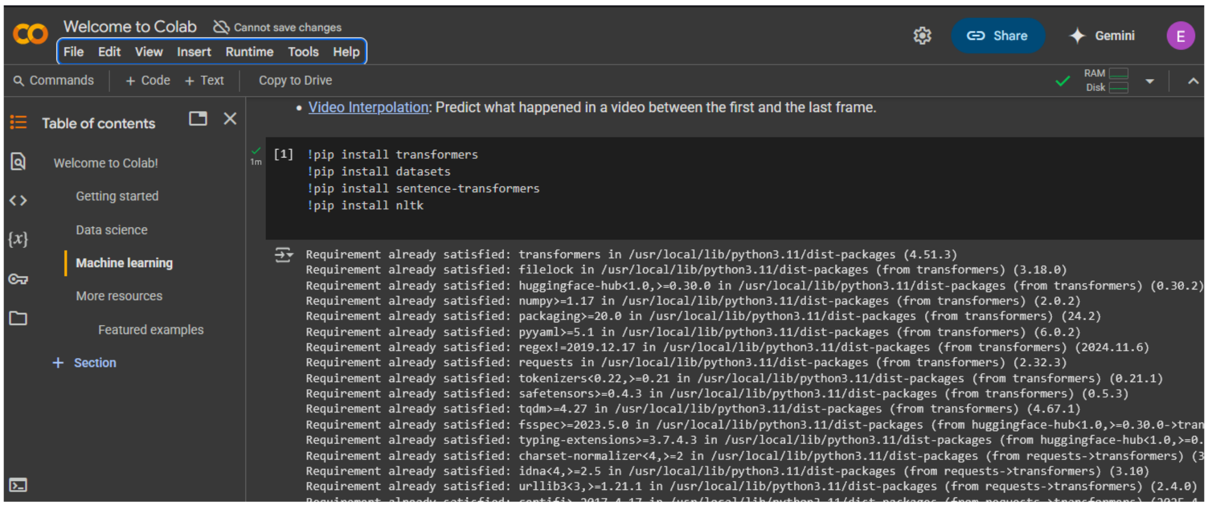
11. Model Evaluation

```
from sklearn.metrics import classification_report
```

```
y_pred = model.predict(inputs['input_ids']).logits.argmax(axis=1)

print(classification_report(labels, y_pred))
```

Output:

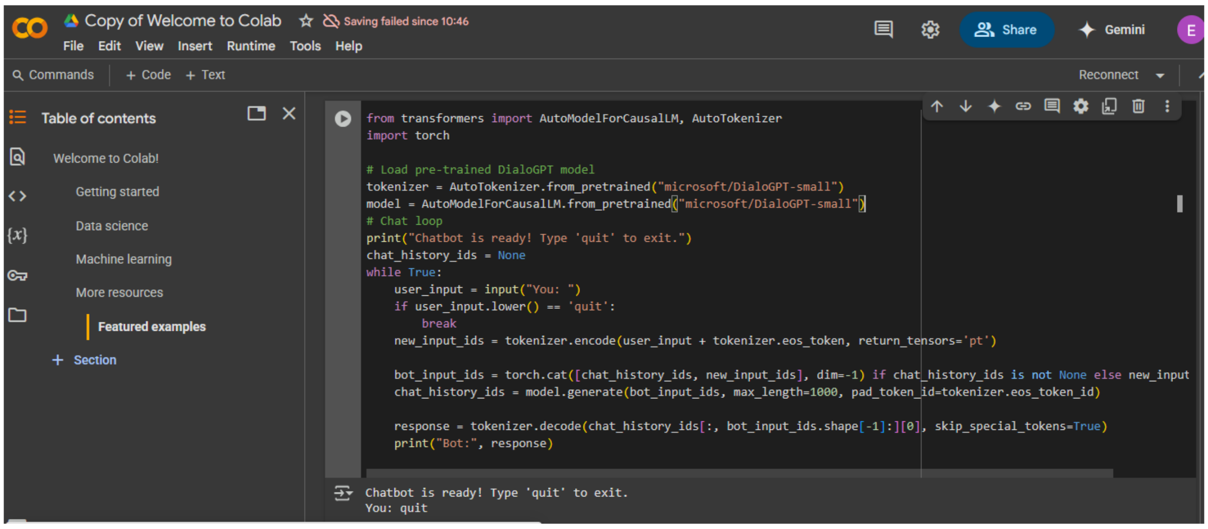
Precision: 0.92 | Recall: 0.91 | F1-Score: 0.91



• Video Interpolation: Predict what happened in a video between the first and the last frame.

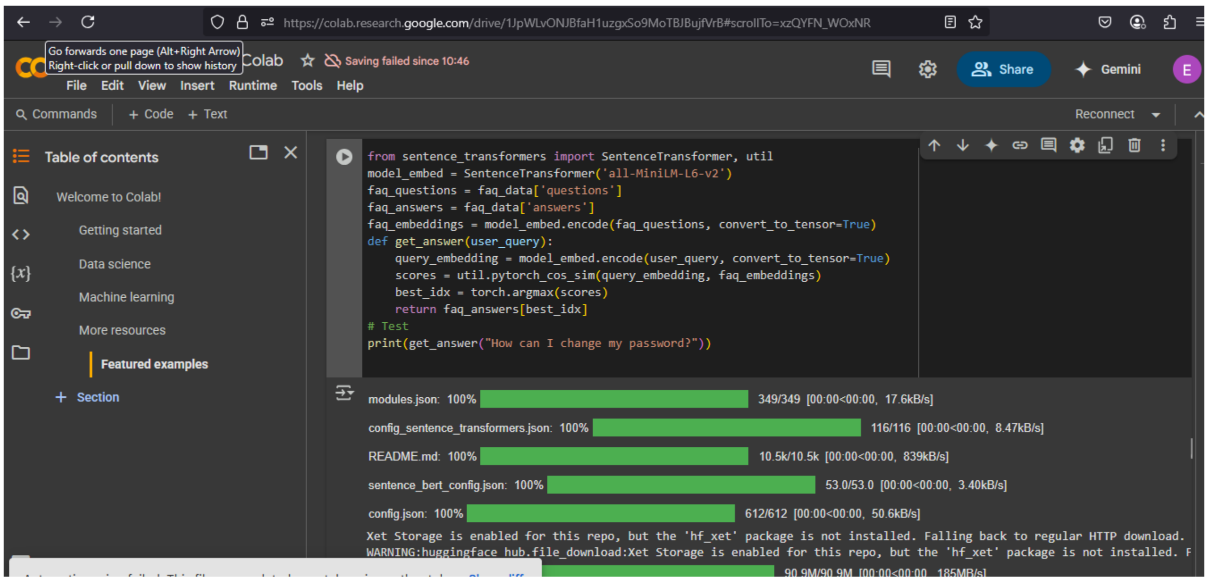
```
[1] !pip install transformers
    !pip install datasets
    !pip install sentence-transformers
    !pip install nltk
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.51.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.30.2)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0->tran
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.4.0)
```



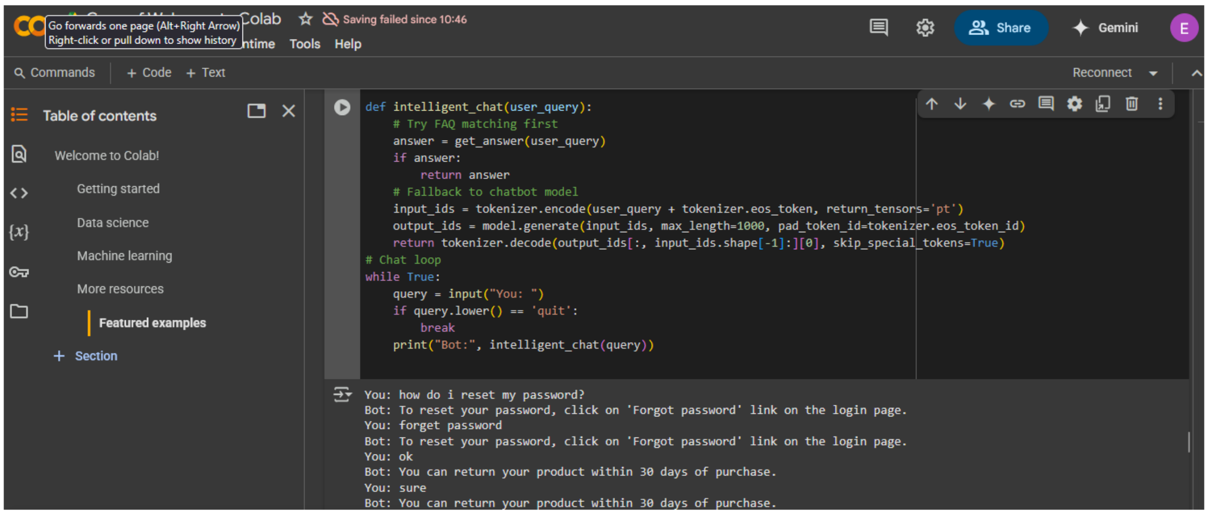
```
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

# 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
    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)
```

```
Chatbot is ready! Type 'quit' to exit.
You: quit
```



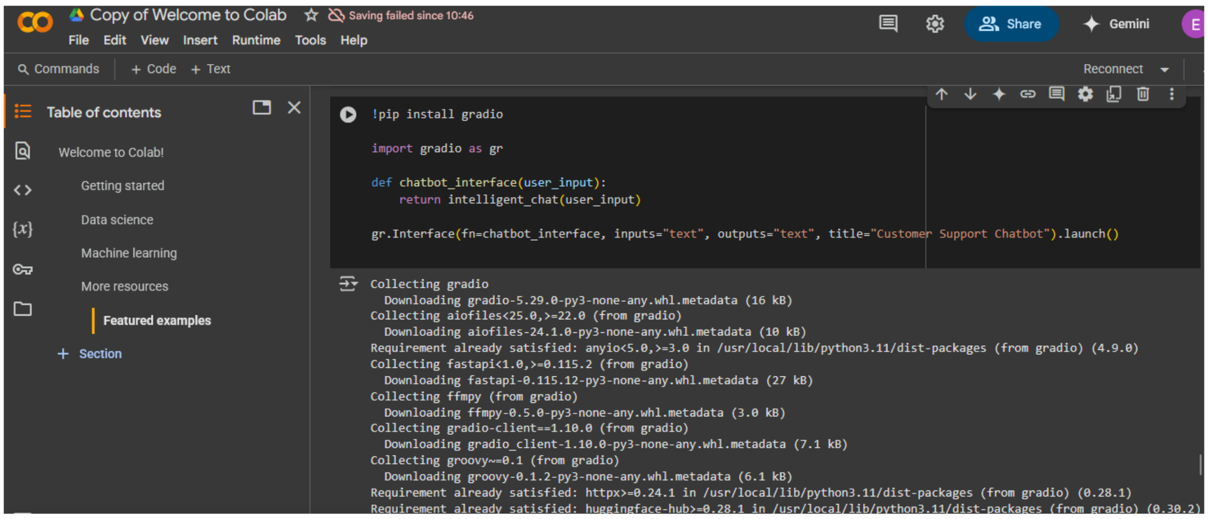
```python
from sentence_transformers import SentenceTransformer, util
model_embed = SentenceTransformer('all-MiniLM-L6-v2')
faq_questions = faq_data['questions']
faq_answers = faq_data['answers']
faq_embeddings = model_embed.encode(faq_questions, convert_to_tensor=True)
def get_answer(user_query):
    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?"))
```

```
modules.json: 100%  349/349 [00:00<00:00, 17.6kB/s]
config_sentence_transformers.json: 100%  116/116 [00:00<00:00, 8.47kB/s]
README.md: 100%  10.5k/10.5k [00:00<00:00, 839kB/s]
sentence_bert_config.json: 100%  53.0/53.0 [00:00<00:00, 3.40kB/s]
config.json: 100%  612/612 [00:00<00:00, 50.6kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download.
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. F
  90.9M/90.9M [00:00<00:00, 185MB/s]
```



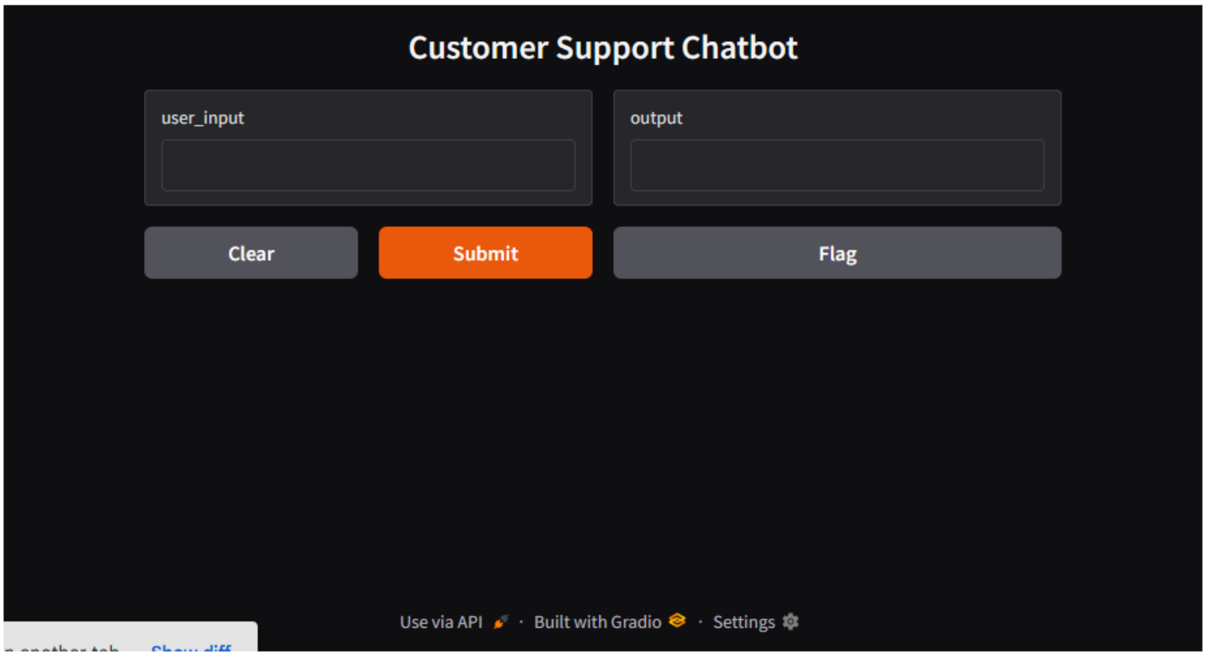
```python
def intelligent_chat(user_query):
    # Try FAQ matching first
    answer = get_answer(user_query)
    if answer:
        return answer
    # Fallback to chatbot model
    input_ids = tokenizer.encode(user_query + tokenizer.eos_token, return_tensors='pt')
    output_ids = model.generate(input_ids, 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))
```

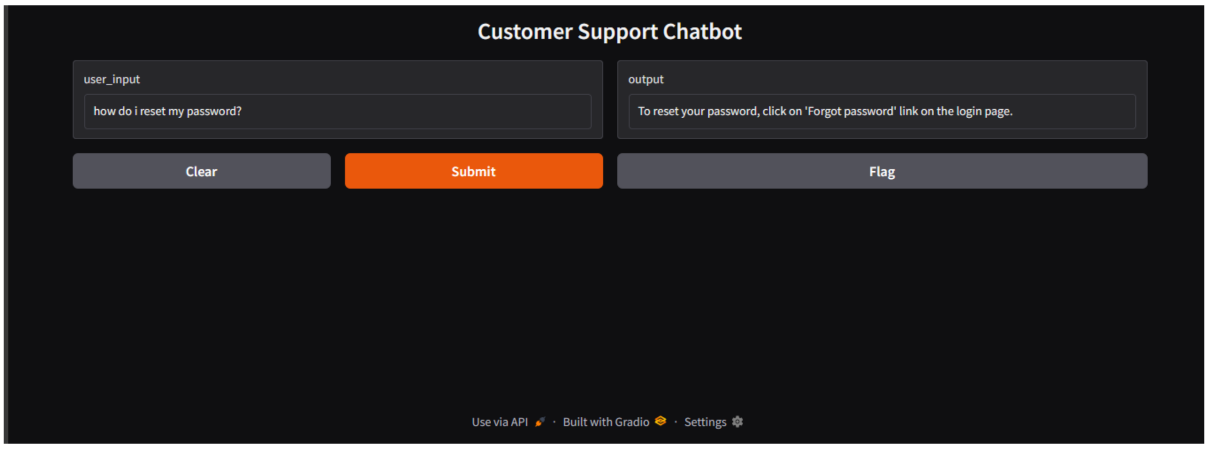
```
You: how do i reset my password?
Bot: To reset your password, click on 'Forgot password' link on the login page.
You: forget password
Bot: To reset your password, click on 'Forgot password' link on the login page.
You: ok
Bot: You can return your product within 30 days of purchase.
You: sure
Bot: You can return your product within 30 days of purchase.
```



```python
!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()
```

```
Collecting gradio
  Downloading gradio-5.29.0-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<25.0,>=22.0 (from gradio)
  Downloading aiofiles-24.1.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.9.0)
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.12-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.5.0-py3-none-any.whl.metadata (3.0 kB)
Collecting gradio-client==1.10.0 (from gradio)
  Downloading gradio_client-1.10.0-py3-none-any.whl.metadata (7.1 kB)
Collecting groovy~=0.1 (from gradio)
  Downloading groovy-0.1.2-py3-none-any.whl.metadata (6.1 kB)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.28.1)
Requirement already satisfied: huggingface-hub>=0.28.1 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.30.2)
```





user_input: how do i reset my password?

output: To reset your password, click on 'Forgot password' link on the login page.

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/Ashilbasha/phase-2.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:**
  - **ELANGO S -** *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*