**Step 1: small dataset with sentence transformer and faiss cpu**

# Install dependencies (run in terminal first if not installed)  
# pip install sentence-transformers faiss-cpu  
  
from sentence\_transformers import SentenceTransformer  
import faiss  
import numpy as np  
  
# ----------------------  
# Step 1: Create Banking FAQs  
# ----------------------  
faqs = [  
 "How can I reset my online banking password?",  
 "How do I check my account balance?",  
 "What should I do if my debit card is lost?",  
 "How can I apply for a personal loan?",  
 "How do I activate international transactions on my credit card?"  
]  
  
answers = {  
 faqs[0]: "You can reset your password by clicking 'Forgot Password' on the login page and following the verification steps.",  
 faqs[1]: "You can check your balance using the mobile app, internet banking, or by visiting an ATM.",  
 faqs[2]: "Report the lost debit card immediately through the customer service helpline or the banking app.",  
 faqs[3]: "You can apply for a personal loan online through the bank’s portal or by visiting your nearest branch.",  
 faqs[4]: "International transactions can be activated via the mobile banking app or by contacting customer care."  
}  
  
# ----------------------  
# Step 2: Load Embedding Model  
# ----------------------  
model = SentenceTransformer('all-MiniLM-L6-v2')  
# lightweight & good for semantic search  
  
# Encode FAQs into embeddings  
faq\_embeddings = model.encode(faqs)  
  
# ----------------------  
# Step 3: Create FAISS Index  
# ----------------------  
dimension = faq\_embeddings.shape[1] # embedding size  
index = faiss.IndexFlatL2(dimension) # L2 distance index  
index.add(np.array(faq\_embeddings))  
  
# ----------------------  
# Step 4: User Query  
# ----------------------  
user\_query = "who are you?"  
query\_embedding = model.encode([user\_query])  
  
# ----------------------  
# Step 5: Search Closest FAQ  
# ----------------------  
# We want the top 1 most similar FAQ  
number\_of\_results = 1  
  
# Search the FAISS index: it returns both distances and indices  
# - distances = how far the query is from each match  
# - indices = the position of the matching FAQ in our 'faqs' list  
distances, indices = index.search(np.array(query\_embedding), number\_of\_results)  
  
# Extract the first (and only) match index  
first\_match\_index = indices[0][0]  
  
# Find the matching FAQ question using the index  
matched\_faq = faqs[first\_match\_index]  
  
# Now retrieve the prepared answer for that FAQ  
matched\_answer = answers[matched\_faq]  
  
# Show results  
print("User Question:", user\_query)  
print("Matched FAQ:", matched\_faq)  
print("Answer:", matched\_answer)

**Step 2: ChromaDb + Sentence Transformers + Large Dataset (openAi/Gemini)**

# rag\_faq\_bot.py  
  
# Install before running:  
# pip install chromadb sentence-transformers openai  
  
import chromadb  
from sentence\_transformers import SentenceTransformer  
from openai import OpenAI  
  
# -----------------------  
# Step 1: FAQ dataset  
# -----------------------  
faqs = [  
 "How can I reset my online banking password?",  
 "How do I check my account balance?",  
 "What should I do if my debit card is lost?",  
 "How do I activate international transactions on my credit card?",  
 "How can I open a new savings account?",  
 "What is the minimum balance required?",  
 "How do I update my registered mobile number?",  
 "How can I apply for a home loan?",  
 "What is the process for closing my bank account?",  
 "How do I check my loan EMI schedule?",  
 "How can I download my account statement?",  
 "What is the daily withdrawal limit from an ATM?",  
 "How do I enable UPI payments?",  
 "Can I increase my credit card limit?",  
 "What is the process to block a stolen credit card?",  
 "How can I register for mobile banking?",  
 "How do I apply for a personal loan?",  
 "What is the penalty for not maintaining minimum balance?",  
 "How can I dispute a wrong transaction?",  
 "What are the bank's working hours?"  
]  
  
# -----------------------  
# Step 2: Setup Vector DB  
# -----------------------  
chroma\_client = chromadb.Client()  
collection = chroma\_client.create\_collection("banking\_faqs")  
  
# Load embeddings model  
model = SentenceTransformer('all-MiniLM-L6-v2')  
  
# Generate embeddings for all FAQs  
embeddings = model.encode(faqs)  
  
# Add FAQs into ChromaDB  
collection.add(  
 documents=faqs,  
 embeddings=embeddings.tolist(),  
 ids=[str(i) for i in range(len(faqs))]  
)  
  
# -----------------------  
# Step 3: Retrieval + RAG  
# -----------------------  
  
def rag\_answer(user\_query: str):  
 # Encode user query  
 query\_embedding = model.encode([user\_query])  
  
 # Search top 3 relevant FAQs  
 results = collection.query(  
 query\_embeddings=query\_embedding.tolist(),  
 n\_results=3  
 )  
  
 # Extract retrieved FAQs  
 retrieved\_faqs = results['documents'][0]  
  
 # Build context  
 context = "\n".join(retrieved\_faqs)  
  
 # Build prompt for LLM  
 prompt = f"""  
 You are a helpful banking assistant.  
 Here are some relevant FAQs from the knowledge base:  
 {context}  
  
 User question: {user\_query}  
 Answer conversationally using the FAQs above.  
 """  
  
 # Call OpenAI LLM  
 client = OpenAI() # Make sure OPENAI\_API\_KEY is set in your environment  
 response = client.chat.completions.create(  
 model="gpt-3.5-turbo",  
 messages=[{"role": "user", "content": prompt}]  
 )  
  
 return response.choices[0].message.content  
  
  
# -----------------------  
# Step 4: Run example  
# -----------------------  
if \_\_name\_\_ == "\_\_main\_\_":  
 user\_query = "I forgot my online banking password. What should I do?"  
 answer = rag\_answer(user\_query)  
 print("User Query:", user\_query)  
 print("AI Answer:", answer)

**Step 3: Same with Flask (use Postman for adding the query)**

# app.py  
from flask import Flask, request, jsonify  
from sentence\_transformers import SentenceTransformer  
import faiss  
import numpy as np  
  
# ----------------------  
# Step 1: Create Banking FAQs  
# ----------------------  
faqs = [  
 "How can I reset my online banking password?",  
 "How do I check my account balance?",  
 "What should I do if my debit card is lost?",  
 "How can I apply for a personal loan?",  
 "How do I activate international transactions on my credit card?"  
]  
  
answers = {  
 faqs[0]: "You can reset your password by clicking 'Forgot Password' on the login page and following the verification steps.",  
 faqs[1]: "You can check your balance using the mobile app, internet banking, or by visiting an ATM.",  
 faqs[2]: "Report the lost debit card immediately through the customer service helpline or the banking app.",  
 faqs[3]: "You can apply for a personal loan online through the bank’s portal or by visiting your nearest branch.",  
 faqs[4]: "International transactions can be activated via the mobile banking app or by contacting customer care."  
}  
  
# ----------------------  
# Step 2: Load Embedding Model  
# ----------------------  
model = SentenceTransformer('all-MiniLM-L6-v2')  
faq\_embeddings = model.encode(faqs)  
  
# ----------------------  
# Step 3: Create FAISS Index  
# ----------------------  
dimension = faq\_embeddings.shape[1]  
index = faiss.IndexFlatL2(dimension)  
index.add(np.array(faq\_embeddings))  
  
# ----------------------  
# Step 4: Initialize Flask  
# ----------------------  
app = Flask(\_\_name\_\_)  
  
@app.route('/')  
def home():  
 return "Welcome to the Banking FAQ Semantic Search API!"  
  
@app.route('/ask', methods=['POST'])  
def ask\_question():  
 try:  
 data = request.json  
 user\_query = data.get('question', '')  
 if not user\_query:  
 return jsonify({"error": "Please provide a question in JSON format: {'question': 'your question here'}"}), 400  
  
 # Encode user query  
 query\_embedding = model.encode([user\_query])  
  
 # Search FAISS index  
 distances, indices = index.search(np.array(query\_embedding), 1)  
 first\_match\_index = indices[0][0]  
 matched\_faq = faqs[first\_match\_index]  
 matched\_answer = answers[matched\_faq]  
  
 return jsonify({  
 "user\_question": user\_query,  
 "matched\_faq": matched\_faq,  
 "answer": matched\_answer,  
 "distance": float(distances[0][0])  
 })  
  
 except Exception as e:  
 return jsonify({"error": str(e)}), 500  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app.run(debug=True)