```
!pip install -q streamlit
!npm install -q localtunnel
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.3/44.3 kB 1.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.9/9.9 MB 42.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.9/6.9 MB 62.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.1/79.1 kB 4.1 MB/s eta 0:00:00
⠙⠹⠸⠼⠴⠦⠧⠇⠏⠋⠙⠹⠸⠼⠴⠦⠧⠇⠏⠋⠙⠹⠸⠼
added 22 packages in 4s
⠼
⠼3 packages are looking for funding
⠼  run `npm fund` for details
⠼
```

```
!apt-get install nodejs
```

```
Get:4 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 libnode72 amd64 12.22.9~dfsg-1ubuntu3.6 [10.8 MB]
Get:5 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 nodejs-doc all 12.22.9~dfsg-1ubuntu3.6 [2,411 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 nodejs amd64 12.22.9~dfsg-1ubuntu3.6 [122 kB]
Fetched 13.7 MB in 1s (10.8 MB/s)
Selecting previously unselected package javascript-common.
(Reading database ... 126102 files and directories currently installed.)
Preparing to unpack .../0-javascript-common_11+nmu1_all.deb ...
Unpacking javascript-common (11+nmu1) ...
Selecting previously unselected package libjs-highlight.js.
Preparing to unpack .../1-libjs-highlight.js_9.18.5+dfsg1-1_all.deb ...
Unpacking libjs-highlight.js (9.18.5+dfsg1-1) ...
Selecting previously unselected package libc-ares2:amd64.
Preparing to unpack .../2-libc-ares2_1.18.1-1ubuntu0.22.04.3_amd64.deb ...
Unpacking libc-ares2:amd64 (1.18.1-1ubuntu0.22.04.3) ...
Selecting previously unselected package libnode72:amd64.
Preparing to unpack .../3-libnode72_12.22.9~dfsg-1ubuntu3.6_amd64.deb ...
Unpacking libnode72:amd64 (12.22.9~dfsg-1ubuntu3.6) ...
Selecting previously unselected package nodejs-doc.
Preparing to unpack .../4-nodejs-doc_12.22.9~dfsg-1ubuntu3.6_all.deb ...
Unpacking nodejs-doc (12.22.9~dfsg-1ubuntu3.6) ...
Selecting previously unselected package nodejs.
Preparing to unpack .../5-nodejs_12.22.9~dfsg-1ubuntu3.6_amd64.deb ...
Unpacking nodejs (12.22.9~dfsg-1ubuntu3.6) ...
Setting up javascript-common (11+nmu1) ...
Setting up libc-ares2:amd64 (1.18.1-1ubuntu0.22.04.3) ...
Setting up libnode72:amd64 (12.22.9~dfsg-1ubuntu3.6) ...
Setting up libjs-highlight.js (9.18.5+dfsg1-1) ...
Setting up nodejs (12.22.9~dfsg-1ubuntu3.6) ...
update-alternatives: using /usr/bin/nodejs to provide /usr/bin/js (js) in auto mode
Setting up nodejs-doc (12.22.9~dfsg-1ubuntu3.6) ...
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for libc-bin (2.35-0ubuntu3.8) ...
/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_0.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libumf.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libur_adapter_opencl.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libhwloc.so.15 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libur_adapter_level_zero.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc.so.2 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtcm_debug.so.1 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtcm.so.1 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libur_loader.so.0 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbbind_2_5.so.3 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbbmalloc_proxy.so.2 is not a symbolic link

/sbin/ldconfig.real: /usr/local/lib/libtbb.so.12 is not a symbolic link
```

```
%%writefile app.py
import streamlit as st
from transformers import pipeline

# Load your fine-tuned model
qa_pipeline = pipeline("text-generation", model="./models/fine_tuned_model")
```

```
def ask_question(question):
    prompt = f"Question: {question}\nAnswer:"
    response = qa_pipeline(prompt, max_length=100, do_sample=True)[0]['generated_text']
    return response.split("Answer:")[-1].strip()

st.title("🩺 Medical QA Chatbot")
user_q = st.text_input("Enter your medical question:")

if user_q:
    with st.spinner("Thinking..."):
        answer = ask_question(user_q)
        st.markdown(f"**Answer:** {answer}")
```

```
Writing app.py
```

```
!streamlit run app.py &>/content/logs.txt &
```

```
pip install openai sentence-transformers pandas scikit-learn
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 71.4 MB/s eta 0:00:00
Downloading nvidia_cuda_runtime_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (883 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 37.6 MB/s eta 0:00:00
Downloading nvidia_cudnn_cu12-9.1.0.70-py3-none-manylinux2014_x86_64.whl (664.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.2 MB/s eta 0:00:00
Downloading nvidia_cufft_cu12-11.2.1.3-py3-none-manylinux2014_x86_64.whl (211.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 6.4 MB/s eta 0:00:00
Downloading nvidia_curand_cu12-10.3.5.147-py3-none-manylinux2014_x86_64.whl (56.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 13.9 MB/s eta 0:00:00
Downloading nvidia_cusolver_cu12-11.6.1.9-py3-none-manylinux2014_x86_64.whl (127.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 9.1 MB/s eta 0:00:00
Downloading nvidia_cusparse_cu12-12.3.1.170-py3-none-manylinux2014_x86_64.whl (207.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 6.7 MB/s eta 0:00:00
Downloading nvidia_nvjitlink_cu12-12.4.127-py3-none-manylinux2014_x86_64.whl (21.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 92.6 MB/s eta 0:00:00
Installing collected packages: nvidia-nvjitlink-cu12, nvidia-curand-cu12, nvidia-cufft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu
  Attempting uninstall: nvidia-nvjitlink-cu12
    Found existing installation: nvidia-nvjitlink-cu12 12.5.82
    Uninstalling nvidia-nvjitlink-cu12-12.5.82:
      Successfully uninstalled nvidia-nvjitlink-cu12-12.5.82
  Attempting uninstall: nvidia-curand-cu12
    Found existing installation: nvidia-curand-cu12 10.3.6.82
    Uninstalling nvidia-curand-cu12-10.3.6.82:
      Successfully uninstalled nvidia-curand-cu12-10.3.6.82
  Attempting uninstall: nvidia-cufft-cu12
    Found existing installation: nvidia-cufft-cu12 11.2.3.61
    Uninstalling nvidia-cufft-cu12-11.2.3.61:
      Successfully uninstalled nvidia-cufft-cu12-11.2.3.61
  Attempting uninstall: nvidia-cuda-runtime-cu12
    Found existing installation: nvidia-cuda-runtime-cu12 12.5.82
    Uninstalling nvidia-cuda-runtime-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-runtime-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-nvrtc-cu12
    Found existing installation: nvidia-cuda-nvrtc-cu12 12.5.82
    Uninstalling nvidia-cuda-nvrtc-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-nvrtc-cu12-12.5.82
  Attempting uninstall: nvidia-cuda-cupti-cu12
    Found existing installation: nvidia-cuda-cupti-cu12 12.5.82
    Uninstalling nvidia-cuda-cupti-cu12-12.5.82:
      Successfully uninstalled nvidia-cuda-cupti-cu12-12.5.82
  Attempting uninstall: nvidia-cublas-cu12
    Found existing installation: nvidia-cublas-cu12 12.5.3.2
    Uninstalling nvidia-cublas-cu12-12.5.3.2:
      Successfully uninstalled nvidia-cublas-cu12-12.5.3.2
  Attempting uninstall: nvidia-cusparse-cu12
    Found existing installation: nvidia-cusparse-cu12 12.5.1.3
    Uninstalling nvidia-cusparse-cu12-12.5.1.3:
      Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
  Attempting uninstall: nvidia-cudnn-cu12
    Found existing installation: nvidia-cudnn-cu12 9.3.0.75
    Uninstalling nvidia-cudnn-cu12-9.3.0.75:
      Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
  Attempting uninstall: nvidia-cusolver-cu12
    Found existing installation: nvidia-cusolver-cu12 11.6.3.83
    Uninstalling nvidia-cusolver-cu12-11.6.3.83:
      Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
Successfully installed nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-runtime-cu12
```

```
import openai
import pandas as pd
import numpy as np
```

```
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity

# Configure OpenAI API
openai.api_key = "your_openai_api_key"

# Load data and model
def load_medical_qa_database(path='/content/medical_qa_large.csv'):
    return pd.read_csv(path)

def embed(text_list, model):
    return model.encode(text_list)

def generate_embeddings_for_database(questions, model):
    return embed(questions, model)

def find_closest_match(user_embedding, question_embeddings, answers):
    similarities = cosine_similarity([user_embedding], question_embeddings)
    best_index = np.argmax(similarities)
    return best_index, answers[best_index]

def generate_response(user_input, context):
    prompt = f"Context: {context}\nUser: {user_input}\nAnswer:"

    response = openai.ChatCompletion.create(
        model="gpt-4",
        messages=[{"role": "user", "content": prompt}],
        temperature=0.5
    )
    return response['choices'][0]['message']['content'].strip()

def MedicalQAChatbot():
    print("Initializing Medical QA Chatbot...")
    df = load_medical_qa_database()
    questions = df['question'].tolist()
    answers = df['answer'].tolist()

    model = SentenceTransformer('all-MiniLM-L6-v2')
    question_embeddings = generate_embeddings_for_database(questions, model)

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

    while True:
        user_input = input("You: ")
        if user_input.lower() == 'exit':
            break

        user_embedding = embed([user_input], model)[0]
        index, best_answer = find_closest_match(user_embedding, question_embeddings, answers)
        final_response = generate_response(user_input, best_answer)

        print("\nBot:", final_response)
        print("Note: This is not a substitute for professional medical advice.\n")

if __name__ == "__main__":
    MedicalQAChatbot()
```

```
Initializing Medical QA Chatbot...
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in y
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json: 100%                                                  349/349 [00:00<00:00, 18.0kB/s]
config_sentence_transformers.json: 100%                             116/116 [00:00<00:00, 8.74kB/s]
README.md: 100%                                                     10.5k/10.5k [00:00<00:00, 623kB/s]
sentence_bert_config.json: 100%                                     53.0/53.0 [00:00<00:00, 4.46kB/s]
config.json: 100%                                                   612/612 [00:00<00:00, 34.0kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better performance
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular
model.safetensors: 100%                                             90.9M/90.9M [00:01<00:00, 100MB/s]
tokenizer_config.json: 100%                                         350/350 [00:00<00:00, 29.4kB/s]
vocab.txt: 100%                                                     232k/232k [00:00<00:00, 6.71MB/s]
tokenizer.json: 100%                                                466k/466k [00:00<00:00, 7.13MB/s]
special_tokens_map.json: 100%                                       112/112 [00:00<00:00, 6.28kB/s]
config.json: 100%                                                   190/190 [00:00<00:00, 8.41kB/s]
Medical QA Chatbot is ready. Type 'exit' to quit.
You: exit
```

```python
import tkinter as tk
from tkinter import messagebox, ttk
import pandas as pd
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
import openai

# OpenAI API key
openai.api_key = "your-api-key-here"

# Load model
embedding_model = SentenceTransformer('all-MiniLM-L6-v2')

# Load CSV data
def load_medical_qa_database(file_path):
    df = pd.read_csv(file_path)
    return df

# Generate embeddings
def generate_embeddings(questions):
    return embedding_model.encode(questions)

# Find closest match
def find_closest_match(user_embedding, question_embeddings, questions, answers):
    similarities = cosine_similarity([user_embedding], question_embeddings)[0]
    best_index = similarities.argmax()
    return questions[best_index], answers[best_index], similarities[best_index]

# Generate GPT response
def generate_response(user_question, matched_answer):
    prompt = f"""You are a helpful medical assistant.
Context: {matched_answer}
User's Question: {user_question}
Answer the user's question based on the context above."""

    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content": prompt}],
        temperature=0.5,
        max_tokens=150
    )
    return response.choices[0].message['content'].strip()

# GUI app
def launch_gui():
    df = load_medical_qa_database("medical_qa_large.csv")
    questions = df["question"].tolist()
    answers = df["answer"].tolist()
    question_embeddings = generate_embeddings(questions)
```

```
    def on_ask():
        user_input = entry.get()
        if not user_input.strip():
            messagebox.showwarning("Input Error", "Please enter a question.")
            return

        user_embedding = embedding_model.encode([user_input])[0]
        matched_q, matched_a, similarity = find_closest_match(user_embedding, question_embeddings, questions, answers)
        response = generate_response(user_input, matched_a)

        result_text.config(state='normal')
        result_text.delete(1.0, tk.END)
        result_text.insert(tk.END, f"Answer: {response}\n\n(Matched Question: {matched_q}\nSimilarity: {similarity:.2f})\n")
        result_text.insert(tk.END, "\nNote: This is not a substitute for professional medical advice.")
        result_text.config(state='disabled')

    # Create GUI window
    root = tk.Tk()
    root.title("Medical QA Chatbot")
    root.geometry("600x400")

    tk.Label(root, text="Ask a medical question:", font=("Arial", 12)).pack(pady=10)
    entry = tk.Entry(root, width=70)
    entry.pack(pady=5)

    ask_button = tk.Button(root, text="Get Answer", command=on_ask)
    ask_button.pack(pady=10)

    result_text = tk.Text(root, wrap=tk.WORD, height=10, width=70, state='disabled')
    result_text.pack(pady=10)

    root.mainloop()

if __name__ == "__main__":
    launch_gui()
```

```
---------------------------------------------------------------------------
TclError                                  Traceback (most recent call last)
<ipython-input-5-0bf094fa2d39> in <cell line: 0>()
     83
     84 if __name__ == "__main__":
---> 85     launch_gui()

 1 frames
/usr/lib/python3.11/tkinter/__init__.py in __init__(self, screenName, baseName, className, useTk, sync, use)
   2324                 baseName = baseName + ext
   2325         interactive = False
-> 2326         self.tk = _tkinter.create(screenName, baseName, className, interactive, wantobjects, useTk, sync, use)
   2327         if useTk:
   2328             self._loadtk()

TclError: no display name and no $DISPLAY environment variable
```

Next steps: Explain error

```
import tkinter as tk
from tkinter import messagebox, ttk
import pandas as pd
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
import openai

# OpenAI API key
openai.api_key = "your-api-key-here"  # Replace with your actual key

# Load model
embedding_model = SentenceTransformer('all-MiniLM-L6-v2')
```


```
import tkinter as tk
from tkinter import messagebox, ttk
import pandas as pd
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity
import openai

# OpenAI API key
openai.api_key = "your-api-key-here"  # Replace with your actual key
```


```
# Load model
```

```
embedding_model = SentenceTransformer('all-MiniLM-L6-v2')

# Load CSV data
def load_medical_qa_database(file_path):
    df = pd.read_csv(file_path)
    return df

# Generate embeddings
def generate_embeddings(questions):
    return embedding_model.encode(questions)

# Find closest match
def find_closest_match(user_embedding, question_embeddings, questions, answers):
    similarities = cosine_similarity([user_embedding], question_embeddings)[0]
    best_index = similarities.argmax()
    return questions[best_index], answers[best_index], similarities[best_index]

# Generate GPT response
def generate_response(user_question, matched_answer):
    prompt = f"""You are a helpful medical assistant.
Context: {matched_answer}
User's Question: {user_question}
Answer the user's question based on the context above."""

    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content": prompt}],
        temperature=0.5,
        max_tokens=150
    )
    return response.choices[0].message['content'].strip()

# GUI app
def launch_gui():
    df = load_medical_qa_database("/content/medical_qa_large.csv")
    questions = df["question"].tolist()
    answers = df["answer"].tolist()
    question_embeddings = generate_embeddings(questions)

    def on_ask():
        user_input = entry.get()
        if not user_input.strip():
            messagebox.showwarning("Input Error", "Please enter a question.")
            return

        user_embedding = embedding_model.encode([user_input])[0]
        matched_q, matched_a, similarity = find_closest_match(user_embedding, question_embeddings, questions, answers)
        response = generate_response(user_input, matched_a)

        result_text.config(state='normal')
        result_text.delete(1.0, tk.END)
        result_text.insert(tk.END, f"Answer: {response}\n\n(Matched Question: {matched_q}\nSimilarity: {similarity:.2f})\n")
        result_text.insert(tk.END, "\nNote: This is not a substitute for professional medical advice.")
        result_text.config(state='disabled')

    # Create GUI window
    root = tk.Tk()
    root.title("Medical QA Chatbot")
    root.geometry("600x400")

    tk.Label(root, text="Ask a medical question:", font=("Arial", 12)).pack(pady=10)
    entry = tk.Entry(root, width=70)
    entry.pack(pady=5)

    ask_button = tk.Button(root, text="Get Answer", command=on_ask)
    ask_button.pack(pady=10)

    result_text = tk.Text(root, wrap=tk.WORD, height=10, width=70, state='disabled')
    result_text.pack(pady=10)

    root.mainloop()

# if __name__ == "__main__":
    # This line is commented out to prevent the GUI from launching
    # in an environment without a display.
    # launch_gui()

# Generate embeddings
def generate_embeddings(questions):
    return embedding_model.encode(questions)
```

```
# Find closest match
def find_closest_match(user_embedding, question_embeddings, questions, answers):
    similarities = cosine_similarity([user_embedding], question_embeddings)[0]
    best_index = similarities.argmax()
    return questions[best_index], answers[best_index], similarities[best_index]

# Generate GPT response
def generate_response(user_question, matched_answer):
    prompt = f"""You are a helpful medical assistant.
Context: {matched_answer}
User's Question: {user_question}
Answer the user's question based on the context above."""

    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content": prompt}],
        temperature=0.5,
        max_tokens=150
    )
    return response.choices[0].message['content'].strip()

# GUI app
def launch_gui():
    df = load_medical_qa_database("medical_qa_large.csv")
    questions = df["question"].tolist()
    answers = df["answer"].tolist()
    question_embeddings = generate_embeddings(questions)

    def on_ask():
        user_input = entry.get()
        if not user_input.strip():
            messagebox.showwarning("Input Error", "Please enter a question.")
            return

        user_embedding = embedding_model.encode([user_input])[0]
        matched_q, matched_a, similarity = find_closest_match(user_embedding, question_embeddings, questions, answers)
        response = generate_response(user_input, matched_a)

        result_text.config(state='normal')
        result_text.delete(1.0, tk.END)
        result_text.insert(tk.END, f"Answer: {response}\n\n(Matched Question: {matched_q}\nSimilarity: {similarity:.2f})\n")
        result_text.insert(tk.END, "\nNote: This is not a substitute for professional medical advice.")
        result_text.config(state='disabled')

    # Create GUI window
    root = tk.Tk()
    root.title("Medical QA Chatbot")
    root.geometry("600x400")

    tk.Label(root, text="Ask a medical question:", font=("Arial", 12)).pack(pady=10)
    entry = tk.Entry(root, width=70)
    entry.pack(pady=5)

    ask_button = tk.Button(root, text="Get Answer", command=on_ask)
    ask_button.pack(pady=10)

    result_text = tk.Text(root, wrap=tk.WORD, height=10, width=70, state='disabled')
    result_text.pack(pady=10)

    root.mainloop()

# if __name__ == "__main__":
    # This line is commented out to prevent the GUI from launching
    # in an environment without a display.
    # launch_gui()


!python medical_chatbot_gui.py
```

```
python3: can't open file '/content/medical_chatbot_gui.py': [Errno 2] No such file or directory
```

```
# ACCURACY

# Mock function to simulate chatbot response (replace this with your chatbot call)
def medical_qa_chatbot(question):
    # Here, call your actual chatbot API or function
    # For example, call API or local model inference
    # return chatbot_api.get_answer(question)

    # Mock responses for demonstration:
```

```
    sample_responses = {
        "What are the symptoms of diabetes?": "Common symptoms include increased thirst, frequent urination, hunger, fatigue, and blurred vision.",
        "How do you treat hypertension?": "Treatment includes lifestyle changes and medications like ACE inhibitors or beta-blockers.",
        "What is the normal body temperature?": "The normal body temperature is around 98.6°F (37°C).",
    }
    return sample_responses.get(question, "Sorry, I don't know the answer to that.")

# Test dataset: list of (question, expected_answer)
test_cases = [
    ("What are the symptoms of diabetes?", "Common symptoms include increased thirst, frequent urination, hunger, fatigue, and blurred vision."),
    ("How do you treat hypertension?", "Treatment includes lifestyle changes and medications like ACE inhibitors or beta-blockers."),
    ("What is the normal body temperature?", "The normal body temperature is around 98.6°F (37°C)."),
]

def evaluate_accuracy(test_cases):
    correct = 0
    for question, expected_answer in test_cases:
        response = medical_qa_chatbot(question)
        print(f"Q: {question}")
        print(f"Expected: {expected_answer}")
        print(f"Response: {response}")
        print()
        # Simple exact match check (can be improved with fuzzy matching)
        if response.strip().lower() == expected_answer.strip().lower():
            correct += 1
    accuracy = correct / len(test_cases) * 100
    print(f"Accuracy: {accuracy:.2f}%")

if __name__ == "__main__":
    evaluate_accuracy(test_cases)
```

```
Q: What are the symptoms of diabetes?
Expected: Common symptoms include increased thirst, frequent urination, hunger, fatigue, and blurred vision.
Response: Common symptoms include increased thirst, frequent urination, hunger, fatigue, and blurred vision.

Q: How do you treat hypertension?
Expected: Treatment includes lifestyle changes and medications like ACE inhibitors or beta-blockers.
Response: Treatment includes lifestyle changes and medications like ACE inhibitors or beta-blockers.

Q: What is the normal body temperature?
Expected: The normal body temperature is around 98.6°F (37°C).
Response: The normal body temperature is around 98.6°F (37°C).

Accuracy: 100.00%
```