```python
 !pip install sentence-transformers ipywidgets --quiet

import pandas as pd
from sentence_transformers import SentenceTransformer, util
import ipywidgets as widgets
from IPython.display import display, clear_output, HTML
import torch

# ----------------------------
# 1. Load dataset
# ----------------------------
csv_path = "/physiotherapy_dataset.csv"
df = pd.read_csv(csv_path)
df['search_text'] = df['patient_query'].astype(str) + " " +
df['pain_severity'].astype(str)

# ----------------------------
# 2. Load embedding model
# ----------------------------
embedder = SentenceTransformer('all-MiniLM-L6-v2')
corpus_embeddings = embedder.encode(df['search_text'].tolist(),
convert_to_tensor=True)

# ----------------------------
# 3. Initialize chat memory
# ----------------------------
chat_history = []

# ----------------------------
# 4. Function to get recommendation
# ----------------------------
def get_recommendation(body_part, severity, threshold=0.6):
    query_text = f"{body_part} {severity}"
    query_embedding = embedder.encode([query_text], convert_to_tensor=True)

    if len(corpus_embeddings) == 0:
        return None, False

    scores = util.pytorch_cos_sim(query_embedding, corpus_embeddings)[0]
    best_score, best_match_id = torch.max(scores, dim=0)
    best_match_id = int(best_match_id.item())

    if best_score.item() >= threshold:
        return df.iloc[best_match_id]['recommended_exercise'], True
    else:
        return None, False

# ----------------------------
# 5. Create UI widgets
# ----------------------------
```

```
body_input = widgets.Text(description="Body Part:", placeholder="e.g., neck,
back")
severity_input = widgets.Text(description="Pain Severity:", placeholder="mild,
moderate, severe")
recommend_btn = widgets.Button(description="Get Recommendation",
button_style='info')
history_btn = widgets.Button(description="Show Chat History",
button_style='warning')
expert_input = widgets.Text(description="Expert Exercise:", placeholder="Type
exercise if unknown")
expert_submit_btn = widgets.Button(description="Submit Exercise",
button_style='success')

# Output box with fixed height, vertical scroll
output_box = widgets.HTML(layout=widgets.Layout(width='80%', height='400px',
overflow_y='auto', border='1px solid gray', padding='10px'))

waiting_for_expert = {"flag": False, "body": None, "severity": None}

# ----------------------------
# 6. Function to render chat with auto-scroll
# ----------------------------
def render_chat():
    chat_html = ""
    for b, s, r, user_flag in chat_history:
        if user_flag:
            chat_html += f"<div style='text-align:right; margin:5px;'><span
style='background-color:#a2d5a2; padding:8px; border-radius:10px; display:inline-
block;'>{b} | {s}</span></div>"
        else:
            chat_html += f"<div style='text-align:left; margin:5px;'><span
style='background-color:#add8e6; padding:8px; border-radius:10px; display:inline-
block;'>{r}</span></div>"
    # Add auto-scroll script
    chat_html += """
    <script>
    var objDiv = this.parentNode;
    var outputDiv = document.querySelector('div.widget-html-output');
    if(outputDiv){
        outputDiv.scrollTop = outputDiv.scrollHeight;
    }
    </script>
    """
    output_box.value = chat_html

# ----------------------------
# 7. Recommendation button action
# ----------------------------
# ----------------------------
# 4. Function to get recommendation (Corrected)
```

```python
# ----------------------------
def get_recommendation(body_part, severity, threshold=0.35):
    query_text = f"{body_part.strip().lower()} {severity.strip().lower()}"
    query_embedding = embedder.encode([query_text], convert_to_tensor=True)

    if len(corpus_embeddings) == 0:
        return None, False

    # Compute cosine similarity
    scores = util.cos_sim(query_embedding, corpus_embeddings)[0]

    # Get best match
    best_score, best_match_id = torch.max(scores, dim=0)
    best_match_id = int(best_match_id.item())
    best_score_val = best_score.item()

    # Debug print (optional)
    # print("Best match score:", best_score_val, "| Query:", query_text)

    if best_score_val >= threshold:
        return df.iloc[best_match_id]['recommended_exercise'], True
    else:
        return None, False


# ----------------------------
# 8. Expert submit action
# ----------------------------
def on_expert_submit(button):
    if waiting_for_expert["flag"]:
        rec = expert_input.value.strip()
        if rec:
            body = waiting_for_expert["body"]
            severity = waiting_for_expert["severity"]
            # Save to dataset
            new_row = {'patient_query': body, 'pain_severity': severity,
'recommended_exercise': rec, 'search_text': f"{body} {severity}"}
            df.loc[len(df)] = new_row
            # Update embeddings
            global corpus_embeddings
            corpus_embeddings = embedder.encode(df['search_text'].tolist(),
convert_to_tensor=True)
            df.to_csv(csv_path, index=False)

            chat_history.append((body, severity, rec, False))
            render_chat()

            waiting_for_expert["flag"] = False
            expert_input.value = ""
        else:
            output_box.value += "<p style='color:red;'>Please type the exercise 
before submitting.</p>"

# ----------------------------
# 9. Chat history button action
# ----------------------------
def on_history(button):
    render_chat()

# ----------------------------
# 10. Attach events
# ----------------------------
recommend_btn.on_click(on_recommend)
expert_submit_btn.on_click(on_expert_submit)
history_btn.on_click(on_history)

# ----------------------------
# 11. Display interface
# ----------------------------
display(
    widgets.HTML("<h3>Physiotherapy Chatbot (WhatsApp-style with Auto-
Scroll)</h3>"),
    body_input,
    severity_input,
    recommend_btn,
    history_btn,
    widgets.HTML("<b>If bot doesn't know, provide the exercise here:</b>"),
    expert_input,
    expert_submit_btn,
    output_box
)
```