Chat with Ollama

The application provides a ChatGPT-like interface powered by Ollama models.
Below are the implemented features:
- Chat creation with automatic titles (first user message becomes the chat title).
- Chats are grouped by recency: Today, Yesterday, and Older.
- Persistent storage of chat history (saved to JSON file) and restored on reload.
- Delete chat option in the sidebar to remove unnecessary conversations.
- Streaming responses: model responses appear in real-time while the model is thinking.
- Interface styled similar to ChatGPT with clear user/assistant message bubbles.

Problems Faced

During development, several challenges were encountered:
- Indentation and formatting issues when embedding custom CSS in Streamlit.
- KeyError caused by missing 'created' field in previously saved chats, requiring safe fallback handling.
- Performance bottlenecks when using larger models, resulting in slower response times.
- Maintaining proper chat history state across sessions and reruns in Streamlit.
- Ensuring real-time streaming responses without breaking the chat display structure.

How It Works

1. The application uses Streamlit for the user interface.
2. Each new chat is stored in a local JSON file, with metadata such as title, creation date, and messages.
3. Messages are displayed in styled chat bubbles (user and assistant roles differentiated).
4. When the user sends a prompt, it is appended to the active chat history and sent to the Ollama API client.
5. The Ollama model streams its response back token by token, which is displayed in real-time.
6. The response is saved into the chat history, ensuring persistence across restarts.

About the Ollama Model

The application integrates with Ollama's language models via the Python client.
In this implementation, the 'llama3:8b' model is used for faster performance compared to
larger models like 'llama3:70b'.

Why it is better than the default/original Ollama setup:

- Optimized for responsiveness: 'llama3:8b' is lightweight yet capable of handling general conversation tasks.
- Streaming support allows users to see partial responses immediately, improving interactivity.

- Reduced latency compared to heavier models, making it more suitable for real-time chat interfaces.
- Balanced trade-off between quality and performance, ensuring users get accurate responses without long delays.

```python
import streamlit as st
import datetime
import json
import os
from ollama import Client

# ----------------------------
# Config
# ----------------------------
st.set_page_config(page_title="Chat with Ollama", page_icon=" ", layout="wide")
DATA_FILE = "chats.json"
client = Client()
```

```python
# ----------------------------
# Persistence helpers
# ----------------------------
def load_chats():
    if os.path.exists(DATA_FILE):
        with open(DATA_FILE, "r", encoding="utf-8") as f:
            return json.load(f)
    return {}
```

```python
def save_chats(chats):
    with open(DATA_FILE, "w", encoding="utf-8") as f:
        json.dump(chats, f, ensure_ascii=False, indent=2, default=str)
```

```python
# ----------------------------
# Init session state
# ----------------------------
```

```
if "chats" not in st.session_state:
    st.session_state.chats = load_chats()
if "active_chat" not in st.session_state:
    st.session_state.active_chat = None
if "theme" not in st.session_state:
    st.session_state.theme = "light"

# ----------------------------
# Sidebar
# ----------------------------
with st.sidebar:
    st.title("  Ollama Chat")

    st.markdown("---")
    st.subheader("  Chat History")

    today = datetime.date.today()
    yesterday = today - datetime.timedelta(days=1)
    groups = {"Today": [], "Yesterday": [], "Older": []}

    for cid, chat in st.session_state.chats.items():
        created_str = chat.get("created")  # might be missing in old data
        if created_str:
            try:
                created_date = datetime.date.fromisoformat(created_str.split(" ")[0])
                if created_date == today:
                    groups["Today"].append((cid, chat))
                elif created_date == yesterday:
                    groups["Yesterday"].append((cid, chat))
                else:
                    groups["Older"].append((cid, chat))
            except Exception:

                groups["Older"].append((cid, chat))

        else:

            groups["Older"].append((cid, chat))
```

```
    for label, chats in groups.items():

        if chats:

            st.markdown(f"**{label}**")

            for cid, chat in chats:

                cols = st.columns([3, 1])

                with cols[0]:

                    if st.button(chat["title"], key=f"open_{cid}"):

                        st.session_state.active_chat = cid

                with cols[1]:

                    if st.button(" ", key=f"del_{cid}"):

                        del st.session_state.chats[cid]

                        save_chats(st.session_state.chats)

                        if st.session_state.active_chat == cid:

                            st.session_state.active_chat = None

                        st.rerun()
```

```
    st.markdown("---")

    if st.button("➕ New Chat"):

        cid = str(len(st.session_state.chats) + 1)

        st.session_state.chats[cid] = {

            "title": "New Chat",

            "messages": [],

            "created": str(datetime.datetime.now())

        }

        st.session_state.active_chat = cid

        save_chats(st.session_state.chats)
```

```
# ----------------------------

# Styles

# ----------------------------

bg_color = "#111" if st.session_state.theme == "dark" else "#fff"

text_color = "#fff" if st.session_state.theme == "dark" else "#000"
```

```
st.markdown(

    f"""

    <style>

    .chat-message {{

        padding: 8px 12px;

        border-radius: 12px;

        margin: 6px 0;

        max-width: 80%;

        word-wrap: break-word;

    }}

    .user {{

        background-color: #2b90d9;

        color: white;

        margin-left: auto;

    }}

    .assistant {{

        background-color: #e5e5e5;

        color: black;

        margin-right: auto;

    }}

    body {{

        background-color: {bg_color};

        color: {text_color};

    }}

    </style>

    """,
    unsafe_allow_html=True,
)
```

```
# ----------------------------
# Main Chat UI
# ----------------------------
if st.session_state.active_chat is None:
    st.info("Start a new chat from the sidebar ➕")
else:
    chat = st.session_state.chats[st.session_state.active_chat]
```

```
    # Show chat history
    for msg in chat["messages"]:
        role = msg["role"]
        content = msg["content"]
        css_class = "user" if role == "user" else "assistant"
        st.markdown(f"<div class='chat-message {css_class}'>{content}</div>", unsafe_allow_html=True)
```

```
    # Input
    user_input = st.chat_input("Type your message...")
```

```
    if user_input:
        # Save user msg
        chat["messages"].append({"role": "user", "content": user_input})
```

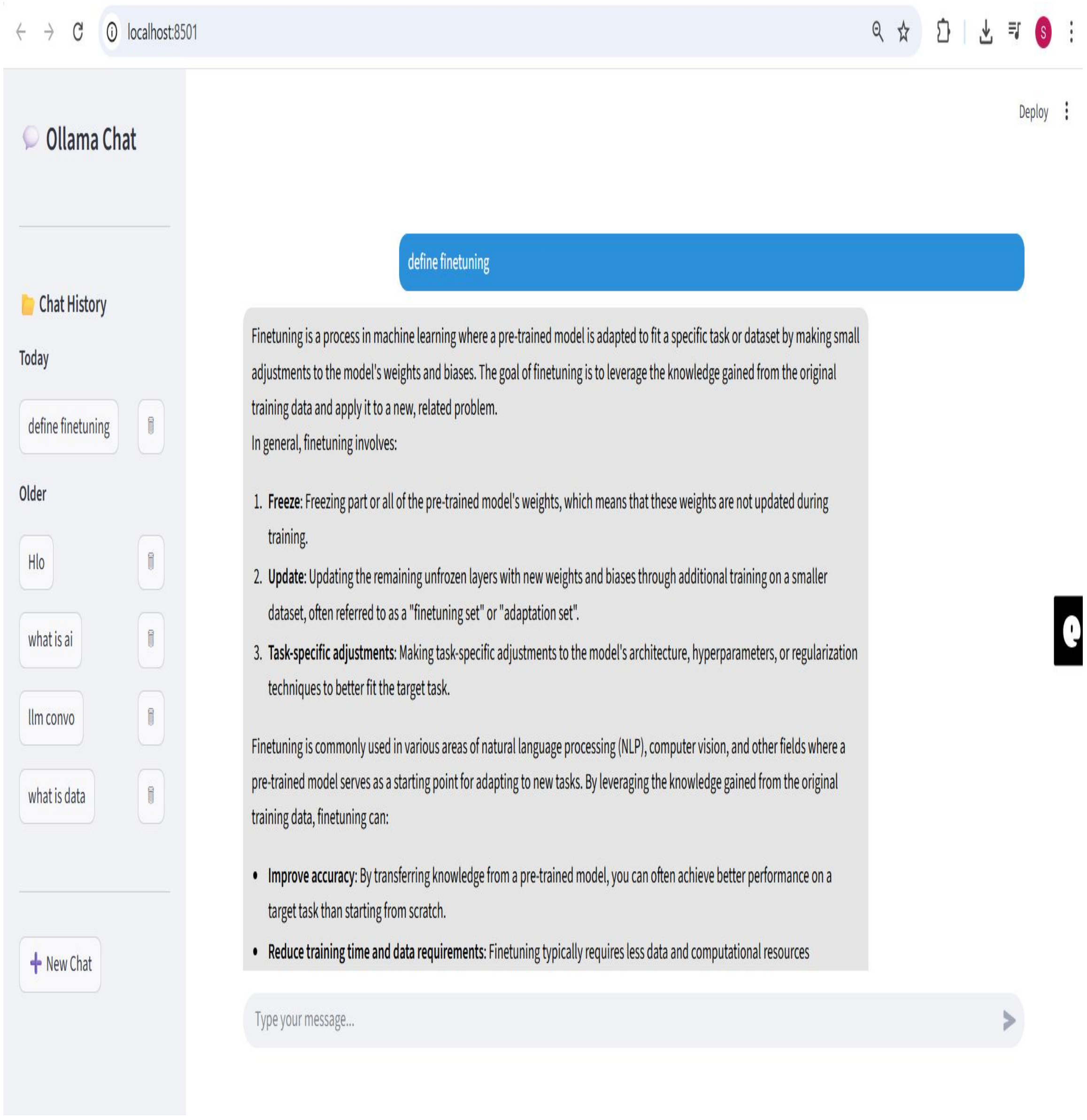
```
        # Automatic title (first user msg)
        if chat["title"] == "New Chat":
            chat["title"] = user_input[:30] + ("..." if len(user_input) > 30 else "")
```

```
        st.markdown(f"<div class='chat-message user'>{user_input}</div>", unsafe_allow_html=True)
        # Stream AI response
        with st.spinner("  Thinking..."):
            response_text = ""
            response_placeholder = st.empty()

            for chunk in client.chat(
                model="llama3:8b",   # ✅ faster model
                messages=chat["messages"],
                stream=True
            ):
                if "message" in chunk:
                    token = chunk["message"]["content"]
                    response_text += token
                    response_placeholder.markdown(
                        f"<div class='chat-message assistant'>{response_text}</div>",
                        unsafe_allow_html=True
                    )

        chat["messages"].append({"role": "assistant", "content": response_text})

        # Save to disk
        st.session_state.chats[st.session_state.active_chat] = chat
        save_chats(st.session_state.chats)
```



define finetuning

Finetuning is a process in machine learning where a pre-trained model is adapted to fit a specific task or dataset by making small adjustments to the model's weights and biases. The goal of finetuning is to leverage the knowledge gained from the original training data and apply it to a new, related problem.
In general, finetuning involves:

1. **Freeze**: Freezing part or all of the pre-trained model's weights, which means that these weights are not updated during training.
2. **Update**: Updating the remaining unfrozen layers with new weights and biases through additional training on a smaller dataset, often referred to as a "finetuning set" or "adaptation set".
3. **Task-specific adjustments**: Making task-specific adjustments to the model's architecture, hyperparameters, or regularization techniques to better fit the target task.

Finetuning is commonly used in various areas of natural language processing (NLP), computer vision, and other fields where a pre-trained model serves as a starting point for adapting to new tasks. By leveraging the knowledge gained from the original training data, finetuning can:

- **Improve accuracy**: By transferring knowledge from a pre-trained model, you can often achieve better performance on a target task than starting from scratch.
- **Reduce training time and data requirements**: Finetuning typically requires less data and computational resources