ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

DAY – 19

Date: Jul 17, 2025

LLaMA 3 Chat (via Ollama)

This project is a simple web-based chat interface that connects to a local LLaMA 3 API using Ollama. It allows you to send messages and receive AI responses in real-time.

STEP 1: Create the HTML file

Open your terminal and create a file named i.html:

```
vim i.html
```

```
step@step-HP-ProDesk-400-G5-SFF:~$ vim i.html
```

STEP 2: Write the HTML and JavaScript

step@step-HP-ProDesk-400-G5-SFF: ~

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>LLaMA 3 Chat</title>
  <style>
    body { font-family: sans-serif; margin: 20px; }
    #chat { width: 100%; height: 300px; border: 1px solid #ccc; padding: 10px; overflow-y: auto; white-space: pre-wrap; background: #f9f9f9; }
    #input { width: 80%; padding: 10px; }
    #send { padding: 10px; }
  </style>
</head>
<body>
  <h2>LLaMA 3 Chat (via Ollama)</h2>
  <div id="chat"></div><br>
  <input type="text" id="input" placeholder="Type a message..." />
  <button id="send">Send</button>

  <script>
    const chatBox = document.getElementById('chat');
    const input = document.getElementById('input');
    const sendBtn = document.getElementById('send');

    sendBtn.onclick = async () => {
      const userText = input.value.trim();
      if (!userText) return;

      appendMessage('You', userText);
      input.value = '';
      input.disabled = true;
      sendBtn.disabled = true;

      let aiMessage = '';
      appendMessage('LLaMA', '');

      const response = await fetch('http://localhost:11434/api/chat', {
        method: 'POST',
                                                                    25,1          Top
```


step@step-HP-ProDesk-400-G5-SFF: ~
headers: { 'Content-Type': 'application/json' },
body: JSON.stringify({
model: 'llama3',
messages: [{ role: 'user', content: userText }],
stream: true
})
});
const reader = response.body.getReader();
const decoder = new TextDecoder();
while (true) {
const { value, done } = await reader.read();
if (done) break;
const chunk = decoder.decode(value);
const lines = chunk.split('\n').filter(line => line.trim());
for (const line of lines) {
try {
const json = JSON.parse(line);
const delta = json?.message?.content || '';
aiMessage += delta;
updateLastMessage('LLaMA', aiMessage);
} catch (e) {
console.error('Invalid JSON chunk:', line);
}
}
}
input.disabled = false;
sendBtn.disabled = false;
input.focus();
};
function appendMessage(sender, message) {
const div = document.createElement('div');
div.className = 'message';
65,1
68%


step@step-HP-ProDesk-400-G5-SFF: ~
for (const line of lines) {
try {
const json = JSON.parse(line);
const delta = json?.message?.content || '';
aiMessage += delta;
updateLastMessage('LLaMA', aiMessage);
} catch (e) {
console.error('Invalid JSON chunk:', line);
}
}
}
input.disabled = false;
sendBtn.disabled = false;
input.focus();
};
function appendMessage(sender, message) {
const div = document.createElement('div');
div.className = 'message';
div.innerHTML = `<strong>${sender}:</strong> <span>${message}</span>`;
chatBox.appendChild(div);
chatBox.scrollTop = chatBox.scrollHeight;
}
function updateLastMessage(sender, newText) {
const messages = chatBox.getElementsByClassName('message');
if (!messages.length) return;
const last = messages[messages.length - 1];
const span = last.querySelector('span');
span.textContent = newText;
chatBox.scrollTop = chatBox.scrollHeight;
}
</script>
</body>
</html>

- This matches your streaming logic exactly.
- It uses fetch with stream: true.
- It updates the last message chunk by chunk.

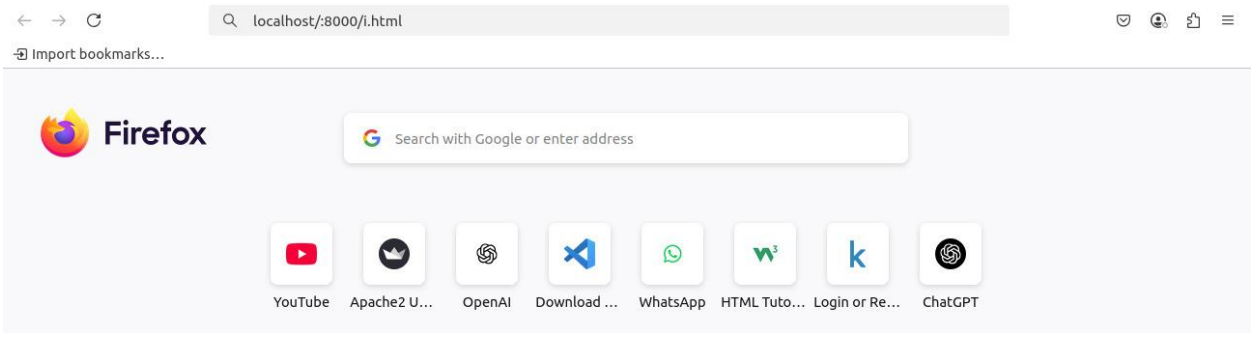
STEP 3: Serve the file locally

Run this in the same folder:

```
python3 -m http.server 8000
```

This starts a simple HTTP server at:

```
http://localhost:8000/
```



Type a message, click Send, and see your conversation appear, streaming in real time!