OCR

What is This Project?

- Chat with an AI
- Upload images and the app reads the text from them (OCR = Optical Character Recognition)
- Ask the AI to analyze the text it found in images

Features

1. AI Chatbot

What it does: Talk to an AI that understands and responds like a human!

Real-world example:

- Us: "What is AI?"
- AI: *Explains artificial intelligence in detail*

2. Image Upload & OCR

What it does: Takes a photo with text and reads it for Us!

Real-world example:

- Upload a photo of handwritten notes
- App extracts
- Now can copy/paste or ask AI about it!

How it works:

- Uses "Tesseract" - free software that reads text from images
- Like how Google Lens scans QR codes or translates signs

3. Chat History

What it does: Saves all Usr conversations (like WhatsApp backup)

Features:

- Groups chats by: Today, Yesterday, Older
- Click any chat to continue where Us left off
- Delete button to remove chats Us don't need

4. Clean User Interface

What it does: Makes the app look nice and easy to use!

Features:

- Blue bubbles for Usr messages (like iPhone)
- Gray bubbles for AI responses
- Special boxes for OCR results
- Sidebar for navigation

Problems Faced (& How I Solved Them!)

Problem 1: "Model requires more memory" Error

What happened:

```
Error: model requires 5.4 GB but only 3.5 GB available
```

Why it happened:

- We used llama3:8b model = 4.7 GB
- Plus system needs memory to run = Total 5.4 GB needed
- Usr computer only had 3.5 GB free

Solution: Switched to smaller model llama3.2:1b = Only 1.3 GB!

Lesson learned: Bigger ≠ Better if it doesn't fit!

Problem 2: "Cannot use bare except" Error

What happened:

```
try:
   # code
except:  # ❌ This is "bare except"
   # handle error
```

Why it's bad:

- Doesn't tell WHAT went wrong
- Hard to debug (find the problem)

- Python's new rules don't allow it

Solution: Be specific about errors:

```
try:
    # code
except FileNotFoundError:  # ✅ Specific!
    # handle missing file
except ValueError:  # ✅ Specific!
    # handle bad data
```

Lesson learned: Be specific when catching errors!

Problem 3: AI Not Responding

What happened: Us uploaded image, saw OCR result, but AI said nothing.

Why it happened:

1. Ollama service wasn't running (the AI's brain was off)
2. Model wasn't pulled (AI brain wasn't installed)
3. Memory error stopped the response

Solution:

- Added connection checks ("Is Ollama running?")
- Added model selector (choose which AI brain to use)
- Added better error messages

Lesson learned: Always check if Usr tools are running!

Problem 4: Tesseract Not Found

What happened:

```
ERROR: Tesseract is not installed or not found
```

Like trying to use Snapchat filters but camera app isn't installed!

Why it happened:

- Tesseract (OCR software) wasn't installed
- Or installed but app couldn't find it

Solution:

1. Install Tesseract from official website
2. Tell our app where to find it:

pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

Lesson learned: Installing software isn't enough - tell Usr code where it is!

Problem 5: Images Not Saving

What happened: Images uploaded but disappeared after refresh.

Why it happened:

- Images are big files
- Can't save directly in JSON (text file)

Solution: Convert image to "Base64" (special code):

- Image → Numbers → Text code → Save in JSON
- Load → Decode text → Numbers → Show image

Think of it like: Taking a photo → Converting to emoji codes → Sending in text → Friend converts back to photo

Lesson learned: Sometimes Us need to transform data to store it!

Technologies Used (In Simple Terms)

1. Python

The programming language (like English for computers)

2. Streamlit

Makes websites/apps quickly (like drag-and-drop website builder)

3. Ollama

Runs AI models on Usr computer (like having ChatGPT offline)

4. Tesseract OCR

Reads text from images (like Google Lens)

5. Pillow

Handles images in Python (resize, convert, edit)

6. JSON

Stores data in organized way (like a digital notebook)

Future Improvements (Ideas!)

1. Dark Mode

Let users choose dark/light theme (easier on eyes at night)

2. Voice Chat

Talk to AI instead of typing (like Siri)

3. Better Image Processing

- Rotate images automatically
- Enhance quality before OCR
- Crop specific areas

4. Multiple AI Models

Switch between different AI personalities:

- Friendly tutor
- Professional writer
- Comedian

5. Mobile App

Convert to phone app (iOS/Android)

6. User Accounts

Login system so everyone has their own chats

7. Share Chats

Send chat links to friends (like Google Docs sharing)

```
import streamlit as st
import datetime
import json
```

```python
import os
from ollama import Client
from PIL import Image
import pytesseract
import io
import base64

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

```python
# Initialize Ollama client
try:
    client = Client()
except Exception as e:
    st.error(f"Ollama connection error: {str(e)}")
    client = None
```

```python
# Configure Tesseract path for Windows
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
```

```python
# ----------------------------
# OCR Helper Functions
# ----------------------------
def extract_text_from_image(image, lang='eng'):
    """Extract text from PIL Image using Tesseract OCR"""
    try:
        if image.mode != 'RGB':
            image = image.convert('RGB')

        custom_config = r'--oem 3 --psm 6'
        text = pytesseract.image_to_string(image, lang=lang, config=custom_config)
        return text.strip()
    except pytesseract.TesseractNotFoundError:
        return "ERROR: Tesseract is not installed or not found. Please install Tesseract OCR."
    except Exception as e:
        return f"Error extracting text: {str(e)}"
```

```python
def image_to_base64(image):
    """Convert PIL Image to base64 for storage"""
    buffered = io.BytesIO()
    image.save(buffered, format="PNG")
    return base64.b64encode(buffered.getvalue()).decode()
```

```python
# ----------------------------
# Persistence helpers
# ----------------------------
def load_chats():
    if os.path.exists(DATA_FILE):
        try:
            with open(DATA_FILE, "r", encoding="utf-8") as f:
                return json.load(f)
        except json.JSONDecodeError:
            return {}
        except IOError as e:
            st.error(f"Error loading chats: {str(e)}")
            return {}
    return {}
```

```python
def save_chats(chats):
    try:
        with open(DATA_FILE, "w", encoding="utf-8") as f:
            json.dump(chats, f, ensure_ascii=False, indent=2, default=str)
    except IOError as e:
        st.error(f"Error saving chats: {str(e)}")
```

```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 "ocr_language" not in st.session_state:
    st.session_state.ocr_language = "eng"
if "selected_model" not in st.session_state:
    st.session_state.selected_model = "llama3.2:1b"
if "show_upload_modal" not in st.session_state:
    st.session_state.show_upload_modal = False
```

```python
# ----------------------------
# Custom CSS - CLEAN & HIGH CONTRAST
# ----------------------------
st.markdown("""
    <style>
    /* Hide default streamlit elements */
    #MainMenu {visibility: hidden;}
    footer {visibility: hidden;}
    header {visibility: hidden;}

    /* Remove all white boxes */
    .stApp {
        background-color: #0a0a0a;
    }

    /* Main container */
    .block-container {
        padding-top: 2rem;
        padding-bottom: 2rem;
        max-width: 900px;
        background-color: transparent !important;
    }

    /* Remove white backgrounds from all elements */
    div[data-testid="stVerticalBlock"] > div {
        background-color: transparent !important;
    }

    /* Sidebar styling - DARK WITH HIGH CONTRAST */
    [data-testid="stSidebar"] {
        background-color: #0f172a !important;
        border-right: 2px solid #1e293b;
    }

    [data-testid="stSidebar"] * {
        color: #e2e8f0 !important;
    }

    [data-testid="stSidebar"] h1 {
        color: #fbbf24 !important;
        font-weight: 700 !important;
        font-size: 1.5rem !important;
        padding: 1rem 0;
    }

    [data-testid="stSidebar"] h2,
    [data-testid="stSidebar"] h3 {
        color: #fbbf24 !important;
        font-weight: 600 !important;
    }

    /* Chat messages - HIGH CONTRAST, NO WHITE BOXES */
    .chat-message {
        padding: 1.5rem;
        margin: 1rem 0;
        border-radius: 12px;
        display: flex;
        flex-direction: column;
        font-size: 1rem;
        line-height: 1.6;
        border: 2px solid;
    }

    .user-message {
        background-color: #1e3a8a !important;
        border-color: #3b82f6 !important;
        color: #e0e7ff !important;
    }

    .user-message * {
        color: #e0e7ff !important;
    }

    .assistant-message {
        background-color: #1e293b !important;
        border-color: #475569 !important;
        color: #f1f5f9 !important;
    }

    .assistant-message * {
        color: #f1f5f9 !important;
    }

    /* OCR Results - HIGH CONTRAST */
    .ocr-container {
        background-color: #78350f !important;
        border: 2px solid #fbbf24 !important;
        padding: 1rem;
        margin: 1rem 0;
        border-radius: 8px;
        font-family: 'Courier New', monospace;
        font-size: 0.95rem;
    }

    .ocr-title {
        font-weight: 700;
        color: #fde68a !important;
        margin-bottom: 0.5rem;
        font-size: 1rem;
    }
    .ocr-container pre {
        color: #fef3c7 !important;
        background-color: transparent !important;
        border: none !important;
        margin: 0 !important;
        padding: 0.5rem 0 !important;
    }

    /* Image preview in chat */
    .image-preview {
        max-width: 400px;
        border-radius: 8px;
        margin: 0.5rem 0;
        border: 2px solid #475569;
    }

    /* Buttons - HIGH CONTRAST */
    .stButton button {
        border-radius: 8px;
        font-weight: 600;
        transition: all 0.2s;
        border: 2px solid;
        background-color: #1e293b !important;
        color: #f1f5f9 !important;
        border-color: #475569 !important;
    }

    .stButton button:hover {
        transform: translateY(-2px);
        box-shadow: 0 4px 8px rgba(251, 191, 36, 0.3);
        background-color: #fbbf24 !important;
        border-color: #f59e0b !important;
        color: #1e293b !important;
    }

    /* Sidebar buttons specific - HIGH CONTRAST */
    [data-testid="stSidebar"] .stButton button {
        background-color: #1e293b !important;
        color: #f1f5f9 !important;
        border: 2px solid #475569 !important;
        font-size: 0.95rem;
        font-weight: 600;
    }

    [data-testid="stSidebar"] .stButton button:hover {
        background-color: #fbbf24 !important;
        color: #1e293b !important;
        border-color: #f59e0b !important;
        transform: translateX(4px);
    }

    /* Delete button hover */
    [data-testid="stSidebar"] .stButton button[key*="del"]:hover {
        background-color: #dc2626 !important;
        color: #ffffff !important;
        border-color: #b91c1c !important;
    }

    /* New chat button - HIGH CONTRAST */
    [data-testid="stSidebar"] button[kind="primary"] {
        background: linear-gradient(to right, #3b82f6, #2563eb) !important;
        border: 2px solid #1d4ed8 !important;
        width: 100%;
        margin-bottom: 1rem;
        color: #ffffff !important;
        font-weight: 700;
        font-size: 1.05rem;
    }

    [data-testid="stSidebar"] button[kind="primary"]:hover {
        background: linear-gradient(to right, #fbbf24, #f59e0b) !important;
        border-color: #d97706 !important;
        color: #1e293b !important;
        transform: scale(1.02);
    }

    /* Selectbox styling - HIGH CONTRAST */
    [data-testid="stSidebar"] .stSelectbox label {
        color: #fbbf24 !important;
        font-weight: 600;
        font-size: 1rem;
    }

    [data-testid="stSidebar"] .stSelectbox div[data-baseweb="select"] {
        background-color: #1e293b !important;
        border: 2px solid #475569 !important;
    }

    [data-testid="stSidebar"] .stSelectbox div[data-baseweb="select"]:hover {
        border-color: #fbbf24 !important;
        background-color: #334155 !important;
    }

    [data-testid="stSidebar"] .stSelectbox [data-baseweb="select"] > div {
        color: #f1f5f9 !important;
    }

    /* Expander styling - HIGH CONTRAST */
    [data-testid="stSidebar"] .streamlit-expanderHeader {
        background-color: #1e293b !important;
        color: #fbbf24 !important;
        font-weight: 600;
        border-radius: 8px;
        border: 2px solid #475569 !important;
    }

    [data-testid="stSidebar"] .streamlit-expanderHeader:hover {
        background-color: #334155 !important;
        border-color: #fbbf24 !important;
    }

    /* Section headers - HIGH CONTRAST */
    [data-testid="stSidebar"] .stMarkdown strong {
        color: #fbbf24 !important;
        font-size: 0.9rem;
        text-transform: uppercase;
        letter-spacing: 0.5px;
        font-weight: 700;
    }

    /* File uploader styling - NO WHITE BOX */
    .uploadedFile {
        background-color: #1e293b !important;
        border: 2px solid #475569 !important;
        color: #f1f5f9 !important;
    }

    [data-testid="stFileUploader"] {
        background-color: #1e293b !important;
        border: 2px solid #475569 !important;
        border-radius: 8px;
        padding: 1rem;
    }

    [data-testid="stFileUploader"] label {
        color: #fbbf24 !important;
        font-weight: 600;
    }

    [data-testid="stFileUploader"] * {
        color: #f1f5f9 !important;
    }

    /* Chat input styling - HIGH CONTRAST */
    .stChatInput > div {
        border: 2px solid #475569 !important;
        background-color: #1e293b !important;
        border-radius: 8px;
    }

    .stChatInput input {
        color: #f1f5f9 !important;
        background-color: #1e293b !important;
    }

    .stChatInput input::placeholder {
        color: #94a3b8 !important;
    }

    /* Welcome screen */
    .welcome-container {
        background-color: transparent !important;
        text-align: center;
        padding: 3rem;
    }

    .welcome-container h1 {
        color: #fbbf24 !important;
        font-weight: 700;
        margin-bottom: 1rem;
    }

    .welcome-container h3 {
        color: #e2e8f0 !important;
        font-weight: 500;
        margin-bottom: 0.5rem;
    }

    .welcome-container p {
        color: #cbd5e1 !important;
        font-size: 1.1rem;
    }
    /* Spinner */
    .stSpinner > div {
        border-top-color: #fbbf24 !important;
    }

    /* Success/Error messages */
    .stSuccess {
        background-color: #166534 !important;
        color: #dcfce7 !important;
        border: 2px solid #22c55e !important;
    }

    .stError {
        background-color: #7f1d1d !important;
        color: #fecaca !important;
        border: 2px solid #dc2626 !important;
    }

    /* Image in messages */
    .stImage {
        border-radius: 8px;
        border: 2px solid #475569;
    }

    /* Horizontal rule */
    hr {
        border-color: #475569 !important;
    }
    </style>
""", unsafe_allow_html=True)
```

```
# ----------------------------
# Sidebar
# ----------------------------
with st.sidebar:
    st.title("  AI Vision Chat")

    # New chat button
    if st.button("➕ New Chat", type="primary", use_container_width=True):
        cid = str(datetime.datetime.now().timestamp())
        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)
        st.rerun()

    st.markdown("---")

    # Model Settings
    with st.expander("  Settings", expanded=False):
        available_models = [
            "llama3.2:1b",
            "llama3.2:3b",
            "llama3:8b",
            "phi3:mini",
        ]

        st.session_state.selected_model = st.selectbox(
            "Model",
            available_models,
            index=0,
        )

        st.session_state.ocr_language = st.selectbox(
            "OCR Language",
            ["eng", "spa", "fra", "deu", "chi_sim", "jpn", "hin"],
            index=0,
        )

    st.markdown("---")
    st.subheader("Recent Chats")

    # 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")
        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 (ValueError, IndexError):
                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([4, 1])
                with cols[0]:
                    if st.button(chat["title"][:30], key=f"open_{cid}", use_container_width=True):
                        st.session_state.active_chat = cid
                        st.rerun()
                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()
```

```
# ----------------------------
# Main Chat Area
# ----------------------------
if st.session_state.active_chat is None:
    st.markdown("""
        <div class='welcome-container'>
            <h1>  AI Vision Chat</h1>
            <h3>Upload images and ask questions!</h3>
            <p>I can read text from images and answer your questions about them</p>
        </div>
    """, unsafe_allow_html=True)
else:
    chat = st.session_state.chats[st.session_state.active_chat]

    # Display chat messages
    for msg in chat["messages"]:
        role = msg.get("role", "user")
        content = msg.get("content", "")

        # Create message container
        msg_class = "user-message" if role == "user" else "assistant-message"

        st.markdown(f"<div class='chat-message {msg_class}'>", unsafe_allow_html=True)

        # Show image if present
        if msg.get("image_data"):
            try:
                img_bytes = base64.b64decode(msg["image_data"])
                st.image(img_bytes, width=300, caption="  Uploaded Image")
            except Exception as e:
                st.error(f"Error displaying image: {str(e)}")

        # Show OCR result if present
        if msg.get("ocr_text"):
            st.markdown(
                f"""<div class='ocr-container'>
                    <div class='ocr-title'>  Extracted Text:</div>
                    <pre>{msg['ocr_text']}</pre>
                </div>""",
                unsafe_allow_html=True
            )

        # Show message content
        if content:
            st.markdown(content)

        st.markdown("</div>", unsafe_allow_html=True)

    # Spacer for fixed input
    st.markdown("<div style='height: 100px;'></div>", unsafe_allow_html=True)
```

```
# ----------------------------
# Image Upload Modal
# ----------------------------
if st.session_state.show_upload_modal and st.session_state.active_chat:
    st.markdown("###   Upload Image")

    uploaded_file = st.file_uploader(
        "Choose an image",
        type=["png", "jpg", "jpeg", "bmp", "tiff"],
        key="image_uploader"
    )

    if uploaded_file:
        try:
            image = Image.open(uploaded_file)
            st.image(image, caption="Preview", use_container_width=True)

            col1, col2, col3 = st.columns(3)
            with col1:
                if st.button("  Extract Text", use_container_width=True):
                    with st.spinner("Extracting text..."):
                        ocr_text = extract_text_from_image(image, st.session_state.ocr_language)

                        if ocr_text and not ocr_text.startswith("ERROR"):
                            img_base64 = image_to_base64(image)
                            chat = st.session_state.chats[st.session_state.active_chat]
                            chat["messages"].append({
                                "role": "user",
                                "content": "  Image uploaded",
                                "image_data": img_base64,
                                "ocr_text": ocr_text
                            })
                            save_chats(st.session_state.chats)
                            st.session_state.show_upload_modal = False
                            st.success("✅ Text extracted successfully!")
                            st.rerun()
                        else:
                            st.error(ocr_text)

            with col2:
                if st.button("  Analyze", use_container_width=True):
                    if client:
                        with st.spinner("Analyzing image..."):
                            ocr_text = extract_text_from_image(image, st.session_state.ocr_language)

                            if ocr_text and not ocr_text.startswith("ERROR"):
                                img_base64 = image_to_base64(image)
                                chat = st.session_state.chats[st.session_state.active_chat]

                                # Add image to chat
                                chat["messages"].append({
                                    "role": "user",
                                    "content": "  Please analyze this image",
                                    "image_data": img_base64,
                                    "ocr_text": ocr_text
                                })

                                # Simple, direct analysis
                                api_messages = [{
                                    "role": "user",
                                    "content": f"Here is text extracted from an image. Please analyze it and tell me 
what it's about:\n\n{ocr_text}"
                                }]

                                try:
                                    response_text = ""
                                    for chunk in client.chat(
                                        model=st.session_state.selected_model,
                                        messages=api_messages,
                                        stream=True
                                    ):
                                        if "message" in chunk and "content" in chunk["message"]:
                                            response_text += chunk["message"]["content"]

                                    chat["messages"].append({
                                        "role": "assistant",
                                        "content": response_text
                                    })
                                    save_chats(st.session_state.chats)
                                    st.session_state.show_upload_modal = False
                                    st.rerun()
                                except Exception as e:
                                    st.error(f"Analysis error: {str(e)}")
                            else:
                                st.error("Could not extract text from image")
                    else:
                        st.error("Ollama not connected")

            with col3:
                if st.button("  Save", use_container_width=True):
                    img_base64 = image_to_base64(image)
                    ocr_text = extract_text_from_image(image, st.session_state.ocr_language)
                    chat = st.session_state.chats[st.session_state.active_chat]
                    chat["messages"].append({
                        "role": "user",
                        "content": "  Image saved",
                        "image_data": img_base64,
                        "ocr_text": ocr_text if not ocr_text.startswith("ERROR") else None
                    })
                    save_chats(st.session_state.chats)
                    st.session_state.show_upload_modal = False
                    st.success("✅ Image saved!")
                    st.rerun()

        except Exception as e:
            st.error(f"Error processing image: {str(e)}")

    if st.button("✖ Close", use_container_width=True):
        st.session_state.show_upload_modal = False
        st.rerun()
```

```
# ----------------------------
# Chat Input with Upload Button
# ----------------------------
if st.session_state.active_chat:
    # Create columns for upload button and input
    col1, col2 = st.columns([1, 20])

    with col1:
        if st.button(" ", key="upload_trigger", help="Upload Image"):
            st.session_state.show_upload_modal = True
            st.rerun()

    with col2:
        user_input = st.chat_input("Ask anything about the images...")

        if user_input:
            chat = st.session_state.chats[st.session_state.active_chat]

            # Add user message
            chat["messages"].append({"role": "user", "content": user_input})

            if chat["title"] == "New Chat":
                chat["title"] = user_input[:30] + ("..." if len(user_input) > 30 else "")

            save_chats(st.session_state.chats)

            if client:
                with st.spinner("Thinking..."):
                    try:
                        # Build conversation with full context
                        api_messages = []

                        # Collect all OCR text from images
                        image_data = []
                        for idx, m in enumerate(chat["messages"]):
                            if m.get("ocr_text"):
                                image_data.append({
                                    "index": len(image_data) + 1,
                                    "content": m["ocr_text"]
                                })

                        # If there are images, add them as context in the conversation
                        if image_data:
                            # Create a context message with all image content
                            context_parts = []
                            for img in image_data:
                                context_parts.append(f"IMAGE {img['index']}:\n{img['content']}")

                            full_image_context = "\n\n---\n\n".join(context_parts)

                            # Add system-like context as a user message (works better with some models)
                            api_messages.append({
                                "role": "user",
                                "content": f"I have uploaded {len(image_data)} image(s) with the following
content:\n\n{full_image_context}"
                            })

                            api_messages.append({
                                "role": "assistant",
                                "content": "I understand. I have processed the image content you provided. I will use
this information to answer your questions accurately."
                            })

                        # Add recent conversation (last 10 messages, excluding image upload notifications)
                        recent_msgs = []
                        for m in chat["messages"][-11:-1]:  # Exclude the current message
                            if m.get("content") and not m["content"].startswith((" ", " ")):
                                recent_msgs.append(m)

                        # Add the filtered conversation
                        for m in recent_msgs:
                            api_messages.append({
                                "role": m["role"],
                                "content": m["content"]
                            })

                        # Add the current user question
                        api_messages.append({
                            "role": "user",
                            "content": user_input
                        })

                        # Get response from model
                        response_text = ""
                        for chunk in client.chat(
                            model=st.session_state.selected_model,
                            messages=api_messages,
                            stream=True
                        ):
                            if "message" in chunk and "content" in chunk["message"]:
                                response_text += chunk["message"]["content"]

                        chat["messages"].append({"role": "assistant", "content": response_text})
                        save_chats(st.session_state.chats)
                        st.rerun()

                    except Exception as e:
                        error_msg = str(e)
                        if "memory" in error_msg.lower() or "500" in error_msg:
                            error_response = "  Memory error! Try switching to llama3.2:1b in Settings or start a new
chat."
                        else:
                            error_response = f"  Error: {error_msg}"

                        chat["messages"].append({"role": "assistant", "content": error_response})
                        save_chats(st.session_state.chats)
                        st.rerun()
            else:
                st.error("Ollama is not connected. Please start Ollama first.")
```