Deepseek Chatbot App 🤖

Welcome to the **Deepseek Chatbot App**! This app allows you to interact with your own **Deepseek LLM model** in a user-friendly interface. Whether you're looking for answers, brainstorming ideas, or just having fun, this chatbot is here to assist you.

Key Features

- **Custom Deepseek Model**: Chat with the powerful Deepseek LLM tailored to your needs.
- **Streamlit Interface**: A clean, interactive, and responsive web-based UI for seamless interaction.
- **Session History**: Maintain a conversation history to keep track of your interactions.
- **Future Enhancements**:
  - **Web Search Integration**: (Coming Soon!) Access real-time information from the web directly within the chat.

How to Use

1. **Install Dependencies**: Ensure you have the required libraries installed (`ollama`, `streamlit`, etc.).
2. **Run the App**: Start the Streamlit app and interact with the chatbot.
3. **Enjoy**: Ask questions, get answers, and explore the capabilities of the Deepseek model.

Stay Updated

Follow me on my social media platforms to stay updated about new features, tutorials, and more:

- **LinkedIn**: Prashant
- **GitHub**: Mr-Dark-debug
- **Instagram**: Prashant

Feedback and Contributions

If you have any feedback, suggestions, or want to contribute to this project, feel free to reach out via the links above. I'd love to hear from you!

Thank you for using the Deepseek Chatbot App!
Happy chatting!

Ollama Chatbot Setup and Explanation

This guide provides a step-by-step explanation of how to set up and run the Ollama chatbot interface using Google Colab. Each step is explained in detail to ensure clarity.

Step 1: Install Required Dependencies

Before we begin, we need to install the necessary libraries and tools to interact with Ollama and run the Streamlit app.

Explanation:

1. **`colab-xterm`**: This library allows us to use a terminal within Google Colab.
2. **`ollama`**: The Python client for interacting with the Ollama API.
3. **`streamlit`**: A framework for building interactive web apps.
4. **`markdown`**: Used for rendering Markdown content.
5. **`localtunnel`**: Exposes your local Streamlit app to the internet so it can be accessed externally.

```
!pip install colab-xterm ollama streamlit markdown
!npm i localtunnel@2.0.2

Collecting colab-xterm
  Downloading colab_xterm-0.2.0-py3-none-any.whl.metadata (1.2 kB)
Requirement already satisfied: ptyprocess~=0.7.0 in
/usr/local/lib/python3.11/dist-packages (from colab-xterm) (0.7.0)
Requirement already satisfied: tornado>5.1 in
/usr/local/lib/python3.11/dist-packages (from colab-xterm) (6.3.3)
Downloading colab_xterm-0.2.0-py3-none-any.whl (115 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 115.6/115.6 kB 2.2 MB/s eta
0:00:00

Successfully installed colab-xterm-0.2.0
Collecting ollama
  Downloading ollama-0.4.7-py3-none-any.whl.metadata (4.7 kB)
Requirement already satisfied: httpx<0.29,>=0.27 in
```

```
/usr/local/lib/python3.11/dist-packages (from ollama) (0.28.1)
Requirement already satisfied: pydantic<3.0.0,>=2.9.0 in
/usr/local/lib/python3.11/dist-packages (from ollama) (2.10.6)
Requirement already satisfied: anyio in
/usr/local/lib/python3.11/dist-packages (from httpx<0.29,>=0.27-
>ollama) (3.7.1)
Requirement already satisfied: certifi in
/usr/local/lib/python3.11/dist-packages (from httpx<0.29,>=0.27-
>ollama) (2024.12.14)
Requirement already satisfied: httpcore==1.* in
/usr/local/lib/python3.11/dist-packages (from httpx<0.29,>=0.27-
>ollama) (1.0.7)
Requirement already satisfied: idna in /usr/local/lib/python3.11/dist-
packages (from httpx<0.29,>=0.27->ollama) (3.10)
Requirement already satisfied: h11<0.15,>=0.13 in
/usr/local/lib/python3.11/dist-packages (from httpcore==1.*-
>httpx<0.29,>=0.27->ollama) (0.14.0)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.9.0-
>ollama) (0.7.0)
Requirement already satisfied: pydantic-core==2.27.2 in
/usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.9.0-
>ollama) (2.27.2)
Requirement already satisfied: typing-extensions>=4.12.2 in
/usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.9.0-
>ollama) (4.12.2)
Requirement already satisfied: sniffio>=1.1 in
/usr/local/lib/python3.11/dist-packages (from anyio-
>httpx<0.29,>=0.27->ollama) (1.3.1)
Downloading ollama-0.4.7-py3-none-any.whl (13 kB)
Installing collected packages: ollama
Successfully installed ollama-0.4.7
```

Step 2: Initialize the Terminal in Google Colab

To interact with Ollama and run commands, we need to initialize a terminal in Google Colab.

Explanation:

- `%load_ext colabxterm`: Loads the `colab-xterm` extension to enable terminal functionality.
- `%xterm`: Launches the terminal interface within the notebook.

Download and Install Ollama

Follow these steps to download and set up Ollama:

1. **Download Ollama**: Run the following command in the terminal to install Ollama:

```
curl -fsSL https://ollama.com/install.sh | sh
```

2. **View Available Models**: Visit the Ollama Model Library to explore available models and choose one that suits your needs.

3. **Check Installed Models**: After installation, check if Ollama is working by listing the installed models:

```
ollama list
```

   If you see the error:

```
Error: could not connect to ollama app, is it running?
```

   Don't worry! Proceed to the next step to start the Ollama service.

Start Ollama and Pull Your Desired Model

1. **Start the Ollama Service**: Run the following command to start the Ollama service:

```
ollama serve &
```

2. **Pull a Model**: Download your desired model using the `ollama pull` command. For example:

```
ollama pull deepseek-r1:32b
```

   Replace `deepseek-r1:70b` with the name of the model you want to use. You can find model names on the Ollama Model Library.

3. **Run the Model**: Once the model is downloaded, you can run it using:

```
ollama run <your-model-name>
```

   Example:

```
ollama run deepseek-r1:32b
```

Stop Ollama (When Finished)

When you're done using Ollama, stop the service to free up resources.

1. **Find the Process ID**: Use the following command to find the process ID of the Ollama service:

```
ps aux | grep ollama
```

2. **Stop the Process**: Use the `kill` command to stop the process. For example:

```
kill <process-id>
```

   Replace `<process-id>` with the actual process ID from the previous step.

Copy-Paste in the Terminal

- **Keyboard Shortcuts**: `Ctrl+C` and `Ctrl+V` won't work in the terminal.
- **Mouse Right-Click**: Use the mouse right-click to copy and paste text into the terminal.

Additional Notes:

- **Model Selection**: Explore the Ollama Model Library to find the best model for your use case.
- **Troubleshooting**: If you encounter any issues, ensure that the Ollama service is running (`ollama serve &`) and that the model is correctly downloaded (`ollama pull`).

By following these steps, you should be able to successfully set up and interact with Ollama in Google Colab terminal.

```
%load_ext colabxterm

%xterm

Launching Xterm...

<IPython.core.display.Javascript object>
```

Step 3: Verify GPU Availability (Optional)

If you're using a GPU-enabled runtime, verify that the GPU is available and properly configured.

Explanation:

1. **`nvidia-smi`**: Displays GPU information, such as memory usage and driver version.

2. **`free -h`**: Shows system memory usage in a human-readable format.

```
!nvidia-smi

/bin/bash: line 1: nvidia-smi: command not found

!free -h

               total        used        free      shared  buff/cache
available
Mem:            12Gi       945Mi       5.8Gi       1.0Mi       6.0Gi
11Gi
Swap:             0B          0B          0B
```

Step 4: Set GPU Environment Variables (Optional)

If you're using a GPU, configure environment variables to optimize performance.

Explanation:

1. **`CUDA_VISIBLE_DEVICES`**: Specifies which GPU(s) to use. `"0"` refers to the first GPU.
2. **`OLLAMA_GPU_OVERHEAD`**: Reduces GPU overhead to improve efficiency.

```
# !export CUDA_VISIBLE_DEVICES=0  # Make sure the right GPU is
selected (0 corresponds to the first GPU)
# !export OLLAMA_GPU_OVERHEAD=0
import os

# Set GPU environment variables
a = os.environ["CUDA_VISIBLE_DEVICES"] = "0"  # Use the first GPU
b = os.environ["OLLAMA_GPU_OVERHEAD"] = "0"  # Set the overhead to use
the GPU efficiently
print(a)
print(b)

0
0
```

Step 5: Test the Ollama API

Before integrating Ollama into the Streamlit app, test the API to ensure it's working correctly.

Explanation:

1. **`ollama.chat`**: Sends a request to the Ollama API with the specified model and user prompt.

2. **Model Name**: Replace `'deepseek-r1:14b'` with the exact model name available in your Ollama instance.
3. **Response**: The API returns the model's response, which is printed to the console.

```
import ollama
prompt = "write a simple html css and js code to create a synth
keybord"
response = ollama.chat(model='deepseek-r1:14b', messages=[{"role":
"user","content": prompt}])
print(response['message']['content'])

---------------------------------------------------------------------
-----
ConnectError                              Traceback (most recent call
last)
/usr/local/lib/python3.11/dist-packages/httpx/_transports/default.py
in map_httpcore_exceptions()
     71     try:
---> 72         yield
     73     except Exception as exc:

/usr/local/lib/python3.11/dist-packages/httpx/_transports/default.py
in handle_request(self, request)
    235         with map_httpcore_exceptions():
--> 236             resp = self._pool.handle_request(req)
    237

/usr/local/lib/python3.11/dist-packages/httpcore/_sync/connection_pool
.py in handle_request(self, request)
    255             self._close_connections(closing)
--> 256             raise exc from None
    257

/usr/local/lib/python3.11/dist-packages/httpcore/_sync/connection_pool
.py in handle_request(self, request)
    235                     # Send the request on the assigned
connection.
--> 236                     response = connection.handle_request(
    237                         pool_request.request

/usr/local/lib/python3.11/dist-packages/httpcore/_sync/connection.py
in handle_request(self, request)
    100             self._connect_failed = True
--> 101             raise exc
    102

/usr/local/lib/python3.11/dist-packages/httpcore/_sync/connection.py
```

```
in handle_request(self, request)
     77                 if self._connection is None:
---> 78                     stream = self._connect(request)
     79

/usr/local/lib/python3.11/dist-packages/httpcore/_sync/connection.py
in _connect(self, request)
    123                     with Trace("connect_tcp", logger, request,
kwargs) as trace:
--> 124                         stream =
self._network_backend.connect_tcp(**kwargs)
    125                         trace.return_value = stream

/usr/local/lib/python3.11/dist-packages/httpcore/_backends/sync.py in
connect_tcp(self, host, port, timeout, local_address, socket_options)
    206
--> 207         with map_exceptions(exc_map):
    208             sock = socket.create_connection(

/usr/lib/python3.11/contextlib.py in __exit__(self, typ, value,
traceback)
    157             try:
--> 158                 self.gen.throw(typ, value, traceback)
    159             except StopIteration as exc:

/usr/local/lib/python3.11/dist-packages/httpcore/_exceptions.py in
map_exceptions(map)
     13             if isinstance(exc, from_exc):
---> 14                 raise to_exc(exc) from exc
     15         raise  # pragma: nocover

ConnectError: [Errno 111] Connection refused

The above exception was the direct cause of the following exception:

ConnectError                              Traceback (most recent call
last)
<ipython-input-4-c599a2d6c1f7> in <cell line: 0>()
      1 import ollama
      2 prompt = "write a simple html css and js code to create a
synth keybord"
----> 3 response = ollama.chat(model='qwen2.5-coder',
messages=[{"role": "user","content": prompt}])
      4 print(response['message']['content'])

/usr/local/lib/python3.11/dist-packages/ollama/_client.py in
chat(self, model, messages, tools, stream, format, options,
keep_alive)
    330     Returns `ChatResponse` if `stream` is `False`, otherwise
returns a `ChatResponse` generator.
```

```
    331     """
--> 332     return self._request(
    333       ChatResponse,
    334       'POST',

/usr/local/lib/python3.11/dist-packages/ollama/_client.py in
_request(self, cls, stream, *args, **kwargs)
    175       return inner()
    176
--> 177     return cls(**self._request_raw(*args, **kwargs).json())
    178
    179   @overload

/usr/local/lib/python3.11/dist-packages/ollama/_client.py in
_request_raw(self, *args, **kwargs)
    116
    117   def _request_raw(self, *args, **kwargs):
--> 118     r = self._client.request(*args, **kwargs)
    119     try:
    120       r.raise_for_status()

/usr/local/lib/python3.11/dist-packages/httpx/_client.py in
request(self, method, url, content, data, files, json, params,
headers, cookies, auth, follow_redirects, timeout, extensions)
    835             extensions=extensions,
    836         )
--> 837         return self.send(request, auth=auth,
follow_redirects=follow_redirects)
    838
    839     @contextmanager

/usr/local/lib/python3.11/dist-packages/httpx/_client.py in send(self,
request, stream, auth, follow_redirects)
    924         auth = self._build_request_auth(request, auth)
    925
--> 926         response = self._send_handling_auth(
    927             request,
    928             auth=auth,

/usr/local/lib/python3.11/dist-packages/httpx/_client.py in
_send_handling_auth(self, request, auth, follow_redirects, history)
    952
    953             while True:
--> 954                 response = self._send_handling_redirects(
    955                     request,
    956                     follow_redirects=follow_redirects,

/usr/local/lib/python3.11/dist-packages/httpx/_client.py in
_send_handling_redirects(self, request, follow_redirects, history)
    989                 hook(request)
```

```
    990
--> 991             response = self._send_single_request(request)
    992             try:
    993                 for hook in self._event_hooks["response"]:

/usr/local/lib/python3.11/dist-packages/httpx/_client.py in
_send_single_request(self, request)
   1025
   1026         with request_context(request=request):
-> 1027             response = transport.handle_request(request)
   1028
   1029         assert isinstance(response.stream, SyncByteStream)

/usr/local/lib/python3.11/dist-packages/httpx/_transports/default.py
in handle_request(self, request)
    233             extensions=request.extensions,
    234         )
--> 235         with map_httpcore_exceptions():
    236             resp = self._pool.handle_request(req)
    237

/usr/lib/python3.11/contextlib.py in __exit__(self, typ, value,
traceback)
    156                 value = typ()
    157             try:
--> 158                 self.gen.throw(typ, value, traceback)
    159             except StopIteration as exc:
    160                 # Suppress StopIteration *unless* it's the
same exception that

/usr/local/lib/python3.11/dist-packages/httpx/_transports/default.py
in map_httpcore_exceptions()
     87
     88         message = str(exc)
---> 89         raise mapped_exc(message) from exc
     90
     91

ConnectError: [Errno 111] Connection refused
```

Step 6: Create the Streamlit App

The Streamlit app provides a user-friendly interface for interacting with the Ollama chatbot.

Explanation:

1. **`query_ollama` Function**:
   - Sends a POST request to the Ollama API with the user's prompt.
   - Handles errors gracefully and provides feedback to the user.
2. **Streamlit UI**:

- Displays the chat history and allows users to input prompts.
- Uses `st.session_state` to maintain chat history across interactions.

3. **Dynamic Updates**:
   - When the user submits a message, the app queries Ollama and dynamically updates the chat interface.

```
%%writefile app.py
import streamlit as st
import requests

# Constants
OLLAMA_BASE_URL = "http://localhost:11434"  # Default Ollama API URL

def query_ollama(prompt):
    """Queries the Ollama model and handles response."""
    try:
        headers = {'Content-Type': 'application/json'}
        data = {
            "model": "deepseek-r1:32b",  # Replace with your Ollama
model name
            "prompt": prompt,
            "stream": False
        }
        response = requests.post(f"{OLLAMA_BASE_URL}/api/generate",
headers=headers, json=data, timeout=60)
        response.raise_for_status()
        result = response.json()
        if 'response' in result:
            return result['response']
        else:
            st.error(f"Unexpected response format from Ollama:
{result}")
            return "Unexpected response format from Ollama."
    except requests.exceptions.RequestException as e:
        st.error(f"Error connecting to Ollama: {e}")
        return "Error connecting to Ollama."

def main():
    # Streamlit app configuration
    st.title("Ollama Chatbot Interface")
    st.write("Chat with the Ollama LLM model!")

    # Initialize session state for chat history
    if "messages" not in st.session_state:
        st.session_state.messages = []

    # Display chat history
    for message in st.session_state.messages:
```

```
        with st.chat_message(message["role"]):
            st.markdown(message["content"])

    # Input prompt from user
    if prompt := st.chat_input("Type your message here..."):
        # Add user message to chat history
        st.session_state.messages.append({"role": "user", "content":
prompt})
        with st.chat_message("user"):
            st.markdown(prompt)

        # Query Ollama for a response
        with st.spinner("Thinking..."):
            response = query_ollama(prompt)

        # Add assistant response to chat history
        st.session_state.messages.append({"role": "assistant",
"content": response})
        with st.chat_message("assistant"):
            st.markdown(response)

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

Step 7: Run the Streamlit App

Finally, run the Streamlit app and expose it to the internet using `localtunnel`.

Explanation:

1. **`streamlit run app.py`**: Starts the Streamlit app on port 8501.
2. **`localtunnel`**: Exposes the app to the internet, providing an external URL for access.

Access the App

Once the app is running, you'll see URLs like:

- **Local URL**: `http://localhost:8501`
- **External URL**: Provided by `localtunnel` (e.g., `https://<random-subdomain>.loca.lt`).

Use the external URL to access the app from any device.

if asked for password or any access code run this command `!wget -q -O - ipv4.icanhazip.com` before running the `streamlit run` command to get your local ipv4 adress that is also your password.

```
!wget -q -O - ipv4.icanhazip.com

!streamlit run app.py & npx localtunnel --port 8501
```

Troubleshooting

1. **Connection Issues**:
   - Ensure the Ollama server is running locally or accessible at `http://localhost:11434`.
   - Check firewall settings if `localtunnel` fails.
2. **Model Errors**:
   - Verify the model name on streamlit (`deepseek-r1:14b`) matches the one in your Ollama instance.
3. **Dependency Conflicts**:
   - If you encounter dependency issues, restart the runtime and reinstall the required libraries.
4. **IPv6 vs. IPv4**:

- If you're working in an environment that uses IPv6, replace ipv4.icanhazip.com with ipv6.icanhazip.com to get your IPv6 address.