

The background of the slide is a light gray gradient. It is decorated with numerous realistic water droplets of various sizes. Some droplets are at the top, some at the bottom, and some on the sides. They have highlights and shadows, giving them a three-dimensional appearance.

BUILDING A CHATBOT WITH SHORT TERM MEMORY (AKA CONVERSATIONAL MEMORY)

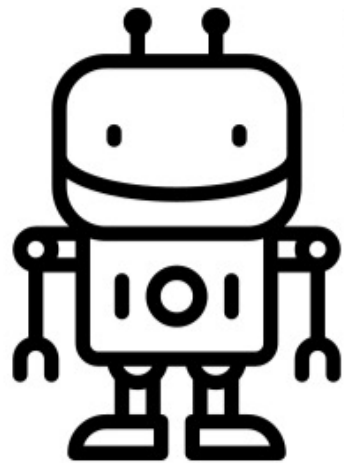
Topics

- Short term memory (conversation memory)
- Using Streamlit to implement a chatbot
- Integrating OpenAI with streamlit
- Three approaches / techniques

LLMS ARE STATELESS

- No built-in mechanism to use information from one interaction to the next.
- Each request to the model is processed independently
 - without any knowledge of previous requests or responses.
 - Applications need to maintain the state and pass it to the LLM.

HOWEVER, WE NEED MEMORY!

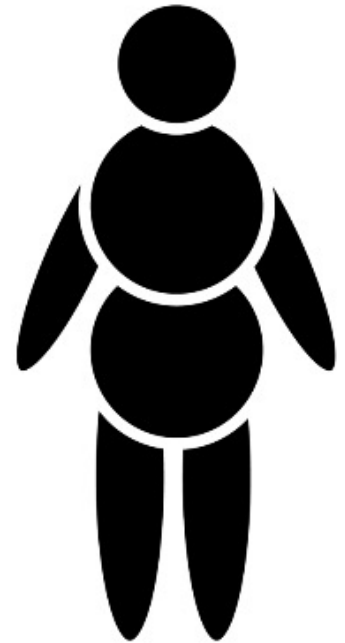


Who was Picasso?

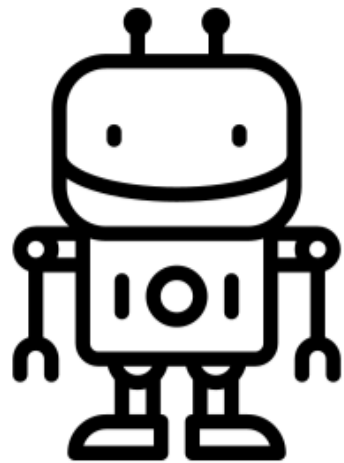
A Spanish artist.

When was he born?

Idk what you're talking about.



HOWEVER, WE NEED MEMORY!



Who was Picasso?

A Spanish artist.

When was he born?

October 25, 1881.



WE NEED MEMORY!

With conversational memory

I'm interested in integrating LLMs with external knowledge.

LLMs are great at generating human-like text. Yet, integrating external knowledge can enhance their capabilities even more.

What are the different possible methods for doing this?

You could use pre-existing knowledge graphs, allow LLMs access to tools like APIs, or retrieval augmentation with vector DBs!

..... Conversation History

Interesting! What was it I wanted to know about again?

You were interested in integrating LLMs with external knowledge.

Without conversational memory

(No conversation history is stored)

..... Conversation History

Interesting! What was it I wanted to know about again?

Sorry I have no idea what you're talking about!

CONVERSATIONAL MEMORY

- In order to create a highly personalized experience for each user, it's critical to **have the ability to reference earlier messages** and prior conversations.

→ This is known as '**short term**' or '**conversational**' memory
(since it is only remembered during this specific 'chat')

- **However: Remember that LLM's are stateless**

- No built-in mechanism to use information from one interaction to the next.
- Each request to the model is processed independently
 - Without any knowledge of previous requests or responses.
 - Applications need to maintain the state and pass it to the LLM.

HISTORY & INPUTS



I'm interested in exploring the potential of integrating Large Language Models with external knowledge



Integrating Large Language Models (LLMs) with external knowledge can improve performance. Techniques include pre-training on domain-specific data, fine-tuning on task-specific data, or retrieving knowledge during inference. This has potential for natural language understanding and generating human-like responses.



I'd like to analyze the different possibilities, what can you think of?



There are several ways to integrate Large Language Models (LLMs) with external knowledge. One approach is to pre-train the model on large amounts of external data, such as domain-specific corpora or knowledge bases. Another approach is to fine-tune the model on task-specific data that contains external knowledge, such as scientific papers or structured databases. Additionally, there are techniques that enable the model to retrieve external knowledge during inference, such as knowledge graph embeddings or query-based methods. Each approach has its own advantages and limitations and can be chosen based on the specific use case.



{input}

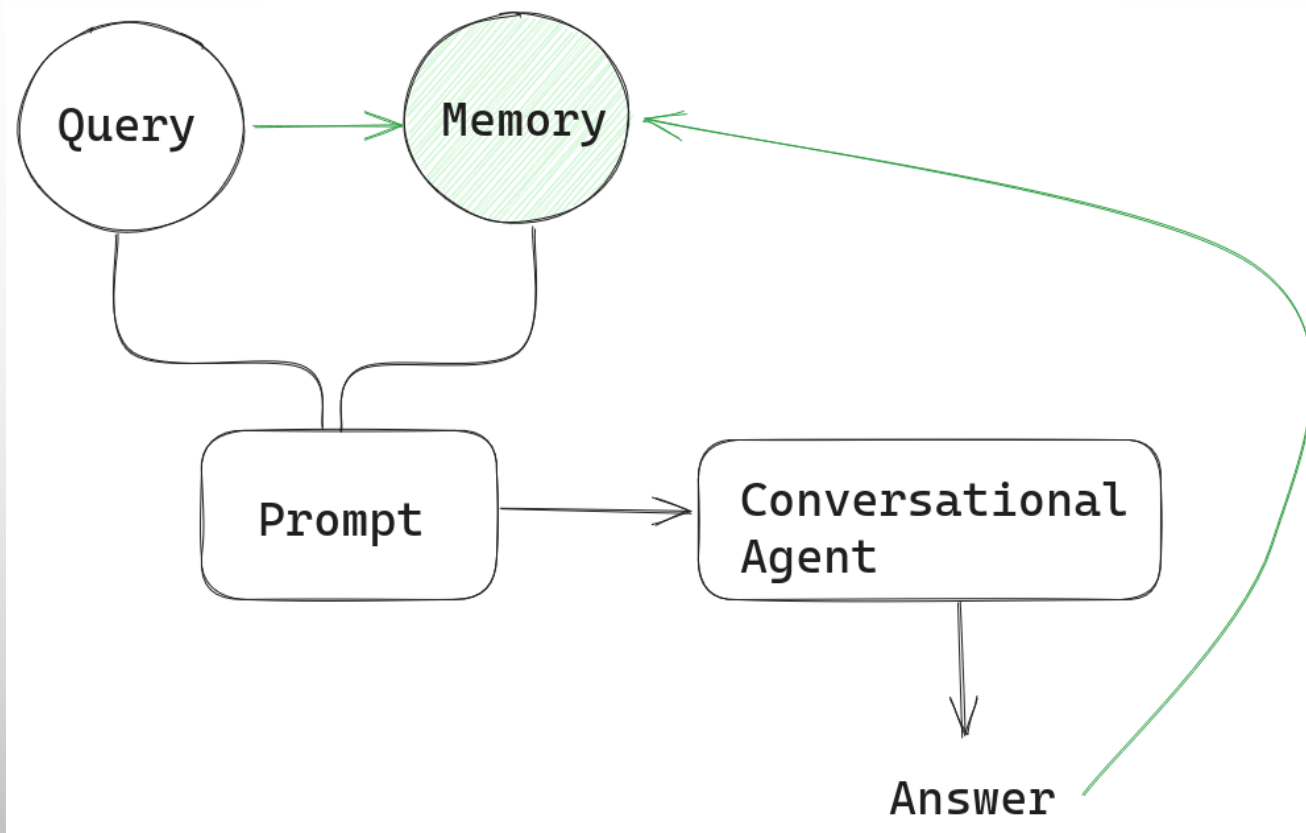


Which data source types could be used to give context to the model?



{history}

ADDING 'MEMORY' TO THE LLM REQUEST



HOW TO ACHIEVE CONVERSATION MEMORY

- Keep track of a list of previous messages
- Pass as part of the prompt to the LLM.
- Note: OpenAPI Assistants API allow you to keep track of a list of messages
→ But we will be using an approach that works for all LLMs

HOW TO IMPLEMENT A CONVERSATION BUFFER

```
completion = client.chat.completions.create(  
    model="gpt-3.5-turbo",  
    messages=[  
        {"role": "system", "content": "You are a helpful assistant."},  
        {"role": "user", "content": "message 1 content."},  
        {"role": "assistant", "content": "message 2 content"},  
        {"role": "user", "content": "message 3 content"},  
        {"role": "assistant", "content": "message 4 content."},  
        {"role": "user", "content": "message 5 content."}  
    ],  
)
```

Buffering: *Pass in the the previous messages*

Note: 'messages' parameter is the history before the most recent user question

<https://community.openai.com/t/how-to-pass-conversation-history-back-to-the-api/697083>

THE CONVERSATION ALSO NEEDS TO BE DISPLAYED!

Remember:

The user expects to **see the history** –

→ Similar to the ‘messages’ to openAI, but in a visual way

The background of the slide features a light gray gradient with several realistic water droplets of various sizes. Some droplets are at the top, some at the bottom, and one large one is on the right side. They have highlights and shadows, giving them a 3D appearance.

STREAMLIT & CHATBOTS

Basics

- **st.chat_message** lets you insert a chat message container into the app so you can display messages from the user or the app (LLM).
- **st.chat_input** lets you display a chat input widget so the user can type in a message.
- Use **st.status** to display output from long-running processes and external API calls.

USING STREAMLIT TO CREATE A CHATBOT

`st.chat_message` (and `st.write`)

```
import streamlit as st

with st.chat_message("user"):
    st.write("Hello 🙋")
```



We could have used `"assistant"` rather than `"user"`

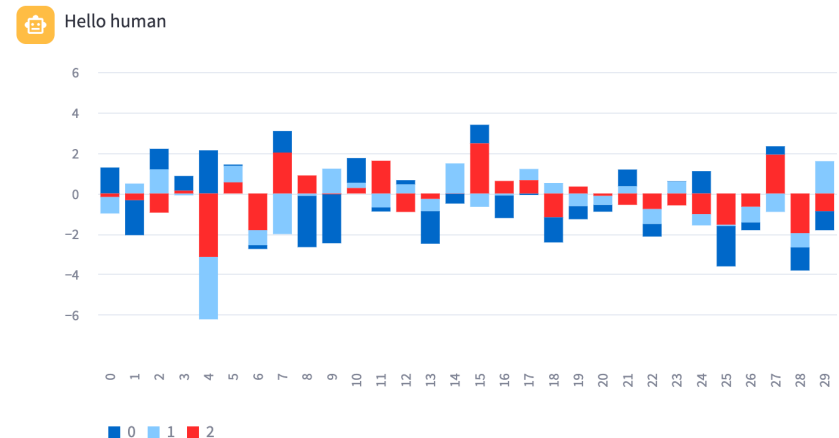
We could have used `markdown` rather than `write`

USING STREAMLIT TO CREATE A CHATBOT

```
import streamlit as st
import numpy as np

with st.chat_message("assistant"):
    st.write("Hello human")
    st.bar_chart(np.random.randn(30, 3))
```

Now, we did use "assistant"



USING STREAMLIT TO CREATE A CHATBOT

```
import streamlit as st
import numpy as np

with st.chat_message("assistant"):
    st.write("Hello human")
    st.bar_chart(np.random.randn(30, 3))
```

These are the same

```
import streamlit as st
import numpy as np

message = st.chat_message("assistant")
message.write("Hello human")
message.bar_chart(np.random.randn(30, 3))
```

USING STREAMLIT TO GET USER INPUT

`st.chat_input`

```
import streamlit as st

prompt = st.chat_input("Say something")
if prompt:
    st.write(f"User has sent the following prompt: {prompt}")
```

Say something



The background of the slide is a light gray gradient. It is decorated with several realistic water droplets of various sizes. Some droplets are at the top left, some are in the middle right, and others are at the bottom. They have highlights and shadows, giving them a 3D appearance.

A SIMPLE BOT THAT REPEATS THE USERS INPUT

A BOT THAT REPEATS THE USER'S INPUT

Check to see if:

- `messages` is in `st.session_state`
 - If it's not, initialize to empty list.
- **Don't want to overwrite the list every time the app reruns**

Loop to iterate through the chat history:

- Display each message
 - In chat message container
 - With role and content

```
import streamlit as st

st.title("Echo Bot")

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

# Display chat messages from history on app rerun
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])
```

But this doesn't collect user input yet...

A BOT THAT REPEATS THE USER'S INPUT

Use `st.chat_input...`

```
# React to user input
if prompt := st.chat_input("What is up?"):
    # Display user message in chat message container
    with st.chat_message("user"):
        st.markdown(prompt)
    # Add user message to chat history
    st.session_state.messages.append({"role": "user", "content": prompt})
```

Using the `:=` operator to assign the user's input to the `prompt` variable
→ this line also checked to make sure it's not `None` in the same line.

When the user has sent a message, display the message in the chat message container
→ **Also append it to the chat history**

<https://docs.streamlit.io/develop/tutorials/llms/build-conversational-apps>

A BOT THAT REPEATS THE USER'S INPUT

Now have the bot answer
(by just repeating the input)

```
response = f"Echo: {prompt}"  
# Display assistant response in chat message container  
with st.chat_message("assistant"):  
    st.markdown(response)  
# Add assistant response to chat history  
st.session_state.messages.append({"role": "assistant", "content": response})
```

Add the chatbot's response within the if block.

Use the same logic as before to display the bot's response (which is just the user's prompt) in the chat message container and add it to the history.

Initialize chat history

```
if "messages" not in st.session_state:  
    st.session_state.messages = []
```

Display chat messages from history on app rerun

```
for message in st.session_state.messages:  
    with st.chat_message(message["role"]):  
        st.markdown(message["content"])
```

React to user input

```
if prompt := st.chat_input("What is up?"):
```

Display user message in chat message container

```
st.chat_message("user").markdown(prompt)
```

Add user message to chat history

```
st.session_state.messages.append({"role": "user", "content": prompt})
```

```
response = f"Echo: {prompt}"
```

Display assistant response in chat message container

```
with st.chat_message("assistant"):  
    st.markdown(response)
```

Add assistant response to chat history

```
st.session_state.messages.append({"role": "assistant", "content": response})
```

The background of the slide is a light gray gradient. It is decorated with several realistic water droplets of various sizes. Some droplets are at the top left, some are at the bottom right, and others are scattered in the center. Each droplet has a highlight and a shadow, giving it a three-dimensional appearance.

STREAMING AN ANSWER

A BOT THAT STREAMS THE ANSWER

Need to import: time, random

```
# Streamed response emulator
def response_generator():
    response = random.choice(
        [
            "Hello there! How can I assist you today?",
            "Hi, human! Is there anything I can help you with?",
            "Do you need help?",
        ]
    )
    for word in response.split():
        yield word + " "
        time.sleep(0.05)
```

- time.sleep create the 'type writer' streaming of text
- random.choice picks one of the three answers

A BOT THAT STREAMS THE ANSWER

```
# Display assistant response in chat message container  
with st.chat_message("assistant"):  
    response = st.write_stream(response_generator())  
# Add assistant response to chat history  
st.session_state.messages.append({"role": "assistant", "content": response})
```

- Use `st.write_stream` to output the text
- Still need to store the response in 'messages'

The background of the slide is a light gray gradient. It is decorated with several realistic water droplets of various sizes. Some droplets are at the top, some at the bottom, and some on the right side. They have highlights and shadows, giving them a three-dimensional appearance.

**TURNING THIS INTO A REAL
CHATBOT**

A BOT THAT STREAMS THE ANSWER

```
import streamlit as st
from openai import OpenAI

# Show title and description.
st.title("MY Lab3 question answering chatbot")

openAI_model = st.sidebar.selectbox("Which Model?",
                                    ("mini", "regular"))

if openAI_model == "mini":
    model_to_use = "gpt-4o-mini"
else:
    model_to_use = "gpt-4o"

# Create an OpenAI client.
if 'client' not in st.session_state:
    api_key = st.secrets["OPENAI_API_KEY"]
    st.session_state.client = OpenAI(api_key=api_key)

if "messages" not in st.session_state:
    st.session_state["messages"] = \
        [{"role": "assistant", "content": "How can I help you?"}]
```

A BOT THAT STREAMS THE ANSWER (PART 2)

```
for msg in st.session_state.messages:  
    #st.chat_message(msg["role"]).write(msg["content"])  
  
    #with st.chat_message(msg["role"]):  
    #    st.write(msg["content"])  
  
    chat_msg = st.chat_message(msg["role"])  
    chat_msg.write(msg["content"])
```

A BOT THAT STREAMS THE ANSWER (PART 3)

```
if prompt := st.chat_input("What is up?"):
    st.session_state.messages.append({"role": "user", "content": prompt})

    with st.chat_message("user"):
        st.markdown(prompt)

    client = st.session_state.client
    stream = client.chat.completions.create(
        model=model_to_use,
        messages = st.session_state.messages,
        stream=True)

    with st.chat_message("assistant"):
        response = st.write_stream(stream)

    st.session_state.messages.append({"role": "assistant", "content": response})
```

LAB 3A

Create a streaming chatbot

- Create a new Lab3.py file in the Labs folder (copy Lab2).
- Add it to the main Streamlit Lab app as a new page.
- Create a chatbot using streamlit and OpenAI