

# Lang Chain memory

---

# Contents

- `OpenAI()` vs `ChatOpenAI()`
- `ConversationBufferMemory()`
- Human, AI and System Message
- `ConversationSummaryMemory`
- `ConversationEntityMemory`
- Simple Chat-Bot App
- `ConversationalRetrievalChain`
- APP- ChatBot talk to Multiple Documents

# A Chat-bot using LangChain

- To build a Chat-bot we need memory.
- We need memory to store the conversations.
- We need memory refer to previous conversations
- We need memory to combine several previously given inputs to produce the current output

```
llm= OpenAI(temperature=0, max_tokens=512)

print(llm.invoke("When did India win the t20 cricket World Cup? "))
print(llm.invoke("Give me some players in that team"))
```

India won the t20 cricket World Cup in 2007.

1. Lionel Messi
2. Sergio Aguero
3. Angel Di Maria
4. Paulo Dybala
5. Nicolas Otamendi
6. Gonzalo Higuain
7. Javier Mascherano
8. Marcos Rojo
9. Ever Banega
10. Mauro Icardi
11. Leandro Paredes
12. Lucas Biglia
13. Eduardo Salvio
14. Cristian Pavon
15. Willy Caballero

# OpenAI() vs ChatOpenAI()

Feature	OpenAI()	ChatOpenAI()
Primary Usage	General-purpose language model API	Specialized and Optimized for chat-based models and Conversations
Integration	Can be integrated into a broader range of applications	Primarily designed for conversational interfaces
Functionality	Broad (text generation, code completion, etc.)	Focused on maintaining context and conversation state
Use Case Examples	Content creation, data analysis, educational tools	Chatbots, customer service assistants, virtual aides
Response Handling	Responses are independent unless specifically managed	Handles context and dialogues over multiple exchanges

# ConversationBufferMemory()

- Simplest form of memory
- The most frequently used memory in LLM based chatbots
- Suitable for small conversations that span up to 10 conversations

```
from langchain_community.chat_models import ChatOpenAI
from langchain.memory import ConversationBufferMemory
from langchain.chains import ConversationChain
```

```
llm= ChatOpenAI(temperature=0, max_tokens=512)
```

```
memory = ConversationBufferMemory()
```

```
conversation = ConversationChain(llm=llm,
                                memory=memory,
                                #verbose=True
                                )
```

```
print(conversation.predict(input="When did India win the t20 cricket World Cup?"))
print(conversation.predict(input="Give me some players in that team"))
```

Instantiate

Result are connected

India won the ICC T20 Cricket World Cup in 2007 and 2011. They defeated Pakistan in the final. Some players in the 2007 and 2011 ICC T20 Cricket World Cup-winning Indian teams were MS Dhoni.

> Entering new ConversationChain chain...

Prompt after formatting:

*The following is a friendly conversation between a human and an AI. The AI is talkative and provides lots of*

*Current conversation:*

*Human: When did India win the t20 cricket World Cup?*

*AI:*

> Finished chain.

India won the ICC T20 Cricket World Cup in 2007 and then again in 2021. The first victory was in South Africa

> Entering new ConversationChain chain...

Prompt after formatting:

*The following is a friendly conversation between a human and an AI. The AI is talkative and provides lots of*

*Current conversation:*

*Human: When did India win the t20 cricket World Cup?*

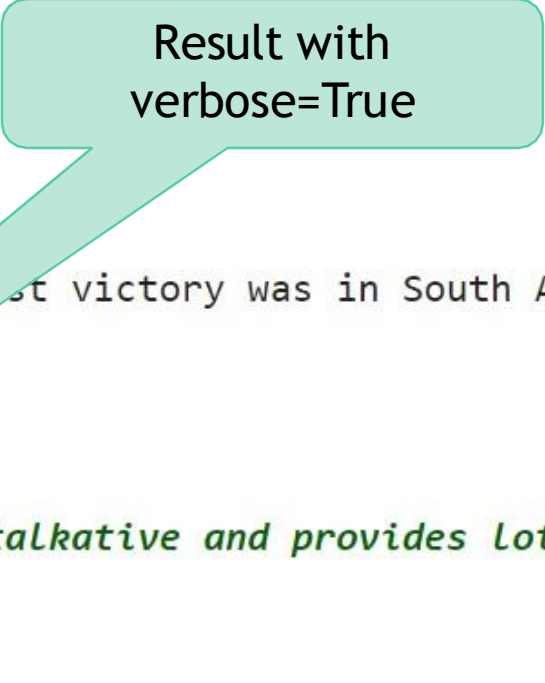
*AI: India won the ICC T20 Cricket World Cup in 2007 and then again in 2021. The first victory was in South Af*

*Human: Give me some players in that team*

*AI:*

> Finished chain.

Some players in the 2007 ICC T20 Cricket World Cup-winning team for India were Yuvraj Singh, Gautam Gambhir,



Result with  
verbose=True



# Human, AI and System Message

- **Human messages**

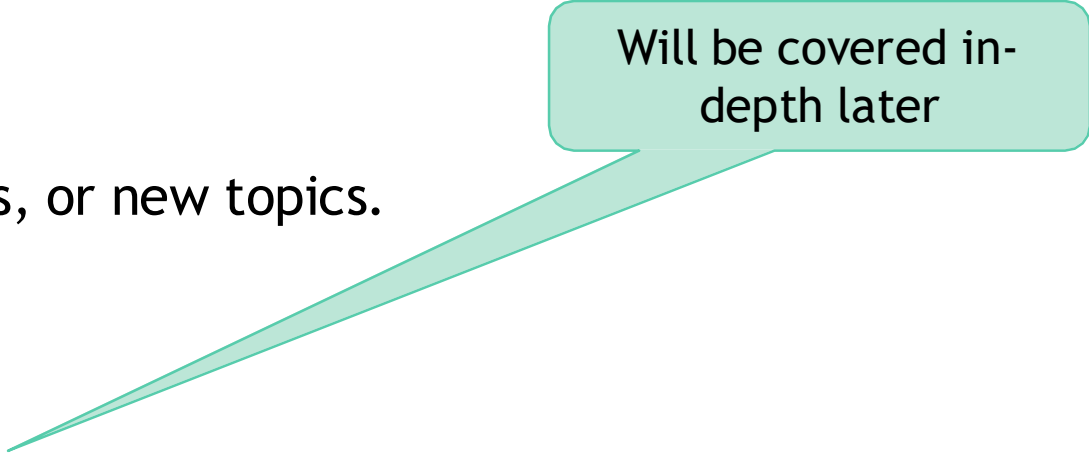
- User inputs like questions or commands.
- They are essential data inputs that AI processes and responds to.

- **AI messages**

- Model's responses to human inputs.
- These responses provide answers, clarifications, or new topics.

- **System messages**

- Neither human nor AI, aid user interaction.
- You can use these to assign a role to the model
- They include guidance, error messages, and status updates.
- Together, these messages create a seamless conversational flow.
- They ensure context relevance and timely system interactions.



Will be covered in-depth later

Memory buffer stores  
both human and AI  
messages

```
print(conversation.memory.buffer)
```

Human: When did India win the t20 cricket World Cup?

AI: India won the ICC T20 Cricket World Cup in 2007 and 2011. They defeated Pakistan in the

Human: Give me some players in that team

AI: Some key players in the 2007 and 2011 ICC T20 Cricket World Cup-winning Indian teams wer

# ConversationBufferMemory()

```
from langchain.memory import ConversationBufferMemory
from langchain.chains import ConversationChain
```

```
llm= OpenAI(temperature=0, max_tokens=512)
```

```
memory = ConversationBufferMemory()
```

```
conversation = ConversationChain(llm=llm, memory=memory)
conversation.predict(input="Hello ")
```

```
' Hello! It's nice to meet you. My name is A and I am an artificial
p you today?'
```

Instantiate

First message

# ConversationBufferMemory()

Second Message

```
conversation.predict(input="Explain three main differences Classification and Regression")
```

```
' The main differences between Classification and Regression are:\n\n- Classification is a type of s  
l or discrete values, while Regression predicts continuous numerical values.\n- Classification algo  
sion trees, support vector machines, and neural networks, while Regression algorithms use linear reg  
l regression.\n- The evaluation metrics for Classification and Regression are different. Classificat  
n, and recall, while Regression uses metrics such as mean squared error, mean absolute error, and R-
```

# ConversationBufferMemory()

Third Message

```
conversation.predict(input="Give the above output in markdown tabular format ")
```

```
' | Main Differences | Classification | Regression |\n| --- | --- | --- |\n| Type of Learning | Categorical or discrete | Continuous numerical |\n| Techniques | Decision trees, support vector machines, neural networks | Linear regression, logistic regression, polynomial regression |\n| Evaluation Metrics | Accuracy, precision, recall | Mean squared error, mean absolute error, R-squared |'
```

Uses Memory

Main Differences	Classification	Regression
Type of Learning	Supervised	Supervised
Predicted Values	Categorical or discrete	Continuous numerical
Techniques	Decision trees, support vector machines, neural networks	Linear regression, logistic regression, polynomial regression
Evaluation Metrics	Accuracy, precision, recall	Mean squared error, mean absolute error, R-squared



# ConversationBufferMemory()

Fourth Message

```
conversation.predict(input="Which one is most widely used in predicting customer churn?")
```

```
' Classification is most widely used in predicting customer churn. This is because it can classify cu  
d or not churned) based on their behavior and characteristics, making it easier for businesses to targ
```

Uses Memory

Fifth Message

```
conversation.predict(input="Give me one more difference")
```

```
' One more difference is that Classification can handle both numerical and  
data. This is because Regression algorithms require a continuous target var
```

Uses Memory

# ConversationBufferMemory()

```
print(conversation.memory.buffer)
```

The whole conversation is stored as a prompt in the backend memory buffer

Human: Hello

AI: Hello! It's nice to meet you. My name is AI and I am an artificial intelligence designed to assist

Human: Explain three main differences Classification and Regression

AI: Sure thing! Classification and regression are two different types of machine learning algorithms

Human: Give the above output in markdown tabular format

AI: Sure, here is the information in a tabular format:

Algorithm Type	Output	Data Type	Evaluation Metrics
Classification	Discrete or categorical values	Labeled data	Accuracy, precision
Regression	Continuous values	Numerical data	Mean squared error, R-squared

Human: Which one is most widely used in predicting customer churn?

AI: Classification algorithms are most commonly used for predicting customer churn. This is because

Human: Give me one more difference

AI: Another difference between classification and regression is the type of models they use. Classification

# ConversationBufferMemory() - Conclusion

- ConversationBufferMemory() offers a simple memory management approach
- The entire dialogue stored in the backend's memory buffer and used as a prompt later.
- As discussions extend over multiple messages, the backend prompt grows, potentially leading to errors or inaccuracies in lengthy conversations.
- It's most effective for brief interactions, ideally up to 10 user messages.



# Other Memory options

- **ConversationBufferWindowMemory:**
  - Keeps only last few(a window) of conversations in the memory buffer.
- **ConversationSummaryMemory:**
  - Keeps a summary of conversations instead of complete conversation.
- **ConversationSummaryBufferMemory:**
  - It's a mix of both. Keeps the overall summary and last few conversations in the memory buffer
- **ConversationKnowledgeGraphMemory a.k.a KGMemory**
  - It's a totally different concept. Utilizes a knowledge graph to store and organize conversation elements in a structured form
- **ConversationEntityMemory**
  - focuses on identifying and storing key entities (such as names, places, dates, and specific terms)
  - This one is the best among all of these; we're going to delve into it deeply.

# Other Memory options – Details

- **ConversationBufferWindowMemory:**

- Implements a sliding window mechanism over the conversation history, keeping only the most recent interactions in memory.
- This method ensures that the memory remains fresh and relevant to the current context, discarding older parts of the conversation that are no longer immediately relevant.

- **ConversationSummaryMemory:**

- This approach summarizes the entire conversation history, storing only the essence or key points of the dialogue.
- It helps in maintaining a compact and relevant context for ongoing interactions without the need for storing the full conversation text.

- **ConversationSummaryBufferMemory:**

- Combines the strategies of buffering recent interactions while summarizing older ones.
- This method keeps a live buffer of the most recent exchanges and compiles older interactions into a summary.
- It ensures a balance between memory efficiency and context retention, providing a comprehensive background for the conversation without overwhelming the memory with details.

- **ConversationKnowledgeGraphMemory:**

- Utilizes a knowledge graph to store and organize conversation elements in a structured form.
- This approach captures the relationships between different pieces of information mentioned throughout the conversation, allowing for more nuanced understanding and retrieval of context.
- It supports complex interactions by providing a dynamic map of the conversation's content and its interconnections.

# ConversationSummaryMemory

- This approach summarizes the entire conversation history, storing only the essence or key points of the dialogue.
- It helps in maintaining a compact and relevant context for ongoing interactions without the need for storing the full conversation text.
- The summary is also created by using LLM.
- Ideal for lengthy and long conversations

# ConversationSummaryMemory

```
from langchain.chains.conversation.memory import ConversationSummaryMemory
```

```
memory = ConversationSummaryMemory(llm=llm)
```

```
conversation = ConversationChain(llm=llm, memory=memory)  
conversation.predict(input="Hello, I need help with my order")
```

Actual conversation

```
' Hello! I am here to assist you with your order. Can you please provide me with yo
```

```
conversation.predict(input="I ordered a pizza")
```

```
' Great! Can I have your order number so I can pull up the details? It should be a  
ation email or on your receipt.'
```

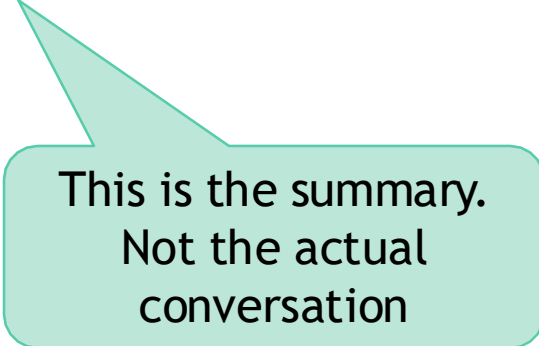
```
conversation.predict(input="I don't remember the order details")
```

```
' That's okay, I can help you find them. Do you have the order number? It should be
```

# ConversationSummaryMemory

```
print(conversation.memory.buffer)
```

The human needs help with their order and the AI offers assistance by asking for the order number



This is the summary.  
Not the actual  
conversation

# ConversationEntityMemory

- This method focuses on identifying and storing key entities (such as names, places, dates, and specific terms) mentioned throughout a conversation.
- This method ensures that important details are retained and easily accessible, enabling the system to reference or utilize these entities in future responses.
- It enhances the relevance and personalization of interactions by keeping track of significant information shared during the dialogue, without needing to remember the entire conversation verbatim.

# ConversationEntityMemory

```
template=ENTITY_MEMORY_CONVERSATION_TEMPLATE.template  
print(template)
```

You are an assistant to a human, powered by a large language model trained by OpenAI.

You are designed to be able to assist with a wide range of tasks, from answering simple question

You are constantly learning and improving, and your capabilities are constantly evolving. You ar

Overall, you are a powerful tool that can help with a wide range of tasks and provide valuable i

Context:  
{entities}

Current conversation:  
{history}

Last line:

Human: {input}

You:

# ConversationEntityMemory

```
template=ENTITY_MEMORY_CONVERSATION_TEMPLATE.template  
print(template)
```

You are an assistant to a human, powered by a large language model trained by OpenAI.

You are designed to be able to assist with a wide range of tasks, from answering simple question

You are constantly learning and improving, and your capabilities are constantly evolving. You ar

Overall, you are a powerful tool that can help with a wide range of tasks and provide valuable i

Context:  
{entities}

Current conversation:  
{history}

Last line:

Human: {input}

You:



```
memory = ConversationEntityMemory(llm=llm)
conversation = ConversationChain(llm=llm,
                                memory=memory,
                                verbose=True,
                                prompt=ENTITY_MEMORY_CONVERSATION_TEMPLATE)
conversation.predict(input="Hello, My Name is Venkat. I am from Bangalore. I need help with my order")
```

> Entering new ConversationChain chain...

Prompt after formatting:

*You are an assistant to a human, powered by a large language model trained by OpenAI.*

*You are designed to be able to assist with a wide range of tasks, from answering simple questions to providing in-depth explanations and discuss*

*You are constantly learning and improving, and your capabilities are constantly evolving. You are able to process and understand large amounts*

*Overall, you are a powerful tool that can help with a wide range of tasks and provide valuable insights and information on a wide range of topi*

Context:

*{'Venkat': '', 'Bangalore': ''}*

Current conversation:

Last line:

*Human: Hello, My Name is Venkat. I am from Bangalore. I need help with my order*

*You:*

> Finished chain.

*' Hello Venkat, it's nice to meet you. I am here to assist you with your order. Can you please provide me with more details about your order so  
lp you?'*

```
conversation.predict(input="I ordered Mobile phone on tuesday night 8pm")
```

```
> Entering new ConversationChain chain...
```

```
Prompt after formatting:
```

```
You are an assistant to a human, powered by a large language model trained by OpenAI.
```

```
You are designed to be able to assist with a wide range of tasks, from answering simple questions to providing in-depth explanations and
```

```
You are constantly learning and improving, and your capabilities are constantly evolving. You are able to process and understand large
```

```
Overall, you are a powerful tool that can help with a wide range of tasks and provide valuable insights and information on a wide range
```

```
Context:
```

```
{'Mobile phone': ''}
```

```
Current conversation:
```

```
Human: Hello, My Name is Venkat. I am from Bangalore. I need help with my order
```

```
AI: Hello Venkat, it's nice to meet you. I am here to assist you with your order. Can you please provide me with more details about your
```

```
Last line:
```

```
Human: I ordered Mobile phone on tuesday night 8pm
```

```
You:
```

```
> Finished chain.
```

```
' Thank you for providing me with that information. Can you please tell me which specific mobile phone you ordered and from which website you ordered it, so I can provide you with more accurate assistance.'
```

# ConversationEntityMemory

```
conversation.memory.buffer
```

```
[HumanMessage(content='Hello, My Name is Venkat. I am from Bangalore. I need help with my order'),  
 AIMessage(content=" Hello Venkat, it's nice to meet you. I am here to assist you with your order. Can  
details about your order so I can better understand how to help you?"),  
 HumanMessage(content='I ordered Mobile phone on tuesday night 8pm'),  
 AIMessage(content=' Thank you for providing me with that information. Can you please tell me which s;  
from which website or store? This will help me provide you with more accurate assistance.'),  
 HumanMessage(content='My order number is IBLC19678456. What is the status of my order?'),  
 AIMessage(content=' Thank you for providing me with your order number. Let me check the status of you  
order is currently being processed and is expected to be shipped within the next 24 hours. Is there an  
with?')] ]
```

# ConversationEntityMemory

```
conversation.memory.entity_store
```

```
InMemoryEntityStore(store={'Venkat': 'Venkat is from Bangalore and needs help with their order.', 'Bangalore': 'Bangalore is the location where the human, Venkat, is from.', 'Mobile phone': 'The human ordered a mobile phone on Tuesday night at 8pm.', 'IBLC19678456': 'The order number is IBLC19678456 and the current status is being processed and expected to be shipped within the next 24 hours.'})
```

# Simple Chat-Bot App

---



```
user_input=input("Your message:")
llm= ChatOpenAI(temperature=0, max_tokens=512)
memory = ConversationBufferMemory()
conversation = ConversationChain(llm=llm,
                                memory=memory,
                                verbose=False
                                )

while user_input!="quit":
    print("AI message ==>", conversation.predict(input=user_input))
    user_input=input("Your message: ")
while user_input == "quit":
    memory.clear()
    break
```

More complex  
conversational apps in  
upcoming sessions

Your message:Hello

AI message ==> Hello! How are you today?

Your message: I want to learn python for working with AI

AI message ==> That's great! Python is a popular programming language for AI

Your message:

# ConversationalRetrievalChain

---

# ConversationalRetrievalChain

- The chain allows conversations with a document via user questions.
- Optional previous history can refine the query for better responses.
- Without history, the system uses only the latest user input.
- An LLM rephrases the conversation into a retriever-friendly query.
- Retrieved documents, along with the conversation, help generate responses.



# APP- ChatBot talk to Multiple Documents

---



# Thank you

---