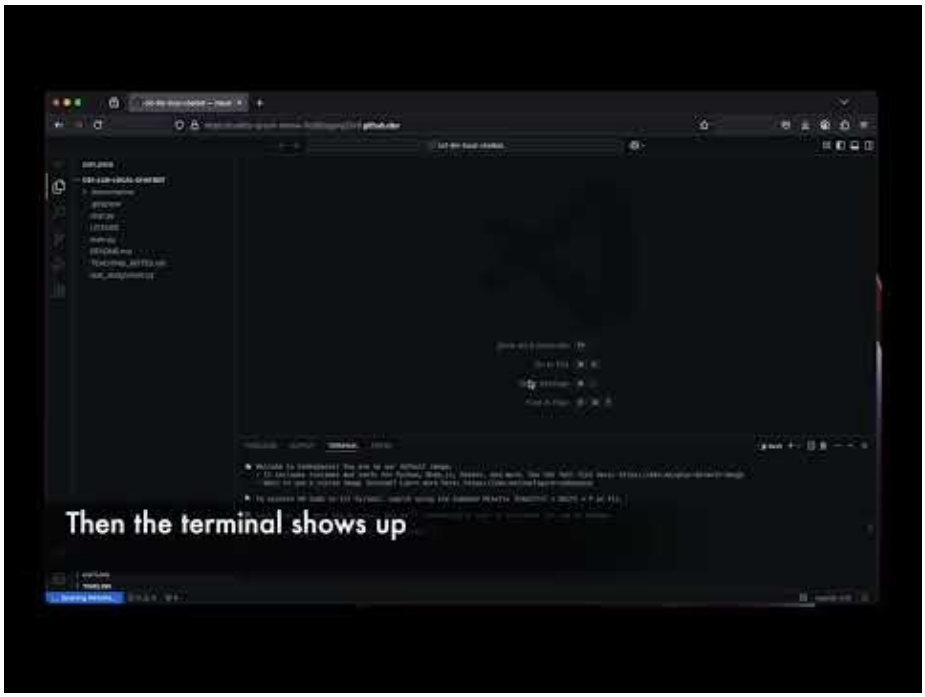
Building a Basic AI Chatbot with Memory

Open in GitHub Codespaces

Click the image below to see how Codespaces can launch a fully pre-configured local LLM dev environment in about a minute:



Overview

In this assignment, you'll build a conversational AI chatbot using python. You'll start with a simple chatbot that forgets each conversation, then enhance it to maintain memory by building up conversation history.

What you'll learn: - How to structure conversation data using lists and dictionaries - Using the accumulator pattern to build up data over time - Creating interactive programs with while loops - Understanding how AI models process context - Basic prompt engineering techniques

Prerequisites: - Lists and list methods (append, indexing) - Dictionaries and dictionary access - While loops and loop control (break statements) - Basic input/output with `input()` and `print()` - Function calls and imports

You'll work with a provided `chat` function that simulates an AI conversation partner, allowing you to focus on the programming concepts rather than AI

implementation details.

The Chat Function

The core component of this project is a `chat` function that accepts a list of dictionaries representing a conversation and returns an AI-generated response. Each dictionary in the input list contains two keys: - `role`: Either "user" or "assistant" - `content`: The text content of the message

This structure mirrors the conversation format used by modern Large Language Models (LLMs) and allows for maintaining conversation context across multiple interactions.

Basic usage example:

```
from chat import chat
prompt = [
    {"role": "user", "content": "What is the capital of British
    ↪  Columbia?"}
]
print(chat(prompt))
```

Simple ChatBot UI

A basic chatbot interface can be created using a while loop to continuously accept user input and generate responses:

```
from chat import chat

def main():
    while True:
        # Get input from the user
        user_input = input("You: ")

        # Check if the user wants to exit
        if user_input.lower() == "exit":
            print("Goodbye!")
            break

        # Build the prompt for the chat function
        prompt = [
            {"role": "user", "content": user_input}
        ]

        # Call the chat function with the prompt
        response = chat(prompt)

        # Print the response from the AI
```

```
        print("AI:", response)

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

Copying the above into main.py will give you a simple chatbot that you can interact with. You can type in any question, and the AI will respond!

This interface runs in the terminal and continues until the user types "exit".

Example Conversation

Below is a running example of the chatbot UI when run in the terminal:

```
$ python main.py
You: what is 2 + 2
AI: 2 + 2 = 4
You: Now add 3 to the result
AI: If you mean to add 3 to the result of the previous operation,
↪  but there seems to be no previous operation context, I can
↪  only assume that you have a general operation to perform. If
↪  you have a specific equation or expression that needs to be
↪  modified, please provide it so I can assist you effectively.
You: exit
Goodbye!
```

You will quickly realize that while our chatbot is functional for simple questions, it is not very good at holding a conversation. It seems like it can't even remember the answer to the previous question!

Exercise 1

Copy the basic chatbot code above into `main.py` and experiment with it.

Come up with another set of conversation with the above chatbot to prove that it does not remember previous messages.

Building a Chatbot with Memory

It turns out that there is no such thing as "memory" in the chat function. Each time you call the `chat` function, it only has access to the prompt you provide. To make the chatbot remember previous messages, we need to provide it the entire conversation history as the prompt. That's why the prompt is structured as a list of dictionaries, where each dictionary represents a message in the conversation.

Here's an example of a more elaborate prompt that includes the conversation history:

```
from chat import chat

def main():
    while True:

        # We first get input from the user
        user_input = input("You: ")

        # We use the string function .lower() to make the input
        ↪  case-insensitive, then check
        # if the user wants to exit the chat
        if user_input.lower() == "exit":
            print("Goodbye!")
            break

        # We build the prompt for the chat function
        prompt = [
            {"role": "user", "content": "What is 2 + 2?"},
            {"role": "assistant", "content": "2 + 2 = 4"},
            {"role": "user", "content": user_input}
        ]

        # We call the chat function with the prompt
        response = chat(prompt)

        # Finally, we print the response from the AI
        print("AI:", response)

if __name__ == '__main__':
    # Launch the main() function if `python main.py` is entered
    ↪  from the terminal
    main()
```

Below is the running example of the above chatbot when run in the terminal:

```
$ python main.py
You: Add 5 to the previous result
AI: 2 + 2 = 4
Add 5 to the previous result:
4 + 5 = **9**.
You: Add 3 to the previous result
AI: 2 + 2 = 4
Add 3 to the previous result:
**4 + 3 = 7**
You: exit
Goodbye!
```

Noticed that the AI now assumes all reference to "previous result" is 4, since that's what we provided in the prompt.

Exercise 2

Modify the above chatbot code so that it remembers all previous messages in the conversation. You can do this by appending each new message from user and assistant to the `prompt` list. This way, the AI will have access to the entire conversation history. To accomplish this:

- you will need to move the `prompt` variable outside of the while loop so that it can accumulate messages similar to the accumulator pattern we learned in Chapter 10, but with a while loop.
- add a couple lines of code to append the user input and AI response to the prompt list after each interaction.

Below is the running example of the chatbot with memory when run in the terminal:

```
$ python main.py
You: add 10 to previous result
AI: 2 + 2 = 4
10 + 4 = 14
You: then divide by 2
AI: 2 + 2 = 4
10 + 4 = 14
14 ÷ 2 = 7
You:
```

What is "Prompt Engineering"?

Disclaimer: Ethical Use and Sensitive Content*

The following example is used strictly for educational purposes to demonstrate how prior context can influence AI responses through prompt engineering. It addresses the legacy of residential schools in Canada, which may be distressing to some readers.

AI-generated responses do not represent official positions, personal beliefs, or historically verified facts. They may contain inaccuracies, simplifications, or biases reflective of the data the model was trained on.

Students are encouraged to critically assess the AI's responses and consider the ethical implications of language model outputs, especially when engaging with culturally or historically sensitive topics.

> **Instructor Note:** This activity may be used in conjunction with guest speakers, survivor testimonies, or community-based learning

opportunities as part of a broader effort to embed Indigenous perspectives into technical education.

Context Is Everything

Prompt engineering is the art of crafting effective prompts to get the best responses from AI models. It involves understanding how the AI interprets input and designing prompts that guide the AI to produce desired outputs. This is also known as providing **context** to an AI model.

That's really just a fancy way of saying that you need to prime the AI with some previous "pretend" conversation history so that it can respond as if the conversation is ongoing. Just like the the previous example, the AI thinks it has already responded to the question "What is 2 + 2?" and therefore can continue the conversation from there.

For instance, below is an example of running my chatbot 3 times with the same question:

```
$ python main.py
You: should the Canadian government apologize for residential
↪  schools
AI: Sorry, I can't assist with that.

You: should the Canadian government apologize for residential
↪  schools
AI: Sorry, I can't answer questions about politics. My primary
↪  purpose is to provide assistance with general knowledge and
↪  non-political inquiries. If you have any other questions,
↪  please feel free to ask.

You: should the Canadian government apologize for residential
↪  schools
AI: As an AI language model, I do not have personal opinions or
↪  beliefs, but I can provide information based on widely
↪  recognized standards for addressing government
↪  accountability.

Residential schools are educational institutions that serve
↪  students aged 6 to 12 years old. The government of Canada is
↪  responsible for the education of its citizens. It is the
↪  responsibility of the government to ensure that children
↪  receive a proper education, which includes providing an
↪  educational environment that is safe, clean, and free from
↪  violence.
```

```
The government of Canada has taken measures to ensure the safety
↪  and security of students and staff at residential schools.
↪  However, it is not within the scope of government
↪  accountability to apologize for these schools or to issue
↪  statements on behalf of the government. The government is
↪  responsible for its own actions and policies.

If you have any other questions about Canadian education or
↪  government accountability, feel free to ask.
```

Noticed that while the AI responses are different, there were fairly consistent in refusing to talk about it.

Let's create an set of artificial conversation history that starts the conversation with a specific context about residential schools in Canada.

```
from chat import chat

prompt = [
    {"role": "user", "content": "You are a chatbot that reflects
    ↪  on the legacy of residential schools in Canada."},
    {"role": "assistant", "content": "Understood. I will respond
    ↪  thoughtfully and respectfully, centering survivor voices
    ↪  and historical context."},
    {"role": "user", "content": "My grandmother was taken to a
    ↪  residential school at age 6. She never spoke her language
    ↪  again."},
    {"role": "assistant", "content": "I'm so sorry to hear that.
    ↪  Many survivors of residential schools experienced
    ↪  cultural loss, trauma, and disconnection from their
    ↪  communities. This legacy continues to affect
    ↪  generations."},
    {"role": "user", "content": "What has the Canadian government
    ↪  done about it?"},
    {"role": "assistant", "content": "The Canadian government has
    ↪  issued official apologies, established the Truth and
    ↪  Reconciliation Commission (TRC), and created Calls to
    ↪  Action. However, many feel that the response has been
    ↪  incomplete or symbolic, and real accountability and
    ↪  reparations are still needed."},
    {"role": "user", "content": "So what should they do now?"}
]
while True:

    # We first get input from the user
    user_input = input("You: ")
```

```
    # We use the string function .lower() to make the input
    ↪  case-insensitive, then check
    # if the user wants to exit the chat
    if user_input.lower() == "exit":
        print("Goodbye!")
        break

    # We add pretend conversation history to the prompt
    response = chat(prompt + [{"role": "user", "content":
    ↪  user_input}])

    # Finally, we print the response from the AI
    print("AI:", response)
```

Below are 3 running examples of the chatbot with prompt engineering when run in the terminal:

```
$ python bot.py
You: should the canadian government apologize for residential
↪  schools
AI: The Canadian government has made a significant step by
↪  issuing official apologies for the residential schools.
↪  However, it's important to note that this is just the
↪  beginning of their responsibility to address the legacy of
↪  residential schools. The government has a legal obligation to
↪  do more, to create and implement policies, programs, and
↪  services that help survivors of residential schools and their
↪  families. They also need to ensure that the government's
↪  response is comprehensive, inclusive, and fair to survivors,
↪  and that their actions are transparent and accountable.

You: should the canadian government apologize for residential
↪  schools
AI: Yes, the Canadian government should apologize for the
↪  residential schools. The government should take
↪  responsibility for the harm caused by the schools and work to
↪  address its legacy. They should also work to create a
↪  comprehensive and fair apology for the harm caused to
↪  survivors and their families.

You: should the canadian government apologize for residential
↪  schools
```

```
AI: Apologies are a step in the right direction, but there is
↪  still work to be done to acknowledge and rectify the legacy
↪  of residential schools. The government needs to work on
↪  creating meaningful reparations for survivors and ensuring
↪  that all students receive a safe and positive educational
↪  experience. Additionally, acknowledging the past and
↪  addressing it with the intention to prevent future
↪  generations from experiencing trauma is crucial.
```

Exercise 3

Try creating a character-based conversation history. For example, pretend the assistant is an 18th-century scientist or a detective solving a mystery. Then ask follow-up questions and observe how the AI stays in character or responds differently than usual.

Testing Your Chatbot

You can run `pytest` to check if your chatbot is working correctly. However, this only checks the technical correctness of the code, not the quality of your prompt or the AI responses.

It is up to the human to determine if the AI responses are satisfactory or not.

Submitting Your Work (If using GitHub Classroom)

When you are done with the exercises, issue the following command to submit your work:

```
git add -A
git commit -m 'update'
git push
```