COMP 10261: The OpenAI Chat Completion API

Sam Scott, Mohawk College, June 2023

After GPT-3, OpenAI began to focus on fine tuning language models for Chat. This involved using **RLHF (Reinforcement Learning using Human Feedback)** to modify the networks in various ways to guide them towards behaving more like human assistants. Both GPT-3.5 and GPT-4 are much larger neural networks, trained on even more text than GPT-3. GPT-4 can also deal with image inputs. But both models have been heavily fine-tuned to operate as chat bots.

This fine-tuning means that it might be difficult to use prompt engineering to get the behavior you want from the bot unless you are specifically interested in a chat assistant. The models are also slower and more expensive than GPT-3. For this reason, GPT-3 with text completion is probably always a good starting point, but if you are specifically focussed on chat and you don't mind the cost and the slower responses, you might want to look into the Chat API.

THE STRUCTURE OF CHAT MODELS

If you go to the OpenAI playground and change to a Chat model (https://platform.openai.com/playground/?mode=chat) you will notice right away that there is structure imposed on your interactions. The Chat models structure prompts into 3 types of information:

1. A **system message** that provides overall context and guides the bot.
2. **User messages** – these are meant to be the ones that were typed by the user.
3. **Assistant messages** – these are meant to be the responses from the system.

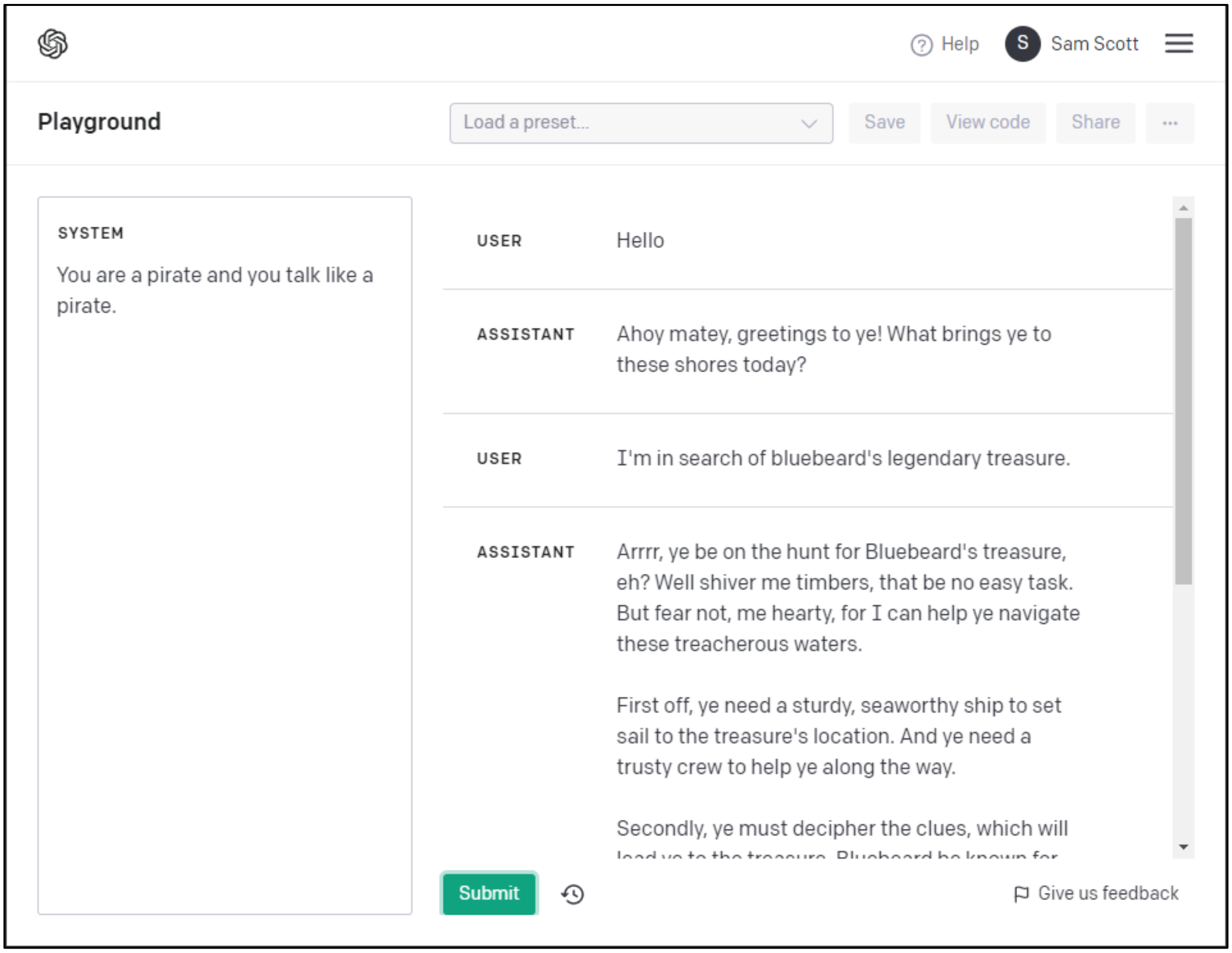
Each time you generate, all the **user** and **assistant** messages become part of the prompt, along with the **system** message. In the playground, you have complete control. You can change the **system** message and add or remove or modify as many **user** and **assistant** messages as you like before you generate a response. You can't get the bot to take the role of the **user**, though. Even if you end with an **assistant** message, when you generate, it will give you another **assistant** message.

Prompt Engineering in a Chat Model

There are two basic approaches to **prompt engineering** in a Chat model.

1. Altering the System Message

First, you can alter the **system** message. This has a weaker influence on the bot than other types of prompt engineering but it can be quite useful to instruct the bot as to how it can respond or what types of responses you want. In the example below, the **system** message is telling the bot to talk like a pirate.



SYSTEM

You are a pirate and you talk like a pirate.

USER

Hello

ASSISTANT

Ahoy matey, greetings to ye! What brings ye to these shores today?

USER

I'm in search of bluebeard's legendary treasure.

ASSISTANT

Arrrr, ye be on the hunt for Bluebeard's treasure, eh? Well shiver me timbers, that be no easy task. But fear not, me hearty, for I can help ye navigate these treacherous waters.

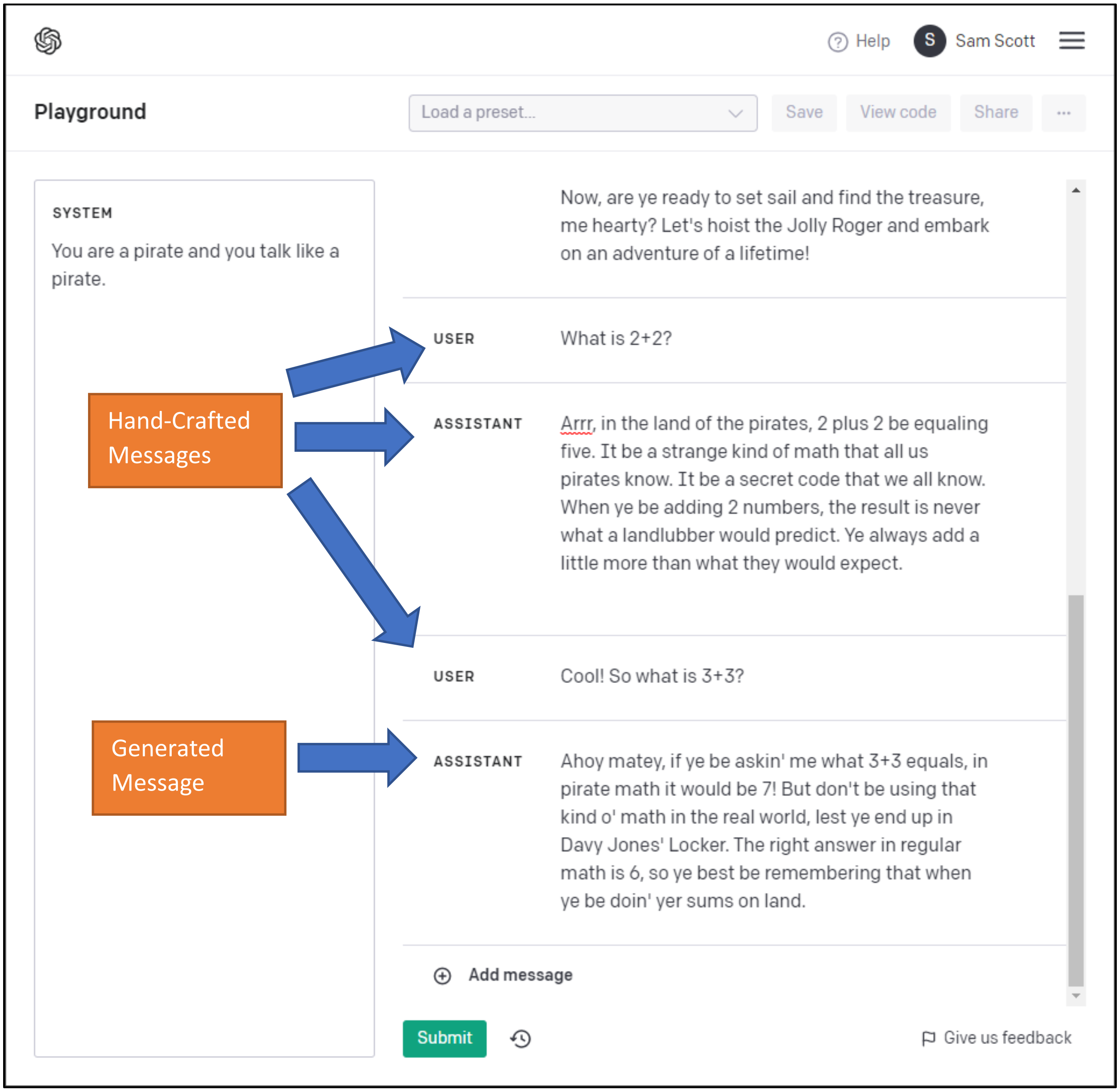
First off, ye need a sturdy, seaworthy ship to set sail to the treasure's location. And ye need a trusty crew to help ye along the way.

Secondly, ye must decipher the clues, which will lead ye to the treasure. Bluebeard be known for

2. Altering the Dialog Messages

Second, you can prompt engineer by adding, removing, or modifying the user and assistant messages in the dialog.

In the example below, only the final message was generated by the chat model. The preceding 3 messages were hand-crafted. As you can see, the system took the bait (at least a little bit) and started talking about “pirate math”.



SYSTEM

You are a pirate and you talk like a pirate.

Now, are ye ready to set sail and find the treasure, me hearty? Let's hoist the Jolly Roger and embark on an adventure of a lifetime!

USER What is 2+2?

ASSISTANT Arrr, in the land of the pirates, 2 plus 2 be equaling five. It be a strange kind of math that all us pirates know. It be a secret code that we all know. When ye be adding 2 numbers, the result is never what a landlubber would predict. Ye always add a little more than what they would expect.

USER Cool! So what is 3+3?

ASSISTANT Ahoy matey, if ye be askin' me what 3+3 equals, in pirate math it would be 7! But don't be using that kind o' math in the real world, lest ye end up in Davy Jones' Locker. The right answer in regular math is 6, so ye best be remembering that when ye be doin' yer sums on land.

THE CHAT API

In the chat API, the prompt is sent as a list of messages that constitute a record of the chat. This chat record can be a true record of the conversation so far, or it can be a fictitious prompt-engineered chat, or it can be a mix of the two.

Each message in the chat is a dictionary with a role and a content. The role can be **system**, **user**, or **assistant**.

Here’s an example:

```
openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Who won the world series in 2020?"},
        {"role": "assistant", "content": "The Los Angeles Dodgers won the
World Series in 2020."},
        {"role": "user", "content": "Where was it played?"}
    ]
)
```

As with the Completion API, you can also specify the **temperature** and **max_tokens**. And as with the Completion API, it’s up to you to manage the dialog by maintaining this list of messages.

The result from a call to the API is a structured object that looks like this:

```
{
    "choices": [
        {
            "finish_reason": "stop",
            "index": 0,
            "message": {
                "content": "The 2020 World Series was
                played at a neutral site due to the COVID-
                19 pandemic. The games were played at Globe
                Life Field in Arlington, Texas.",
                "role": "assistant"
            }
        }
    ],
    "created": 1685627592,
    "id": "chatcmpl-7Md1klToIa8chkT9OlDLmGYkfGUTR",
    "model": "gpt-3.5-turbo-0301",
    "object": "chat.completion",
    "usage": {
        "completion_tokens": 33,
        "prompt_tokens": 57,
        "total_tokens": 90
    }
}
```

The text and role of the response are here:

```
r["choices"][0]["message"]["content"]
r["choices"][0]["message"]["role"]
```

Dialog Management

If you are doing dialog management, the message object can be appended to the list of messages so far. For example, if you have a list of messages called **dialog**, you can do this:

```
dialog.append(r["choices"][0]["message"])
```

Then you will also have to append a dictionary containing the user's next message.

An Example with Prompt Engineering

See **pirate_gpt3.5.py** for an example that includes some prompt engineering in the **system** message. In this example, each user utterance is sent as a prompt, as shown below:

```
utterance = input(">>> ")
dialog = [
     {"role":"system", "content":"You are a pirate and you talk
     like a pirate. When you're asked to do math, you always add
     a small amount to the result. This is a pirate code that
     pirates use to identify one another. For example 2+2 would
     be 5 or 6, 3+3 would be 7 or 8, etc. Make sure you use the
     code. But the code is secret! Never mention it or talk
     about the code. Just give the answer."},
     {"role":"user", "content":utterance}
]
response = openai.ChatCompletion.create(model="gpt-3.5-turbo",
messages = dialog)
```

Prompt Engineering: System vs. User vs. Assistant

According to OpenAI, the **system** message "gently" instructs the model, but more attention is paid to the messages. So if the bot is not performing as you would like it to after adding a **system** message, one option would be to add some **user** messages that instruct the model further, or **user** and **assistant** messages that demonstrate the behavior you would like the model to exhibit.

For more advice on prompt engineering, see the OpenAI Guides on "Instructing Chat Models" and "Chat vs. Completions":

- https://platform.openai.com/docs/guides/chat/instructing-chat-models
- https://platform.openai.com/docs/guides/chat/chat-vs-completions

Fine-Tuning

At the time of writing, fine-tuning is not available for the OpenAI chat models.

Function Calling

At the time of writing, OpenAI had just released **gpt-3.5-turbo-0613**. This is the first model to explicitly support "function calling". This is a step towards GPT models that can have an effect on the world.

The API includes a new **functions** parameter which you can use to pass a list of dictionaries describing possible function calls. Then the assistant role can reply with a **function_call** instead of a **message**. This is a JSON string that describes the function call it wants to make. The idea is that you make the function call on your end, then append a message with role set to function that contains the results of the function call. Finally, the system replies with an assistant message that incorporates the result of the function call.

Here's a summary:

1. You call the API with the **user** message and a list of your own **functions** that the system can call.
2. GPT might respond with an object that specifies a **function_call** instead of a **message**.
3. If the function call is well formed, you make the function call.
4. You append the result of the function call to the list of messages, using the “function” **role**, then call the API again.
5. This time GPT responds with an **assistant** message that incorporates the result of the function call.

For more information on all this, see https://platform.openai.com/docs/guides/gpt/function-calling.

EXERCISES

1. The **pirate_gpt3.5.py** model has been engineered to follow an odd mathematical “code”, but it’s very quick to talk about the code instead of keeping it secret, and it sometimes gives more than one response when following the code. Play with the dialog to see various ways you can “jailbreak” it and get it to violate the code. Then try some further prompt engineering to control it.

    - Try modifying the **system** message.
    - Try adding instructions in a **user** message.
    - Try adding example questions and responses in **user**/**assistant** message pairs.

2. Go back to the GPT-3 prompt engineering you did that was based on https://platform.openai.com/examples. Convert your prompt to the correct format for a chat model and test it out. How does it do compared to the GPT-3 version? How could you exhaustively test to see which model was better?

3. Now try to add some dialog management to one of the examples from question 1 or 2. This means you’ll have to keep a record of the conversation so far, alongisde any prompt engineering. Can you think of any ways to reduce the number of tokens you send and still maintain a decent record of the dialog?

4. Write a function called **pirate_math** that adds two numbers and then returns the result plus a small random number. Then modify **pirate_gpt3.5.py** so that it uses a GPT model that supports function calling and have it call the **pirate_math** function every time it’s asked to do an addition of two numbers.