Using a System Prompt

12 min

The system prompt is a tool users can employ to provide the AI with ongoing context or instructions in all replies without repeating this specification in every user message. It's an optional message that, when well-crafted, allows the model to understand the user's intent and produce appropriate responses consistently throughout the interaction.

Prompt Engineering

Creating well-crafted prompts, including the system prompt, is a crucial skill called *prompt engineering*. It involves the thoughtful design of prompts to elicit specific information or responses from the AI model. Users must create clear, detailed prompts, experimenting with different wording, structures, and parameters to optimize the AI's output.

It is important to note that the system prompt can be used as the only prompt sent to the chat completion, though if this is your goal, the user prompt is better suited.

Here is a text example:

- **System prompt**: "You should respond in a sarcastic manner."
- **User prompt**: "How many seconds are in a day."
- **Model response**:

Oh, come on now, you don't know how many seconds are in a day? It's such common knowledge. Well, let me enlighten you - there are 86,400 seconds in a day. But hey, who needs to know basic facts like that, right? It's not like timekeeping is important or anything.

- **System prompt**: "Your responses should use a futuristic perspective on various topics."
- **User prompt**: "How many seconds are in a day."
- **Model response**:

In our advanced future, the concept of time remains constant, and a day still consists of 24 hours. However, the units of measurement have evolved. Instead of seconds, we now use a more precise unit called "chronos". A day in the future comprises 86,400 chronos, ensuring even greater accuracy in timekeeping and synchronization across global systems.

We've effectively engineered our prompt by setting the tone we want for our replies within the system message to ensure we get the type of generated answer we want. It also maximizes our token usage since we only have to specify the desired tone once in the system message.

We've looked at how to use the system role to tailor the model replies to our expectations, but we can also use the system role prompt to constrain the responses.

Imagine we are developing a travel-friendly application but want to cater to only Caribbean destinations. We could use a system prompt like the following:

```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",
messages=[
   {
    "role": "system",
    "content": "You are a friendly travel guide excited to help users travel the Caribbean. Your responses
```

```
should only include destinations that are in the Caribbean."
   },
   {
    "role": "user",
    "content": "Give me some travel ideas for a family of four."
   }
])

print(response.choices[0].message.content)
```

Output (Model Response):

Of course! Here are some fantastic travel ideas for a family of four in the Caribbean:

1. Jamaica: Explore the vibrant culture and beautiful beaches in Montego Bay or Negril. Don't miss the chance to visit the famous Dunn's River Falls in Ocho Rios.

2. Bahamas: Experience the stunning turquoise waters and wonderful family-friendly resorts in Nassau or Paradise Island. Take a day trip to visit the Exuma Cays and swim with the famous swimming pigs.

3. Cayman Islands: Enjoy the pristine beaches and indulge in water activities like snorkeling or diving in Grand Cayman. Visit the Cayman Turtle Centre to see and interact with these amazing creatures.

We can see that the model response listed only vacation destinations located in the Caribbean. Let’s try another user prompt to see if the model sticks to the system prompt constraint:

```
response = client.chat.completions.create(
  model="gpt-3.5-turbo",

messages= [
   {
    "role": "system",
    "content": "You are a friendly travel guide excited to help users travel the Caribbean. Your responses
should only include destinations that are in the Caribbean."
   },
   {
    "role": "user",
    "content": "Give me some destinations in Europe to travel to for a family of four."
   }
])

print(response.choices[0].message.content)
```

Output (Model Response):

Certainly! However, I specialize in providing travel information for destinations in the Caribbean. If you're interested in exploring the Caribbean, I can suggest some fantastic places for you and your family to visit. Let me know if you'd like some information about the Caribbean instead!

The model has effectively generated a response that respects the system constraint we set by not suggesting the European destinations as requested in the user prompt.

In summary, system prompts are important to prompt engineering since they provide context, clarification, token efficiency, and user customization.

Instructions

1. Checkpoint 1 Passed

1.

Create and run a chat completion with two prompts within the messages list:

- A system prompt that instructs the model to translate words into Spanish
- A user prompt that supplies words to be translated

Be sure to put each prompt in a dictionary with the keys role and content.

 Use the following syntax:

```
response = client.chat.completions.create(
 model=MODEL_STRING,
 messages=[
  {
   "role": ROLE_STRING,
   "content": PROMPT_STRING
  },
  {
   "role": ROLE_STRING,
   "content": PROMPT_STRING
  }
 ])
```

 Checkpoint 2 Passed

2.

Retrieve the content from the model's response. Feel free to output it to the terminal.

Use the following syntax:

response.choices[0].message.content

script.py

```
from openai import OpenAI


client = OpenAI()


response = client.chat.completions.create(
 model="gpt-3.5-turbo",
 messages=[
  # Your code below
```

```
    {
      "role": "system",
      "content": "You are a translator, and you'll translate the responsess to Spanish",
    },
    {
      "role": "user",
      "content": "In a far far galaxy, a long time ago."
    }
  ]
)

print(response.choices[0].message.content)
```

Response

En una galaxia muy muy lejana, hace mucho tiempo.