Learn how to work with the GPT-35-Turbo and GPT-4 models

Article • 07/18/2023

The GPT-35-Turbo and GPT-4 models are language models that are optimized for conversational interfaces. The models behave differently than the older GPT-3 models. Previous models were text-in and text-out, meaning they accepted a prompt string and returned a completion to append to the prompt. However, the GPT-35-Turbo and GPT-4 models are conversation-in and message-out. The models expect input formatted in a specific chat-like transcript format, and return a completion that represents a model-written message in the chat. While this format was designed specifically for multi-turn conversations, you'll find it can also work well for non-chat scenarios too.

In Azure OpenAI there are two different options for interacting with these type of models:

- Chat Completion API.
- Completion API with Chat Markup Language (ChatML).

The Chat Completion API is a new dedicated API for interacting with the GPT-35-Turbo and GPT-4 models. This API is the preferred method for accessing these models. **It is also the only way to access the new GPT-4 models**.

ChatML uses the same completion API that you use for other models like text-davinci-002, it requires a unique token based prompt format known as Chat Markup Language (ChatML). This provides lower level access than the dedicated Chat Completion API, but also requires additional input validation, only supports gpt-35-turbo models, and **the underlying format is more likely to change over time**.

This article walks you through getting started with the GPT-35-Turbo and GPT-4 models. It's important to use the techniques described here to get the best results. If you try to interact with the models the same way you did with the older model series, the models will often be verbose and provide less useful responses.

Working with the GPT-3.5-Turbo and GPT-4 models

The following code snippet shows the most basic way to use the GPT-3.5-Turbo and GPT-4 models with the Chat Completion API. If this is your first time using these models programmatically, we recommend starting with our GPT-3.5-Turbo & GPT-4 Quickstart.

OpenAI Python 0.28.1

Python

```python
import os
import openai
openai.api_type = "azure"
openai.api_version = "2023-05-15"
openai.api_base = os.getenv("AZURE_OPENAI_ENDPOINT")  # Your Azure
OpenAI resource's endpoint value.
openai.api_key = os.getenv("AZURE_OPENAI_KEY")

response = openai.ChatCompletion.create(
    engine="gpt-35-turbo", # The deployment name you chose when you de-
ployed the GPT-3.5-Turbo or GPT-4 model.
    messages=[
        {"role": "system", "content": "Assistant is a large language
model trained by OpenAI."},
        {"role": "user", "content": "Who were the founders of
Microsoft?"}
    ]
)

print(response)

# To print only the response content text:
# print(response['choices'][0]['message']['content'])
```

Output

JSON formatting added artificially for ease of reading.

JSON

```json
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "content": "The founders of Microsoft are Bill Gates and Paul
Allen. They co-founded the company in 1975.",
        "role": "assistant"
      }
    }
  ],
  "created": 1679014551,
  "id": "chatcmpl-6usfn2yyjkbmESe3G4jaQR6bsScO1",
  "model": "gpt-3.5-turbo-0301",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 86,
    "prompt_tokens": 37,
    "total_tokens": 123
  }
}
```

ⓘ **Note**

The following parameters aren't available with the new GPT-35-Turbo and GPT-4 models: `logprobs`, `best_of`, and `echo`. If you set any of these parameters, you'll get an error.

Every response includes a `finish_reason`. The possible values for `finish_reason` are:

- **stop**: API returned complete model output.
- **length**: Incomplete model output due to max_tokens parameter or token limit.
- **content_filter**: Omitted content due to a flag from our content filters.
- **null**:API response still in progress or incomplete.

Consider setting `max_tokens` to a slightly higher value than normal such as 300 or 500. This ensures that the model doesn't stop generating text before it reaches the end of the message.

Model versioning

ⓘ **Note**

`gpt-35-turbo` is equivalent to the `gpt-3.5-turbo` model from OpenAI.

Unlike previous GPT-3 and GPT-3.5 models, the `gpt-35-turbo` model as well as the `gpt-4` and `gpt-4-32k` models will continue to be updated. When creating a deployment of these models, you'll also need to specify a model version.

You can find the model retirement dates for these models on our models page.

Working with the Chat Completion API

OpenAI trained the GPT-35-Turbo and GPT-4 models to accept input formatted as a conversation. The messages parameter takes an array of message objects with a conversation organized by role. When using the Python API a list of dictionaries is used.

The format of a basic Chat Completion is as follows:

```
{"role": "system", "content": "Provide some context and/or instructions to
the model"},
{"role": "user", "content": "The users messages goes here"}
```

A conversation with one example answer followed by a question would look like:

```
{"role": "system", "content": "Provide some context and/or instructions to
the model."},
{"role": "user", "content": "Example question goes here."},
{"role": "assistant", "content": "Example answer goes here."},
{"role": "user", "content": "First question/message for the model to actu-
ally respond to."}
```

System role

The system role also known as the system message is included at the beginning of the array. This message provides the initial instructions to the model. You can provide various information in the system role including:

- A brief description of the assistant
- Personality traits of the assistant
- Instructions or rules you would like the assistant to follow
- Data or information needed for the model, such as relevant questions from an FAQ

You can customize the system role for your use case or just include basic instructions. The system role/message is optional, but it's recommended to at least include a basic one to get the best results.

Messages

After the system role, you can include a series of messages between the **user** and the **assistant**.

```
{"role": "user", "content": "What is thermodynamics?"}
```

To trigger a response from the model, you should end with a user message indicating that it's the assistant's turn to respond. You can also include a series of example messages between the user and the assistant as a way to do few shot learning.

Message prompt examples

The following section shows examples of different styles of prompts that you could use with the GPT-35-Turbo and GPT-4 models. These examples are just a starting point, and you can experiment with different prompts to customize the behavior for your own use cases.

Basic example

If you want the GPT-35-Turbo model to behave similarly to chat.openai.com, you can use a basic system message like "Assistant is a large language model trained by OpenAI."

```
{"role": "system", "content": "Assistant is a large language model trained
by OpenAI."},
{"role": "user", "content": "Who were the founders of Microsoft?"}
```

Example with instructions

For some scenarios, you might want to give additional instructions to the model to define guardrails for what the model is able to do.

```
{"role": "system", "content": "Assistant is an intelligent chatbot designed
to help users answer their tax related questions.
Instructions:
- Only answer questions related to taxes.
- If you're unsure of an answer, you can say "I don't know" or "I'm not
sure" and recommend users go to the IRS website for more information. "},
{"role": "user", "content": "When are my taxes due?"}
```

Using data for grounding

You can also include relevant data or information in the system message to give the model extra context for the conversation. If you only need to include a small amount of information, you can hard code it in the system message. If you have a large amount of data that the model should be aware of, you can use embeddings or a product like Azure Cognitive Search to retrieve the most relevant information at query time.

```
{"role": "system", "content": "Assistant is an intelligent chatbot designed
to help users answer technical questions about Azure OpenAI Serivce. Only
answer questions using the context below and if you're not sure of an an-
swer, you can say 'I don't know'.

Context:
- Azure OpenAI Service provides REST API access to OpenAI's powerful lan-
guage models including the GPT-3, Codex and Embeddings model series.
- Azure OpenAI Service gives customers advanced language AI with OpenAI GPT-
3, Codex, and DALL-E models with the security and enterprise promise of
Azure. Azure OpenAI co-develops the APIs with OpenAI, ensuring compatibility
and a smooth transition from one to the other.
- At Microsoft, we're committed to the advancement of AI driven by princi-
ples that put people first. Microsoft has made significant investments to
help guard against abuse and unintended harm, which includes requiring ap-
plicants to show well-defined use cases, incorporating Microsoft's princi-
ples for responsible AI use."
},
{"role": "user", "content": "What is Azure OpenAI Service?"}
```

Few shot learning with Chat Completion

You can also give few shot examples to the model. The approach for few shot learning has changed slightly because of the new prompt format. You can now include a series of messages between the user and the assistant in the prompt as few shot examples. These examples can be used to seed answers to common questions to prime the model or teach particular behaviors to the model.

This is only one example of how you can use few shot learning with GPT-35-Turbo and GPT-4. You can experiment with different approaches to see what works best for your use case.

```
{"role": "system", "content": "Assistant is an intelligent chatbot designed
to help users answer their tax related questions. "},
{"role": "user", "content": "When do I need to file my taxes by?"},
{"role": "assistant", "content": "In 2023, you will need to file your taxes
by April 18th. The date falls after the usual April 15th deadline because
April 15th falls on a Saturday in 2023. For more details, see
https://www.irs.gov/filing/individuals/when-to-file."},
{"role": "user", "content": "How can I check the status of my tax refund?"},
{"role": "assistant", "content": "You can check the status of your tax re-
fund by visiting https://www.irs.gov/refunds"}
```

Using Chat Completion for non-chat scenarios

The Chat Completion API is designed to work with multi-turn conversations, but it also works well for non-chat scenarios.

For example, for an entity extraction scenario, you might use the following prompt:

```
{"role": "system", "content": "You are an assistant designed to extract en-
tities from text. Users will paste in a string of text and you will respond
with entities you've extracted from the text as a JSON object. Here's an ex-
ample of your output format:
{
   "name": "",
   "company": "",
   "phone_number": ""
}"},
{"role": "user", "content": "Hello. My name is Robert Smith. I'm calling
from Contoso Insurance, Delaware. My colleague mentioned that you are inter-
ested in learning about our comprehensive benefits policy. Could you give me
a call back at (555) 346-9322 when you get a chance so we can go over the
benefits?"}
```

Creating a basic conversation loop

The examples so far have shown you the basic mechanics of interacting with the Chat Completion API. This example shows you how to create a conversation loop that performs the following actions:

- Continuously takes console input, and properly formats it as part of the messages list as user role content.
- Outputs responses that are printed to the console and formatted and added to the messages list as assistant role content.

This means that every time a new question is asked, a running transcript of the conversation so far is sent along with the latest question. Since the model has no memory, you need to send an updated transcript with each new question or the model will lose context of the previous questions and answers.

OpenAI Python 0.28.1

Python

```python
import os
import openai
openai.api_type = "azure"
openai.api_version = "2023-05-15"
openai.api_base = os.getenv("AZURE_OPENAI_ENDPOINT")  # Your Azure
OpenAI resource's endpoint value.
openai.api_key = os.getenv("AZURE_OPENAI_KEY")

conversation=[{"role": "system", "content": "You are a helpful assist-
ant."}]

while True:
    user_input = input()
    conversation.append({"role": "user", "content": user_input})

    response = openai.ChatCompletion.create(
        engine="gpt-35-turbo", # The deployment name you chose when you
deployed the GPT-35-turbo or GPT-4 model.
        messages=conversation
    )

    conversation.append({"role": "assistant", "content":
response["choices"][0]["message"]["content"]})
    print("\n" + response['choices'][0]['message']['content'] + "\n")
```

When you run the code above you will get a blank console window. Enter your first question in the window and then hit enter. Once the response is returned, you can repeat the process and keep asking questions.

Managing conversations

The previous example will run until you hit the model's token limit. With each question asked, and answer received, the `messages` list grows in size. The token limit for `gpt-35-turbo` is 4096 tokens, whereas the token limits for `gpt-4` and `gpt-4-32k` are 8192 and 32768 respectively. These limits include the token count from both the message list sent and the model response. The number of tokens in the messages list combined with the value of the `max_tokens` parameter must stay under these limits or you'll receive an error.

It's your responsibility to ensure the prompt and completion falls within the token limit. This means that for longer conversations, you need to keep track of the token count and only send the model a prompt that falls within the limit.

> ⓘ **Note**
>
> We strongly recommend staying within the **documented input token limit** for all models even if you find you are able to exceed that limit.

The following code sample shows a simple chat loop example with a technique for handling a 4096 token count using OpenAI's tiktoken library.

The code uses tiktoken `0.5.1`. If you have an older version run `pip install tiktoken --upgrade`.

OpenAI Python 0.28.1

Python

```python
import tiktoken
import openai
import os

openai.api_type = "azure"
openai.api_version = "2023-05-15"
openai.api_base = os.getenv("AZURE_OPENAI_ENDPOINT")  # Your Azure
OpenAI resource's endpoint value.
openai.api_key = os.getenv("AZURE_OPENAI_KEY")

system_message = {"role": "system", "content": "You are a helpful as-
sistant."}
max_response_tokens = 250
token_limit = 4096
conversation = []
conversation.append(system_message)

def num_tokens_from_messages(messages, model="gpt-3.5-turbo-0613"):
    """Return the number of tokens used by a list of messages."""
    try:
        encoding = tiktoken.encoding_for_model(model)
    except KeyError:
        print("Warning: model not found. Using cl100k_base encoding.")
        encoding = tiktoken.get_encoding("cl100k_base")
    if model in {
        "gpt-3.5-turbo-0613",
        "gpt-3.5-turbo-16k-0613",
        "gpt-4-0314",
        "gpt-4-32k-0314",
        "gpt-4-0613",
        "gpt-4-32k-0613",
        }:
```

```
        tokens_per_message = 3
        tokens_per_name = 1
    elif model == "gpt-3.5-turbo-0301":
        tokens_per_message = 4  # every message follows <|start|>
{role/name}\n{content}<|end|>\n
        tokens_per_name = -1  # if there's a name, the role is omitted
    elif "gpt-3.5-turbo" in model:
        print("Warning: gpt-3.5-turbo may update over time. Returning
num tokens assuming gpt-3.5-turbo-0613.")
        return num_tokens_from_messages(messages, model="gpt-3.5-turbo-
0613")
    elif "gpt-4" in model:
        print("Warning: gpt-4 may update over time. Returning num tokens
assuming gpt-4-0613.")
        return num_tokens_from_messages(messages, model="gpt-4-0613")
    else:
        raise NotImplementedError(
            f"""num_tokens_from_messages() is not implemented for model
{model}. See https://github.com/openai/openai-python/blob/main/chatml.md
for information on how messages are converted to tokens."""
        )
    num_tokens = 0
    for message in messages:
        num_tokens += tokens_per_message
        for key, value in message.items():
            num_tokens += len(encoding.encode(value))
            if key == "name":
                num_tokens += tokens_per_name
    num_tokens += 3  # every reply is primed with
<|start|>assistant<|message|>
    return num_tokens

while True:
    user_input = input("")
    conversation.append({"role": "user", "content": user_input})
    conv_history_tokens = num_tokens_from_messages(conversation)

    while conv_history_tokens + max_response_tokens >= token_limit:
        del conversation[1]
        conv_history_tokens = num_tokens_from_messages(conversation)

    response = openai.ChatCompletion.create(
        engine="gpt-35-turbo", # The deployment name you chose when you
deployed the GPT-35-Turbo or GPT-4 model.
        messages=conversation,
        temperature=0.7,
        max_tokens=max_response_tokens,
    )

    conversation.append({"role": "assistant", "content":
response['choices'][0]['message']['content']})
    print("\n" + response['choices'][0]['message']['content'] + "\n")
```

In this example, once the token count is reached, the oldest messages in the conversation transcript will be removed. `del` is used instead of `pop()` for efficiency, and we start at index 1 so as to always preserve the system message and only remove user/assistant messages. Over time, this method of managing the conversation can cause the conversation quality to degrade as the model will gradually lose context of the earlier portions of the conversation.

An alternative approach is to limit the conversation duration to the max token length or a certain number of turns. Once the max token limit is reached and the model would lose context if you were to allow the conversation to continue, you can prompt the user that they need to begin a new conversation and clear the messages list to start a brand new conversation with the full token limit available.

The token counting portion of the code demonstrated previously is a simplified version of one of OpenAI's cookbook examples .

Next steps

- Learn more about Azure OpenAI.
- Get started with the GPT-35-Turbo model with the GPT-35-Turbo quickstart.
- For more examples, check out the Azure OpenAI Samples GitHub repository