Open in Colab
Open on GitHub

ChatOpenAI

This notebook provides a quick overview for getting started with OpenAI chat models. For detailed documentation of all ChatOpenAI features and configurations head to the API reference.

OpenAI has several chat models. You can find information about their latest models and their costs, context windows, and supported input types in the OpenAI docs.

AZURE OPENAI

Note that certain OpenAI models can also be accessed via the Microsoft Azure platform. To use the Azure OpenAI service use the AzureChatOpenAI integration.

Overview

Integration details

Class	Package	Local	Serializable	JS support	Package downloads	Package latest
ChatOpenAI	langchain-openai	❌	beta	✅	6.8M/month	v0.3.1

Model features

Tool calling	Structured output	JSON mode	Image input	Audio input	Video input	Token-level streaming	Native async	Token usage
✅	✅	✅	✅	❌	❌	✅	✅	✅

Setup

To access OpenAI models you'll need to create an OpenAI account, get an API key, and install the `langchain-openai` integration package.

Credentials

Head to https://platform.openai.com to sign up to OpenAI and generate an API key. Once you've done this set the OPENAI_API_KEY environment variable:

```
import getpass
import os

if not os.environ.get("OPENAI_API_KEY"):
    os.environ["OPENAI_API_KEY"] = getpass.getpass("Enter your OpenAI API
key: ")
```

If you want to get automated tracing of your model calls you can also set your LangSmith API key by uncommenting below:

```
# os.environ["LANGSMITH_API_KEY"] = getpass.getpass("Enter your LangSmith
API key: ")
# os.environ["LANGSMITH_TRACING"] = "true"
```

Installation

The LangChain OpenAI integration lives in the `langchain-openai` package:

```
%pip install -qU langchain-openai
```

Instantiation

Now we can instantiate our model object and generate chat completions:

```python
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(
    model="gpt-4o",
    temperature=0,
    max_tokens=None,
    timeout=None,
    max_retries=2,
    # api_key="...",  # if you prefer to pass api key in directly instaed
of using env vars
    # base_url="...",
    # organization="...",
    # other params...
)
```

API Reference: ChatOpenAI

Invocation

```python
messages = [
    (
        "system",
        "You are a helpful assistant that translates English to French.
Translate the user sentence.",
    ),
    ("human", "I love programming."),
]
ai_msg = llm.invoke(messages)
ai_msg
```

```
AIMessage(content="J'adore la programmation.", additional_kwargs=
{'refusal': None}, response_metadata={'token_usage': {'completion_tokens':
5, 'prompt_tokens': 31, 'total_tokens': 36}, 'model_name': 'gpt-4o-2024-
05-13', 'system_fingerprint': 'fp_3aa7262c27', 'finish_reason': 'stop',
'logprobs': None}, id='run-63219b22-03e3-4561-8cc4-78b7c7c3a3ca-0',
usage_metadata={'input_tokens': 31, 'output_tokens': 5, 'total_tokens':
36})
```

```
print(ai_msg.content)
```

```
J'adore la programmation.
```

Chaining

We can chain our model with a prompt template like so:

```
from langchain_core.prompts import ChatPromptTemplate

prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            "You are a helpful assistant that translates {input_language}
to {output_language}.",
        ),
        ("human", "{input}"),
    ]
)

chain = prompt | llm
chain.invoke(
    {
        "input_language": "English",
        "output_language": "German",
        "input": "I love programming.",
    }
)
```

API Reference: ChatPromptTemplate

```
AIMessage(content='Ich liebe das Programmieren.', additional_kwargs=
{'refusal': None}, response_metadata={'token_usage': {'completion_tokens':
6, 'prompt_tokens': 26, 'total_tokens': 32}, 'model_name': 'gpt-4o-2024-
05-13', 'system_fingerprint': 'fp_3aa7262c27', 'finish_reason': 'stop',
'logprobs': None}, id='run-350585e1-16ca-4dad-9460-3d9e7e49aaf1-0',
usage_metadata={'input_tokens': 26, 'output_tokens': 6, 'total_tokens':
32})
```

Tool calling

OpenAI has a tool calling (we use "tool calling" and "function calling" interchangeably here) API that lets you describe tools and their arguments, and have the model return a JSON object with a tool to invoke and the inputs to that tool. tool-calling is extremely useful for building tool-using chains and agents, and for getting structured outputs from models more generally.

ChatOpenAI.bind_tools()

With `ChatOpenAI.bind_tools`, we can easily pass in Pydantic classes, dict schemas, LangChain tools, or even functions as tools to the model. Under the hood these are converted to an OpenAI tool schemas, which looks like:

```
{
    "name": "...",
    "description": "...",
    "parameters": {...}  # JSONSchema
}
```

and passed in every model invocation.

```python
from pydantic import BaseModel, Field


class GetWeather(BaseModel):
    """Get the current weather in a given location"""

    location: str = Field(..., description="The city and state, e.g. San Francisco, CA")


llm_with_tools = llm.bind_tools([GetWeather])
```

```python
ai_msg = llm_with_tools.invoke(
    "what is the weather like in San Francisco",
)
ai_msg
```

```
AIMessage(content='', additional_kwargs={'tool_calls': [{'id':
'call_o9udf3EVOWiV4Iupktpbpofk', 'function': {'arguments':
'{"location":"San Francisco, CA"}', 'name': 'GetWeather'}, 'type':
'function'}], 'refusal': None}, response_metadata={'token_usage':
{'completion_tokens': 17, 'prompt_tokens': 68, 'total_tokens': 85},
'model_name': 'gpt-4o-2024-05-13', 'system_fingerprint': 'fp_3aa7262c27',
'finish_reason': 'tool_calls', 'logprobs': None}, id='run-1617c9b2-dda5-
4120-996b-0333ed5992e2-0', tool_calls=[{'name': 'GetWeather', 'args':
{'location': 'San Francisco, CA'}, 'id': 'call_o9udf3EVOWiV4Iupktpbpofk',
'type': 'tool_call'}], usage_metadata={'input_tokens': 68,
'output_tokens': 17, 'total_tokens': 85})
```

`strict=True`

REQUIRES `langchain-openai>=0.1.21rc1`

As of Aug 6, 2024, OpenAI supports a `strict` argument when calling tools that will enforce that the tool argument schema is respected by the model. See more here: https://platform.openai.com/docs/guides/function-calling

Note: If `strict=True` the tool definition will also be validated, and a subset of JSON schema are accepted. Crucially, schema cannot have optional args (those with default values). Read the full docs on what types of schema are supported here: https://platform.openai.com/docs/guides/structured-outputs/supported-schemas.

```
llm_with_tools = llm.bind_tools([GetWeather], strict=True)
ai_msg = llm_with_tools.invoke(
    "what is the weather like in San Francisco",
)
ai_msg
```

```
AIMessage(content='', additional_kwargs={'tool_calls': [{'id':
'call_jUqhd8wzAIzInTJl72Rla8ht', 'function': {'arguments':
'{"location":"San Francisco, CA"}', 'name': 'GetWeather'}, 'type':
'function'}], 'refusal': None}, response_metadata={'token_usage':
{'completion_tokens': 17, 'prompt_tokens': 68, 'total_tokens': 85},
'model_name': 'gpt-4o-2024-05-13', 'system_fingerprint': 'fp_3aa7262c27',
'finish_reason': 'tool_calls', 'logprobs': None}, id='run-5e3356a9-132d-
4623-8e73-dd5a898cf4a6-0', tool_calls=[{'name': 'GetWeather', 'args':
{'location': 'San Francisco, CA'}, 'id': 'call_jUqhd8wzAIzInTJl72Rla8ht',
'type': 'tool_call'}], usage_metadata={'input_tokens': 68,
'output_tokens': 17, 'total_tokens': 85})
```

AIMessage.tool_calls

Notice that the AIMessage has a `tool_calls` attribute. This contains in a standardized ToolCall format that is model-provider agnostic.

```
ai_msg.tool_calls
```

```
[{'name': 'GetWeather',
  'args': {'location': 'San Francisco, CA'},
  'id': 'call_jUqhd8wzAIzInTJl72Rla8ht',
  'type': 'tool_call'}]
```

For more on binding tools and tool call outputs, head to the tool calling docs.

Fine-tuning

You can call fine-tuned OpenAI models by passing in your corresponding `modelName` parameter.

This generally takes the form of `ft:{OPENAI_MODEL_NAME}:{ORG_NAME}::{MODEL_ID}`. For example:

```
fine_tuned_model = ChatOpenAI(
    temperature=0, model_name="ft:gpt-3.5-turbo-0613:langchain::7qTVM5AR"
)

fine_tuned_model.invoke(messages)
```

```
AIMessage(content="J'adore la programmation.", additional_kwargs=
{'refusal': None}, response_metadata={'token_usage': {'completion_tokens':
8, 'prompt_tokens': 31, 'total_tokens': 39}, 'model_name': 'ft:gpt-3.5-
turbo-0613:langchain::7qTVM5AR', 'system_fingerprint': None,
'finish_reason': 'stop', 'logprobs': None}, id='run-0f39b30e-c56e-4f3b-
af99-5c948c984146-0', usage_metadata={'input_tokens': 31, 'output_tokens':
8, 'total_tokens': 39})
```

Multimodal Inputs

OpenAI has models that support multimodal inputs. You can pass in images or audio to these models. For more information on how to do this in LangChain, head to the multimodal inputs docs.

You can see the list of models that support different modalities in OpenAI's documentation.

At the time of this doc's writing, the main OpenAI models you would use would be:

- Image inputs: `gpt-4o`, `gpt-4o-mini`
- Audio inputs: `gpt-4o-audio-preview`

For an example of passing in image inputs, see the multimodal inputs how-to guide.

Below is an example of passing audio inputs to `gpt-4o-audio-preview`:

```
import base64

from langchain_openai import ChatOpenAI

llm = ChatOpenAI(
    model="gpt-4o-audio-preview",
    temperature=0,
)

with open(

"../../../../libs/partners/openai/tests/integration_tests/chat_models/audio
    "rb",
) as f:
    # b64 encode it
    audio = f.read()
    audio_b64 = base64.b64encode(audio).decode()


output_message = llm.invoke(
    [
        (
            "human",
            [
                {"type": "text", "text": "Transcribe the following:"},
                # the audio clip says "I'm sorry, but I can't create..."
                {
                    "type": "input_audio",
                    "input_audio": {"data": audio_b64, "format": "wav"},
                },
            ],
        ),
    ]
)
output_message.content
```

API Reference: ChatOpenAI

```
"I'm sorry, but I can't create audio content that involves yelling. Is
there anything else I can help you with?"
```

Predicted output

INFO

Requires `langchain-openai>=0.2.6`

Some OpenAI models (such as their `gpt-4o` and `gpt-4o-mini` series) support Predicted Outputs, which allow you to pass in a known portion of the LLM's expected output ahead of time to reduce latency. This is useful for cases such as editing text or code, where only a small part of the model's output will change.

Here's an example:

```
code = """
/// <summary>
/// Represents a user with a first name, last name, and username.
/// </summary>
public class User
{
    /// <summary>
    /// Gets or sets the user's first name.
    /// </summary>
    public string FirstName { get; set; }

    /// <summary>
    /// Gets or sets the user's last name.
    /// </summary>
    public string LastName { get; set; }
    /// <summary>
    /// Gets or sets the user's username.
    /// </summary>
    public string Username { get; set; }
}
"""

llm = ChatOpenAI(model="gpt-4o")
query = (
    "Replace the Username property with an Email property. "
    "Respond only with code, and with no markdown formatting."
)
response = llm.invoke(
    [{"role": "user", "content": query}, {"role": "user", "content":
code}],
    prediction={"type": "content", "content": code},
)
print(response.content)
print(response.response_metadata)
```

```
/// <summary>
/// Represents a user with a first name, last name, and email.
/// </summary>
public class User
{
    /// <summary>
    /// Gets or sets the user's first name.
    /// </summary>
    public string FirstName { get; set; }

    /// <summary>
    /// Gets or sets the user's last name.
    /// </summary>
    public string LastName { get; set; }

    /// <summary>
    /// Gets or sets the user's email.
    /// </summary>
    public string Email { get; set; }
}
{'token_usage': {'completion_tokens': 226, 'prompt_tokens': 166,
'total_tokens': 392, 'completion_tokens_details':
```

```
{'accepted_prediction_tokens': 49, 'audio_tokens': None,
'reasoning_tokens': 0, 'rejected_prediction_tokens': 107},
'prompt_tokens_details': {'audio_tokens': None, 'cached_tokens': 0}},
'model_name': 'gpt-4o-2024-08-06', 'system_fingerprint': 'fp_45cf54deae',
'finish_reason': 'stop', 'logprobs': None}
```

Note that currently predictions are billed as additional tokens and may increase your usage and costs in exchange for this reduced latency.

Audio Generation (Preview)

INFO

Requires `langchain-openai>=0.2.3`

OpenAI has a new audio generation feature that allows you to use audio inputs and outputs with the `gpt-4o-audio-preview` model.

```python
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(
    model="gpt-4o-audio-preview",
    temperature=0,
    model_kwargs={
        "modalities": ["text", "audio"],
        "audio": {"voice": "alloy", "format": "wav"},
    },
)

output_message = llm.invoke(
    [
        ("human", "Are you made by OpenAI? Just answer yes or no"),
    ]
)
```

API Reference: ChatOpenAI

`output_message.additional_kwargs['audio']` will contain a dictionary like

```
{
    'data': '<audio data b64-encoded',
    'expires_at': 1729268602,
    'id': 'audio_67127d6a44348190af62c1530ef0955a',
    'transcript': 'Yes.'
}
```

and the format will be what was passed in `model_kwargs['audio']['format']`.

We can also pass this message with audio data back to the model as part of a message history before openai `expires_at` is reached.

NOTE

Output audio is stored under the `audio` key in `AIMessage.additional_kwargs`, but input content blocks are typed with an `input_audio` type and key in `HumanMessage.content` lists.

For more information, see OpenAI's audio docs.

```
history = [
    ("human", "Are you made by OpenAI? Just answer yes or no"),
    output_message,
    ("human", "And what is your name? Just give your name."),
]
second_output_message = llm.invoke(history)
```

API reference

For detailed documentation of all ChatOpenAI features and configurations head to the API reference:

https://python.langchain.com/api_reference/openai/chat_models/langchain_openai.chat_models.base.ChatOpenAI.html

Related

- Chat model conceptual guide
- Chat model how-to guides

Edit this page

Was this page helpful?

3 comments · 1 reply – *powered by giscus*

Oldest Newest

syarif1977 Oct 26, 2024

berikan contoh pembangunan infrastruktur