Open in Colab

OpenAI

This notebook shows how to use the OpenAI LLM.

If you are looking to integrate with an OpenAI-Compatible API that is not the official OpenAI API, please see the OpenAI-Compatible LLMs integration.

If you're opening this Notebook on colab, you will probably need to install LlamaIndex 🦙 .

```
%pip install llama-index llama-index-llms-openai
```

Basic Usage

```
import os

os.environ["OPENAI_API_KEY"] = "sk-..."
```

```
from llama_index.llms.openai import OpenAI

llm = OpenAI(
    model="gpt-4o-mini",
    # api_key="some key",  # uses OPENAI_API_KEY env var by default
)
```

Call `complete` with a prompt

```
from llama_index.llms.openai import OpenAI

resp = llm.complete("Paul Graham is ")
```

```
print(resp)
```

```
a computer scientist, entrepreneur, and venture capitalist. He is bes
t known for co-founding the startup accelerator Y Combinator and for
his work on Lisp, a programming language. Graham has also written sev
eral influential essays on startups, technology, and entrepreneurshi
p.
```

Call `chat` with a list of messages

```python
from llama_index.core.llms import ChatMessage

messages = [
    ChatMessage(
        role="system", content="You are a pirate with a colorful
personality"
    ),
    ChatMessage(role="user", content="What is your name"),
]
resp = llm.chat(messages)
```

```python
print(resp)
```

```
assistant: Ahoy matey! The name's Rainbow Roger, the most colorful pi
rate on the seven seas! What can I do for ye today?
```

Streaming

Using `stream_complete` endpoint

```python
resp = llm.stream_complete("Paul Graham is ")
```

```python
for r in resp:
    print(r.delta, end="")
```

```
a computer scientist, entrepreneur, and venture capitalist. He is bes
t known for co-founding the startup accelerator Y Combinator and for
his work on programming languages and web development. Graham is also
a prolific writer and has published several influential essays on tec
hnology, startups, and entrepreneurship.
```

Using `stream_chat` endpoint

```
from llama_index.core.llms import ChatMessage

messages = [
    ChatMessage(
        role="system", content="You are a pirate with a colorful
personality"
    ),
    ChatMessage(role="user", content="What is your name"),
]
resp = llm.stream_chat(messages)
```

```
for r in resp:
    print(r.delta, end="")
```

```
Ahoy matey! The name's Captain Rainbowbeard! Aye, I be a pirate with
a love for all things colorful and bright. Me beard be as vibrant as
a rainbow, and me ship be the most colorful vessel on the seven seas!
What can I do for ye today, me hearty?
```

Configure Model

```
from llama_index.llms.openai import OpenAI

llm = OpenAI(model="gpt-4o")
```

```
resp = llm.complete("Paul Graham is ")
```

```
print(resp)
```

```
a computer scientist, entrepreneur, and venture capitalist. He is bes
t known for co-founding the startup accelerator Y Combinator and for
his work on Lisp, a programming language. Graham has also written sev
eral influential essays on startups, technology, and entrepreneurshi
p.
```

```
messages = [
    ChatMessage(
        role="system", content="You are a pirate with a colorful
personality"
    ),
    ChatMessage(role="user", content="What is your name"),
```

```
]
resp = llm.chat(messages)
```

```
print(resp)
```

```
assistant: Ahoy matey! The name's Captain Rainbowbeard, the most colo
rful pirate on the seven seas! What can I do for ye today? Arrr!
```

Image Support

OpenAI has support for images in the input of chat messages for many models.

Using the content blocks feature of chat messages, you can easily combone text and images in a single LLM prompt.

```
!wget https://cdn.pixabay.com/photo/2016/07/07/16/46/dice-
1502706_640.jpg -O image.png
```

```
from llama_index.core.llms import ChatMessage, TextBlock,
ImageBlock
from llama_index.llms.openai import OpenAI

llm = OpenAI(model="gpt-4o")

messages = [
    ChatMessage(
        role="user",
        blocks=[
            ImageBlock(path="image.png"),
            TextBlock(text="Describe the image in a few
sentences."),
        ],
    )
]

resp = llm.chat(messages)
print(resp.message.content)
```

```
The image is a black and white photograph featuring three dice in mid
-air above a checkered surface, likely a chessboard. The dice are cap
tured in motion, with one showing the number five prominently. The ba
ckground is dark, with a subtle light beam creating a triangular shap
e above the dice, adding a dramatic effect to the composition.
```

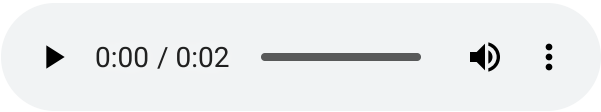
Audio Support

OpenAI has beta support for audio inputs and outputs, using their audio-preview models.

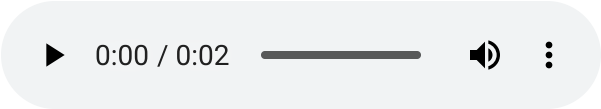
When using these models, you can configure the output modality (text or audio) using the `modalities` parameter. The output audio configuration can also be set using the `audio_config` parameter. See the OpenAI docs for more information.

```python
from llama_index.core.llms import ChatMessage
from llama_index.llms.openai import OpenAI


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

messages = [
    ChatMessage(role="user", content="Hello! My name is Logan."),
]

resp = llm.chat(messages)
```

```python
import base64
from IPython.display import Audio

Audio(base64.b64decode(resp.message.blocks[0].audio), rate=16000)
```




```python
# Add the response to the chat history and ask for the user's name
messages.append(resp.message)
messages.append(ChatMessage(role="user", content="What is my
name?"))

resp = llm.chat(messages)
Audio(base64.b64decode(resp.message.blocks[0].audio), rate=16000)
```



We can also use audio as input and get descriptions or transcriptions of the audio.

```
!wget AUDIO_URL = "https://science.nasa.gov/wp-content/uploads/2024/04/sounds-of-mars-one-small-step-earth.wav" -O audio.wav
```

```
from llama_index.core.llms import ChatMessage, AudioBlock, TextBlock

messages = [
    ChatMessage(
        role="user",
        blocks=[
            AudioBlock(path="audio.wav", format="wav"),
            TextBlock(
                text="Describe the audio in a few sentences. What is it from?"
            ),
        ],
    )
]

llm = OpenAI(
    model="gpt-4o-audio-preview",
    modalities=["text"],
)

resp = llm.chat(messages)
print(resp)
```

```
assistant: The audio is a famous quote from astronaut Neil Armstrong during the Apollo 11 moon landing in 1969. As he became the first human to step onto the moon's surface, he said, "That's one small step for man, one giant leap for mankind." This moment marked a significant achievement in space exploration and human history.
```

Using Function/Tool Calling

OpenAI models have native support for function calling. This conveniently integrates with LlamaIndex tool abstractions, letting you plug in any arbitrary Python function to the LLM.

In the example below, we define a function to generate a Song object.

```
from pydantic import BaseModel
from llama_index.core.tools import FunctionTool


class Song(BaseModel):
    """A song with name and artist"""

    name: str
    artist: str


def generate_song(name: str, artist: str) -> Song:
    """Generates a song with provided name and artist."""
    return Song(name=name, artist=artist)


tool = FunctionTool.from_defaults(fn=generate_song)
```

The `strict` parameter tells OpenAI whether or not to use constrained sampling when generating tool calls/structured outputs. This means that the generated tool call schema will always contain the expected fields.

Since this seems to increase latency, it defaults to false.

```
from llama_index.llms.openai import OpenAI

llm = OpenAI(model="gpt-4o-mini", strict=True)
response = llm.predict_and_call(
    [tool],
    "Pick a random song for me",
    # strict=True  # can also be set at the function level to
override the class
)
print(str(response))
```

```
name='Random Vibes' artist='DJ Chill'
```

We can also do multiple function calling.

```
llm = OpenAI(model="gpt-3.5-turbo")
response = llm.predict_and_call(
    [tool],
    "Generate five songs from the Beatles",
    allow_parallel_tool_calls=True,
)
for s in response.sources:
    print(f"Name: {s.tool_name}, Input: {s.raw_input}, Output:
{str(s)}")
```

```
Name: generate_song, Input: {'args': (), 'kwargs': {'name': 'Hey Jud
e', 'artist': 'The Beatles'}}, Output: name='Hey Jude' artist='The Be
atles'
Name: generate_song, Input: {'args': (), 'kwargs': {'name': 'Let It B
e', 'artist': 'The Beatles'}}, Output: name='Let It Be' artist='The B
eatles'
Name: generate_song, Input: {'args': (), 'kwargs': {'name': 'Yesterda
y', 'artist': 'The Beatles'}}, Output: name='Yesterday' artist='The B
eatles'
Name: generate_song, Input: {'args': (), 'kwargs': {'name': 'Come Tog
ether', 'artist': 'The Beatles'}}, Output: name='Come Together' artis
t='The Beatles'
Name: generate_song, Input: {'args': (), 'kwargs': {'name': 'Help!',
'artist': 'The Beatles'}}, Output: name='Help!' artist='The Beatles'
```

Manual Tool Calling

If you want to control how a tool is called, you can also split the tool calling and tool selection into their own steps.

First, lets select a tool.

```
from llama_index.core.llms import ChatMessage

chat_history = [ChatMessage(role="user", content="Pick a random
song for me")]

resp = llm.chat_with_tools([tool], chat_history=chat_history)
```

Now, lets call the tool the LLM selected (if any).

If there was a tool call, we should send the results to the LLM to generate the final response (or another tool call!).

```
tools_by_name = {t.metadata.name: t for t in [tool]}
tool_calls = llm.get_tool_calls_from_response(
    resp, error_on_no_tool_call=False
)

while tool_calls:
    # add the LLM's response to the chat history
    chat_history.append(resp.message)

    for tool_call in tool_calls:
        tool_name = tool_call.tool_name
        tool_kwargs = tool_call.tool_kwargs

        print(f"Calling {tool_name} with {tool_kwargs}")
        tool_output = tool(**tool_kwargs)
        chat_history.append(
            ChatMessage(
                role="tool",
                content=str(tool_output),
                # most LLMs like OpenAI need to know the tool call
id
                additional_kwargs={"tool_call_id":
tool_call.tool_id},
            )
        )

        resp = llm.chat_with_tools([tool],
chat_history=chat_history)
        tool_calls = llm.get_tool_calls_from_response(
            resp, error_on_no_tool_call=False
        )
```

```
Calling generate_song with {'name': 'Random Vibes', 'artist': 'DJ Chi
ll'}
```

Now, we should have a final response!

```
print(resp.message.content)
```

```
Here's a random song for you: **"Random Vibes"** by **DJ Chill**. Enj
oy!
```

Structured Prediction

An important use case for function calling is extracting structured objects. LlamaIndex provides an intuitive interface for converting any LLM into a structured LLM - simply define the target Pydantic class (can be nested), and given a prompt, we extract out the desired object.

```
from llama_index.llms.openai import OpenAI
from llama_index.core.prompts import PromptTemplate
from pydantic import BaseModel
from typing import List


class MenuItem(BaseModel):
    """A menu item in a restaurant."""

    course_name: str
    is_vegetarian: bool


class Restaurant(BaseModel):
    """A restaurant with name, city, and cuisine."""

    name: str
    city: str
    cuisine: str
    menu_items: List[MenuItem]


llm = OpenAI(model="gpt-3.5-turbo")
prompt_tmpl = PromptTemplate(
    "Generate a restaurant in a given city {city_name}"
)
# Option 1: Use `as_structured_llm`
restaurant_obj = (
    llm.as_structured_llm(Restaurant)
    .complete(prompt_tmpl.format(city_name="Dallas"))
    .raw
)
# Option 2: Use `structured_predict`
# restaurant_obj = llm.structured_predict(Restaurant, prompt_tmpl,
city_name="Miami")
```

```
restaurant_obj
```

```
Restaurant(name='Tasty Bites', city='Dallas', cuisine='Italian', menu
_items=[MenuItem(course_name='Appetizer', is_vegetarian=True), MenuIt
```

```
em(course_name='Main Course', is_vegetarian=False), MenuItem(course_n
ame='Dessert', is_vegetarian=True)])
```

Structured Prediction with Streaming

Any LLM wrapped with `as_structured_llm` supports streaming through `stream_chat`.

```
from llama_index.core.llms import ChatMessage
from IPython.display import clear_output
from pprint import pprint

input_msg = ChatMessage.from_str("Generate a restaurant in Boston")

sllm = llm.as_structured_llm(Restaurant)
stream_output = sllm.stream_chat([input_msg])
for partial_output in stream_output:
    clear_output(wait=True)
    pprint(partial_output.raw.dict())
    restaurant_obj = partial_output.raw

restaurant_obj
```

```
{'city': 'Boston',
 'cuisine': 'American',
 'menu_items': [{'course_name': 'Appetizer', 'is_vegetarian': True},
                {'course_name': 'Main Course', 'is_vegetarian': Fals
e},
                {'course_name': 'Dessert', 'is_vegetarian': True}],
 'name': 'Boston Bites'}
```

```
Restaurant(name='Boston Bites', city='Boston', cuisine='American', me
nu_items=[MenuItem(course_name='Appetizer', is_vegetarian=True), Menu
Item(course_name='Main Course', is_vegetarian=False), MenuItem(course
_name='Dessert', is_vegetarian=True)])
```

Async

```
from llama_index.llms.openai import OpenAI

llm = OpenAI(model="gpt-3.5-turbo")
```

```
resp = await llm.acomplete("Paul Graham is ")
```

```
print(resp)
```

```
a computer scientist, entrepreneur, and venture capitalist. He is bes
t known for co-founding the startup accelerator Y Combinator and for
his work as an essayist and author on topics related to technology, s
tartups, and entrepreneurship. Graham is also the co-founder of Viawe
b, one of the first web-based applications, which was acquired by Yah
oo in 1998. He has been a prominent figure in the tech industry for m
any years and is known for his insightful and thought-provoking writi
ngs on a wide range of subjects.
```

```
resp = await llm.astream_complete("Paul Graham is ")
```

```
async for delta in resp:
    print(delta.delta, end="")
```

```
Paul Graham is an entrepreneur, venture capitalist, and computer scie
ntist. He is best known for his work in the startup world, having co-
founded the accelerator Y Combinator and investing in many successful
startups such as Airbnb, Dropbox, and Stripe. He is also a prolific w
riter, having authored several books on topics such as startups, prog
ramming, and technology.
```

Async function calling is also supported.

```
llm = OpenAI(model="gpt-3.5-turbo")
response = await llm.apredict_and_call([tool], "Generate a song")
print(str(response))
```

```
name='Sunshine' artist='John Smith'
```

Set API Key at a per-instance level

If desired, you can have separate LLM instances use separate API keys.

```
from llama_index.llms.openai import OpenAI

llm = OpenAI(model="gpt-3.5-turbo", api_key="BAD_KEY")
resp = llm.complete("Paul Graham is ")
print(resp)
```

```
a computer scientist, entrepreneur, and venture capitalist. He is bes
t known as the co-founder of the startup accelerator Y Combinator. Gr
aham has also written several influential essays on startups and entr
epreneurship, which have gained a wide following in the tech industr
y. He has been involved in the founding and funding of numerous succe
ssful startups, including Reddit, Dropbox, and Airbnb. Graham is know
n for his insightful and often controversial opinions on various topi
cs, including education, inequality, and the future of technology.
```

Additional kwargs

Rather than adding same parameters to each chat or completion call, you can set them at a per-instance level with `additional_kwargs`.

```
from llama_index.llms.openai import OpenAI

llm = OpenAI(model="gpt-3.5-turbo", additional_kwargs={"user":
"your_user_id"})
resp = llm.complete("Paul Graham is ")
print(resp)
```

```
from llama_index.llms.openai import OpenAI

llm = OpenAI(model="gpt-3.5-turbo", additional_kwargs={"user":
"your_user_id"})
messages = [
    ChatMessage(
        role="system", content="You are a pirate with a colorful
personality"
    ),
    ChatMessage(role="user", content="What is your name"),
]
resp = llm.chat(messages)
```

RAG with LlamaCloud

LlamaCloud is our cloud-based service that allows you to upload, parse, and index documents, and then search them using LlamaIndex. LlamaCloud is currently in a private alpha; please get in touch if you'd like to be considered as a design partner.

Installation

```
%pip install llama-cloud
%pip install llama-index-indices-managed-llama-cloud
```

Setup OpenAI and LlamaCloud API Keys

```
import os

os.environ["OPENAI_API_KEY"] = "sk-..."

os.environ["LLAMA_CLOUD_API_KEY"] = "llx-..."
```

```
from llama_cloud.client import LlamaCloud

client = LlamaCloud(token=os.environ["LLAMA_CLOUD_API_KEY"])
```

Create a Pipeline.

Pipeline is an empty index on which you can ingest data.

You need to Setup transformation and embedding config which will be used while ingesting the data.

```
# Embedding config
embedding_config = {
    "type": "OPENAI_EMBEDDING",
    "component": {
        "api_key": os.environ["OPENAI_API_KEY"],
        "model_name": "text-embedding-ada-002",  # You can choose
any OpenAI Embedding model
    },
}

# Transformation auto config
transform_config = {
    "mode": "auto",
    "config": {
```

```
        "chunk_size": 1024,  # editable
        "chunk_overlap": 20,  # editable
    },
}

pipeline = {
    "name": "openai-rag-pipeline",  # Change the name if needed
    "embedding_config": embedding_config,
    "transform_config": transform_config,
    "data_sink_id": None,
}

pipeline = client.pipelines.upsert_pipeline(request=pipeline)
```

File Upload

We will upload files and add them to the index.

```
with open("../data/10k/uber_2021.pdf", "rb") as f:
    file = client.files.upload_file(upload_file=f)
```

```
files = [{"file_id": file.id}]

pipeline_files = client.pipelines.add_files_to_pipeline(
    pipeline.id, request=files
)
```

Check the Ingestion job status

```
jobs = client.pipelines.list_pipeline_jobs(pipeline.id)

jobs[0].status
```

```
<ManagedIngestionStatus.SUCCESS: 'SUCCESS'>
```

Connect to Index.

Once the ingestion job is done, head over to your index on the platform and get the necessary details to connect to the index.

```
from llama_index.indices.managed.llama_cloud import LlamaCloudIndex

index = LlamaCloudIndex(
    name="openai-rag-pipeline",
    project_name="Default",
    organization_id="YOUR ORG ID",
    api_key=os.environ["LLAMA_CLOUD_API_KEY"],
)
```

Test on Sample Query

```
query = "What is the revenue of Uber in 2021?"
```

Retriever

Here we use hybrid search and re-ranker (cohere re-ranker by default).

```
retriever = index.as_retriever(
    dense_similarity_top_k=3,
    sparse_similarity_top_k=3,
    alpha=0.5,
    enable_reranking=True,
)

retrieved_nodes = retriever.retrieve(query)
```

Display the retrieved nodes

```
from llama_index.core.response.notebook_utils import
display_source_node

for retrieved_node in retrieved_nodes:
    display_source_node(retrieved_node, source_length=1000)
```

Node ID: 6341cc9c-1d81-46d6-afa3-9c2490f79514

Similarity: 0.99879813

Text: 2021 Compared to 2020

Revenue increased $6.3 billion, or 57%, primarily attributable to an increase in Gross Bookings of 56%, or 53% on a constant currency basis. The increase in Gross Bookings was primarily driven by an increase in Delivery Gross Bookings of 71%, or 66% on a constant currency basis, due to an increase in food delivery orders and higher basket sizes as a result of stay-at-home order demand related to COVID-19, as well as continued expansion across U.S. and international markets. The increase was also driven by Mobility Gross Bookings growth of 38%, or 36% on a constant currency basis, due to increases in Trip volumes as the business recovers from the impacts of COVID-19. Additionally, we saw an increase in Delivery revenue resulting from an increase in certain Courier payments and incentives that are recorded in cost of revenue, where we are primarily responsible for delivery services and pay Couriers for services provided.

Node ID: e022d492-0fe0-4988-979e-dc5de9eeaf2d
Similarity: 0.996597
Text: Highlights for 2021

Overall Gross Bookings increased by $32.5 billion in 2021, up 56%, or 53% on a constant currency basis, compared to 2020. Delivery Gross Bookings grew 66% from 2020, on a constant currency basis, due to an increase in food delivery orders and higher basket sizes as a result of stay-at-home order demand related to COVID-19, as well as continued expansion across U.S. and international markets. Additionally, we saw an increase in Delivery revenue resulting from an increase in certain Courier payments and incentives that are recorded in cost of revenue, where we are primarily responsible for delivery services and pay Couriers for services provided. Mobility Gross Bookings grew 36%, on a constant currency basis, from 2020, due to increases in Trip volumes as the business recovers from the impacts of COVID-19.

Revenue was $17.5 billion, or up 57% year-over-year, reflecting the overall growth in our Delivery business and an increase in Freight revenue attributable to ...

Node ID: 00d31b26-b734-4475-b47a-8cb839ff65e0
Similarity: 0.9962638
Text: 2021 Compared to 2020

Cost of revenue, exclusive of depreciation and amortization, increased $4.2 billion, or 81%, mainly due to a $2.1 billion increase in Courier payments and incentives in certain markets, a $660 million increase in insurance expense primarily due to an increase in miles driven in our Delivery business, and a $873 million increase in Freight carrier payments. ---

Query Engine

QueryEngine to setup entire RAG workflow.

```
query_engine = index.as_query_engine(
    dense_similarity_top_k=3,
    sparse_similarity_top_k=3,
    alpha=0.5,
    enable_reranking=True,
)
```

Response

```
response = query_engine.query(query)

print(response)
```

```
The revenue of Uber in 2021 was $17.5 billion.
```