test

December 26, 2023

[1]:
```python
import base64
from io import BytesIO

from IPython.display import HTML, display
from PIL import Image


def convert_to_base64(pil_image):
    """
    Convert PIL images to Base64 encoded strings

    :param pil_image: PIL image
    :return: Re-sized Base64 string
    """

    buffered = BytesIO()
    pil_image.save(buffered, format="JPEG")  # You can change the format if␣
 ↪needed
    img_str = base64.b64encode(buffered.getvalue()).decode("utf-8")
    return img_str


def plt_img_base64(img_base64):
    """
    Disply base64 encoded string as image

    :param img_base64:  Base64 string
    """
    # Create an HTML img tag with the base64 string as the source
    image_html = f'<img src="data:image/jpeg;base64,{img_base64}" />'
    # Display the image by rendering the HTML
    display(HTML(image_html))


file_path = "download.jpeg"
pil_image = Image.open(file_path)
```

```
image_b64 = convert_to_base64(pil_image)
plt_img_base64(image_b64)
```

```
<IPython.core.display.HTML object>
```

[2]:
```
from langchain.chat_models import ChatOllama
from langchain_core.messages import HumanMessage

chat_model = ChatOllama(
    model="bakllava:latest",
)

# Call the chat model with both messages and images
content_parts = []
image_part = {
    "type": "image_url",
    "image_url": f"data:image/jpeg;base64,{image_b64}",
}
text_part = {"type": "text", "text": "What is the Daollar-based gross retention␣
 ↪rate?"}

content_parts.append(image_part)
content_parts.append(text_part)
prompt = [HumanMessage(content=content_parts)]
chat_model(prompt)
```

```
---------------------------------------------------------------------------
OllamaEndpointNotFoundError               Traceback (most recent call last)
Cell In[2], line 19
     17 content_parts.append(text_part)
     18 prompt = [HumanMessage(content=content_parts)]
---> 19 chat_model(prompt)

File ~/.local/lib/python3.10/site-packages/langchain_core/language_models/
 ↪chat_models.py:636, in BaseChatModel.__call__(self, messages, stop, callbacks ␣
 ↪**kwargs)
    629 def __call__(
    630     self,
    631     messages: List[BaseMessage],
   (…)
    634     **kwargs: Any,
    635 ) -> BaseMessage:
--> 636     generation = self.generate(
    637         [messages], stop=stop, callbacks=callbacks, **kwargs
    638     ).generations[0][0]
    639     if isinstance(generation, ChatGeneration):
    640         return generation.message
```

```
File ~/.local/lib/python3.10/site-packages/langchain_core/language_models/
 ↪chat_models.py:382, in BaseChatModel.generate(self, messages, stop, callbacks ␣
 ↪tags, metadata, run_name, **kwargs)
    380         if run_managers:
    381             run_managers[i].on_llm_error(e,␣
 ↪response=LLMResult(generations=[]))
--> 382         raise e
    383 flattened_outputs = [
    384     LLMResult(generations=[res.generations], llm_output=res.llm_output)
    385     for res in results
    386 ]
    387 llm_output = self._combine_llm_outputs([res.llm_output for res in␣
 ↪results])

File ~/.local/lib/python3.10/site-packages/langchain_core/language_models/
 ↪chat_models.py:372, in BaseChatModel.generate(self, messages, stop, callbacks ␣
 ↪tags, metadata, run_name, **kwargs)
    369 for i, m in enumerate(messages):
    370     try:
    371         results.append(
--> 372             self._generate_with_cache(
    373                 m,
    374                 stop=stop,
    375                 run_manager=run_managers[i] if run_managers else None,
    376                 **kwargs,
    377             )
    378         )
    379     except BaseException as e:
    380         if run_managers:

File ~/.local/lib/python3.10/site-packages/langchain_core/language_models/
 ↪chat_models.py:528, in BaseChatModel._generate_with_cache(self, messages,␣
 ↪stop, run_manager, **kwargs)
    524     raise ValueError(
    525         "Asked to cache, but no cache found at `langchain.cache`."
    526     )
    527 if new_arg_supported:
--> 528     return self._generate(
    529         messages, stop=stop, run_manager=run_manager, **kwargs
    530     )
    531 else:
    532     return self._generate(messages, stop=stop, **kwargs)

File ~/.local/lib/python3.10/site-packages/langchain_community/chat_models/
 ↪ollama.py:209, in ChatOllama._generate(self, messages, stop, run_manager,␣
 ↪**kwargs)
    185 def _generate(
    186     self,
    187     messages: List[BaseMessage],
   (…)
    190     **kwargs: Any,
    191 ) -> ChatResult:
    192     """Call out to Ollama's generate endpoint.
    193 
    194     Args:
   (…)
    206             ])
    207     """
--> 209     final_chunk = self._chat_stream_with_aggregation(
    210         messages,
    211         stop=stop,
    212         run_manager=run_manager,
    213         verbose=self.verbose,
    214         **kwargs,
    215     )
    216     chat_generation = ChatGeneration(
    217         message=AIMessage(content=final_chunk.text),
    218         generation_info=final_chunk.generation_info,
    219     )
    220     return ChatResult(generations=[chat_generation])

File ~/.local/lib/python3.10/site-packages/langchain_community/chat_models/ollama.py:168, in ChatOllama._chat_stream_with_aggregation(self, messages, stop, run_manager, verbose, **kwargs)
    159 def _chat_stream_with_aggregation(
    160     self,
    161     messages: List[BaseMessage],
   (…)
    165     **kwargs: Any,
    166 ) -> ChatGenerationChunk:
    167     final_chunk: Optional[ChatGenerationChunk] = None
--> 168     for stream_resp in self._create_chat_stream(messages, stop, **kwargs):
    169         if stream_resp:
    170             chunk = _chat_stream_response_to_chat_generation_chunk(stream_resp)

File ~/.local/lib/python3.10/site-packages/langchain_community/chat_models/ollama.py:155, in ChatOllama._create_chat_stream(self, messages, stop, **kwargs)
    146 def _create_chat_stream(
    147     self,
    148     messages: List[BaseMessage],
    149     stop: Optional[List[str]] = None,
    150     **kwargs: Any,
    151 ) -> Iterator[str]:
```

```
    152     payload = {
    153         "messages": self._convert_messages_to_ollama_messages(messages)
    154     }
--> 155     yield from self._create_stream(
    156␣
  ↪        payload=payload, stop=stop, api_url=f"{self.base_url}/api/chat/", **kwargs
    157     )

File ~/.local/lib/python3.10/site-packages/langchain_community/llms/ollama.py:
 ↪198, in _OllamaCommon._create_stream(self, api_url, payload, stop, **kwargs)
    196 if response.status_code != 200:
    197     if response.status_code == 404:
--> 198         raise OllamaEndpointNotFoundError(
    199             "Ollama call failed with status code 404."
    200         )
    201     else:
    202         optional_detail = response.json().get("error")

OllamaEndpointNotFoundError: Ollama call failed with status code 404.
```

[3]:
```
!ollama list
```

```
NAME                    ID              SIZE    MODIFIED
bakllava:latest         3dd68bd4447c    4.7 GB  29 hours ago
mistral:instruct        4d9f4b269c33    4.1 GB  20 hours ago
neural-chat:latest      73940af9fe02    4.1 GB  4 weeks ago
orca2:13b               a8dcfac3ac32    7.4 GB  4 weeks ago
starling-lm:latest      0eab7e16513a    4.1 GB  3 weeks ago
yi:latest               59e2d70c6939    3.5 GB  4 weeks ago
zephyr:7b-beta-q4_0     1629f2a8a495    4.1 GB  6 weeks ago
zephyr:latest           1629f2a8a495    4.1 GB  4 weeks ago
```

[4]:
```
from langchain.callbacks.manager import CallbackManager
from langchain.callbacks.streaming_stdout import StreamingStdOutCallbackHandler
from langchain.llms import Ollama

llm = Ollama(
    model="mistral:instruct",
    callback_manager=CallbackManager([StreamingStdOutCallbackHandler()])
)

llm("Tell me about the history of AI")
```

```
 The concept of artificial intelligence (AI) can be traced back to ancient
Greece, where philosophers like Talbot and Cornelius Agrippa wrote about the
possibility of creating automated beings. However, the modern history of AI
begins in the mid-20th century.
```

```
1. Early Beginnings: In 1943, Warren McCulloch and Walter Pitts created the
first artificial neuron model. This marked the beginning of efforts to develop
machines that could mimic human intelligence. In 1950, Alan Turing proposed the
concept of a "universal machine" that could perform any task a human being can
do. He also introduced the famous "Turing Test," which measures a machine's
ability to mimic human conversation.

2. First AI Projects: The first formal AI research project was initiated at
Dartmouth College in 1956, led by Marvin Minsky and John McCarthy. They aimed to
create a machine that could learn from experience. Around the same time, Allen
Newell and Herbert A. Simmons developed the Logic Theorist program, which could
prove mathematical theorems.

3. AI Winter: By the late 1960s and early 1970s, it became clear that achieving
true AI was much more complex than initially thought. Funding for AI research
dried up, leading to a period known as "AI winter." During this time,
researchers focused on narrower applications of AI, such as expert systems and
machine vision.

4. Advancements and Success Stories: In the late 1980s and early 1990s, advances
in computing power and machine learning algorithms led to renewed interest in
AI. IBM's Deep Blue defeated world champion Garry Kasparov at chess in 1997,
demonstrating a machine's ability to surpass human intelligence in specific
tasks. In the late 1990s and early 2000s, AI was applied to various industries,
including healthcare, finance, and manufacturing.

5. Modern AI: With the advent of big data, cloud computing, and the
proliferation of smartphones, AI has become an essential part of our daily
lives. Advancements in deep learning and neural networks have enabled AI systems
to perform tasks such as image recognition, speech recognition, and natural
language processing with remarkable accuracy. In recent years, AI has achieved
notable successes, including AlphaGo's victory over world champion Go players,
autonomous vehicles, and chatbots like ChatGPT.

6. Ethical and Societal Concerns: As AI continues to evolve, ethical and
societal concerns have arisen, such as privacy, job displacement, and the
potential for misuse. These issues are being addressed by governments, academic
institutions, and industry leaders through research, regulation, and ethical
guidelines.

Today, AI is transforming industries and improving our lives in numerous ways.
However, it's essential to remember that true AI, which can match or surpass
human intelligence across all domains, remains a work in progress.
```

```
[4]: ' The concept of artificial intelligence (AI) can be traced back to ancient
     Greece, where philosophers like Talbot and Cornelius Agrippa wrote about the
```

```
possibility of creating automated beings. However, the modern history of AI
begins in the mid-20th century.\n\n1. Early Beginnings: In 1943, Warren
McCulloch and Walter Pitts created the first artificial neuron model. This
marked the beginning of efforts to develop machines that could mimic human
intelligence. In 1950, Alan Turing proposed the concept of a "universal machine"
that could perform any task a human being can do. He also introduced the famous
"Turing Test," which measures a machine\'s ability to mimic human
conversation.\n\n2. First AI Projects: The first formal AI research project was
initiated at Dartmouth College in 1956, led by Marvin Minsky and John McCarthy.
They aimed to create a machine that could learn from experience. Around the same
time, Allen Newell and Herbert A. Simmons developed the Logic Theorist program,
which could prove mathematical theorems.\n\n3. AI Winter: By the late 1960s and
early 1970s, it became clear that achieving true AI was much more complex than
initially thought. Funding for AI research dried up, leading to a period known
as "AI winter." During this time, researchers focused on narrower applications
of AI, such as expert systems and machine vision.\n\n4. Advancements and Success
Stories: In the late 1980s and early 1990s, advances in computing power and
machine learning algorithms led to renewed interest in AI. IBM\'s Deep Blue
defeated world champion Garry Kasparov at chess in 1997, demonstrating a
machine\'s ability to surpass human intelligence in specific tasks. In the late
1990s and early 2000s, AI was applied to various industries, including
healthcare, finance, and manufacturing.\n\n5. Modern AI: With the advent of big
data, cloud computing, and the proliferation of smartphones, AI has become an
essential part of our daily lives. Advancements in deep learning and neural
networks have enabled AI systems to perform tasks such as image recognition,
speech recognition, and natural language processing with remarkable accuracy. In
recent years, AI has achieved notable successes, including AlphaGo\'s victory
over world champion Go players, autonomous vehicles, and chatbots like
ChatGPT.\n\n6. Ethical and Societal Concerns: As AI continues to evolve, ethical
and societal concerns have arisen, such as privacy, job displacement, and the
potential for misuse. These issues are being addressed by governments, academic
institutions, and industry leaders through research, regulation, and ethical
guidelines.\n\nToday, AI is transforming industries and improving our lives in
numerous ways. However, it\'s essential to remember that true AI, which can
match or surpass human intelligence across all domains, remains a work in
progress.'
```

[5]:
```python
from langchain.callbacks.manager import CallbackManager
from langchain.callbacks.streaming_stdout import StreamingStdOutCallbackHandler
from langchain.chat_models import ChatOllama

chat_model = ChatOllama(
    model="mistral:instruct",
    format="json",
    callback_manager=CallbackManager([StreamingStdOutCallbackHandler()]),
)
```

```python
from langchain.schema import HumanMessage

messages = [
    HumanMessage(
        content="What color is the sky at different times of the day? Respond␣
 ↪using JSON"
    )
]

chat_model_response = chat_model(messages)
chat_model_response
```

```
---------------------------------------------------------------------------
OllamaEndpointNotFoundError               Traceback (most recent call last)
Cell In[5], line 19
     11 from langchain.schema import HumanMessage
     13 messages = [
     14     HumanMessage(
     15         content="What color is the sky at different times of the day?␣
 ↪Respond using JSON"
     16     )
     17 ]
---> 19 chat_model_response = chat_model(messages)
     20 chat_model_response

File ~/.local/lib/python3.10/site-packages/langchain_core/language_models/
 ↪chat_models.py:636, in BaseChatModel.__call__(self, messages, stop, callbacks␣
 ↪**kwargs)
    629 def __call__(
    630     self,
    631     messages: List[BaseMessage],
   (…)
    634     **kwargs: Any,
    635 ) -> BaseMessage:
--> 636     generation = self.generate(
    637         [messages], stop=stop, callbacks=callbacks, **kwargs
    638     ).generations[0][0]
    639     if isinstance(generation, ChatGeneration):
    640         return generation.message

File ~/.local/lib/python3.10/site-packages/langchain_core/language_models/
 ↪chat_models.py:382, in BaseChatModel.generate(self, messages, stop, callbacks␣
 ↪tags, metadata, run_name, **kwargs)
    380         if run_managers:
    381             run_managers[i].on_llm_error(e,␣
 ↪response=LLMResult(generations=[]))
--> 382         raise e
```

```
    383 flattened_outputs = [
    384     LLMResult(generations=[res.generations], llm_output=res.llm_output)
    385     for res in results
    386 ]
    387 llm_output = self._combine_llm_outputs([res.llm_output for res in␣
  ↪results])

File ~/.local/lib/python3.10/site-packages/langchain_core/language_models/
  ↪chat_models.py:372, in BaseChatModel.generate(self, messages, stop, callbacks ␣
  ↪tags, metadata, run_name, **kwargs)
    369 for i, m in enumerate(messages):
    370     try:
    371         results.append(
--> 372             self._generate_with_cache(
    373                 m,
    374                 stop=stop,
    375                 run_manager=run_managers[i] if run_managers else None,
    376                 **kwargs,
    377             )
    378         )
    379     except BaseException as e:
    380         if run_managers:

File ~/.local/lib/python3.10/site-packages/langchain_core/language_models/
  ↪chat_models.py:528, in BaseChatModel._generate_with_cache(self, messages,␣
  ↪stop, run_manager, **kwargs)
    524     raise ValueError(
    525         "Asked to cache, but no cache found at `langchain.cache`."
    526     )
    527 if new_arg_supported:
--> 528     return self._generate(
    529         messages, stop=stop, run_manager=run_manager, **kwargs
    530     )
    531 else:
    532     return self._generate(messages, stop=stop, **kwargs)

File ~/.local/lib/python3.10/site-packages/langchain_community/chat_models/
  ↪ollama.py:209, in ChatOllama._generate(self, messages, stop, run_manager,␣
  ↪**kwargs)
    185 def _generate(
    186     self,
    187     messages: List[BaseMessage],
   (…)
    190     **kwargs: Any,
    191 ) -> ChatResult:
    192     """Call out to Ollama's generate endpoint.
    193
    194     Args:
```

```
   (…)
    206             ])
    207     """
--> 209     final_chunk = self._chat_stream_with_aggregation(
    210         messages,
    211         stop=stop,
    212         run_manager=run_manager,
    213         verbose=self.verbose,
    214         **kwargs,
    215     )
    216     chat_generation = ChatGeneration(
    217         message=AIMessage(content=final_chunk.text),
    218         generation_info=final_chunk.generation_info,
    219     )
    220     return ChatResult(generations=[chat_generation])

File ~/.local/lib/python3.10/site-packages/langchain_community/chat_models/
  ↪ollama.py:168, in ChatOllama._chat_stream_with_aggregation(self, messages,␣
  ↪stop, run_manager, verbose, **kwargs)
    159 def _chat_stream_with_aggregation(
    160     self,
    161     messages: List[BaseMessage],
   (…)
    165     **kwargs: Any,
    166 ) -> ChatGenerationChunk:
    167     final_chunk: Optional[ChatGenerationChunk] = None
--> 168     for stream_resp in self._create_chat_stream(messages, stop,␣
  ↪**kwargs):
    169         if stream_resp:
    170             chunk =␣
  ↪_chat_stream_response_to_chat_generation_chunk(stream_resp)

File ~/.local/lib/python3.10/site-packages/langchain_community/chat_models/
  ↪ollama.py:155, in ChatOllama._create_chat_stream(self, messages, stop,␣
  ↪**kwargs)
    146 def _create_chat_stream(
    147     self,
    148     messages: List[BaseMessage],
    149     stop: Optional[List[str]] = None,
    150     **kwargs: Any,
    151 ) -> Iterator[str]:
    152     payload = {
    153         "messages": self._convert_messages_to_ollama_messages(messages)
    154     }
--> 155     yield from self._create_stream(
    156␣
  ↪        payload=payload, stop=stop, api_url=f"{self.base_url}/api/chat/", **kwargs
    157     )
```

```
File ~/.local/lib/python3.10/site-packages/langchain_community/llms/ollama.py:
 ↪198, in _OllamaCommon._create_stream(self, api_url, payload, stop, **kwargs)
    196 if response.status_code != 200:
    197     if response.status_code == 404:
--> 198         raise OllamaEndpointNotFoundError(
    199             "Ollama call failed with status code 404."
    200         )
    201     else:
    202         optional_detail = response.json().get("error")

OllamaEndpointNotFoundError: Ollama call failed with status code 404.
```