Langchain's Custom Chat Model Class Inheritance and method overriding for Minimax

```python
from typing import Any, Dict, Iterator, List, Optional
from langchain_core.callbacks import CallbackManagerForLLMRun
from langchain_core.language_models import BaseChatModel
from langchain_core.messages import AIMessage, AIMessageChunk,
BaseMessage
from langchain_core.messages.ai import UsageMetadata
from langchain_core.outputs import ChatGeneration,
ChatGenerationChunk, ChatResult
from pydantic import Field, BaseModel, SecretStr
from langchain_core.utils.utils import secret_from_env
from openai import OpenAI

class ChatMinimax(BaseChatModel):
    model_name: str = Field(default="MiniMax-Text-01")
    temperature: Optional[float] = 0.5
    max_tokens: Optional[int] = 1024
    timeout: Optional[int] = None
    stop: Optional[List[str]] = None
    max_retries: int = 2
    minimax_api_key: Optional[SecretStr] = Field(
        alias="api_key",
default_factory=secret_from_env("MINIMAX_API_KEY", default=None)
    )

    def _generate(
        self,
        messages: List[BaseMessage],
        stop: Optional[List[str]] = None,
        run_manager: Optional[CallbackManagerForLLMRun] = None,
        **kwargs: Any,
    ) -> ChatResult:
        client =
OpenAI(api_key=self.minimax_api_key.get_secret_value(),
base_url="https://api.minimaxi.chat/v1")
        response = client.chat.completions.create(
            model=self.model_name,
            messages=[{"role": "user", "content": message.content} for
message in messages],
            temperature=self.temperature,
            max_tokens=self.max_tokens,
        )

        aimessage = AIMessage(
            content=response.choices[0].message.content,
            usage_metadata={
                "input_tokens": response.usage.prompt_tokens,
                "output_tokens": response.usage.completion_tokens,
                "total_tokens": response.usage.total_tokens,
            },
        )

        return
ChatResult(generations=[ChatGeneration(message=aimessage)])

    def __repr__(self):
        return f"ChatMinimax(model_name='{self.model_name}',
temperature={self.temperature})"

    def _stream(
        self,
        messages: List[BaseMessage],
        stop: Optional[List[str]] = None,
        run_manager: Optional[CallbackManagerForLLMRun] = None,
        **kwargs: Any,
    ) -> Iterator[ChatGenerationChunk]:
        client =
OpenAI(api_key=self.minimax_api_key.get_secret_value(),
base_url="https://api.minimaxi.chat/v1")

        stream = client.chat.completions.create(
            model=self.model_name,
            messages=[{"role": "user", "content": message.content} for
message in messages],
            temperature=self.temperature,
            max_tokens=self.max_tokens,
            stream=True,
        )

        for chunk in stream:
            if chunk.choices[0].delta.content:
                yield ChatGenerationChunk(
                    message=AIMessageChunk(
                        content=chunk.choices[0].delta.content
                    )
                )

    @property
    def _llm_type(self) -> str:
        return "Minimax-Text-01"

    @property
    def _identifying_params(self) -> Dict[str, Any]:
        return {
            "model_name": self.model_name,
        }
```

Invoking calls (Asynchronous & Asynchronous)

```
from google.colab import userdata

model = ChatMinimax(model_name="MiniMax-Text-01", temperature=0.7,
max_output_tokens=2048,api_key=userdata.get("MINIMAX_API_KEY"))
model

response = model.invoke("Hi how can I help you")
print(response)
```

```
content='Hi there! It’s great to hear from you. I’m here to help
answer any questions you might have or engage in a conversation on any
topic you’re interested in. Whether it’s about technology, hobbies,
current events, or just a friendly chat, I’m all ears. How can I
assist you today?' additional_kwargs={} response_metadata={} id='run-
76b08514-8b35-424c-aba8-eda88197f85d-0'
usage_metadata={'input_tokens': 749, 'output_tokens': 64,
'total_tokens': 813}
```

Streaming

```
for chunk in model.stream("Hi who are you"):
  print(chunk.content, end="",flush=True)
```

```
Hello! I am 海螺AI, an AI assistant developed to help answer
questions, provide information, and engage in discussions on a wide
```

```
range of topics. I'm here to assist you with whatever you need,
whether it's learning something new, solving a problem, or just having
a friendly chat. How can I assist you today?
```

Unit Tests

```
import unittest
from langchain_core.messages import HumanMessage

class TestChatMinimax(unittest.TestCase):
    def test_chat_response(self):
      model = ChatMinimax(model_name="MiniMax-Text-01",
temperature=0.7,
max_output_tokens=2048,api_key=userdata.get("MINIMAX_API_KEY"))
      response = model.invoke("Hello how can I help you")

# Run tests manually
suite = unittest.TestLoader().loadTestsFromTestCase(TestChatMinimax)
unittest.TextTestRunner(verbosity=2).run(suite)
```

```
test_chat_response (__main__.TestChatMinimax.test_chat_response) ...
ok

----------------------------------------------------------------------
Ran 1 test in 5.144s

OK

<unittest.runner.TextTestResult run=1 errors=0 failures=0>
```

```
import base64
import httpx
from typing import Any, Dict, Iterator, List, Optional, Union
from langchain_core.callbacks import CallbackManagerForLLMRun
from langchain_core.language_models import BaseChatModel
from langchain_core.messages import AIMessage, AIMessageChunk,
BaseMessage
from langchain_core.messages.ai import UsageMetadata
from langchain_core.outputs import ChatGeneration,
ChatGenerationChunk, ChatResult
from pydantic import Field, SecretStr
from langchain_core.utils.utils import secret_from_env
from openai import OpenAI

class ChatMinimax(BaseChatModel):
    model_name: str = Field(default="MiniMax-Text-01")
    temperature: Optional[float] = 0.5
    max_tokens: Optional[int] = 1024
    timeout: Optional[int] = None
    stop: Optional[List[str]] = None
    max_retries: int = 2
    minimax_api_key: Optional[SecretStr] = Field(
        alias="api_key",
default_factory=secret_from_env("MINIMAX_API_KEY", default=None)
    )

    @staticmethod
    def encode_image(image_path: str) -> str:
      if not image_path.startswith("http"):
        with open(image_path, "rb") as image_file:
          return base64.b64encode(image_file.read()).decode('utf-8')
      else:
        response = httpx.get(image_path)
        response.raise_for_status()
        return base64.b64encode(response.content).decode('utf-8')

    def _generate(
        self,
        messages: List[BaseMessage],
        image: Optional[Union[str, bytes]] = None,
        image_is_base64: bool = False,
        stop: Optional[List[str]] = None,
        run_manager: Optional[CallbackManagerForLLMRun] = None,
        **kwargs: Any,
    ) -> ChatResult:
        """Generate a response from MiniMax's API, with optional image
input."""

        client =
OpenAI(api_key=self.minimax_api_key.get_secret_value(),
base_url="https://api.minimaxi.chat/v1")

        # Format messages
        formatted_messages = [
            {"role": "system", "content": "MM Intelligent Assistant is
a self-developed MiniMax model."},
            {
                "role": "user",
                "name": "user",
                "content": [{"type": "text", "text": msg.content} for
msg in messages],
            }
        ]

        # Add image input if provided
        if image:
            image_payload = {
                "type": "image_url",
                "image_url": {
                    "url": f"data:image/jpeg;base64,{image}" if
image_is_base64 else image
                }
            }
            formatted_messages[1]["content"].append(image_payload)

        # API call
        response = client.chat.completions.create(
            model=self.model_name,
            messages=formatted_messages,
            temperature=self.temperature,
            max_tokens=self.max_tokens,
        )

        aimessage = AIMessage(
            content=response.choices[0].message.content,
            usage_metadata=UsageMetadata(
                input_tokens=response.usage.prompt_tokens,
                output_tokens=response.usage.completion_tokens,
                total_tokens=response.usage.total_tokens,
            ),
        )

        return
ChatResult(generations=[ChatGeneration(message=aimessage)])

    def _stream(
        self,
        messages: List[BaseMessage],
        image: Optional[Union[str, bytes]] = None,
        image_is_base64: bool = False,
        stop: Optional[List[str]] = None,
        run_manager: Optional[CallbackManagerForLLMRun] = None,
        **kwargs: Any,
    ) -> Iterator[ChatGenerationChunk]:
        """Stream responses from MiniMax API with optional image
input."""

        client =
OpenAI(api_key=self.minimax_api_key.get_secret_value(),
base_url="https://api.minimaxi.chat/v1")

        formatted_messages = [
            {"role": "system", "content": "MM Intelligent Assistant is
a self-developed MiniMax model."},
            {
                "role": "user",
                "name": "user",
                "content": [{"type": "text", "text": msg.content} for
msg in messages],
            }
        ]

        if image:
            image_payload = {
                "type": "image_url",
                "image_url": {
                    "url": f"data:image/jpeg;base64,{image}" if
image_is_base64 else image
                }
            }
            formatted_messages[1]["content"].append(image_payload)

        stream = client.chat.completions.create(
            model=self.model_name,
            messages=formatted_messages,
            temperature=self.temperature,
            max_tokens=self.max_tokens,
            stream=True,
        )

        for chunk in stream:
            if chunk.choices[0].delta.content:
                yield ChatGenerationChunk(

message=AIMessageChunk(content=chunk.choices[0].delta.content)
                )

    @property
    def _llm_type(self) -> str:
        return "Minimax-Text-01"

    @property
    def _identifying_params(self) -> Dict[str, Any]:
        return {"model_name": self.model_name}
```

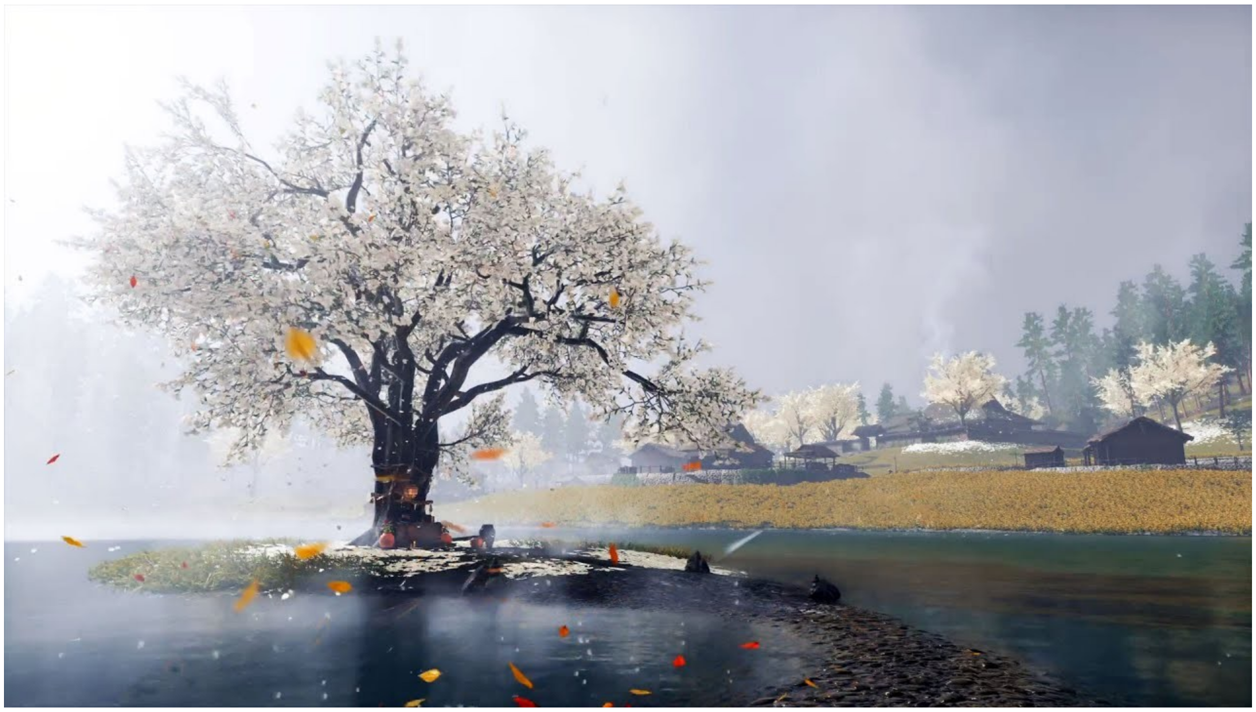
Minimax's Multimodal Understanding on different types of images

Google's Titan Architecture aiming to enhance cognitive memory mimicking that of human's

```
image_base64 =
ChatMinimax.encode_image("https://miro.medium.com/v2/resize:fit:1400/1
*_YLjLN1GDHtAFWhr3rW5Pw.png")
response = model.invoke("What is in this image?", image=image_base64,
image_is_base64=True)
response
```

```
AIMessage(content='This image illustrates a neural memory architecture
used in machine learning models, particularly those involving
attention mechanisms and memory retrieval processes. It shows the flow
of information between different components such as neural memory,
core sequence, and persistent memory, and how they interact during
learning and testing phases. The diagram highlights the retrieval and
updating of neural memory, the sequence processing in the core, and
the use of learnable data-independent weights in persistent memory.',
additional_kwargs={}, response_metadata={}, id='run-791b1a2a-9400-
4d2a-82a4-1cdc953f1040-0', usage_metadata={'input_tokens': 3723,
'output_tokens': 85, 'total_tokens': 3808})
```

Ghost of Tsushima Scenery



```
image_base64 =
ChatMinimax.encode_image("https://i.ytimg.com/vi/U54zml7zwng/maxresdef
ault.jpg")
response = model.invoke("What is in this image?", image=image_base64,
image_is_base64=True)
response
```

```
AIMessage(content="The image depicts a serene and picturesque scene of
a large tree covered in white blossoms, situated on a small piece of
land that juts out into a calm body of water. The tree is the central
focus of the image, with its branches spreading wide and covered in
delicate white flowers, suggesting that it might be a cherry blossom
tree in full bloom. The tree trunk is dark and sturdy, contrasting
with the lightness of the blossoms.\n\nAround the tree, there are a
few scattered leaves in various colors, including orange and red,
which could indicate the transition from spring to autumn or simply
the natural shedding of leaves. The ground around the tree base
appears slightly damp or muddy, suggesting recent rain or the
proximity to the water.\n\nThe body of water surrounding the tree is
calm and reflective, with a misty or foggy atmosphere that adds a
dreamy and tranquil quality to the scene. This mist partially obscures
the background, creating a sense of depth and mystery. The water's
surface is relatively still, with only slight ripples disturbing its
mirror-like quality.\n\nIn the background, there are several small
wooden structures, possibly cabins or cottages, nestled among more
trees that also appear to be in bloom. These structures are simple and
rustic, blending harmoniously with the natural surroundings. The
background trees are also covered in white blossoms, similar to the
central tree, suggesting a landscape dominated by cherry blossoms or a
similar flowering species.\n\nThe overall color palette of the image
is soft and muted, with the white blossoms, misty atmosphere, and calm
water creating a peaceful and almost ethereal ambiance. The scattered
autumn leaves add a touch of warmth and contrast to the predominantly
cool tones of the scene.\n\nThe image evokes a sense of tranquility
and natural beauty, with the blooming tree as the centerpiece of a
serene and idyllic landscape. The misty atmosphere and calm water
enhance the feeling of peace and quiet, making it a visually soothing
and aesthetically pleasing scene.", additional_kwargs={},
response_metadata={}, id='run-26a203a7-2f70-44c3-bfdc-0a56fb1d4201-0',
usage_metadata={'input_tokens': 5857, 'output_tokens': 383,
'total_tokens': 6240})
```