```
try:
    from litellm import completion
except ImportError:
    import subprocess

    subprocess.check_call(["pip", "install", "litellm"])
    import litellm
    from litellm import completion

    litellm.set_verbose = True
    litellm.ssl_verify = False


class LiteLLM:
    """
    This class represents a LiteLLM.
    It is used to interact with the LLM model for various tasks.
    """

    def __init__(
        self,
        model_name: str = "gpt-4o",
        system_prompt: str = None,
        stream: bool = False,
        temperature: float = 0.5,
        max_tokens: int = 4000,
```

```
    ssl_verify: bool = False,
):
    """
    Initialize the LiteLLM with the given parameters.

    Args:
        model_name (str, optional): The name of the model to use. Defaults to "gpt-4o".
        system_prompt (str, optional): The system prompt to use. Defaults to None.
        stream (bool, optional): Whether to stream the output. Defaults to False.
        temperature (float, optional): The temperature for the model. Defaults to 0.5.
        max_tokens (int, optional): The maximum number of tokens to generate. Defaults to 4000.
    """
    self.model_name = model_name
    self.system_prompt = system_prompt
    self.stream = stream
    self.temperature = temperature
    self.max_tokens = max_tokens
    self.ssl_verify = ssl_verify

def _prepare_messages(self, task: str) -> list:
    """
    Prepare the messages for the given task.

    Args:
        task (str): The task to prepare messages for.
```

```
    Returns:
        list: A list of messages prepared for the task.
    """
    messages = []

    if self.system_prompt:  # Check if system_prompt is not None
        messages.append(
            {"role": "system", "content": self.system_prompt}
        )

    messages.append({"role": "user", "content": task})

    return messages

def run(self, task: str, *args, **kwargs):
    """
    Run the LLM model for the given task.

    Args:
        task (str): The task to run the model for.
        *args: Additional positional arguments to pass to the model.
        **kwargs: Additional keyword arguments to pass to the model.

    Returns:
        str: The content of the response from the model.
    """
```

```
    try:

        messages = self._prepare_messages(task)

        response = completion(
            model=self.model_name,
            messages=messages,
            stream=self.stream,
            temperature=self.temperature,
            max_tokens=self.max_tokens,
            *args,
            **kwargs,
        )

        content = response.choices[
            0
        ].message.content  # Accessing the content

        return content
    except Exception as error:
        print(error)

def __call__(self, task: str, *args, **kwargs):
    """
    Call the LLM model for the given task.
```

Args:

task (str): The task to run the model for.

*args: Additional positional arguments to pass to the model.

**kwargs: Additional keyword arguments to pass to the model.

Returns:

str: The content of the response from the model.

"""

return self.run(task, *args, **kwargs)