```
# !pip install accelerate
# !pip install torch
# !pip install transformers
# !pip install bitsandbytes

from typing import Callable, Dict, List

import torch
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig,
    TextStreamer,
)
from swarm_models.base_llm import BaseLLM


class LlamaFunctionCaller(BaseLLM):
    """
    A class to manage and execute Llama functions.

    Attributes:
    ----------
    model: transformers.AutoModelForCausalLM
        The loaded Llama model.
    tokenizer: transformers.AutoTokenizer
```

    The tokenizer for the Llama model.

functions: Dict[str, Callable]

    A dictionary of functions available for execution.

Methods:

__init__(self, model_id: str, cache_dir: str, runtime: str)

    Initializes the LlamaFunctionCaller with the specified model.

add_func(self, name: str, function: Callable, description: str, arguments: List[Dict])

    Adds a new function to the LlamaFunctionCaller.

call_function(self, name: str, **kwargs)

    Calls the specified function with given arguments.

stream(self, user_prompt: str)

    Streams a user prompt to the model and prints the response.

Example:

```
# Example usage
model_id = "Your-Model-ID"
cache_dir = "Your-Cache-Directory"
runtime = "cuda"  # or 'cpu'

llama_caller = LlamaFunctionCaller(model_id, cache_dir, runtime)
```

```
# Add a custom function
def get_weather(location: str, format: str) -> str:
    # This is a placeholder for the actual implementation
    return f"Weather at {location} in {format} format."



llama_caller.add_func(
    name="get_weather",
    function=get_weather,
    description="Get the weather at a location",
    arguments=[
        {
            "name": "location",
            "type": "string",
            "description": "Location for the weather",
        },
        {
            "name": "format",
            "type": "string",
            "description": "Format of the weather data",
        },
    ],
)

# Call the function
result = llama_caller.call_function("get_weather", location="Paris", format="Celsius")
```

```
print(result)

# Stream a user prompt
llama_caller("Tell me about the tallest mountain in the world.")

"""

def __init__(
    self,
    model_id: str = "Trelis/Llama-2-7b-chat-hf-function-calling-v2",
    cache_dir: str = "llama_cache",
    runtime: str = "auto",
    max_tokens: int = 500,
    streaming: bool = False,
    *args,
    **kwargs,
):
    self.model_id = model_id
    self.cache_dir = cache_dir
    self.runtime = runtime
    self.max_tokens = max_tokens
    self.streaming = streaming

    # Load the model and tokenizer
    self.model = self._load_model()
    self.tokenizer = AutoTokenizer.from_pretrained(
```

```
        model_id, cache_dir=cache_dir, use_fast=True
    )
    self.functions = {}


def _load_model(self):
    # Configuration for loading the model
    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_use_double_quant=True,
        bnb_4bit_quant_type="nf4",
        bnb_4bit_compute_dtype=torch.bfloat16,
    )
    return AutoModelForCausalLM.from_pretrained(
        self.model_id,
        quantization_config=bnb_config,
        device_map=self.runtime,
        trust_remote_code=True,
        cache_dir=self.cache_dir,
    )


def add_func(
    self,
    name: str,
    function: Callable,
    description: str,
    arguments: List[Dict],
```

```
):
    """
    Adds a new function to the LlamaFunctionCaller.

    Args:
        name (str): The name of the function.
        function (Callable): The function to execute.
        description (str): Description of the function.
        arguments (List[Dict]): List of argument specifications.
    """
    self.functions[name] = {
        "function": function,
        "description": description,
        "arguments": arguments,
    }

def call_function(self, name: str, **kwargs):
    """
    Calls the specified function with given arguments.

    Args:
        name (str): The name of the function to call.
        **kwargs: Keyword arguments for the function call.

    Returns:
        The result of the function call.
```

```
    """
    if name not in self.functions:
        raise ValueError(f"Function {name} not found.")

    func_info = self.functions[name]
    return func_info["function"](**kwargs)

def __call__(self, task: str, **kwargs):
    """
    Streams a user prompt to the model and prints the response.

    Args:
        task (str): The user prompt to stream.
    """
    # Format the prompt
    prompt = f"{task}\n\n"

    # Encode and send to the model
    inputs = self.tokenizer([prompt], return_tensors="pt").to(
        self.runtime
    )

    streamer = TextStreamer(self.tokenizer)

    if self.streaming:
        out = self.model.generate(
```

```
                **inputs,
                streamer=streamer,
                max_new_tokens=self.max_tokens,
                **kwargs,
            )

            return out
        else:
            out = self.model.generate(
                **inputs, max_length=self.max_tokens, **kwargs
            )
            # return self.tokenizer.decode(out[0], skip_special_tokens=True)
            return out


# llama_caller = LlamaFunctionCaller()


# # Add a custom function
# def get_weather(location: str, format: str) -> str:
#     # This is a placeholder for the actual implementation
#     return f"Weather at {location} in {format} format."


# llama_caller.add_func(
#     name="get_weather",
```

```
#     function=get_weather,
#     description="Get the weather at a location",
#     arguments=[
#         {
#             "name": "location",
#             "type": "string",
#             "description": "Location for the weather",
#         },
#         {
#             "name": "format",
#             "type": "string",
#             "description": "Format of the weather data",
#         },
#     ],
# )

# # Call the function
# result = llama_caller.call_function("get_weather", location="Paris", format="Celsius")
# print(result)

# # Stream a user prompt
# llama_caller("Tell me about the tallest mountain in the world.")
```