```
from dataclasses import dataclass, field
from typing import Optional, Tuple


from PIL import Image
from transformers import AutoModelForCausalLM, AutoTokenizer


from swarm_models.base_multimodal_model import BaseMultiModalModel



@dataclass
class QwenVLMultiModal(BaseMultiModalModel):
    """
    QwenVLMultiModal is a class that represents a multi-modal model for Qwen chatbot.
    It inherits from the BaseMultiModalModel class.



    Args:
        model_name (str): The name of the model to be used.
        device (str): The device to run the model on.
        args (tuple): Additional positional arguments.
        kwargs (dict): Additional keyword arguments.
        quantize (bool): A flag to indicate whether to quantize the model.
            return_bounding_boxes (bool): A flag to indicate whether to return bounding boxes for the
image.
```

```
Examples:
>>> qwen = QwenVLMultiModal()
>>> response = qwen.run("Hello", "https://example.com/image.jpg")
>>> print(response)
"""

model_name: str = "Qwen/Qwen-VL"
device: str = "cuda"
args: tuple = field(default_factory=tuple)
kwargs: dict = field(default_factory=dict)
quantize: bool = False
return_bounding_boxes: bool = False

def __post_init__(self):
    """
    Initializes the QwenVLMultiModal object.
    It initializes the tokenizer and the model for the Qwen chatbot.
    """

    if self.quantize:
        self.model_name = "Qwen/Qwen-VL-Chat-Int4"

    self.tokenizer = AutoTokenizer.from_pretrained(
        self.model_name, trust_remote_code=True
    )
    self.model = AutoModelForCausalLM.from_pretrained(
            self.model_name,
            device_map=self.device,
            trust_remote_code=True,
        ).eval()


    def run(
        self, text: str, img: str, *args, **kwargs
    ) -> Tuple[Optional[str], Optional[Image.Image]]:
        """
        Runs the Qwen chatbot model on the given text and image inputs.


        Args:
            text (str): The input text for the chatbot.
            img (str): The input image for the chatbot.
            *args: Additional positional arguments.
            **kwargs: Additional keyword arguments.


        Returns:
              Tuple[Optional[str], Optional[Image.Image]]: A tuple containing the response generated by
the chatbot
            and the image associated with the response (if any).
        """
        try:
            if self.return_bounding_boxes:
                query = self.tokenizer.from_list_format(
                    [
            {"image": img, "text": text},
        ]
    )

    inputs = self.tokenizer(query, return_tensors="pt")
    inputs = inputs.to(self.model.device)
    pred = self.model.generate(**inputs)
    response = self.tokenizer.decode(
        pred.cpu()[0], skip_special_tokens=False
    )

    image_bb = self.tokenizer.draw_bbox_on_latest_picture(
        response
    )

    if image_bb:
        image_bb.save("output.jpg")
    else:
        print("No bounding boxes found in the image.")

    return response, image_bb
else:
    query = self.tokenizer.from_list_format(
        [
            {"image": img, "text": text},
        ]
```

```
            )

            inputs = self.tokenizer(query, return_tensors="pt")
            inputs = inputs.to(self.model.device)
            pred = self.model.generate(**inputs)
            response = self.tokenizer.decode(
                pred.cpu()[0], skip_special_tokens=False
            )
            return response
    except Exception as error:
        print(f"[ERROR]: [QwenVLMultiModal]: {error}")

def chat(
    self, text: str, img: str, *args, **kwargs
) -> tuple[str, list]:
    """
    Chat with the model using text and image inputs.

    Args:
        text (str): The text input for the chat.
        img (str): The image input for the chat.
        *args: Additional positional arguments.
        **kwargs: Additional keyword arguments.

    Returns:
        tuple[str, list]: A tuple containing the response and chat history.
```

```
Raises:
    Exception: If an error occurs during the chat.

"""
try:
    response, history = self.model.chat(
        self.tokenizer,
        query=f"<img>{img}</img>",
        history=None,
    )
    return response, history
except Exception as e:
    raise Exception(
        "An error occurred during the chat."
    ) from e
```