```
from io import BytesIO

import requests
import torch
from PIL import Image
from transformers import (
    AutoProcessor,
    VipLlavaForConditionalGeneration,
)

from swarm_models.base_multimodal_model import BaseMultiModalModel


class VipLlavaMultiModal(BaseMultiModalModel):
    """
    A multi-modal model for VIP-LLAVA.

    Args:
        model_name (str): The name or path of the pre-trained model.
        max_new_tokens (int): The maximum number of new tokens to generate.
        device_map (str): The device mapping for the model.
        torch_dtype: The torch data type for the model.
        *args: Additional positional arguments.
        **kwargs: Additional keyword arguments.
    """
```

```
def __init__(
    self,
    model_name: str = "llava-hf/vip-llava-7b-hf",
    max_new_tokens: int = 500,
    device_map: str = "auto",
    torch_dtype=torch.float16,
    *args,
    **kwargs,
):
    super().__init__(*args, **kwargs)
    self.model_name = model_name
    self.max_new_tokens = max_new_tokens
    self.device_map = device_map
    self.torch_dtype = torch_dtype

    self.model = VipLlavaForConditionalGeneration.from_pretrained(
        model_name,
        device_map=device_map,
        torch_dtype=torch_dtype,
        *args,
        **kwargs,
    )
    self.processor = AutoProcessor.from_pretrained(
        model_name, *args, **kwargs
    )
```

```
def run(self, text: str, img: str, *args, **kwargs):
    """
    Run the VIP-LLAVA model.

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

    Returns:
        str: The generated output text.
        tuple: A tuple containing None and the error message if an error occurs.
    """
    try:
        response = requests.get(img, stream=True)
        response.raise_for_status()
        image = Image.open(BytesIO(response.content))

        inputs = self.processor(
            text=text,
            images=image,
            return_tensors="pt",
            *args,
            **kwargs,
        ).to(0, self.torch_dtype)
```

```
    # Generate
    generate_ids = self.model.generate(
        **inputs, max_new_tokens=self.max_new_tokens, **kwargs
    )

    return self.processor.decode(
        generate_ids[0][len(inputs["input_ids"][0]) :],
        skip_special_tokens=True,
    )

except requests.RequestException as error:
    return None, f"Error fetching image: {error}"

except Exception as error:
    return None, f"Error during model inference: {error}"
```