```
from io import BytesIO
from typing import Tuple, Union

import requests
from PIL import Image
from transformers import AutoProcessor, LlavaForConditionalGeneration

from swarm_models.base_multimodal_model import BaseMultiModalModel


class LavaMultiModal(BaseMultiModalModel):
    """
    A class to handle multi-modal inputs (text and image) using the Llava model for conditional generation.

    Attributes:
        model_name (str): The name or path of the pre-trained model.
        max_length (int): The maximum length of the generated sequence.

    Args:
        model_name (str): The name of the pre-trained model.
        max_length (int): The maximum length of the generated sequence.
        *args: Additional positional arguments.
        **kwargs: Additional keyword arguments.

    Examples:
```

```
>>> model = LavaMultiModal()
>>> model.run("A cat", "https://example.com/cat.jpg")

"""

def __init__(
    self,
    model_name: str = "llava-hf/llava-1.5-7b-hf",
    max_length: int = 30,
    *args,
    **kwargs,
) -> None:
    super().__init__(*args, **kwargs)
    self.model_name = model_name
    self.max_length = max_length

    self.model = LlavaForConditionalGeneration.from_pretrained(
        model_name, *args, **kwargs
    )
    self.processor = AutoProcessor.from_pretrained(model_name)

def run(
    self, text: str, img: str, *args, **kwargs
) -> Union[str, Tuple[None, str]]:
    """
    Processes the input text and image, and generates a response.
```

```
    Args:
        text (str): The input text for the model.
        img (str): The URL of the image to process.
        max_length (int): The maximum length of the generated sequence.

    Returns:
        Union[str, Tuple[None, str]]: The generated response string or a tuple (None, error message)
in case of an error.
    """
    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"
        )

        # Generate
        generate_ids = self.model.generate(
            **inputs, max_length=self.max_length, **kwargs
        )
        return self.processor.batch_decode(
            generate_ids,
            skip_special_tokens=True,
```

```
            clean_up_tokenization_spaces=False,
            *args,
        )[0]

    except requests.RequestException as e:
        return None, f"Error fetching image: {str(e)}"
    except Exception as e:
        return None, f"Error during model processing: {str(e)}"
```