```
import requests
from PIL import Image
from transformers import AutoModelForVision2Seq, AutoProcessor

from swarm_models.base_multimodal_model import BaseMultiModalModel


# utils
def is_overlapping(rect1, rect2):
    x1, y1, x2, y2 = rect1
    x3, y3, x4, y4 = rect2
    return not (x2 < x3 or x1 > x4 or y2 < y3 or y1 > y4)


class Kosmos(BaseMultiModalModel):
    """A class representing the Kosmos model.

    This model is used for multi-modal tasks such as grounding, referring expression comprehension,
    referring expression generation, grounded VQA, grounded image captioning, and more.

    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.
        verbose (bool): Whether to print verbose output.
        *args: Variable length argument list.
        **kwargs: Arbitrary keyword arguments.
```

```
    Attributes:
        max_new_tokens (int): The maximum number of new tokens to generate.
        model (AutoModelForVision2Seq): The pre-trained model for vision-to-sequence tasks.
        processor (AutoProcessor): The pre-trained processor for vision-to-sequence tasks.
    """

    def __init__(
        self,
        model_name="ydshieh/kosmos-2-patch14-224",
        max_new_tokens: int = 64,
        verbose: bool = False,
        *args,
        **kwargs,
    ):
        super().__init__(*args, **kwargs)

        self.max_new_tokens = max_new_tokens

        self.model = AutoModelForVision2Seq.from_pretrained(
            model_name, trust_remote_code=True, *args, **kwargs
        )
        self.processor = AutoProcessor.from_pretrained(
            model_name, trust_remote_code=True, *args, **kwargs
        )
```

```
def get_image(self, url: str):
    """Get image from url

    Args:
        url (str): The URL of the image.

    Returns:
        PIL.Image: The image object.
    """
    return Image.open(requests.get(url, stream=True).raw)

def run(self, task: str, image: str, *args, **kwargs):
    """Run the model

    Args:
        task (str): The task to run.
        image (str): The URL of the image.
    """
    inputs = self.processor(
        text=task, images=image, return_tensors="pt"
    )
    generated_ids = self.model.generate(
        pixel_values=inputs["pixel_values"],
        input_ids=inputs["input_ids"][:, :-1],
        attention_mask=inputs["attention_mask"][:, :-1],
        image_embeds=None,
```

```
            img_attn_mask=inputs["img_attn_mask"][:, :-1],
            use_cache=True,
            max_new_tokens=self.max_new_tokens,
        )

        generated_texts = self.processor.batch_decode(
            generated_ids,
            skip_special_tokens=True,
        )[0]

        processed_text, entities = (
            self.processor.post_process_generation(generated_texts)
        )

        return processed_text, entities

    # tasks
    def multimodal_grounding(self, phrase, image_url):
        task = f"<grounding><phrase> {phrase} </phrase>"
        self.run(task, image_url)

    def referring_expression_comprehension(self, phrase, image_url):
        task = f"<grounding><phrase> {phrase} </phrase>"
        self.run(task, image_url)

    def referring_expression_generation(self, phrase, image_url):
```

```
    task = (
        "<grounding><phrase>"
        " It</phrase><object><patch_index_0044><patch_index_0863></object> is"
    )
    self.run(task, image_url)

def grounded_vqa(self, question, image_url):
    task = f"<grounding> Question: {question} Answer:"
    self.run(task, image_url)

def grounded_image_captioning(self, image_url):
    task = "<grounding> An image of"
    self.run(task, image_url)

def grounded_image_captioning_detailed(self, image_url):
    task = "<grounding> Describe this image in detail"
    self.run(task, image_url)

def generate_boxees(self, task, image_url):
    image = self.get_image(image_url)
    processed_text, entities = self.process_task(task, image)
    self.draw_entity_boxes_on_image(image, entities, show=True)
```