```
import json
import os
from typing import Dict, Optional
from loguru import logger
from pydantic import BaseModel
import importlib
import subprocess


# Define a Pydantic model to handle JSON output format
class AnnotationResult(BaseModel):
    image_name: str
    annotated_image_path: Optional[str] = None
    detections: list


class GroundedSAMTwo:
    def __init__(
        self,
        ontology_dict: Dict[str, str],
        model_name: str = None,
        grounding_dino_box_threshold: float = 0.25,
        extension: str = "jpg",
        output_dir: Optional[str] = None,
    ):
        """
```

```
        Initialize the GroundedSAMTwo class with a caption ontology and load the model.

        :param ontology_dict: Dictionary for mapping captions to classes, e.g., {"shipping container": "container"}.
        :param model_name: Name of the model to use (default: "Grounding DINO").
        :param grounding_dino_box_threshold: Threshold for the bounding box confidence (default: 0.25).
        """
        self.ontology_dict = ontology_dict
        self.model_name = model_name
        self.grounding_dino_box_threshold = (
            grounding_dino_box_threshold
        )
        self.extension = extension
        self.output_dir = output_dir
        self.base_model = None  # Model will be loaded lazily

        logger.info(
            "GroundedSAMTwo initialized with model: {}, box threshold: {}",
            model_name,
            grounding_dino_box_threshold,
        )

    def _install_and_import(self):
        """
        Install and import the necessary packages at runtime.
```

```
    """
    try:
        # Dynamically import required modules
        global cv2, sv, CaptionOntology, GroundedSAM2


        cv2 = importlib.import_module("cv2")
        sv = importlib.import_module("supervision")
        CaptionOntology = importlib.import_module(
            "autodistill.detection"
        ).CaptionOntology
        GroundedSAM2 = importlib.import_module(
            "autodistill_grounded_sam_2"
        ).GroundedSAM2
    except ImportError:
        logger.warning(
                "Some packages are missing. Installing required packages: supervision, autodistill,
autodistill-grounded-sam-2"
        )
        subprocess.check_call(
            [
                "python",
                "-m",
                "pip",
                "install",
                "-U",
                "supervision",
```

```
            "autodistill",
            "autodistill-grounded-sam-2",
        ]
    )
    # Retry imports after installation
    self._install_and_import()

def _load_model(self):
    """
    Lazily load the GroundedSAM2 model.
    """
    if self.base_model is None:
        self._install_and_import()  # Install and import required packages
        self.base_model = GroundedSAM2(
            ontology=CaptionOntology(self.ontology_dict),
            # grounding_dino_box_threshold=self.grounding_dino_box_threshold,
        )
        logger.info("GroundedSAM2 model loaded.")

def run(self, input_path: str) -> Optional[str]:
    """
    Annotate an image or label a dataset directory.

    :param input_path: Path to an image or directory of images.
    :return: JSON string output of the annotation results or None.
    """
```

```
    self._load_model()  # Load the model if not already loaded

    if os.path.isdir(input_path):
        logger.info("Processing directory: {}", input_path)
        return self._label_dataset(input_path, self.extension)
    else:
        logger.info("Processing single image: {}", input_path)
        return self._annotate_single_image(
            input_path, self.output_dir
        )

def _label_dataset(
    self, image_dir: str, extension: str = "jpg"
) -> str:
    """
    Label all images in the provided directory.

    :param image_dir: Directory containing images to label.
    :param extension: Image file extension (default: "jpg").
    :return: JSON string of the annotated results for the dataset.
    """
    self._load_model()
    logger.info(
        "Labeling dataset in directory: {} with extension: {}",
        image_dir,
        extension,
```

```
    )
    self.base_model.label(image_dir, extension=extension)

    # Output results (could be adjusted to store or handle annotations)
    result = {"directory": image_dir, "status": "Labeled"}
    return json.dumps(result, indent=4)

def _annotate_single_image(
    self, image_path: str, output_dir: Optional[str] = None
) -> str:
    """
    Annotate a single image and optionally save the annotated image.

    :param image_path: Path to the image.
    :param output_dir: Optional directory to save the annotated image.
    :return: JSON string of the annotation result.
    """
    self._load_model()
    logger.info("Annotating image: {}", image_path)

    try:
        # Make predictions and apply non-max suppression
        results = self.base_model.predict(image_path).with_nms()
        results = results[results.confidence > 0.3]

        # Read the image and annotate
```

```
image = cv2.imread(image_path)
mask_annotator = sv.BoxAnnotator()
annotated_image = mask_annotator.annotate(
    image.copy(), detections=results
)

# Display the annotated image
sv.plot_image(image=annotated_image, size=(8, 8))

# Save the annotated image if an output directory is provided
annotated_image_path = None
if output_dir:
    os.makedirs(output_dir, exist_ok=True)
    annotated_image_path = os.path.join(
        output_dir, os.path.basename(image_path)
    )
    cv2.imwrite(annotated_image_path, annotated_image)
    logger.info(
        "Annotated image saved to: {}",
        annotated_image_path,
    )

# Prepare the JSON result using Pydantic
annotation_result = AnnotationResult(
    image_name=os.path.basename(image_path),
    annotated_image_path=annotated_image_path,
```

```
                detections=results.to_dict(
                    orient="records"
                ),  # Assuming the results object supports conversion to dict
            )

            # Return the result as a JSON string
            return annotation_result.json(indent=4)
        except Exception as e:
            logger.error("Error during image annotation: {}", e)
            return json.dumps({"error": str(e)})


# Example usage:
# ontology = {"shipping container": "container"}
# runner = GroundedSAMTwo(ontology)
#
# # Run on a single image
# image_path = "path/to/your/image.jpg"
# json_output = runner.run(image_path, output_dir="annotated_images")
# logger.info("Annotation result: \n{}", json_output)
#
# # Run on a dataset (directory)
# image_dir = "path/to/your/dataset"
# json_output = runner.run(image_dir)
# logger.info("Dataset labeling result: \n{}", json_output)
```