Configuration Manual - Street Navigation for Visual Impairment using CNN and Transformer Models

MSc Research Project
Masters of Science in Artificial Intelligence

Hasan Ali
Student ID: 22142291

School of Computing
National College of Ireland

Supervisor: Faithful Onwuegbuche

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Hasan Ali
Student ID:	22142291
Programme:	Masters of Science in Artificial Intelligence
Year:	2024
Module:	MSc Research Project
Supervisor:	Faithful Onwuegbuche
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual - Street Navigation for Visual Impairment using CNN and Transformer Models
Word Count:	XXX
Page Count:	16

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Hasan Ali
Date:	16th September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual - Street Navigation for Visual Impairment using CNN and Transformer Models

Hasan Ali
22142291

1 Introduction

This manual provides a detailed step-by-step guide to reproduce the experiments and results described in the object detection thesis using the DETR and YOLO models. It focuses on the specific configurations, tools, and procedures required to replicate the study, from environment setup to running the models.

1.1 Datasets:

Please find the datasets in those location:

Google Drive:

1. Link

RoboFlow:

1. Phase 1 Dataset
2. Phase 2 Dataset

1.2 Detailed Output Runs

Please find here a link to the original model runs outputs, including model predictions, trained model weights, visualizations of image predictions for all runs in this experiment:

1. Link

Please note this was not included in Moodle due to large size.

2 Prerequisites

Before proceeding with the steps outlined in this manual, ensure you have the following prerequisites in place:

1. Google Account: Required for using Google Colab.
2. Local Machine: A computer with the minimum recommended specifications (detailed below) for running local tasks.

3 Environment Setup

3.1 Version Control System

We use GitHub for VCS but in this Config manual we refer to your usage of the ZIP file attahced in Moodle. Firstly, download the ZIP file containing the code artifacts, and secondly, unzip it in a place suitable for you.

3.2 Google Colab Setup

Google Colab is used for training and running predictions for object detection models, as it provides access to powerful GPU instances.

1. **Google Colab Configuration**:
   - **Login to Google Colab**: Access Google Colab and log in with your Google account.
   - **Set Hardware Accelerator**:
     (a) Navigate to `Runtime > Change runtime type`.
     (b) Select `GPU` from the `Hardware accelerator` dropdown menu.
2. **Colab Resources**:
   - **Hardware**:
     - Use an Nvidia A100 GPU for optimal performance.
     - Opt for the High RAM option for better handling of large datasets.
   - **Colab Pro+ Subscription**:
     - Ensure you have a Colab Pro+ subscription, which provides background task capabilities and longer runtime durations.
   - **Google Drive Integration**:
     - Mount Google Drive in Colab to store large datasets and model checkpoints.
     - Ensure you have Google Drive Premium for adequate storage.
   - **Action**: Use the following code snippet to mount Google Drive in Colab: from google.colab import drive drive.mount('/content/drive')

3.3 Local Resources

Local resources are used for non-intensive tasks such as data preprocessing.

1. **System Requirements**:
   - **Operating System**: Linux Fedora 39 GNOME (64-bit).
   - **Processor**: Intel i5 8th Generation or higher.
   - **RAM**: Minimum 8GB.
   - **Storage**: Minimum 256GB SSD with 50GB free space.

2. **Software Setup**:
   - **Python Libraries**: Ensure the necessary libraries are installed. Use the `requirements.txt` file provided in the Zip repository: pip install -r requirements.txt
   - **CUDA Toolkit**: Install appropriate GPU drivers and CUDA toolkit for GPU acceleration.
3. **Integrated Development Environment (IDE)**:
   - **Install VSCodium**:
     - VSCodium is a free and open-source code editor, which you will use for editing and managing your scripts.
     - **Install and Run VSCodium** and open your project folder for editing and development.

4 Labeling Tools Setup

4.1 LabelImg

1. **Installation**:
   - Download and install LabelImg from its official repository.
   - Follow the installation instructions for your operating system.
2. **Configuration**:
   - Use default settings for labeling images.
   - Save the labeled images in YOLO format.

4.2 Label Studio

1. **Installation**: pip install label-studio
2. **Configuration**:
   - Start the Label Studio server: label-studio start
   - Configure your project to label images, and export labels after auditing.

4.3 Roboflow

1. **Account Setup**:
   - Sign up for a Roboflow account at Roboflow.
2. **Dataset Management**:
   - Upload your dataset to Roboflow.
   - Perform any necessary labeling or format conversion.

- Use the Roboflow API to manage your dataset within your codebase.
- **Example API Usage**: from roboflow import Roboflow rf = Roboflow(api_key="YOUR_AP
  project = rf.workspace().project("PROJECT_NAME") dataset = project.version("VERSION

5 Dataset Management

5.1 Local Storage

1. **Dataset Organization**:
   - Organize your datasets into directories (e.g., `train`, `test`, `validation`).
   - Ensure data is properly formatted for training (e.g., YOLO format).
2. **Accessing the Dataset**:
   - Download the dataset from the provided link and extract it to the appropriate directory.

5.2 Roboflow Dataset Management

1. **Upload and Preprocessing**:
   - Upload the dataset to Roboflow and perform preprocessing steps such as resizing, augmentation, or format conversion.
2. **Downloading the Dataset**:
   - Use the Roboflow API to download datasets directly to your local environment or Google Colab.
   - **Example**: dataset = project.version("VERSION_NUMBER").download("coco")

5.3 Datasets Details

The dataset will look like below:

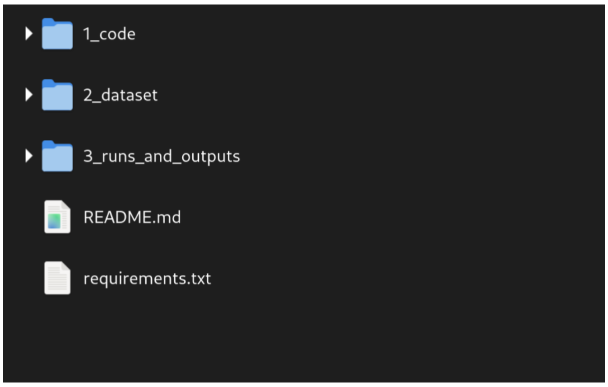


Figure 1: Dataset overview

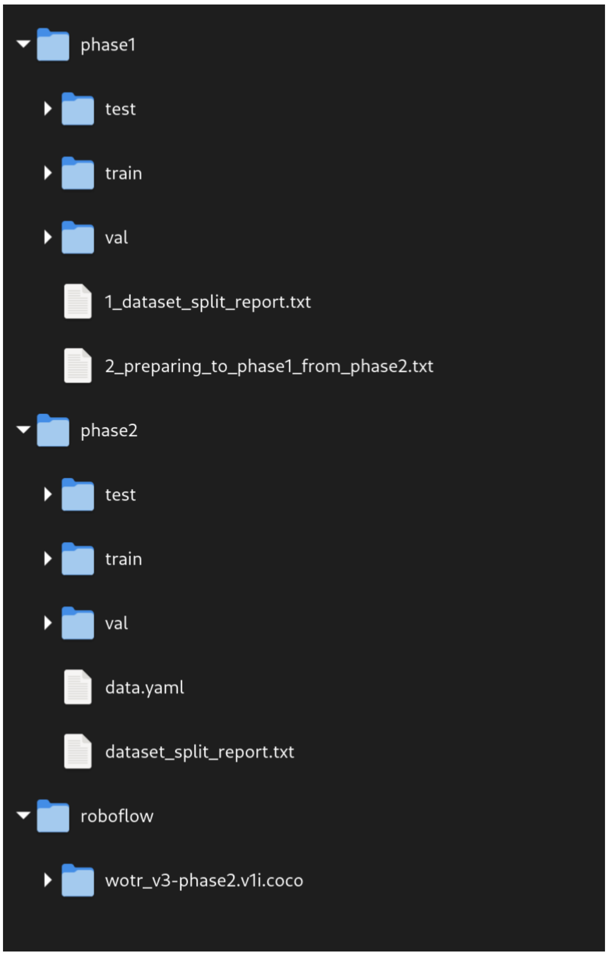
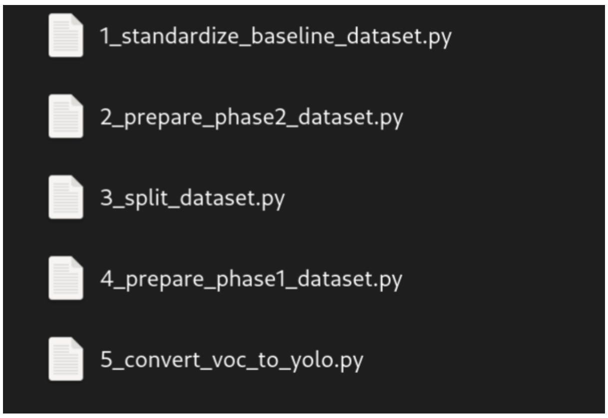


Figure 2: Dataset details

6 Running Experiments and Evaluations



6.0.1 Data Preparation

Below are the steps we took to prepare our dataset:

1. **Conduct an audit on the dataset:**
   (a) We manually assessed the accuracy of the ground truth labels to ensure that the dataset is of high quality.

2. **Standardize the baseline dataset and improve it:**

```python
import os
import xml.etree.ElementTree as ET

def process_voc_labels(labels_folder, images_folder):
    # Define class renaming dictionary
    class_rename_dict = {
        'fire_hydrant': 'fire hydrant',
        'reflective_cone': 'reflective cone',
        'warning_column': 'warning column',
        'blind_road': 'tactile pavement',
        'ashcan': 'litter bin',
        'tricycle': 'bicycle',
        'red_light': 'traffic light',
        'green_light': 'traffic light'
    }

    # Initialize counters
    removed_sign_count = 0
    renamed_classes_count = {old: 0 for old in class_rename_dict}
    deleted_files_count = 0

    for label_file in os.listdir(labels_folder):
        if label_file.endswith('.xml'):
            label_path = os.path.join(labels_folder, label_file)
            tree = ET.parse(label_path)
            root = tree.getroot()

            # Find all object elements
            objects = root.findall('object')
            to_remove = []
            renamed = False

            for obj in objects:
                name = obj.find('name').text

                # Remove objects of class "sign"
                if name == 'sign':
                    to_remove.append(obj)
                    removed_sign_count += 1
                # Rename classes
                elif name in class_rename_dict:
                    obj.find('name').text = class_rename_dict[name]
                    renamed_classes_count[name] += 1
                    renamed = True

            # Remove marked objects
            for obj in to_remove:
                root.remove(obj)

            # Check if there are any objects left
            objects_left = root.findall('object')

            if not objects_left:
                # No objects left, delete the XML file and corresponding image
                os.remove(label_path)
                deleted_files_count += 1
                image_file = label_file.replace('.xml', '.jpg')  # Assuming images are in .jpg format
                image_path = os.path.join(images_folder, image_file)
                if os.path.exists(image_path):
                    os.remove(image_path)
            else:
                # Save the modified XML file
                tree.write(label_path)

    # Print the report
    print("Processing Report:")
    print(f"Total 'sign' classes removed: {removed_sign_count}")
    for old_name, count in renamed_classes_count.items():
        new_name = class_rename_dict[old_name]
        print(f"Total '{old_name}' classes renamed to '{new_name}': {count}")
    print(f"Total files deleted: {deleted_files_count}")

# Define your folder paths
labels_folder = '/home/hasan/projects/1_obj_det_proj/01_dataset_groundtruth/new/WOTR/labels_voc'
images_folder = '/home/hasan/projects/1_obj_det_proj/01_dataset_groundtruth/new/WOTR/images'

# Process the labels
process_voc_labels(labels_folder, images_folder)
```

(a) Remove sign class instances from our labels:

i. We conducted a sense check on the sign class labels and found that it is not a reliable class as it mixes between stop signs and other directional signs. Moreover, we asserted that the sign class is not relevant for our use-case as VIP will not benefit a lot from it.

ii. To be more specific, the "sign" labels had two issues:

A. They included all signs, meaning they included directional and stop signs, which is problematic in the evaluation as the COCO dataset only contains "stop signs".

B. They are not useful for our purpose: There is no major use for VIP in regards to signs (be it stop signs or directional car signs) - in other words, this class is not a Class of Concern for the VIP when they engage in street navigation.

iii. In this step, we scan our PASCAL VOC labels for the sign class and remove them.

iv. In this step, we delete empty labels (along with their image pair): We also delete the images which contain only sign classes because if we keep them, they might increase the noise in our dataset.

(b) Consolidate classes: There are classes in the WOTR dataset which were extended from the COCO dataset, such as red/green light instead of the original COCO class traffic light.

i. After checking the classes in the WOTR dataset, we asserted that some classes would be better combined and consolidated to ensure better detection and more meaningful prediction and training data.

ii. We consolidate "tricycle" into "bicycle", "red light" into "traffic light", and "green light" into "traffic light".

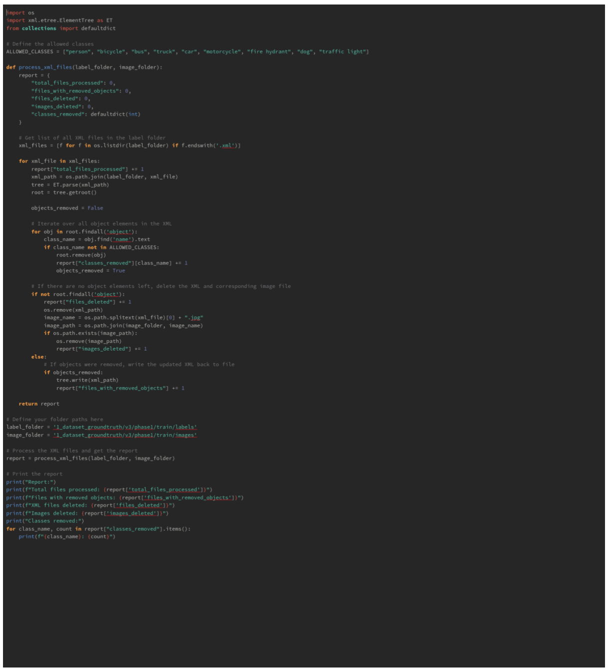
(c) Rename classes: The names in the dataset were slightly different from the COCO dataset, and therefore we renamed them to match the COCO dataset. For example, "fire_hydrant" to "fire hydrant".

i. We will rename the class names to make them more meaningful, as well as matching the case and style with the COCO dataset.

ii. We rename 1) "fire_hydrant" to "fire hydrant", 2) "reflective_cone" to "reflective cone", 3) "warning_column" to "warning column", 4) "blind_road" to "tactile pavement", and 5) "ashcan" to "litter bin".

3. **Prepare Phase-based datasets:**

```python
import os
import xml.etree.ElementTree as ET
from collections import defaultdict

# Define the allowed classes
ALLOWED_CLASSES = ["person", "bicycle", "bus", "truck", "car", "motorcycle", "fire hydrant", "dog", "traffic light", "tree", "reflective cone", "crosswalk",
"tactile pavement", "pole", "warning column", "roadblock", "litter bin"]

def process_xml_files(label_folder, image_folder):
    report = {
        "total_files_processed": 0,
        "files_with_removed_objects": 0,
        "files_deleted": 0,
        "images_deleted": 0,
        "classes_removed": defaultdict(int)
    }

    # Get list of all XML files in the label folder
    xml_files = [f for f in os.listdir(label_folder) if f.endswith('.xml')]

    for xml_file in xml_files:
        report["total_files_processed"] += 1
        xml_path = os.path.join(label_folder, xml_file)
        tree = ET.parse(xml_path)
        root = tree.getroot()

        objects_removed = False

        # Iterate over all object elements in the XML
        for obj in root.findall('object'):
            class_name = obj.find('name').text
            if class_name not in ALLOWED_CLASSES:
                root.remove(obj)
                report["classes_removed"][class_name] += 1
                objects_removed = True

        # If there are no object elements left, delete the XML and corresponding image file
        if not root.findall('object'):
            report["files_deleted"] += 1
            os.remove(xml_path)
            image_name = os.path.splitext(xml_file)[0] + ".jpg"
            image_path = os.path.join(image_folder, image_name)
            if os.path.exists(image_path):
                os.remove(image_path)
                report["images_deleted"] += 1
        else:
            # If objects were removed, write the updated XML back to file
            if objects_removed:
                tree.write(xml_path)
                report["files_with_removed_objects"] += 1

    return report

# Define your folder paths here
label_folder = '/home/hasan/projects/1_obj_det_proj/01_dataset_groundtruth/new/WOTR/phase2/labels_voc'
image_folder = '/home/hasan/projects/1_obj_det_proj/01_dataset_groundtruth/new/WOTR/phase2/images'

# Process the XML files and get the report
report = process_xml_files(label_folder, image_folder)

# Print the report
print("Report:")
print(f"Total files processed: {report['total_files_processed']}")
print(f"Files with removed objects: {report['files_with_removed_objects']}")
print(f"XML files deleted: {report['files_deleted']}")
print(f"Images deleted: {report['images_deleted']}")
print("Classes removed:")
for class_name, count in report["classes_removed"].items():
    print(f"{class_name}: {count}")
```



```python
import os
import xml.etree.ElementTree as ET
from collections import defaultdict

# Define the allowed classes
ALLOWED_CLASSES = ["person", "bicycle", "bus", "truck", "car", "motorcycle", "fire hydrant", "dog", "traffic light"]

def process_xml_files(label_folder, image_folder):
    report = {
        "total_files_processed": 0,
        "files_with_removed_objects": 0,
        "files_deleted": 0,
        "images_deleted": 0,
        "classes_removed": defaultdict(int)
    }

    # Get list of all XML files in the label folder
    xml_files = [f for f in os.listdir(label_folder) if f.endswith('.xml')]

    for xml_file in xml_files:
        report["total_files_processed"] += 1
        xml_path = os.path.join(label_folder, xml_file)
        tree = ET.parse(xml_path)
        root = tree.getroot()

        objects_removed = False

        # Iterate over all object elements in the XML
        for obj in root.findall('object'):
            class_name = obj.find('name').text
            if class_name not in ALLOWED_CLASSES:
                root.remove(obj)
                report["classes_removed"][class_name] += 1
                objects_removed = True

        # If there are no object elements left, delete the XML and corresponding image file
        if not root.findall('object'):
            report["files_deleted"] += 1
            os.remove(xml_path)
            image_name = os.path.splitext(xml_file)[0] + ".jpg"
            image_path = os.path.join(image_folder, image_name)
            if os.path.exists(image_path):
                os.remove(image_path)
                report["images_deleted"] += 1
        else:
            # If objects were removed, write the updated XML back to file
            if objects_removed:
                tree.write(xml_path)
                report["files_with_removed_objects"] += 1

    return report

# Define your folder paths here
label_folder = '1_dataset_groundtruth/v3/phase1/train/labels'
image_folder = '1_dataset_groundtruth/v3/phase1/train/images'

# Process the XML files and get the report
report = process_xml_files(label_folder, image_folder)

# Print the report
print("Report:")
print(f"Total files processed: {report['total_files_processed']}")
print(f"Files with removed objects: {report['files_with_removed_objects']}")
print(f"XML files deleted: {report['files_deleted']}")
print(f"Images deleted: {report['images_deleted']}")
print("Classes removed:")
for class_name, count in report["classes_removed"].items():
    print(f"{class_name}: {count}")
```

Figure 3: Enter Caption

(a) In this step, we prepare the dataset for each phase. We aim to have the phase images in both phases and apply respective differences when needed, such as keeping only classes in scope for each phase.

(b) Because Phase 2 is a bigger set, we start by preparing the Phase 2 dataset, and then prepare the Phase 1 dataset out of it, by removing classes out of scope in Phase 1.

(c) Steps are:

   i. Prepare the Phase 2 dataset:

      A. Identify classes in scope and remove classes that are not in scope.

      B. Classes in scope are: person, bicycle, bus, truck, car, motorcycle, fire hydrant, dog, traffic light, tree, reflective cone, crosswalk, tactile pavement, pole, warning column, roadblock, litter bin.

      C. Delete empty labels (along with their image pair).

   ii. Prepare the Phase 1 dataset:

      A. Identify classes and remove classes that are not in scope.

      B. Classes are: person, bicycle, bus, truck, car, motorcycle, fire hydrant, dog, traffic light.

      C. Delete empty labels (along with their image pair).

4. **Split each phase-based dataset into 3 main collections**: Training, Validation, and Testing datasets. The split we chose is 80/10/10. We used SKLearn to ensure randomness and avoidance of bias in the image splitting process.

```python
import os
import random
import shutil
from sklearn.model_selection import train_test_split

def create_folders(root, folders):
    for folder in folders:
        os.makedirs(os.path.join(root, folder, 'images'), exist_ok=True)
        os.makedirs(os.path.join(root, folder, 'labels'), exist_ok=True)

def copy_files(data_pairs, src_images_path, src_labels_path, dest_folder):
    for img, lbl in data_pairs:
        shutil.copy2(os.path.join(src_images_path, img), os.path.join(dest_folder, 'images', img))
        shutil.copy2(os.path.join(src_labels_path, lbl), os.path.join(dest_folder, 'labels', lbl))

def generate_report(train_data, val_data, test_data, report_path):
    report = f"""
    Dataset Split Report
    ====================
    Total number of samples: {len(train_data) + len(val_data) + len(test_data)}

    Training set:
        Number of samples: {len(train_data)}
        Percentage: {len(train_data) / (len(train_data) + len(val_data) + len(test_data)) * 100:.2f}%
        Filenames: {[img for img, _ in train_data]}

    Validation set:
        Number of samples: {len(val_data)}
        Percentage: {len(val_data) / (len(train_data) + len(val_data) + len(test_data)) * 100:.2f}%
        Filenames: {[img for img, _ in val_data]}

    Test set:
        Number of samples: {len(test_data)}
        Percentage: {len(test_data) / (len(train_data) + len(val_data) + len(test_data)) * 100:.2f}%
        Filenames: {[img for img, _ in test_data]}
    """

    with open(report_path, 'w') as report_file:
        report_file.write(report)

if __name__ == '__main__':
    images_path = '/home/hasan/projects/1_obj_det_proj/01_dataset_groundtruth/new/WOTR/phase2/images'
    labels_path = '/home/hasan/projects/1_obj_det_proj/01_dataset_groundtruth/new/WOTR/phase2/labels_voc'

    base_path = os.path.dirname(images_path)

    train_folder = os.path.join(base_path, 'train')
    val_folder = os.path.join(base_path, 'val')
    test_folder = os.path.join(base_path, 'test')

    # Create folders
    create_folders(base_path, [train_folder, val_folder, test_folder])

    images = sorted(os.listdir(images_path))
    labels = sorted(os.listdir(labels_path))

    # Ensure the lists are paired and in sync
    assert len(images) == len(labels), "Number of images and labels must be equal"
    assert all(img.split('.')[0] == lbl.split('.')[0] for img, lbl in zip(images, labels)), "Images and labels filenames must match"

    data_pairs = list(zip(images, labels))

    # Shuffle the data pairs to ensure randomness
    random.seed(42)
    random.shuffle(data_pairs)

    # Split dataset
    train_val_data, test_data = train_test_split(data_pairs, test_size=0.1, random_state=42)
    train_data, val_data = train_test_split(train_val_data, test_size=0.1111, random_state=42)  # 0.1111 of 0.9 is 0.1

    # Copy files to respective folders
    copy_files(train_data, images_path, labels_path, train_folder)
    copy_files(val_data, images_path, labels_path, val_folder)
    copy_files(test_data, images_path, labels_path, test_folder)

    # Generate and save report
    report_path = os.path.join(base_path, 'dataset_split_report.txt')
    generate_report(train_data, val_data, test_data, report_path)

    print(f"Dataset split completed successfully. Report saved to {report_path}.")
```

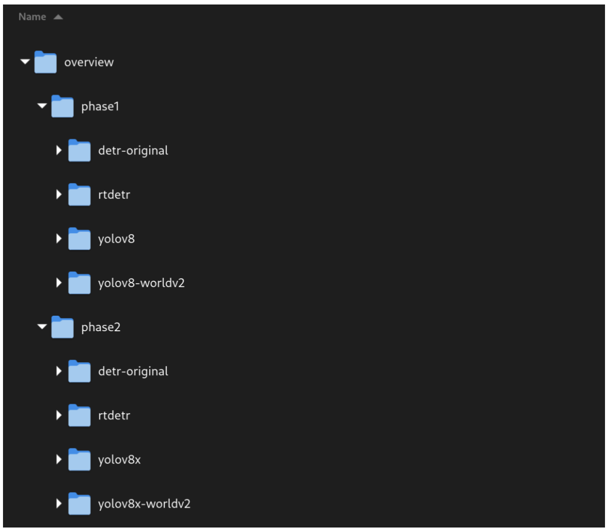
(a) Dataset was split into:
   i. 80% training set
   ii. 10% validation set
   iii. 10% test set

5. **Convert the labels from PASCAL-VOC to YOLO format:** The reason we chose the YOLO format is that it is native to the YOLO family of models and can be easily converted at a later stage to other formats.

```python
import os
import xml.etree.ElementTree as ET

# Define the path to your VOC dataset and the target folder for YOLO labels
voc_folder = '1_dataset_groundtruth/v3/phase1/train/labels'  # Update this to your VOC folder path
yolo_labels_folder = os.path.join(os.path.dirname(voc_folder), 'labels_yolo')

# Create the target directory if it doesn't exist
os.makedirs(yolo_labels_folder, exist_ok=True)

# Define the mapping from Pascal VOC classes to YOLO class IDs
class_mapping = {
    'person': 0,
    'bicycle': 1,
    'car': 2,
    'motorcycle': 3,
    'bus': 5,
    'truck': 7,
    'traffic light': 9,
    'fire hydrant': 10,
    'dog': 16,
    'tree': 80,
    'reflective cone': 81,
    'crosswalk': 82,
    'tactile pavement': 83,
    'pole': 84,
    'warning column': 85,
    'roadblock': 86,
    'litter bin': 87
}

def voc_to_yolo(xml_file):
    tree = ET.parse(xml_file)
    root = tree.getroot()

    # Extract image dimensions from XML
    size = root.find('size')
    image_width = int(size.find('width').text)
    image_height = int(size.find('height').text)

    yolo_labels = []

    for obj in root.findall('object'):
        class_name = obj.find('name').text
        class_id = class_mapping.get(class_name, None)
        if class_id is None:
            print(f"Class '{class_name}' not found in class mapping.")
            continue

        bndbox = obj.find('bndbox')
        xmin = int(bndbox.find('xmin').text)
        ymin = int(bndbox.find('ymin').text)
        xmax = int(bndbox.find('xmax').text)
        ymax = int(bndbox.find('ymax').text)

        # Convert to YOLO format
        x_center = (xmin + xmax) / 2.0 / image_width
        y_center = (ymin + ymax) / 2.0 / image_height
        width = (xmax - xmin) / image_width
        height = (ymax - ymin) / image_height

        yolo_labels.append(f"{class_id} {x_center} {y_center} {width} {height}")

    return yolo_labels

def main():
    for root, dirs, files in os.walk(voc_folder):
        for file in files:
            if file.endswith('.xml'):
                xml_path = os.path.join(root, file)

                yolo_labels = voc_to_yolo(xml_path)

                # Save YOLO labels to a text file
                yolo_file = os.path.join(yolo_labels_folder, file.replace('.xml', '.txt'))
                with open(yolo_file, 'w') as f:
                    f.write("\n".join(yolo_labels))

                print(f"Converted {xml_path} to {yolo_file}")

if __name__ == "__main__":
    main()
```

After performing the steps above, we are ready to utilize the datasets for our predictions and fine-tuning, and evaluating model performance.

6.1 Notebooks



1. **Structure**:
   - All experiments are organized in Jupyter notebooks stored in the repository.
   - The notebooks are pre-run with results already available in the cell outputs.
   - Experiment results and outputs are stored in the "**3_runs_and_outputs**" folders.
   - The experiments are split into two phases: **Phase 1** and **Phase 2**.
   - Each phase is further divided by the model type (e.g., YOLOv8, DETR, RT-DETR, etc.).
   - Each notebook includes the necessary library requirements within the first few cells.

   Phase 1 Code snippets and outputs (please note entire code is in the code repository as it is too big):

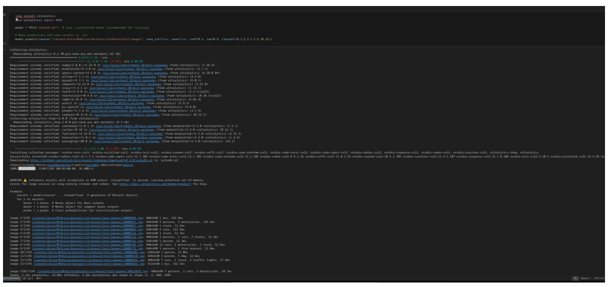


Figure 4: YOLO Code Snippet and Output

Figure 5: DETR Code Snippet

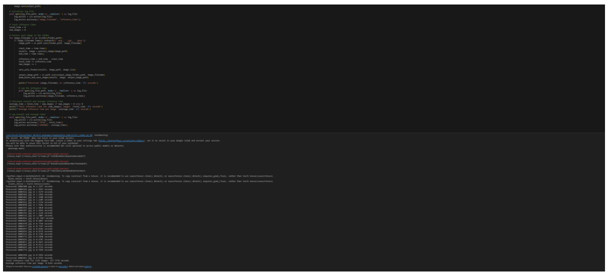


Figure 6: DETR Code Snippet and Output

Phase 2 Code snippets and outputs ((please note entire code is in the code repository as it is too big):

Figure 7: YOLOv8 Training: Code Snippet and Output 1/2

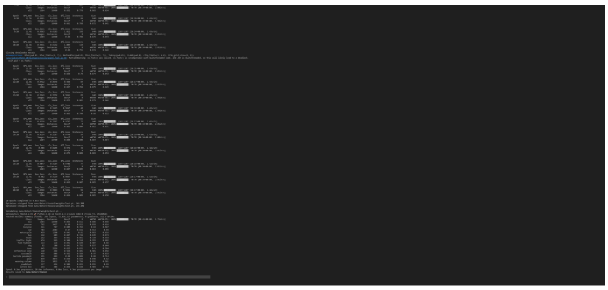


Figure 8: YOLOv8 Training: Code Snippet and Output 2/2

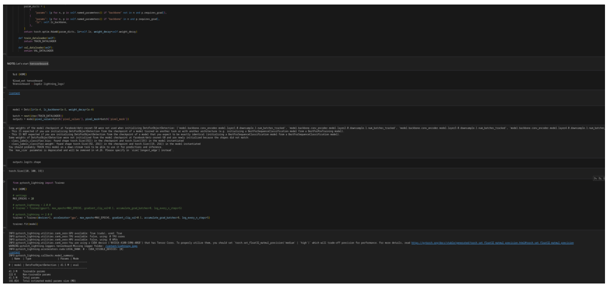


Figure 9: DETR Training: Code Snippet and Output

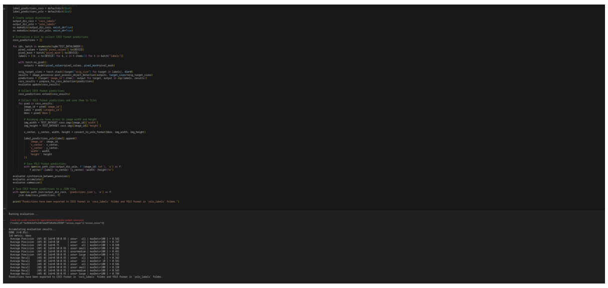


Figure 10: RT-DETR Training: Code Snippet and Output

2. **Using Datasets**:
   - **YOLO Models**: Use the dataset stored on Google Drive directly for both phases.
   - **DETR Implementation**:
     - **Phase 1**: Use the Google Drive dataset directly.
     - **Phase 2**: Use the dataset from Roboflow in the COCO format. The code is already in the notebook, just run it.

6.2 Running the Experiments Again

1. **Steps to Re-run Experiments**:
   - **Open the Experiment Notebook**:
     - Navigate to the specific experiment folder in the repository.
     - Open the notebook in VSCodium or your preferred IDE (e.g., Jupyter Notebook).
   - **Run the Required Cells**:
     - Execute the cells sequentially as needed to reproduce the results.
     - Note that some notebooks may include a final cell that exports or zips the output folder. Running this cell is optional and is used to compile prediction images, labels, and other outputs into a zipped file for easier downloading.
2. **Ensure Proper Configuration**:

- Use the Nvidia A100 GPU in Google Colab to ensure that the experiments run efficiently.
- Ensure Google Drive is mounted in Colab to store and access large datasets during the experiments.

7 Evaluating Model Performance

7.1 Output and Evaluation

1. **Prediction Labels**:
   - After running the experiments, you will find the prediction labels for each image in the `output` folder.
   - These prediction labels are saved in `.txt` format, corresponding to the images on which the experiment was run (typically the test set).
2. **Post-Processing for DETR/RT-DETR Models**:
   - **Phase 1**:
     - After obtaining the prediction labels, run the `postprocess_detr_step1.py` script.
     - **Input Required**:
       * The original folder containing the prediction labels.
       * A new empty folder where the post-processed labels will be stored.
     - The output folder from this step will contain labels ready for metrics evaluation using `detr_phase1_postprocess_step1.py`.
   - **Phase 2**:
     - After obtaining the prediction labels, run both `detr_phase2_postprocess_step1.py` and `detr_phase2_postprocess_step2.py`.
     - **Step 1**:
       * Input the original folder containing the prediction labels.
       * Specify a new empty folder where the post-processed labels from step 1 will be stored.
     - **Step 2**:
       * Input the folder containing the post-processed labels from step 1.
       * Specify another new empty folder where the final post-processed labels will be stored.
     - The output folder from step 2 will contain labels ready for metrics evaluation using `calculate_metrics.py`.
3. **Running Evaluation Metrics**:
   - To calculate the evaluation metrics, run the `calculate_metrics.py` script.
   - **Input Required**:
     - **Ground Truth Labels**: The actual labels for the test set.

– **Prediction Labels**: The labels generated by the model, located in the `output` folder or the post-processed folder.

- **R**un the script
- The script will output the evaluation results, which include metrics such as mAP, Precision, and Recall.

8 Results Reproduction

1. **Evaluation Metrics**:

    - Ensure that you evaluate the models using the same metrics used in the thesis (e.g., mAP, Precision, Recall).
    - Compare the results with those reported in the thesis.

2. **Reporting Results**:

    - Document any deviations from the original results and analyze potential causes.

- **Common Issues**:

    – GPU Memory Errors: Consider reducing batch size or using a smaller model variant.

    – Dataset Format Errors: Ensure the dataset is in the correct format as required by the model.

9 Appendix

I this section, you can find the full results of the experiment. The original file can be found in the Google Drive "Detailed Output Runs" folder. Link.

9.1 Phase 1 and 2 Results:

Phase	Model_Archit	Model_Size	Params (M	Class	mAP	Precision	Recall	F1	FNR	TP	FP	FN	Average Inference Time (m
Phase 2	YOLO	yolov8x	68.2	0	0.3405	0.6871	0.5696	0.6228	0.4304	1952	889	1475	
Phase 2	YOLO	yolov8x	68.2	1	0.5396	0.6547	0.554	0.6001	0.446	436	230	351	
Phase 2	YOLO	yolov8x	68.2	2	0.3618	0.7522	0.6609	0.7036	0.3391	1709	563	877	
Phase 2	YOLO	yolov8x	68.2	3	0.427	0.6778	0.5852	0.6281	0.4148	749	356	531	
Phase 2	YOLO	yolov8x	68.2	9	0.4985	0.73	0.7202	0.7251	0.2798	695	257	270	
Phase 2	YOLO	yolov8x	68.2	80	0.3111	0.6142	0.5449	0.5775	0.4551	1213	762	1013	
Phase 2	YOLO	yolov8x	68.2	82	0.6321	0.8281	0.7871	0.8071	0.2129	636	132	172	
Phase 2	YOLO	yolov8x	68.2	84	0.2983	0.6747	0.6008	0.6357	0.3992	1838	886	1221	
Phase 2	YOLO	yolov8x	68.2	85	0.3722	0.7034	0.5376	0.6094	0.4624	543	229	467	
Phase 2	YOLO	yolov8x	68.2	16	0.6085	0.7356	0.6598	0.6957	0.3402	64	23	33	
Phase 2	YOLO	yolov8x	68.2	81	0.3891	0.7531	0.6389	0.6913	0.3611	299	98	169	
Phase 2	YOLO	yolov8x	68.2	86	0.4046	0.814	0.7123	0.7597	0.2877	302	69	122	
Phase 2	YOLO	yolov8x	68.2	87	0.6205	0.839	0.7273	0.7791	0.2727	224	43	84	
Phase 2	YOLO	yolov8x	68.2	7	0.426	0.7066	0.4858	0.5758	0.5142	171	71	181	
Phase 2	YOLO	yolov8x	68.2	83	0.5856	0.751	0.6608	0.703	0.3392	187	62	96	
Phase 2	YOLO	yolov8x	68.2	5	0.5262	0.8095	0.5749	0.6723	0.4251	119	28	88	
Phase 2	YOLO	yolov8x	68.2	10	0.6654	0.7757	0.7034	0.7378	0.2966	83	24	35	
Phase 2	YOLO	yolov8x	68.2	Overall	0.471								
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	0	0.3468	0.6965	0.5686	0.6261	0.4314	1948	849	1478	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	1	0.5177	0.6471	0.5032	0.5661	0.4968	396	216	391	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	2	0.3662	0.7581	0.652	0.701	0.348	1686	538	900	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	3	0.4207	0.697	0.5805	0.6334	0.4195	743	323	537	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	9	0.5009	0.743	0.7131	0.7278	0.2869	691	239	278	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	80	0.3126	0.6476	0.5384	0.588	0.4616	1198	652	1027	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	82	0.6405	0.8395	0.7834	0.8105	0.2166	633	121	175	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	84	0.302	0.6957	0.5887	0.6377	0.4113	1808	791	1263	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	85	0.3538	0.7035	0.5227	0.5998	0.4773	529	223	483	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	16	0.5904	0.7011	0.6224	0.6595	0.3776	61	26	37	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	81	0.3605	0.8066	0.6239	0.7036	0.3761	292	70	176	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	86	0.4031	0.8361	0.7099	0.7679	0.2901	301	59	123	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	87	0.613	0.8427	0.7377	0.7867	0.2623	225	42	80	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	7	0.4448	0.7608	0.5465	0.6361	0.4535	194	61	161	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	83	0.5677	0.7948	0.6431	0.7109	0.3569	182	47	101	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	5	0.5571	0.8497	0.6404	0.7303	0.3596	130	23	73	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	10	0.6606	0.7353	0.6356	0.6818	0.3644	75	27	43	
Phase 2	YOLO-WORLD	yolov8x-worldv2	68.2	Overall	0.4681								

Phase	Model_Archi	Model_Size	Params (M	Class	mAP	Precision	Recall	F1	FNR	TP	FP	FN	Average Inference Time (m
Phase 2	DETR	DETR-Original	41.6	0	0.2334	0.1844	0.3195	0.2338	0.6805	1095	4843	2332	
Phase 2	DETR	DETR-Original	41.6	1	0.3559	0.2066	0.3037	0.2459	0.6963	239	918	548	
Phase 2	DETR	DETR-Original	41.6	2	0.2793	0.2635	0.4157	0.3226	0.5843	1082	3024	1521	
Phase 2	DETR	DETR-Original	41.6	3	0.2711	0.2	0.3	0.24	0.7	384	1536	896	
Phase 2	DETR	DETR-Original	41.6	9	0.2771	0.1672	0.2374	0.1962	0.7626	230	1146	739	
Phase 2	DETR	DETR-Original	41.6	80	0.1841	0.0926	0.1878	0.124	0.8122	418	4098	1808	
Phase 2	DETR	DETR-Original	41.6	82	0.4849	0.2597	0.5408	0.3509	0.4592	437	1246	371	
Phase 2	DETR	DETR-Original	41.6	84	0.204	0.124	0.225	0.1599	0.775	692	4887	2383	
Phase 2	DETR	DETR-Original	41.6	85	0.1951	0.128	0.166	0.1446	0.834	168	1144	844	
Phase 2	DETR	DETR-Original	41.6	16	0.6299	0.6509	0.697	0.6732	0.303	69	37	30	
Phase 2	DETR	DETR-Original	41.6	81	0.1965	0.2855	0.3825	0.3269	0.6175	179	448	289	
Phase 2	DETR	DETR-Original	41.6	5	0.3575	0.3917	0.4498	0.4187	0.5502	94	146	115	
Phase 2	DETR	DETR-Original	41.6	7	0.264	0.2936	0.3664	0.326	0.6336	133	320	230	
Phase 2	DETR	DETR-Original	41.6	86	0.2685	0.3321	0.4292	0.3745	0.5708	182	366	242	
Phase 2	DETR	DETR-Original	41.6	87	0.3728	0.451	0.4984	0.4735	0.5016	152	185	153	
Phase 2	DETR	DETR-Original	41.6	83	0.4498	0.3926	0.4523	0.4204	0.5477	128	198	155	
Phase 2	DETR	DETR-Original	41.6	10	0.4781	0.5269	0.4153	0.4645	0.5847	49	44	69	
Phase 2	DETR	DETR-Original	41.6	Overall	0.3236								
Phase 2	DETR	RTDETR	41.6	0	0.321	0.5551	0.4818	0.5159	0.5182	1651	1323	1776	
Phase 2	DETR	RTDETR	41.6	1	0.4527	0.4567	0.4562	0.4565	0.5438	359	427	428	
Phase 2	DETR	RTDETR	41.6	2	0.3541	0.6371	0.6045	0.6204	0.3955	1573	896	1029	
Phase 2	DETR	RTDETR	41.6	3	0.3598	0.4948	0.4813	0.4879	0.5188	616	629	664	
Phase 2	DETR	RTDETR	41.6	9	0.4624	0.6249	0.6017	0.613	0.3983	583	350	386	
Phase 2	DETR	RTDETR	41.6	80	0.2834	0.4416	0.4142	0.4274	0.5858	922	1166	1304	
Phase 2	DETR	RTDETR	41.6	82	0.6161	0.7538	0.7277	0.7406	0.2723	588	192	220	
Phase 2	DETR	RTDETR	41.6	84	0.2858	0.5439	0.5002	0.5211	0.4998	1537	1289	1536	
Phase 2	DETR	RTDETR	41.6	85	0.3676	0.5515	0.4921	0.5201	0.5079	498	405	514	
Phase 2	DETR	RTDETR	41.6	16	0.6733	0.7895	0.7576	0.7732	0.2424	75	20	24	
Phase 2	DETR	RTDETR	41.6	81	0.3053	0.6077	0.5427	0.5734	0.4573	254	164	214	
Phase 2	DETR	RTDETR	41.6	5	0.4856	0.7	0.5694	0.628	0.4306	119	51	90	
Phase 2	DETR	RTDETR	41.6	7	0.4086	0.659	0.4751	0.5522	0.5249	172	89	190	
Phase 2	DETR	RTDETR	41.6	10	0.5863	0.7157	0.6186	0.6636	0.3814	73	29	45	
Phase 2	DETR	RTDETR	41.6	86	0.4078	0.7402	0.6651	0.7006	0.3349	282	99	142	
Phase 2	DETR	RTDETR	41.6	87	0.5128	0.6871	0.6558	0.6711	0.3442	202	92	106	
Phase 2	DETR	RTDETR	41.6	83	0.575	0.7143	0.636	0.6729	0.364	180	72	103	
Phase 2	DETR	RTDETR	41.6	Overall	0.4387								

Phase	Model_Archi	Model_Size	Params (M	Class	mAP	Precision	Recall	F1	FNR	TP	FP	FN	Average Inference Time (m
Phase 1	DETR	DETR-Original	41.6	0	0.2579	0.1892	0.3668	0.2496	0.6332	1257	5387	2170	
Phase 1	DETR	DETR-Original	41.6	1	0.1989	0.175	0.1919	0.183	0.8081	151	712	636	
Phase 1	DETR	DETR-Original	41.6	2	0.2974	0.243	0.4618	0.3184	0.5382	1202	3745	1401	
Phase 1	DETR	DETR-Original	41.6	3	0.2586	0.235	0.2875	0.2586	0.7125	368	1198	912	
Phase 1	DETR	DETR-Original	41.6	9	0.2215	0.0912	0.2291	0.1305	0.7709	222	2211	747	
Phase 1	DETR	DETR-Original	41.6	7	0.3231	0.4087	0.4683	0.4365	0.5317	170	246	193	
Phase 1	DETR	DETR-Original	41.6	16	0.7118	0.6991	0.79	0.7418	0.21	79	34	21	
Phase 1	DETR	DETR-Original	41.6	5	0.4522	0.5584	0.5263	0.5419	0.4737	110	87	99	
Phase 1	DETR	DETR-Original	41.6	10	0.3831	0.3401	0.5678	0.4254	0.4322	67	130	51	
Phase 1	DETR	DETR-Original	41.6	Overall	0.3449								818.2
Phase 1	DETR	RTDETR	41.6	0	0.3093	0.5927	0.4692	0.5238	0.5308	1608	1105	1819	
Phase 1	DETR	RTDETR	41.6	1	0.2408	0.5401	0.2313	0.3238	0.7687	182	155	605	
Phase 1	DETR	RTDETR	41.6	2	0.3506	0.6503	0.5866	0.6168	0.4134	1527	821	1076	
Phase 1	DETR	RTDETR	41.6	3	0	0	0	0	1	0	0	1280	
Phase 1	DETR	RTDETR	41.6	9	0.3263	0.348	0.4747	0.4016	0.5253	460	862	509	
Phase 1	DETR	RTDETR	41.6	7	0.3759	0.755	0.5179	0.6144	0.4821	188	61	175	
Phase 1	DETR	RTDETR	41.6	16	0.7254	0.8947	0.85	0.8718	0.15	85	10	15	
Phase 1	DETR	RTDETR	41.6	5	0.5206	0.7911	0.5981	0.6812	0.4019	125	33	84	
Phase 1	DETR	RTDETR	41.6	10	0.5715	0.7037	0.6441	0.6726	0.3559	76	32	42	
Phase 1	DETR	RTDETR	41.6	Overall	0.3801								560.2

Phase	Model_Archi	Model_Size	Params (M	Class	mAP	Precision	Recall	F1	FNR	TP	FP	FN	Average Inference Time (m
Phase 1	YOLO-WORLD	yolov8l-worldv2	43.7	0	0.292335	0.737979	0.402594	0.520976	0.597406	1366	485	2027	
Phase 1	YOLO-WORLD	yolov8l-worldv2	43.7	1	0.205553	0.768473	0.20339	0.321649	0.79661	156	47	611	
Phase 1	YOLO-WORLD	yolov8l-worldv2	43.7	2	0.317872	0.862014	0.424444	0.568812	0.575556	1087	174	1474	
Phase 1	YOLO-WORLD	yolov8l-worldv2	43.7	3	0.286663	0.697552	0.317928	0.436782	0.682072	399	173	856	
Phase 1	YOLO-WORLD	yolov8l-worldv2	43.7	5	0.465948	0.866667	0.58794	0.700599	0.41206	117	18	82	
Phase 1	YOLO-WORLD	yolov8l-worldv2	43.7	7	0.288579	0.837209	0.313043	0.455696	0.686957	108	21	237	
Phase 1	YOLO-WORLD	yolov8l-worldv2	43.7	9	0.283719	0.465177	0.369906	0.412107	0.630094	354	407	603	
Phase 1	YOLO-WORLD	yolov8l-worldv2	43.7	10	0.490705	0.575472	0.530435	0.552036	0.469565	61	45	54	
Phase 1	YOLO-WORLD	yolov8l-worldv2	43.7	16	0.676327	0.818182	0.684783	0.745562	0.315217	63	14	29	
Phase 1	YOLO-WORLD	yolov8l-worldv2	43.7	Overall	0.367522								18.3
Phase 1	YOLO-WORLD	yolov8x-worldv2	68.2	0	0.300171	0.723996	0.412801	0.525804	0.587199	1406	536	2000	
Phase 1	YOLO-WORLD	yolov8x-worldv2	68.2	1	0.230097	0.744493	0.222076	0.342105	0.777924	169	58	592	
Phase 1	YOLO-WORLD	yolov8x-worldv2	68.2	2	0.31881	0.866005	0.408028	0.554702	0.591972	1047	162	1519	
Phase 1	YOLO-WORLD	yolov8x-worldv2	68.2	3	0.296901	0.701874	0.325949	0.445165	0.674051	412	175	852	
Phase 1	YOLO-WORLD	yolov8x-worldv2	68.2	5	0.452376	0.845588	0.569307	0.680473	0.430693	115	21	87	
Phase 1	YOLO-WORLD	yolov8x-worldv2	68.2	7	0.274644	0.804511	0.305714	0.443064	0.694286	107	26	243	
Phase 1	YOLO-WORLD	yolov8x-worldv2	68.2	9	0.293867	0.451501	0.405602	0.427322	0.594398	391	475	573	
Phase 1	YOLO-WORLD	yolov8x-worldv2	68.2	10	0.495735	0.612245	0.526316	0.566038	0.473684	60	38	54	
Phase 1	YOLO-WORLD	yolov8x-worldv2	68.2	16	0.692859	0.866667	0.714286	0.783133	0.285714	65	10	26	
Phase 1	YOLO-WORLD	yolov8x-worldv2	68.2	Overall	0.372829								19.3
Phase 1	YOLO-WORLD	yolov8m-worldv2	25.9	0	0.290232	0.735388	0.408863	0.525536	0.591137	1384	498	2001	
Phase 1	YOLO-WORLD	yolov8m-worldv2	25.9	1	0.198103	0.681373	0.181226	0.286303	0.818774	139	65	628	
Phase 1	YOLO-WORLD	yolov8m-worldv2	25.9	2	0.315204	0.852021	0.434969	0.575922	0.565031	1117	194	1451	
Phase 1	YOLO-WORLD	yolov8m-worldv2	25.9	3	0.266202	0.681275	0.274038	0.390857	0.725962	342	160	906	
Phase 1	YOLO-WORLD	yolov8m-worldv2	25.9	5	0.421766	0.846154	0.552764	0.668693	0.447236	110	20	89	
Phase 1	YOLO-WORLD	yolov8m-worldv2	25.9	7	0.250463	0.826087	0.270655	0.407725	0.729345	95	20	256	
Phase 1	YOLO-WORLD	yolov8m-worldv2	25.9	9	0.262676	0.463127	0.330526	0.385749	0.669474	314	364	636	
Phase 1	YOLO-WORLD	yolov8m-worldv2	25.9	10	0.465432	0.508772	0.513274	0.511013	0.486726	58	56	55	
Phase 1	YOLO-WORLD	yolov8m-worldv2	25.9	16	0.651777	0.885714	0.681319	0.770186	0.318681	62	8	29	
Phase 1	YOLO-WORLD	yolov8m-worldv2	25.9	Overall	0.346873								16.7
Phase 1	YOLO-WORLD	yolov8s-worldv2	11.2	0	0.260713	0.730117	0.334328	0.45864	0.665672	1120	414	2230	
Phase 1	YOLO-WORLD	yolov8s-worldv2	11.2	1	0.148836	0.738806	0.133784	0.226545	0.866216	99	35	641	
Phase 1	YOLO-WORLD	yolov8s-worldv2	11.2	2	0.277987	0.864496	0.323252	0.470555	0.676748	823	129	1723	
Phase 1	YOLO-WORLD	yolov8s-worldv2	11.2	3	0.22107	0.653012	0.224152	0.333744	0.775848	271	144	938	
Phase 1	YOLO-WORLD	yolov8s-worldv2	11.2	5	0.349133	0.778761	0.446701	0.567742	0.553299	88	25	109	
Phase 1	YOLO-WORLD	yolov8s-worldv2	11.2	7	0.189426	0.795181	0.19186	0.309133	0.80814	66	17	278	
Phase 1	YOLO-WORLD	yolov8s-worldv2	11.2	9	0.209923	0.454198	0.252654	0.324693	0.747346	238	286	704	
Phase 1	YOLO-WORLD	yolov8s-worldv2	11.2	10	0.449316	0.550459	0.535714	0.542986	0.464286	60	49	52	
Phase 1	YOLO-WORLD	yolov8s-worldv2	11.2	16	0.623186	0.830986	0.641304	0.723926	0.358696	59	12	33	
Phase 1	YOLO-WORLD	yolov8s-worldv2	11.2	Overall	0.303288								15

Phase	Model_Archi	Model_Size	Params (M	Class	mAP	Precision	Recall	F1	FNR	TP	FP	FN	Average Inference Time (m
Phase 1	YOLO	yolov8l	43.7	0	0.308661	0.746988	0.454812	0.565384	0.545188	1550	525	1858	
Phase 1	YOLO	yolov8l	43.7	1	0.228015	0.748031	0.245161	0.369291	0.754839	190	64	585	
Phase 1	YOLO	yolov8l	43.7	2	0.337451	0.765854	0.54556	0.637204	0.45444	1413	432	1177	
Phase 1	YOLO	yolov8l	43.7	3	0.284563	0.659744	0.32571	0.436114	0.67429	413	213	855	
Phase 1	YOLO	yolov8l	43.7	5	0.481322	0.824324	0.600985	0.695157	0.399015	122	26	81	
Phase 1	YOLO	yolov8l	43.7	7	0.355916	0.87	0.490141	0.627027	0.509859	174	26	181	
Phase 1	YOLO	yolov8l	43.7	9	0.284175	0.429392	0.387565	0.407407	0.612435	374	497	591	
Phase 1	YOLO	yolov8l	43.7	10	0.563768	0.767442	0.573913	0.656716	0.426087	66	20	49	
Phase 1	YOLO	yolov8l	43.7	16	0.714487	0.906977	0.8125	0.857143	0.1875	78	8	18	
Phase 1	YOLO	yolov8l	43.7	Overall	0.395373								13.9
Phase 1	YOLO	yolov8x	68.2	0	0.306875	0.718157	0.465457	0.564831	0.534543	1590	624	1826	
Phase 1	YOLO	yolov8x	68.2	1	0.224315	0.703297	0.246154	0.364672	0.753846	192	81	588	
Phase 1	YOLO	yolov8x	68.2	2	0.343296	0.779173	0.573133	0.660457	0.426867	1489	422	1109	
Phase 1	YOLO	yolov8x	68.2	3	0.287508	0.656695	0.361569	0.466363	0.638431	461	241	814	
Phase 1	YOLO	yolov8x	68.2	5	0.507678	0.848684	0.635468	0.726761	0.364532	129	23	74	
Phase 1	YOLO	yolov8x	68.2	7	0.346996	0.846512	0.504155	0.631944	0.495845	182	33	179	
Phase 1	YOLO	yolov8x	68.2	9	0.297327	0.406445	0.404762	0.405602	0.595238	391	571	575	
Phase 1	YOLO	yolov8x	68.2	10	0.55522	0.727273	0.62069	0.669767	0.37931	72	27	44	
Phase 1	YOLO	yolov8x	68.2	16	0.728328	0.921348	0.854167	0.886486	0.145833	82	7	14	
Phase 1	YOLO	yolov8x	68.2	Overall	0.399727								14
Phase 1	YOLO	yolov8m	25.9	0	0.292957	0.722787	0.440059	0.547052	0.559941	1494	573	1901	
Phase 1	YOLO	yolov8m	25.9	1	0.202875	0.742358	0.219638	0.338983	0.780362	170	59	604	
Phase 1	YOLO	yolov8m	25.9	2	0.3245	0.765537	0.523773	0.621988	0.476227	1355	415	1232	
Phase 1	YOLO	yolov8m	25.9	3	0.270763	0.67053	0.320665	0.433851	0.679335	405	199	858	
Phase 1	YOLO	yolov8m	25.9	5	0.47821	0.852113	0.60199	0.705539	0.39801	121	21	80	
Phase 1	YOLO	yolov8m	25.9	7	0.326355	0.858696	0.44382	0.585185	0.55618	158	26	198	
Phase 1	YOLO	yolov8m	25.9	9	0.261101	0.412545	0.35625	0.382337	0.64375	342	487	618	
Phase 1	YOLO	yolov8m	25.9	10	0.565126	0.741935	0.6	0.663462	0.4	69	24	46	
Phase 1	YOLO	yolov8m	25.9	16	0.672009	0.864198	0.744681	0.8	0.255319	70	11	24	
Phase 1	YOLO	yolov8m	25.9	Overall	0.377099								11.7
Phase 1	YOLO	yolov8n	3.2	0	0.238345	0.719675	0.296556	0.42003	0.703444	973	379	2308	
Phase 1	YOLO	yolov8n	3.2	1	0.122597	0.6875	0.105191	0.182464	0.894809	77	35	655	
Phase 1	YOLO	yolov8n	3.2	2	0.284387	0.749188	0.365255	0.491088	0.634745	923	309	1604	
Phase 1	YOLO	yolov8n	3.2	3	0.171031	0.686275	0.152306	0.249288	0.847694	175	80	974	
Phase 1	YOLO	yolov8n	3.2	5	0.384431	0.891304	0.418367	0.569444	0.581633	82	10	114	
Phase 1	YOLO	yolov8n	3.2	7	0.223539	0.771186	0.269231	0.399123	0.730769	91	27	247	
Phase 1	YOLO	yolov8n	3.2	9	0.140447	0.415301	0.167401	0.238619	0.832599	152	214	756	
Phase 1	YOLO	yolov8n	3.2	10	0.505734	0.786667	0.541284	0.641304	0.458716	59	16	50	
Phase 1	YOLO	yolov8n	3.2	16	0.585123	0.814286	0.626374	0.708075	0.373626	57	13	34	
Phase 1	YOLO	yolov8n	3.2	Overall	0.295071								9.4
Phase 1	YOLO	yolov8s	11.2	0	0.269482	0.728465	0.380399	0.499804	0.619601	1277	476	2080	
Phase 1	YOLO	yolov8s	11.2	1	0.160281	0.686813	0.163399	0.263992	0.836601	125	57	640	
Phase 1	YOLO	yolov8s	11.2	2	0.31246	0.761034	0.471853	0.582529	0.528147	1207	379	1351	
Phase 1	YOLO	yolov8s	11.2	3	0.226833	0.664414	0.2436	0.356495	0.7564	295	149	916	
Phase 1	YOLO	yolov8s	11.2	5	0.438073	0.788321	0.556701	0.652568	0.443299	108	29	86	
Phase 1	YOLO	yolov8s	11.2	7	0.285052	0.836478	0.382184	0.524655	0.617816	133	26	215	
Phase 1	YOLO	yolov8s	11.2	9	0.203576	0.416933	0.278252	0.33376	0.721748	261	365	677	
Phase 1	YOLO	yolov8s	11.2	10	0.48545	0.63	0.557522	0.591549	0.442478	63	37	50	
Phase 1	YOLO	yolov8s	11.2	16	0.690023	0.875	0.736842	0.8	0.263158	70	10	25	
Phase 1	YOLO	yolov8s	11.2	Overall	0.341248								9.6

10 Conclusion

By following this manual, you should be able to replicate the experimental setup

References