YOLO Workflow for Custom Object Detection

This guide provides a step-by-step workflow for training a custom YOLO (You Only Look Once) model to detect specific objects of interest. It covers:

1. Preparing a robust dataset by recording videos, extracting frames, and organizing them for training.
2. Annotating images using **LabelImg** in YOLO format.
3. Configuring the Great Lakes HPC environment, including virtual environments and dependencies.
4. Transferring your dataset to the HPC and verifying its structure.
5. Training the model with YOLOv8 using SLURM job scripts.
6. Interpreting training logs and evaluating performance metrics like mAP.
7. Retrieving the trained model and artifacts for local use.
8. Properly closing your HPC session.

Contents

[Phase 0: Extracting & Preparing The Data 2](#_heading=h.lo2f4hw8qbgm)

[Step 0.1 Modify the Arduino Code 2](#_heading=h.a5wyeqd2d4g4)

[Step 0.2 Configure Xamera on Your Mobile Device 2](#_heading=h.rn850przrset)

[Step 0.3 Record Video 2](#_heading=h.6yv9u17cdtrt)

[Step 0.4 Transfer the Video 3](#_heading=h.nkukukej2gh5)

[Step 0.5 Extract Frames from the Video 3](#_heading=h.mojb7qomjain)

[Step 0.6 Select High-Quality Images 3](#_heading=h.picba2uykm1g)

[Phase 1: Labeling Images Locally 3](#_heading=h.gjdgxs)

[Step 1.1: Install LabelImg 3](#_heading=h.30j0zll)

[Step 1.2: Label the Images 4](#_heading=h.1fob9te)

[Step 1.3: Organize the Dataset 4](#_heading=h.3znysh7)

[Phase 2: Setting Up the HPC Environment 5](#_heading=h.2et92p0)

[Step 2.1: Access the HPC 5](#_heading=h.tyjcwt)

[Step 2.2: Set Up the Virtual Environment 5](#_heading=h.3dy6vkm)

[Step 2.3: Install Dependencies 5](#_heading=h.1t3h5sf)

[Phase 3: Transferring Data to the HPC 5](#_heading=h.4d34og8)

[Step 3.1: Transfer Files 5](#_heading=h.2s8eyo1)

[Step 3.2: Verify the Dataset 6](#_heading=h.17dp8vu)

[Phase 4: Training the YOLO Model on the HPC 7](#_heading=h.3rdcrjn)

[Step 4.1: Create the Configuration File 7](#_heading=h.26in1rg)

[Step 4.2: Create a SLURM Job Script 7](#_heading=h.lnxbz9)

[Step 4.3: Submit the SLURM Job 8](#_heading=h.35nkun2)

[Step 4.4: Monitor the Job 8](#_heading=h.1ksv4uv)

[Phase 5: Understanding the Output 9](#_heading=h.44sinio)

[Phase 6: Transferring Results Back to Local Machine 9](#_heading=h.2jxsxqh)

[Step 6.1: Transfer Trained Model 9](#_heading=h.z337ya)

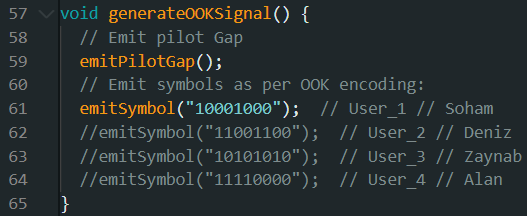
[Phase 7: Exit 10](#_heading=h.3j2qqm3)

# Phase 0: Extracting & Preparing The Data

## Step 0.1 Modify the Arduino Code

1. Open the Arduino code titled **“optimized\_glove”** from GitHub
2. Scroll to the **generateOOKSignal()** function at the bottom of the code.
3. Refer to the provided table and insert your unique Arduino code into the **emitSymbol()** call.
   * Alternatively, comment out the existing line and uncomment your custom line.
4. Upload the modified code to the Arduino Micro.

| YOLO/LabelImg Label | Arduino Code | Responsible |
| --- | --- | --- |
| User\_1 | 10101010 | Zaynab |
| User\_2 | 10001000 | Soham |



## Step 0.2 Configure Xamera on Your Mobile Device

1. Open the **Xamera app** on your mobile device.
2. Adjust the shutter rate to **1,000 Hz (1K Hz).**

## Step 0.3 Record Video

1. Start recording a **10-minute video** using Xamera.
   * The recording will automatically begin when you select **“Start Tracking”** and will save upon selecting **“Stop Tracking.”**
2. Ensure the video captures diverse scenarios to maximize data quality. Consider variations in:
   * Distance between the phone and the LED.
   * Rotation and angle of the phone relative to the LED.
   * Ambient lighting conditions (e.g., bright windows, indoor lighting, or darkness).
3. Plan your recording carefully to include a wide range of unique situations.

## Step 0.4 Transfer the Video

1. Move the saved video file from your mobile device to your laptop.

## Step 0.5 Extract Frames from the Video

1. Use a tool like **FFmpeg** or a Python script with **OpenCV** to split the video into individual frames.

| import cv2  import os  def extract\_frames(video\_path, output\_folder):  os.makedirs(output\_folder, exist\_ok=True)  cap = cv2.VideoCapture(video\_path)  if not cap.isOpened():  print(f"Error: Could not open video {video\_path}")  return  frame\_count = int(cap.get(cv2.CAP\_PROP\_FRAME\_COUNT))  print(f"Total frames in the video: {frame\_count}")  frame\_idx = 0  while True:  ret, frame = cap.read()  if not ret:  print("Finished extracting frames.")  break  frame\_filename = os.path.join(output\_folder, f"frame\_{frame\_idx:04d}.jpg")  cv2.imwrite(frame\_filename, frame)  frame\_idx += 1  if frame\_idx % 100 == 0: # Log every 100 frames  print(f"Extracted {frame\_idx}/{frame\_count} frames...")  cap.release()  print(f"All frames are saved in {output\_folder}")  video\_path = "path/to/your/video.mp4"  output\_folder = "extracted\_frames"  extract\_frames(video\_path, output\_folder) |
| --- |

## Step 0.6 Select High-Quality Images

1. Review the extracted frames and manually select **100 high-quality images** that represent diverse scenarios.

# Phase 1: Labeling Images Locally

## Step 1.1: Install LabelImg

LabelImg is a graphical image annotation tool that supports YOLO format. Install it on your local machine:

* GitHub Repository: <https://github.com/HumanSignal/labelImg>.

On Linux:

* pip install labelImg
* labelImg # Launch the application

On Windows:

1. Download the precompiled executable from <https://github.com/HumanSignal/labelImg/releases>.
2. Run the executable to launch the application.

## Step 1.2: Label the Images

1. Organize your images:
   * Place all images in a folder (e.g., images/).
2. Open LabelImg:
   * Set the image directory to your images/ folder.
   * Set the label directory to a new folder (e.g., labels/) where the .txt files will be saved.
3. Set LabelImg to YOLO mode:
   * In LabelImg, go to View > Auto Save Mode, and ensure the format is set to YOLO.
4. Set predefined classes:
   * In data/predefined\_classes.txt, define your class (e.g., your\_object\_name).
5. Label the images:

* Use the Create RectBox tool to draw bounding boxes around your object of interest.
* Assign the class name ***(refer to table)***.
* Save the annotations. Each image will have a corresponding .txt file in YOLO format:
  + <class\_id> <x\_center> <y\_center> <width> <height>

1. Verify the labels:
   * Ensure each image has a corresponding .txt file with accurate bounding box information.

## Step 1.3: Organize the Dataset

Organize the labeled dataset into the following structure:

dataset/

|── images/

│ |── train/ # Training images

│ └── val/ # Validation images

└── labels/

|── train/ # Labels for training images

└── val/ # Labels for validation images

* Split the dataset (80% for training, 20% for validation).
  + You can select these however you'd like; it doesn't matter which images/label data are used for training or validation, just split them!
* Ensure each image in images/train has a corresponding .txt file in labels/train.

# Phase 2: Setting Up the HPC Environment

## Step 2.1: Access the HPC

* Connect to the VPN (if working remotely):
  + Use <https://its.umich.edu/enterprise/wifi-networks/vpn/getting-started>.
* Access Great Lakes:
  + Via local terminal: ssh sohamn[@greatlakes.arc-ts.umich.edu](mailto:your_username@greatlakes.arc-ts.umich.edu)
  + Or via the web: <https://greatlakes.arc-ts.umich.edu/>.

## Step 2.2: Set Up the Virtual Environment

* Create a virtual environment:
  + python -m venv ~/YOLO-Soham
* Activate the virtual environment:
  + source ~/YOLO-Soham/bin/activate
* Load required modules:
  + module load python/3.12.1
  + module load cuda/12.1.1

## Step 2.3: Install Dependencies

* Install Ultralytics YOLOv8:
  + pip install ultralytics
* Install PyTorch:
  + pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121

# Phase 3: Transferring Data to the HPC

## Step 3.1: Transfer Files

* Use scp to transfer the dataset (from a local terminal):
  + scp C:\Users\ [your\_username@greatlakes.arc-ts.umich.edu:/home/user/](about:blank)
  + scp -r: (flag to transfer folders)
  + scp -r C:\Users\soham\Desktop\User\_1 2
  + scp -r F:\GitHub\Adaptive-HCI\YOLOv8\Tools\yolo\_v1 sohamn[@greatlakes.arc-ts.umich.edu:/home/sohamn/](about:blank)datasets

## Step 3.2: Verify the Dataset

Check the dataset structure on the HPC:

dataset/

|── images/

│ |── train/

│ └── val/

└── labels/

|── train/

└── val/

# Phase 4: Training the YOLO Model on the HPC

## Step 4.1: Create the Configuration File

* Create custom\_dataset.yaml:
  + nano yolo\_v1.yaml
* Add the following content:

| train: /home/sohamn/datasets/yolo\_v1/images/train val: /home/sohamn/datasets/yolo\_v1/images/val nc: 1 # Number of classes names: ['User\_1'] # Class names |
| --- |

## Step 4.2: Create a SLURM Job Script

* Create yolo\_job.slurm:
  + nano yolo\_v1.slurm
* Add the following content:

| #!/bin/bash  #SBATCH --job-name=yolo\_v1 # Name of the job  #SBATCH --partition=gpu # GPU partition  #SBATCH --gres=gpu:1 # Request 1 GPU  #SBATCH --time=04:00:00 # Max runtime (4 hours)  #SBATCH --mem=16G # Memory allocation  #SBATCH --output=/home/sohamn/logs/yolo\_cuda\_test.out # Standard output log  #SBATCH --error=/home/sohamn/logs/yolo\_cuda\_test.err # Error log (separate from output)  # Load necessary modules  module load python/3.12.1  module load cuda/12.1.1  # Activate the virtual environment  source ~/YOLOv11-Soham/bin/activate  # Run the YOLO training script  yolo train \  data=/home/sohamn/yaml/yolo\_v1.yaml \  model=/home/sohamn/models/yolov8n.pt \  epochs=100 \  imgsz=640 \  project=/home/sohamn/models \  name=yolo\_v1 |
| --- |

| **Model** | **Size (Parameters)** | **Speed (ms)** | **Accuracy (mAP)** | **Use Case** |
| --- | --- | --- | --- | --- |
| YOLOv8n | ~3.2M | ~6.3ms | ~37.3 | Edge devices, real-time apps |
| YOLOv8s | ~11.2M | ~6.4ms | ~44.9 | Lightweight applications |
| YOLOv8m | ~25.9M | ~8.2ms | ~50.2 | General-purpose detection |
| YOLOv8l | ~43.7M | ~10.1ms | ~52.9 | High-accuracy tasks |
| YOLOv8x | ~68.2M | ~12.3ms | ~53.9 | Maximum accuracy, high-resources |

## Step 4.3: Submit the SLURM Job

* Submit the job:
  + sbatch yolo\_job.slurm

## Step 4.4: Monitor the Job

* Check Job Status
  + squeue -u sohamn
    - This will display information about all your running and queued jobs.
    - Look for your job’s JOBID and STATE (e.g., RUNNING, PENDING, etc.).
  + Refresh Job Status:
    - watch -n 5 squeue -u your\_username
      * This command refreshes the job status every 5 seconds, displaying the most up-to-date information.
* View Logs (Completed Jobs)
  + cat yolo\_cuda\_test.out
    - This prints the entire log file to the terminal.
  + For large log files, use less to view them page by page:
    - less yolo\_cuda\_test.out
      * Use the arrow keys to navigate and press ‘q’ to quit.
* Stream Logs in Real-Time (During Execution)
  + tail -f yolo\_cuda\_test.out
    - This command streams new lines to your terminal as they are added to the log file.
    - To stop streaming, press Ctrl+C.

# Phase 5: Understanding the Output

During training, YOLO provides detailed logs which are written to the terminal and saved in the SLURM output file (e.g., yolo\_cuda\_test.out).

* **Epoch Progress**: Shows the current epoch and total number of epochs.
* **Loss Values**:
  + **Box Loss**: Measures the accuracy of the bounding box predictions.
  + **Objectness Loss**: Evaluates how well the model detects objects versus the background.
  + **Classification Loss**: Assesses the accuracy of class predictions.
* **Metrics**:
  + **Precision**: The proportion of correct object detections out of all detections.
  + **Recall**: The proportion of actual objects detected by the model.
  + **mAP (Mean Average Precision)**:
    - **mAP@0.5**: Average precision for IoU (Intersection over Union) threshold 0.5.
    - **mAP@0.5:0.95**: Average precision averaged across IoU thresholds from 0.5 to 0.95.

# Phase 6: Transferring Results Back to Local Machine

## Step 6.1: Transfer Trained Model

* At the end of training, the trained model and additional files are saved in the following directory:
  + runs/detect/train/
* Key Files:
  + best.pt: The best-performing model based on validation metrics.
    - This is selected based on the lowest validation loss or the highest **mAP (mean Average Precision)**, depending on YOLO's configuration.
    - Use for scenarios where the validation dataset reflects real-world data.
    - ***Generally, use best.pt!***
  + wlast.pt: The final model at the end of training.
    - It may or may not be better than best.pt since the model could overfit or diverge in the later epochs.
    - Use if you believe the validation dataset isn't fully representative of real-world scenarios and want to experiment with the final model.
* Additional logs and visualizations, such as loss curves and detection examples, are also saved in this folder.
* Transfer the trained model to your local machine (from a local terminal):
  + scp -r sohamn[@greatlakes.arc-ts.umich.edu:](about:blank)/home/sohamn/models/yolo\_v1 F:\GitHub\Adaptive-HCI\YOLOv8\Models
  + scp -r: (flag to transfer folders)

# Phase 7: Exit

To exit the HPC:

exit