GSOC’25-JDEROBOT

Extend DetectionMetrics: GUI, CI Workflow, and Object Detection

Mentors : David Pascual Hernández, Sergio Paniego, Santiago Montiel Marín

Contact Details:

Name	:	Sakhineti Praveena
Country	:	India
Timezone	:	IST (GMT +5:30)
Email	:	sakhinetipraveena@gmail.com
Github	:	https://github.com/SakhinetiPraveena
LinkedIn	:	https://www.linkedin.com/in/sakhinetipraveena/
Resume	:	Resume link
Phone no	:	+91 9652924569

Contents

Introduction

Hey! I am Praveena, currently working full time as an **Applications Developer** at **Oracle, India**. I earned my Bachelor of Technology in Electronics and Communication Engineering from the **National Institute of Technology, Calicut**. In the summer of 2022, during my third year of college, I completed my **summer internship** at Oracle. During this time, I developed a highly efficient HCM chatbot by integrating Oracle Digital Assistant with Oracle Connections, leveraging OCI’s NLP models, and deploying it across the IDCS-ODA platform, web, and Slack. After graduating in 2023, I joined Oracle full-time. I work in product development for Human Capital Management (HCM), an Oracle Fusion application. My role involves handling both frontend and backend development for CoreHR modules based on the design team’s requirements.

Project Overview

DetectionMetrics is a toolkit designed to unify and streamline the evaluation of perception models across different frameworks and datasets. The older version i.e DetectionMetrics v1 provides objective performance metrics such as mean average precision and mean inference time for object detection models. Now

DetectionMetrics v2 was introduced in which the previous version was redesigned with an expanded focus on image and LiDAR segmentation.

Currently, DetectionMetrics v2 functions as both a Python library and CLI, focusing on the quantitative evaluation of image and LiDAR segmentation models, with plans to expand into object detection. It supports PyTorch and TensorFlow models, along with multiple public datasets. Its modular design allows easy integration of new models and datasets.
This project aims to bring back the key functionalities from v1 that were lost while redesigning, build a GUI and set up a CI workflow.

Goals/Deliverables

1. Recovering support for object detection evaluation.
2. Building a GUI.
3. Adding a GitHub Action to generate a PIP package and push to PyPI automatically.
4. Moving the docs to gh_pages branch (issue #236)
5. Adding more tutorials for image and LiDAR segmentation evaluation. Also for the newly added object detection model evaluation.

Implementation

1. Recovering support for object detection evaluation

A. Defining *DetectionModel* Class

I have two ways to do this in mind.

a. One way is to create a base class *PerceptionModel* containing the key common functionalities from which *DetectionModel* class and *SegmentationModel* inherit. And then further create an *ImageDetectionModel* class that inherits from *DetectionModel,* where we implement object detection logic.

b. As an alternative, instead of creating a separate DetectionModel class, we can modify *SegmentationModel* to serve as a base class for both segmentation and detection.

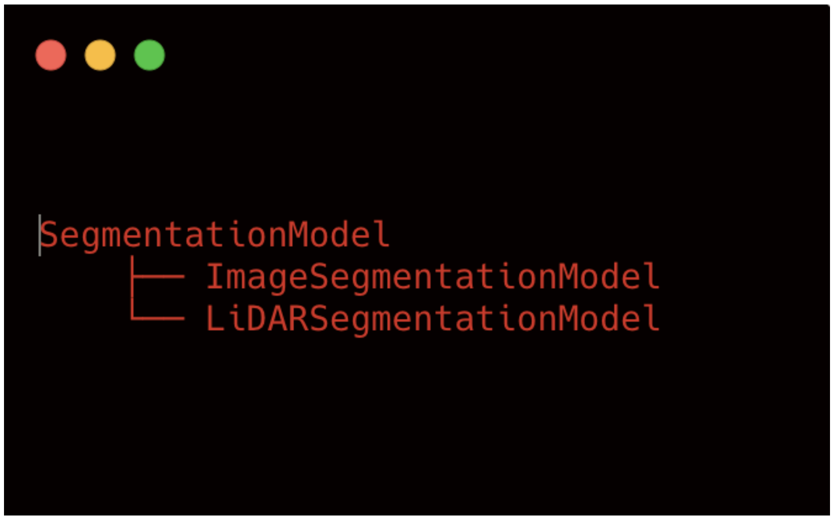
Why use the first approach?

- While both segmentation and detection are perception tasks, they differ significantly in output format, Evaluation metrics, Dataset structure and preprocessing.
- Creating *PerceptionModel* will help us maintain the shared parts between segmentation and detection, mainly model loading, config and ontology parsing and file existence checks.
- In future, if we want to add other models like Classification, Depth Estimation or Keypoint Detection. It's cleaner if they all inherit from a common base class.

Addressing key differences between Segmentation and Detection models :

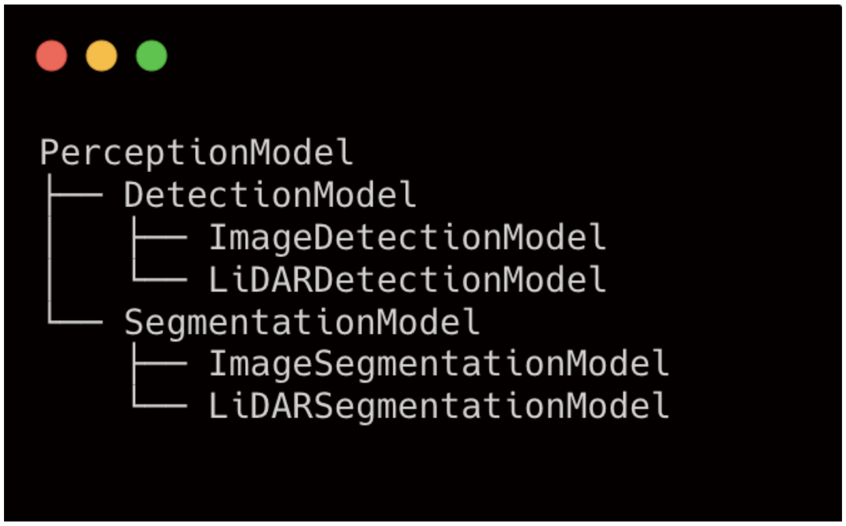
- Input handling differs as segmentation models handle Image→Pixel-wise classification but detection models handle Image→Object-level classification.
- Output Type for segmentation models is Per-pixel classification but for detection models it is Bounding boxes + class labels
- Evaluation metrics such as mAP should be added for detection models.

Older class structure:



```
SegmentationModel
      ├── ImageSegmentationModel
      └── LiDARSegmentationModel
```

Proposed Class Structure:



```
PerceptionModel
├── DetectionModel
│   ├── ImageDetectionModel
│   └── LiDARDetectionModel
└── SegmentationModel
    ├── ImageSegmentationModel
    └── LiDARSegmentationModel
```

Final Approach :

- While defining *PerceptionModel*, we will only define the core shared logic there, and let the subclasses define their domain-specific logic.
- I am planning to keep *get_computational_cost()* and *get_lut_ontology()* as abstract methods in *PerceptionModel*, and will override them in the child classes.
- Since segmentation and detection inherently have different tasks for *inference()* and *eval()* I will directly define them as abstract methods in their specific model classes.

- The base class would be something similar to

```python
class PerceptionModel:
    """Parent perception model class

    :param model: Segmentation model object
    :type model: Any
    :param model_type: Model type (e.g. scripted, compiled, etc.)
    :type model_type: str
    :param model_cfg: JSON file containing model configuration
    :type model_cfg: str
    :param ontology_fname: JSON file containing model output ontology
    :type ontology_fname: str
    :param model_fname: Model file or directory, defaults to None
    :type model_fname: Optional[str], optional
    """

    def __init__(
        self,
        model: Any,
        model_type: str,
        model_cfg: str,
        ontology_fname: str,
        model_fname: Optional[str] = None,
    ):
        self.model = model
        self.model_type = model_type
        self.model_fname = model_fname

        # Check that provided paths exist
        assert os.path.isfile(ontology_fname), "Ontology file not
found"
        assert os.path.isfile(model_cfg), "Model configuration not
found"
        if self.model_fname is not None:
            assert os.path.exists(model_fname), "Model file or
directory not found"

        # Read ontology and model configuration
        self.ontology = uio.read_json(ontology_fname)
        self.model_cfg = uio.read_json(model_cfg)
        self.n_classes = len(self.ontology)

     def get_lut_ontology(
        self, dataset_ontology: dict, ontology_translation: 
Optional[str] = None
    ):
        """Build ontology lookup table (leave empty if ontologies 
match).

        :param dataset_ontology: Image or LiDAR dataset ontology
        :type dataset_ontology: dict
        :param ontology_translation: JSON file containing translation 
between model and dataset ontologies, defaults to None
        :type ontology_translation: Optional[str], optional
        :return: Lookup table for ontology conversion or None if 
ontologies match
        :rtype: dict or None
        """
        lut_ontology = None
        if dataset_ontology != self.ontology:
            if ontology_translation is not None:
                ontology_translation = 
uio.read_json(ontology_translation)
            lut_ontology = uc.get_ontology_conversion_lut(
                dataset_ontology,
                self.ontology,
                ontology_translation,
                self.model_cfg.get("ignored_classes", []),
            )
        return lut_ontology

    @abstractmethod
    def get_computational_cost(self, runs: int = 30, warm_up_runs: 
int = 5) -> dict:
        """Get different metrics related to the computational cost of 
the model.

        :param runs: Number of runs to measure inference time,
defaults to 30
        :type runs: int, optional
        :param warm_up_runs: Number of warm-up runs, defaults to 5
        :type warm_up_runs: int, optional
        :return: Dictionary containing computational cost information
        :rtype: dict
        """
        raise NotImplementedError
```

- Define *DetectionModel* similar to *SegmentationModel*, to hold the logic specific to detection models and modify both of them to inherit from the new base class.
- Design for methods in *DetectionModel* class:
    - Inference Method (inference) : Returns bounding boxes, class labels, and confidence scores

```
    @abstractmethod
    def inference(
        self, input_data: Union[np.ndarray, Image.Image]
    ) -> List[Dict]:
        """Perform object detection inference.

        :param input_data: Either an image (PIL.Image) or a numpy
array (point cloud)
        :type input_data: Union[np.ndarray, Image.Image]
        :return: List of detected objects with bounding boxes,
labels, and confidence scores.
        :rtype: List[Dict]
        """
        raise NotImplementedError
```

- Evaluation Method (eval) : Matches bounding boxes with ground truth using IoU, then calculates mAP.

```
    @abstractmethod
    def eval(
        self,
        dataset: dm_dataset.ImageDetectionDataset,
        split: str | List[str] = "test",
        iou_thresholds: List[float] = [0.5, 0.75],
        predictions_outdir: Optional[str] = None,
        results_per_sample: bool = False,
    ) -> pd.DataFrame:
        """Perform evaluation for an object detection dataset.

        :param dataset: Detection dataset for evaluation
        :type dataset: ImageDetectionDataset
        :param split: Dataset split(s) for evaluation, defaults to
"test"
        :type split: str | List[str], optional
        :param iou_thresholds: List of IoU thresholds for mAP
calculation, defaults to [0.5, 0.75]
        :type iou_thresholds: List[float], optional
        :param predictions_outdir: Directory to save predictions per
sample, defaults to None
        :type predictions_outdir: Optional[str], optional
        :param results_per_sample: Store per-sample results if True
(requires predictions_outdir)
        :type results_per_sample: bool, optional
        :return: DataFrame containing evaluation results (e.g., mAP,
Precision, Recall)
        :rtype: pd.DataFrame
        """
        raise NotImplementedError
```

- Data Handling & Post-processing : Uses bounding box coordinates instead of a dense mask.
- Computational cost (get_computational_cost) : Returns computational cost similar to segmentation models except the logic for number of bounding boxes processed per second is added as well.

- Ontology matching (get_lut_ontology) : Maps class names to integer labels and ensures bounding box formats match. This is an abstract method in *PerceptionModel* and is being overridden in this class

```
 def get_lut_ontology(
        self, dataset_ontology: dict, ontology_translation:
Optional[str] = None
    ):
        """Build an ontology lookup table for object detection
models.

        :param dataset_ontology: Dataset ontology containing object
classes
        :type dataset_ontology: dict
        :param ontology_translation: JSON file containing translation
between dataset and model ontologies
        :type ontology_translation: Optional[str], optional
        :return: Lookup table for ontology conversion or None if
ontologies match
        :rtype: dict or None
        """
        lut_ontology = super().get_lut_ontology(dataset_ontology,
ontology_translation)

        # Extra step: Ensure class-to-ID mapping for object detection
        if lut_ontology:
            class_to_id_map = {cls: i for i, cls in
enumerate(self.ontology)}
            lut_ontology = {k: class_to_id_map[v] for k, v in
lut_ontology.items()}

        return lut_ontology
```

B. Defining DetectionDataset

Following a similar approach that I used to define *ImageDetectionModel,* we can create a *PerceptionDataset* class to serve as a base class for both segmentation and detection classes.

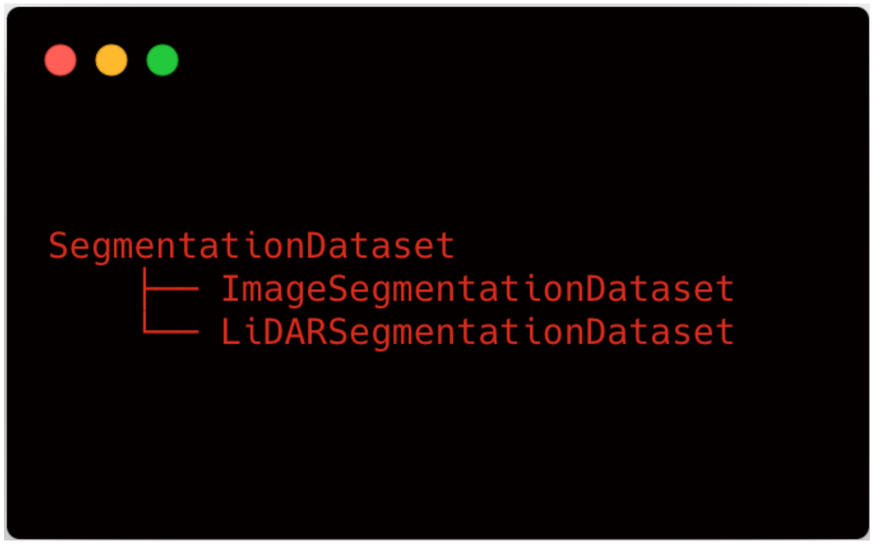
Common functionalities that stay common between segmentation and detection datasets :

- Loading images
- Handling dataset paths & file structures
- Basic transformations (resizing images, normalizing pixel values)
- Supporting multiple datasets/formats

Addressing key differences between Segmentation and Detection datasets :

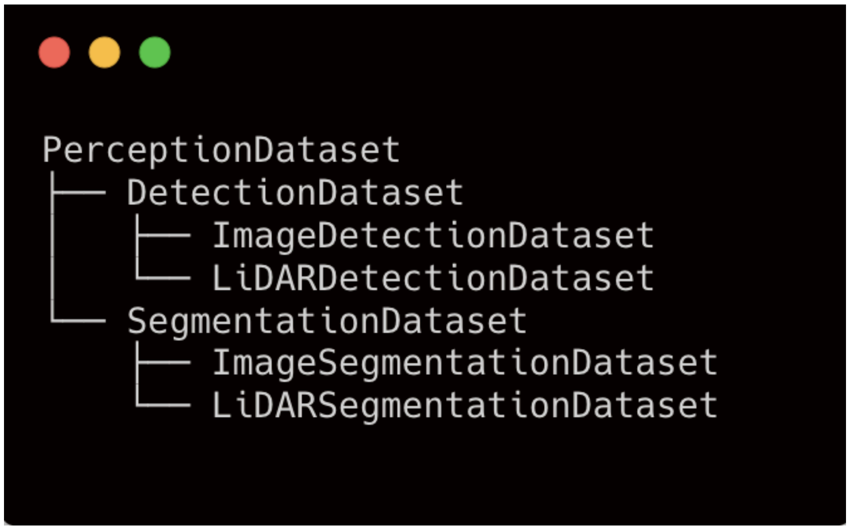
- Annotation data is stored as segmentation masks for segmentation but stored as bounding boxes for detection.
- While preprocessing normalising masks wouldn't be necessary and involves resizing bounding boxes proportionally when augmenting images in case of detection.
- The label format also differs because bounding box coordinates + class labels are used for detection models (e.g., [x_min, y_min, x_max, y_max, class]).

  Old Structure:



```
SegmentationDataset
      ├── ImageSegmentationDataset
      └── LiDARSegmentationDataset
```

Proposed new structure:



```
PerceptionDataset
├── DetectionDataset
│   ├── ImageDetectionDataset
│   └── LiDARDetectionDataset
└── SegmentationDataset
    ├── ImageSegmentationDataset
    └── LiDARSegmentationDataset
```

Final Approach :

- Defining PerceptionDataset, we will only define the core shared logic there, and let the subclasses define their domain-specific logic cleanly.
- PerceptionDataset will be something similar to (will modify accordingly as I progress further with implementation)

```
class PerceptionDataset(ABC):
"""Abstract perception dataset class

    :param dataset: Segmentation dataset as a pandas DataFrame
    :type dataset: pd.DataFrame
    :param dataset_dir: Dataset root directory
    :type dataset_dir: str
    :param ontology: Dataset ontology definition
    :type ontology: dict
    """


    def __init__(self, dataset: pd.DataFrame, dataset_dir: str,
ontology: dict):
        self.dataset = dataset
        self.dataset_dir = os.path.abspath(dataset_dir)
        self.ontology = ontology
        self.has_label_count = all("label_count" in v for v in
self.ontology.values())

    def __len__(self):
        return len(self.dataset)

    @abstractmethod
    def make_fname_global(self):
        """Convert all relative filenames in the dataset to absolute
paths."""
        raise NotImplementedError

    @abstractmethod
    def get_label_count(self, splits: Optional[List[str]] = None) ->
np.ndarray:
        """Get label count for each class in the dataset.

        :param splits: Dataset splits to consider (e.g., ["train",
"val"])
        :type splits: List[str], optional
        :return: Label count for the dataset
        :rtype: np.ndarray
        """
        raise NotImplementedError

    def append(self, new_dataset: Self):
        """Append another dataset with a common ontology.

        :param new_dataset: Dataset to be appended
        :type new_dataset: Self
        """
        if not self.has_label_count:
            assert self.ontology == new_dataset.ontology, "Ontologies
don't match"
        else:
            assert (
                self.ontology.keys() == new_dataset.ontology.keys()
            ), "Ontologies don't match"
            for class_name in self.ontology:
                assert (
                    self.ontology[class_name]["idx"]
                    == new_dataset.ontology[class_name]["idx"]
                ), f"Ontologies don't match for class {class_name}
(idx mismatch)"
                assert (
                    self.ontology[class_name]["rgb"]
                    == new_dataset.ontology[class_name]["rgb"]
                ), f"Ontologies don't match for class {class_name}
(RGB mismatch)"

                # Accumulate label count
                self.ontology[class_name]["label_count"] +=
new_dataset.ontology[
                    class_name
                ]["label_count"]

        self.make_fname_global()
        new_dataset.make_fname_global()

        # Concatenate pandas dataframes
        self.dataset = pd.concat(
            [self.dataset, new_dataset.dataset],
verify_integrity=True
        )
```

- Design for methods in DetectionDataset class :
    - make_fname_global() : Needs to be adapted for object detection to work with image filenames and annotation files instead of segmentation masks or point cloud files.
    - export() : Needs to handle exporting object detection datasets in the required format (e.g., COCO, Pascal VOC). It should ensure that annotations are converted correctly, and images are saved properly.
    - read_label() : This needs to be adapted for reading object detection annotations instead of segmentation masks.
    - get_label_count() : This should count the occurrences of object classes in the dataset, rather than pixel-level segmentation labels.

C. Implementing evaluation metrics for object detection
Evaluating segmentation models operates at the pixel level. Each pixel is assigned a class label. We compare predicted pixel labels to ground truth labels using a confusion matrix. But evaluating detection models operates at the object instance level. Each prediction is a bounding box + class label + confidence score, and we need to match predicted boxes to ground truth boxes using IoU thresholding.
For segmentation in *utils/ metrics.py* we already compute TP, FP, FN, TN using a confusion matrix. Metrics like IoU, accuracy, precision, recall are calculated per class across the whole image (pixel-wise).

For detection models :

- We need to first perform IoU-based matching of predicted boxes and ground truth boxes for each image.
- Metrics include:
    - Precision–Recall curve
    - Average Precision (AP) per class
    - Mean Average Precision (mAP) across all classes
    - Average Recall (if needed)
- The expected workflow for detection evaluation is
    - For each image:
        - predictions: list of boxes, labels, scores
        - ground_truths: list of boxes, labels
    - Match predicted boxes with GT using IoU > threshold (e.g., 0.5)
        - Label matches as TP or FP
        - Unmatched GTs are FN
    - Track TP, FP, FN per class
    - Feed into MetricsFactory
    - Generate final metrics and dataframe

D. Make object detection evaluation compatible with COCO and Pascal VOC-style datasets

- I am planning to create *coco.py* and *pascalVoc.py* under the datasets folder.
- We need to parse one JSON in case of COCO whereas many XMLs in case of Pascal VOC.

- And the bounding box format is also a bit different, for COCO the format is *[x_min, y_min, width, height]*, but for pascal VOC it is *[xmin, ymin, xmax, ymax]*
- In *coco.py* we can define COCOImageDetectionDataset that is derived from ImageDetectionDataset, and use COCO API's to load the dataset. Something similar to the code below
  from pycocotools.coco import COCO (further methods are to be added still).
  We can similarly use the COCO API for evaluating as well.
- A row in the final dataframe for COCO would be something like

```
{
    "image": "path/to/coco/images/000000012345.jpg",
    "labels": [
        {
            "category_id": 1,
            "bbox": [48.0, 30.5, 150.0, 200.0],  # [x, y, width, height]
            "area": 30000.0,
            "iscrowd": 0,
            "segmentation": [],  # optional
        },
        {
            "category_id": 3,
            "bbox": [300.0, 100.0, 100.0, 120.0],
            "area": 12000.0,
            "iscrowd": 0,
            "segmentation": []
        }
    ],
    "image_id": 12345,
    "width": 640,
    "height": 480,
    "split": "train",
    "scene": "coco_2017_train"
}
```

- In *pascalVoc.py* we can define PascalVOCImageDetectionDataset that is derived from ImageDetectionDataset. Since Pascal VOC uses XML annotations, we need to parse them with xml.etree.ElementTree.

- A row in the final dataframe for Pascal-VOC would be something like

```
{
    "image": "path/to/VOC2012/JPEGImages/2007_000027.jpg",
    "labels": [
        {
            "name": "person",
            "bbox": [104, 78, 375, 183],  # [xmin, ymin, xmax, ymax]
            "pose": "Left",
            "truncated": 0,
            "difficult": 0
        },
        {
            "name": "dog",
            "bbox": [8, 12, 400, 300],
            "pose": "Frontal",
            "truncated": 1,
            "difficult": 0
        }
    ],
    "image_id": "2007_000027",
    "width": 500,
    "height": 333,
    "split": "train",
    "scene": "voc_2012"
}
```

E. Make object detection evaluation compatible with Mask R-CNN and YOLO models

- We need to modularize model support, create mask_rcnn.py and yolo.py.
- Mask R-CNN model can be loaded using torchvision (*torchvision.models.detection.maskrcnn_resnet50_fpn()*)
- YOLO model can be loaded using (*cv2.dnn.readNet("yolov4.weights", "yolov4.cfg")*)
- We need to normalize predictions , we can probably create a *normalize.py* in utils that converts different outputs into a unified format.

- We need to integrate evaluation, we can use a single evaluation pipeline that takes in the normalized predictions. We can extend our current logic to support bounding boxes as well.
- Something like :

```
if "masks" in predictions:
    evaluate_segmentation(predictions["masks"], ...)
else:
    evaluate_detection(predictions["boxes"], predictions["labels"], 
...)
```

2. Building a GUI using streamlit

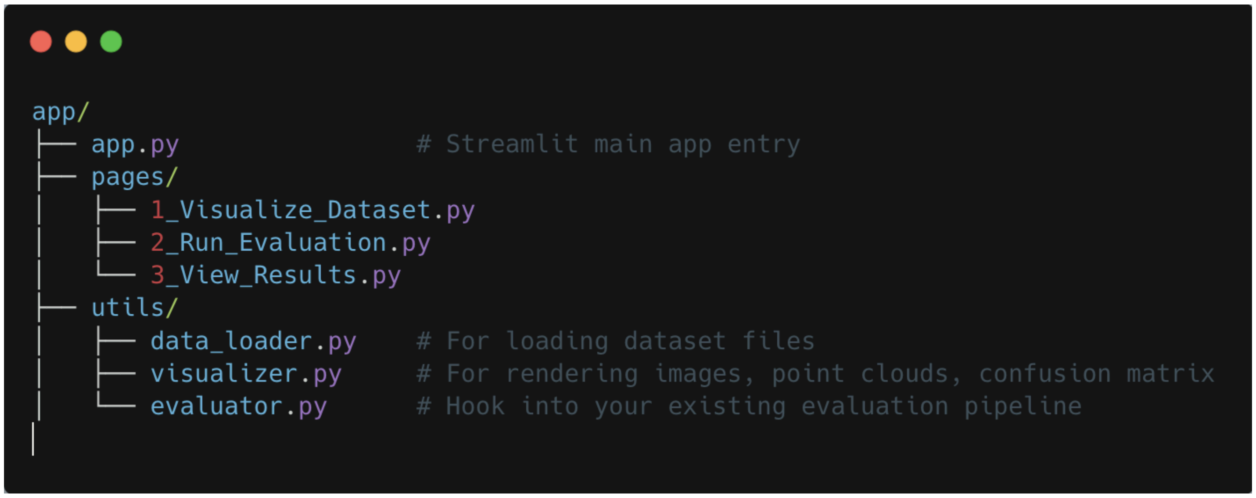
Moving on I will get started with creating a streamlined and interactive GUI for:

- Visualizing evaluation results (metrics, confusion matrix, etc.)
- Browsing dataset samples (images & point clouds)
- Running evaluation jobs in batch (select model + dataset + split)

Why use streamlit?

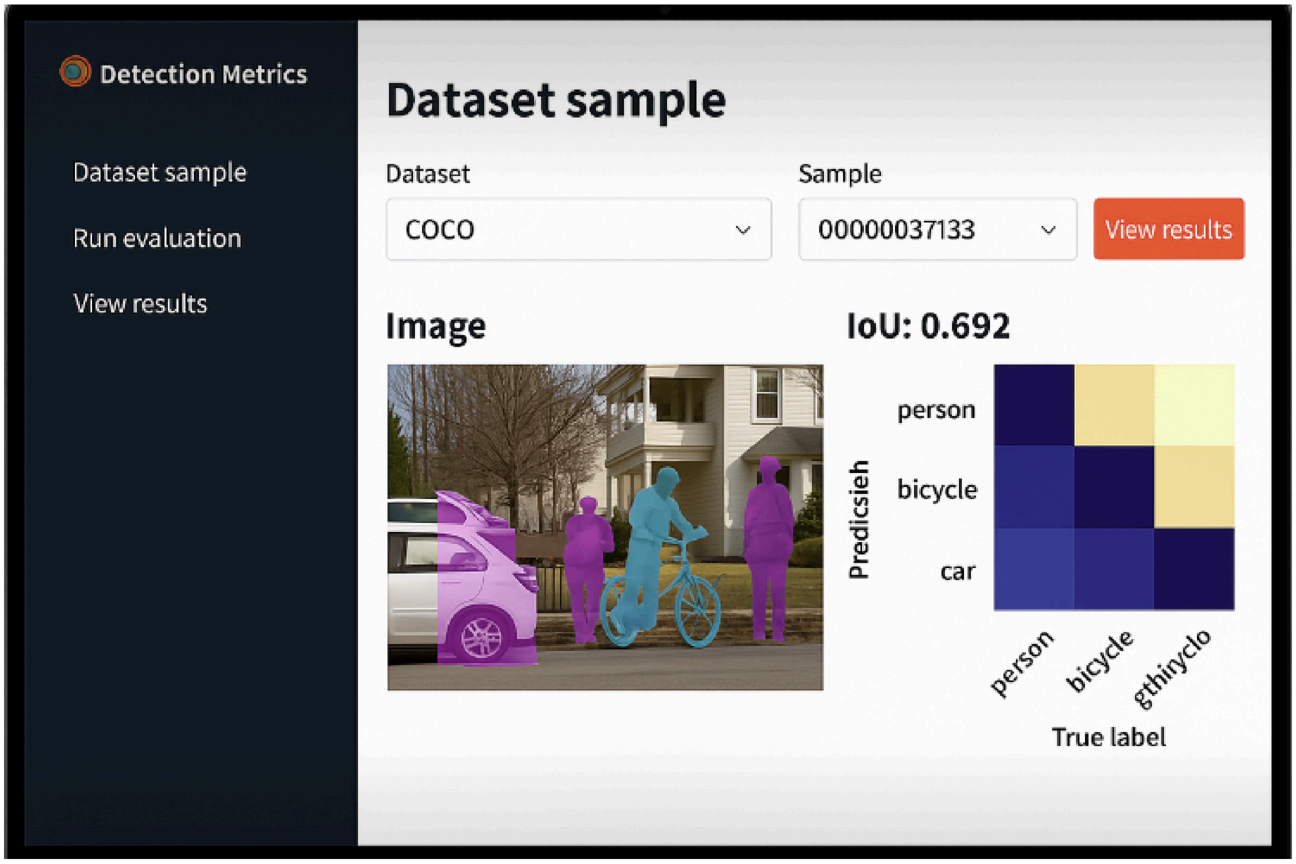
- Streamlit is better suited for a dashboard-style tool that handles various inputs, visualizations, and navigation.
- We have full control (e.g., tabs for "Run Model", "Evaluate", "Visualize") in streamlit, whereas in Gradio there are fixed UI patterns—limited customization.
- In streamlit we have st.selectbox() for model/dataset selection and st.file_uploader() for custom inputs
- We have tabs or sidebar for switching between "Inference", "Evaluation", "Visualization"
- And finally streamlit has embedded plotly, matplotlib, or even st.dataframe() for results.

The app structure I have in mind is



```
app/
├── app.py              # Streamlit main app entry
├── pages/
│   ├── 1_Visualize_Dataset.py
│   ├── 2_Run_Evaluation.py
│   └── 3_View_Results.py
├── utils/
│   ├── data_loader.py    # For loading dataset files
│   ├── visualizer.py     # For rendering images, point clouds, confusion matrix
│   └── evaluator.py      # Hook into your existing evaluation pipeline
│
```

The UI would look something similar to



Page Features:

Visualize Dataset Samples

- Inputs:
  - Dataset selector
  - Split selector (train/val/test)
  - Sample browser (slider, dropdown, or random)
- Views:
  - Images + annotations overlay
  - Point clouds with class colors (if LiDAR)
  - Class histogram

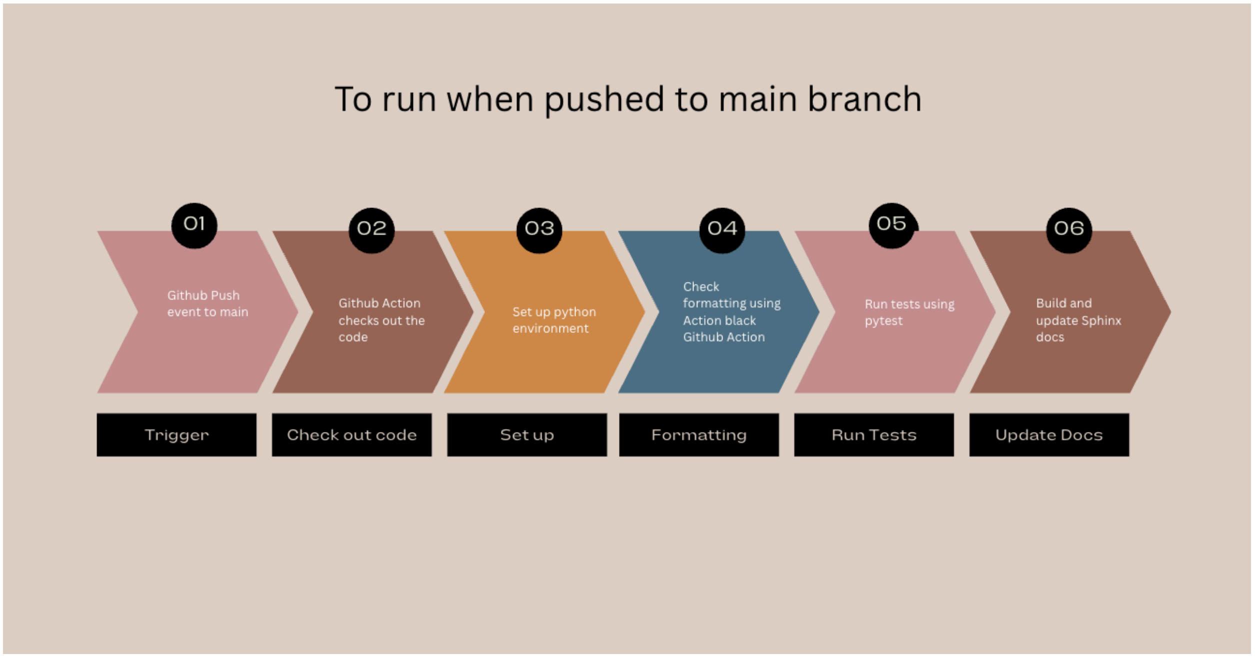
Run Evaluation

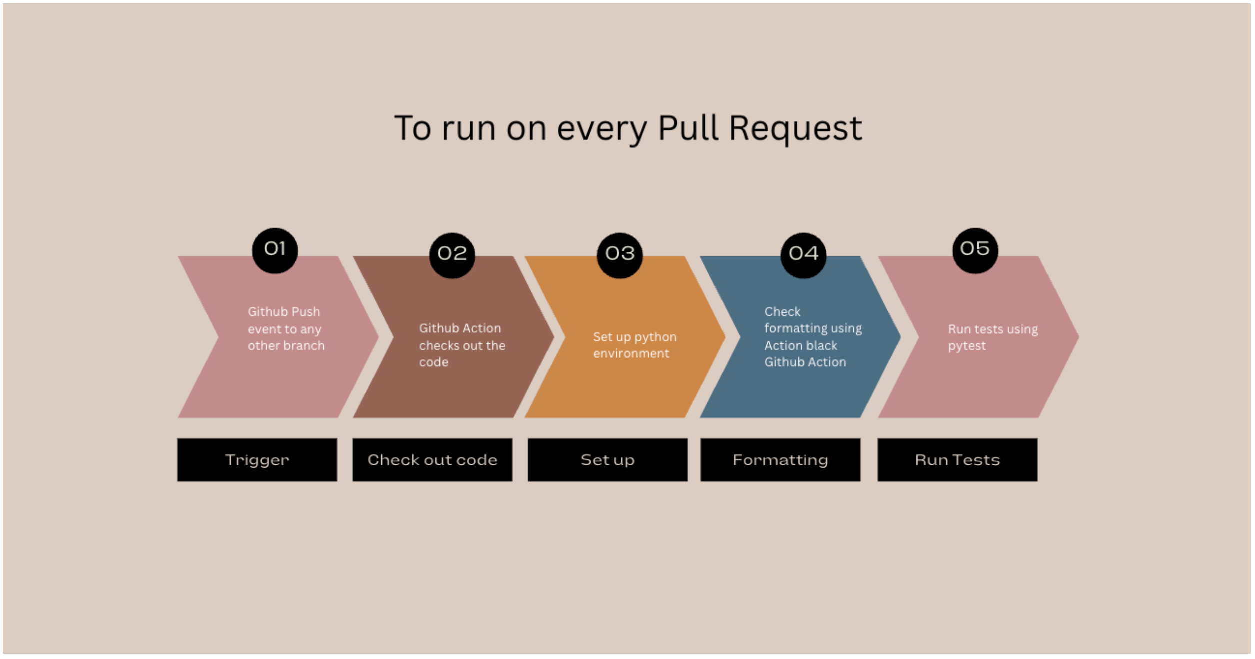
- Inputs:
  - Model selector (Mask R-CNN, YOLO, etc.)
  - Dataset selector
  - Split selector
  - Evaluation config (batch size, confidence thresholds)
- Actions:
  - Button to launch evaluation (with progress bar)
  - Live logging in expandable box

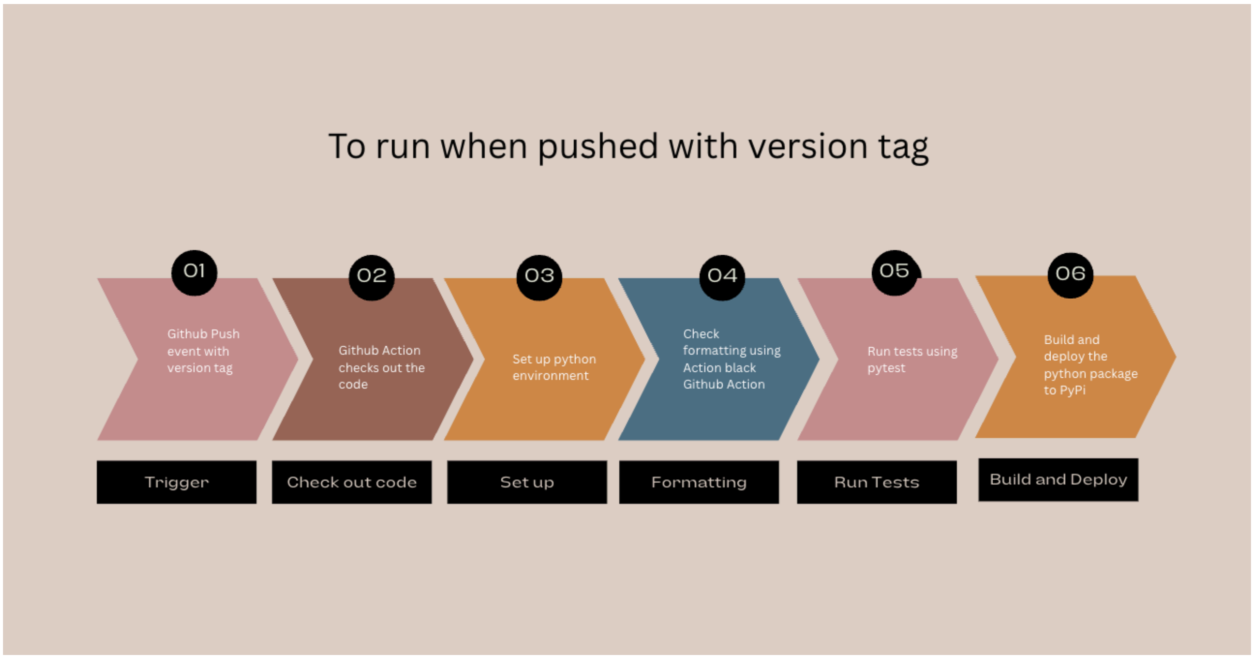
View Results

- Inputs:
  - Select past evaluation run from dropdown (load from saved path)
- Views:
  - mIoU, mAP, precision-recall, per-class metrics
  - Confusion matrix heatmap (interactive)
  - Class-wise IoU bars
  - Sample-wise error analysis (with predicted vs. true overlay)

3. CI Workflow


To run when pushed to main branch
01
Github Push event to main
Trigger
02
Github Action checks out the code
Check out code
03
Set up python environment
Set up
04
Check formatting using Action black Github Action
Formatting
05
Run tests using pytest
Run Tests
06
Build and update Sphinx docs
Update Docs


To run on every Pull Request
01
Github Push event to any other branch
Trigger
02
Github Action checks out the code
Check out code
03
Set up python environment
Set up
04
Check formatting using Action black Github Action
Formatting
05
Run tests using pytest
Run Tests


To run when pushed with version tag
01
Github Push event with version tag
Trigger
02
Github Action checks out the code
Check out code
03
Set up python environment
Set up
04
Check formatting using Action black Github Action
Formatting
05
Run tests using pytest
Run Tests
06
Build and deploy the python package to PyPi
Build and Deploy

A. **Check Black formatting**

I recently came across Action-black GitHub Action , using which we can check formatting of the code every time a code is pushed.

I can implement any one of the approaches to achieve this.

a. Add a GitHub action to check formatting with a specific black version.
b. I can use the *rickstaa/action-black action*, which provides more customization options, such as specifying additional Black arguments or combining it with other actions like *reviewdog/action-suggester* to annotate changes.This setup allows you to check the code and see a diff of the changes that Black would make without applying them.
c. Another option is to use the *datadog/action-py-black-formatter* action, which can be combined with *peter-evans/create-pull-request* to automatically create a pull request with the formatted changes. (But I am guessing that this might be an overkill)
d. As another alternative, if you want to format only the files that were changed in a pull request, I can use the

rgasper/python-black-pull-request-action to automatically check formatting, change the files if the formatting needs to be changed and raise a pull request.

B. **Adding a GitHub Action to generate a PIP package and push to PyPI automatically.**
So the idea is to automatically create a GitHub release everytime we add a version tag in the repository. And once the release is done on GitHub, we can automatically publish that new package version on PyPi using a GitHub Action. Which can later be installed using pip or poetry or any other python package manager.

The GitHub Action will get triggered when the version tag follows PEP440 naming convention. Will verify using regular expressions given below.

```
on:
  push:
    tags:
      - "[0-9]+.[0-9]+.[0-9]+"
      - "[0-9]+.[0-9]+.[0-9]+a[0-9]+"
      - "[0-9]+.[0-9]+.[0-9]+b[0-9]+"
      - "[0-9]+.[0-9]+.[0-9]+rc[0-9]+"
```

Jobs in the GitHub Action :

- Extract tag and Details
- Check PyPi
  - Fetch information from PyPI
  - Compare versions and exit if not newer
- Set up and Build
  - Set up Python
  - Install Poetry
  - Set project version with Poetry
  - Install dependencies
  - Build source and wheel distribution
  - Upload artifacts

- Upload release to PyPI
    - Download artifacts
    - Publish distribution to PyPI
- Create GitHub Release
    - Checkout Code
    - Download artifacts
    - Create GitHub Release

4. **Adding tutorials for image and LiDAR segmentation evaluation. Also for the newly added object detection model evaluation.**
   I plan to add Jupyter Notebooks with tutorials on how to uptake and use the tool, with different models and datasets. Along with tutorials on the newly added object detection evaluation, I also plan to add LiDAR segmentation, dataset conversion/merging and image segmentation with tensorflow.

Timeline

Community Bonding Period	
May 8 - June 1	• I would try to get a better grasp of the codebase, brush up on some computer vision concepts & Python essentials. • As I am not too well versed with building a GUI in python (I am pretty comfortable with Javascript and React though), I would like to surf through some tutorials for the same. • I plan on starting my coding for the project during this time to save some time for later. • Keep contributing to DetectionMetrics github repository in general and be actively present in the discussions/issues.
Coding period begins	

June 2 - June 15 **Week 1-2**	● Defining the ImageDetectionModel and ImageDetectionDataset classes. ● Restoring compatibility with COCO style dataset ● Implementing object detection evaluation for COCO dataset and Mask R-CNN model. ● Implementing mAP and IoU for the same.
June 16 - June 29 **Week 3-4**	● Restoring compatibility with Pascal VOC-style datasets. ● Restoring compatibility with YOLO - OpenCV framework. ● Thoroughly testing the newly added functionalities.
June 30 - July 13 **Week 5-6**	● Finish any pending work from the first four weeks, basically finish Object Detection Evaluation completely. ● Adding a few tutorials for the same. ● Including test cases related to any newly added functions in utils. ● Design the basic GUI.
Midterm Evaluations	
July 14 - July 27 **Week 7-8**	● Get the GUI approved by the mentors ● Finish building the GUI. ● Thoroughly test the built GUI.
July 28 - Aug 10 **Week 9-10**	● Add a GitHub Action to generate a PIP package and push to PyPI automatically. ● Moving the docs to *gh_pages* branch ● Add more tutorials based on the requirement for image and LiDAR segmentation evaluation.
Aug 11 - Aug 24 **Week 11-12**	● Thoroughly test all the functionalities of DetectionMetrics. ● Document all the work that I could accomplish. ● Identify any kind of further potential enhancements that can be done on the tool and create respective issues in the repository. ● Leaving last week as a buffer to complete any pending tasks or address the issues that might come up last minute.

Final Evaluations	
Post GSOC	● I see the potential to document and submit to publish this work in a journal similar to sensors. I would be thrilled to work on the same. ● I am willing to further work on restoring the compatibility for other popular object detection frameworks and datasets.

Contributions Pre-Gsoc

1. **Added GPU acceleration using MPS in MacOS (#279)**
   DetectionMetrics was failing to load torch models in MacOS and was only using CUDA for gpu acceleration. I have modified the condition where MPS is used in the absence of CUDA, only in the absence of MPS and CUDA, CPU is used. Also added map_location argument in torch.jit.load() to avoid errors due to device mismatch. This change improves cross-platform compatibility of DetectionMetrics.
2. **Identified and fixed errors in utils/conversion.py(#283)**
   a. In the function *hex_to_rgb* I have added the code to handle invalid hex codes of wrong length or invalid characters.
   b. In the function *get_ontology_conversion_lut* : Ignores_classes was not getting implemented when ontology_translation is provided, fixed that as well.
3. **Added tests for conversion.py (#283)**
   Added tests that are compatible to run with Pytest to verify positive and negative cases for all the functions in conversion.py.
4. **Added github actions for automatically running tests.(#294)**

Motivation behind participation in Gsoc'25

- **General :** I love building applications and solving complex problems.

While my full-time role has deepened my expertise in certain areas, I want to expand my skill set by working on diverse projects and learning new technologies. GSoC provides the perfect environment to step out of my comfort zone, gain hands-on experience, and upskill through mentorship while collaborating with experienced developers.

- **DetectionMetrics :**
  While application development excites me, I am more passionate about AI/ML and have been shifting my focus to this domain since last year. I see GSoC 2025 as a gateway to strengthening my profile and gaining hands-on experience in the domain.

  I also plan to pursue higher studies abroad and will be applying for Fall 2026 admissions. This project aligns perfectly with my interests and offers a strong foundation in the field, which I believe will enhance my chances of securing admission.

  Over the past six weeks, I have been actively exploring and contributing to the repository's issues, which has given me a solid understanding of the codebase. With DetectionMetrics in its early development stage, I believe my contributions can make a meaningful impact and lay the groundwork for long-term involvement beyond GSoC.

Personal Information

Studies:

- **What is your School and degree?**

Degree	:	Bachelor of Technology in Electronics and Communication Engineering
College	:	National Institute Of Technology, Calicut
Graduation	:	2023

- **Would your application contribute to your ongoing studies/degree? If so, how?**
  N/A

Programming background:

- **Computing experience: operating systems you use on a daily basis, known programming languages, hardware, etc.**
  I use MacOS and Linux on a regular basis. I am very much comfortable with python, C++, JavaScript and Java.
- **Robot or Computer Vision programming experience.**
  Although my major was in Electronics, I opted for several computer science electives in college that align with my career goals and interests. In my fourth year, I took a Computer Vision course, where I implemented:
  - Image-to-Image Translation with cGAN using PyTorch and Torchvision
  - Emotion Detection Model with DCNN using TensorFlow/Keras

  These projects gave me hands-on experience in deep learning and equipped me with the relevant skills required for this project.
- **Other software programming skills.**
  I have developed strong software programming skills through my academic projects, coursework, and nearly two years of professional experience.
  - Programming Languages: C, C++, Python, Java, Javascript, SQL, PHP
  - Web Tools: HTML, CSS, React, React Native, Express.js, Node.js, MYSQL, MongoDB
  - Relevant Coursework: Natural Language Processing, Computer Vision, Artificial Intelligence, Advanced Algorithms and Data Structures, Operating Systems, Computer Networks and Databases.
  - Developer Tools: Git, Docker, Anaconda, Google Colab, Postman, Unity

GSOC Participation:

- **Have you participated in GSoC before?**
  No

- **How many times, which year, which project?**
  N/A
- **Have you applied but were not selected? When?**
  N/A
- **Have you submitted/will you submit another proposal for GSoC 2025 to a different org?**
  Nope

Commitments:

I am committed to dedicating **30 - 35 hours per week** to my project, with the flexibility to increase my input as needed. Should personal or miscellaneous commitments arise, I will notify my mentor in advance, providing a timeframe for any additional hours required to meet deadlines.

Working hours (open to adjustments according to your work timings):

- 08:00 - 11:00 IST
- 18:00 - 00:00 IST

I plan to finish the Gsoc project with almost two weeks of buffer. This will give me more time to address anything left or give more importance to any issue that demands more time and debugging. This extra time will also enable me to thoroughly test and document my work.

I would also like to take this opportunity to express my gratitude to David Pascual-Hernández (@dpascualhe) and Sergio Paniego Blanco (@sergiopaniego) for provided me with tremendous support and guidance during the past few months. Truly grateful for their invaluable contributions in reviewing and merging my PRs and providing assistance whenever I encountered difficulties.