Module Interface Specification for ProgName

Team #, Team Name
Student 1 name
Student 2 name
Student 3 name
Student 4 name

March 21, 2025

1 Revision History

Date	Version	Notes
Date 1	1.0	Notes
Date 2	1.1	Notes

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at [give url —SS]

[Also add any additional symbols, abbreviations or acronyms —SS]

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **ii**

3 Introduction **1**

4 Notation **1**

5 Module Decomposition **1**

6 MIS of Training Module **3**
- 6.1 Module . . . 3
- 6.2 Uses . . . 3
- 6.3 Syntax . . . 3
  - 6.3.1 Exported Constants . . . 3
  - 6.3.2 Exported Access Programs . . . 3
- 6.4 Semantics . . . 3
  - 6.4.1 State Variables . . . 3
  - 6.4.2 Environment Variables . . . 3
  - 6.4.3 Assumptions . . . 3
  - 6.4.4 Access Routine Semantics . . . 3
  - 6.4.5 Local Functions . . . 4

7 MIS of Inference Module **5**
- 7.1 Module . . . 5
- 7.2 Uses . . . 5
- 7.3 Syntax . . . 5
  - 7.3.1 Exported Constants . . . 5
  - 7.3.2 Exported Access Programs . . . 5
- 7.4 Semantics . . . 5
  - 7.4.1 State Variables . . . 5
  - 7.4.2 Environment Variables . . . 5
  - 7.4.3 Assumptions . . . 5
  - 7.4.4 Access Routine Semantics . . . 5
  - 7.4.5 Local Functions . . . 5

8 MIS of Evaluation Module **7**
- 8.1 Module . . . 7
- 8.2 Uses . . . 7
- 8.3 Syntax . . . 7
  - 8.3.1 Exported Constants . . . 7
  - 8.3.2 Exported Access Programs . . . 7

8.4 Semantics . . . 7
8.4.1 State Variables . . . 7
8.4.2 Environment Variables . . . 7
8.4.3 Assumptions . . . 7
8.4.4 Access Routine Semantics . . . 7
8.4.5 Local Functions . . . 7

9 MIS of Model Module **8**
9.1 Module . . . 8
9.2 Uses . . . 8
9.3 Syntax . . . 8
9.3.1 Exported Constants . . . 8
9.3.2 Exported Access Programs . . . 8
9.4 Semantics . . . 8
9.4.1 State Variables . . . 8
9.4.2 Environment Variables . . . 8
9.4.3 Assumptions . . . 8
9.4.4 Access Routine Semantics . . . 8
9.4.5 Local Functions . . . 9

10 MIS of Loss Module **10**
10.1 Module . . . 10
10.2 Uses . . . 10
10.3 Syntax . . . 10
10.3.1 Exported Constants . . . 10
10.3.2 Exported Access Programs . . . 10
10.4 Semantics . . . 10
10.4.1 State Variables . . . 10
10.4.2 Environment Variables . . . 10
10.4.3 Assumptions . . . 10
10.4.4 Access Routine Semantics . . . 10
10.4.5 Local Functions . . . 11

11 MIS of Optimizer Module **12**
11.1 Module . . . 12
11.2 Uses . . . 12
11.3 Syntax . . . 12
11.3.1 Exported Constants . . . 12
11.3.2 Exported Access Programs . . . 12
11.4 Semantics . . . 12
11.4.1 State Variables . . . 12
11.4.2 Environment Variables . . . 12
11.4.3 Assumptions . . . 12

11.4.4 Access Routine Semantics . . . 12
11.4.5 Local Functions . . . 12

12 MIS of Plotting Module **13**
12.1 Module . . . 13
12.2 Uses . . . 13
12.3 Syntax . . . 13
12.3.1 Exported Constants . . . 13
12.3.2 Exported Access Programs . . . 13
12.4 Semantics . . . 13
12.4.1 State Variables . . . 13
12.4.2 Environment Variables . . . 13
12.4.3 Assumptions . . . 13
12.4.4 Access Routine Semantics . . . 13
12.4.5 Local Functions . . . 13

13 MIS of Checkpoint Module **14**
13.1 Module . . . 14
13.2 Uses . . . 14
13.3 Syntax . . . 14
13.3.1 Exported Constants . . . 14
13.3.2 Exported Access Programs . . . 14
13.4 Semantics . . . 14
13.4.1 State Variables . . . 14
13.4.2 Environment Variables . . . 14
13.4.3 Assumptions . . . 14
13.4.4 Access Routine Semantics . . . 14
13.4.5 Local Functions . . . 15

14 MIS of Data Module **16**
14.1 Module . . . 16
14.2 Uses . . . 16
14.3 Syntax . . . 16
14.3.1 Exported Constants . . . 16
14.3.2 Exported Access Programs . . . 16
14.4 Semantics . . . 16
14.4.1 State Variables . . . 16
14.4.2 Environment Variables . . . 16
14.4.3 Assumptions . . . 16
14.4.4 Access Routine Semantics . . . 16
14.4.5 Local Functions . . . 17

15 MIS of Equivariant Layers Module **18**
15.1 Module . . . 18
15.2 Uses . . . 18
15.3 Syntax . . . 18
15.3.1 Exported Constants . . . 18
15.3.2 Exported Access Programs . . . 18
15.4 Semantics . . . 18
15.4.1 State Variables . . . 18
15.4.2 Environment Variables . . . 18
15.4.3 Assumptions . . . 18
15.4.4 Access Routine Semantics . . . 18
15.4.5 Local Functions . . . 19

16 MIS of OpenPCDet Layers Module **20**

17 MIS of PyTorch Module **21**

18 MIS of Config Module **22**
18.1 Module . . . 22
18.2 Uses . . . 22
18.3 Syntax . . . 22
18.3.1 Exported Constants . . . 22
18.3.2 Exported Access Programs . . . 22
18.4 Semantics . . . 22
18.4.1 State Variables . . . 22
18.4.2 Environment Variables . . . 22
18.4.3 Assumptions . . . 22
18.4.4 Access Routine Semantics . . . 22
18.4.5 Local Functions . . . 23

19 MIS of Data Processing Module **24**
19.1 Module . . . 24
19.2 Uses . . . 24
19.3 Syntax . . . 24
19.3.1 Exported Constants . . . 24
19.3.2 Exported Access Programs . . . 24
19.4 Semantics . . . 24
19.4.1 State Variables . . . 24
19.4.2 Environment Variables . . . 24
19.4.3 Assumptions . . . 24
19.4.4 Access Routine Semantics . . . 25
19.4.5 Local Functions . . . 25

20 References **25**

21 Appendix **27**

3 Introduction

The following document details the Module Interface Specifications for [Fill in your project name and description —SS]

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at .... [provide the url for your repo —SS]

4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1|c_2 \Rightarrow r_2|...|c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by ProgName.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)

The specification of ProgName uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, ProgName uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Config Loading Module Data Loading Module Model Module Checkpointing Module Training Module Inference Module Loss Module Evaluation Module Optimization Module Data Processing Module
Software Decision Module	Plotting Module PyTorch Module Logging Module

Table 1: Module Hierarchy

6 MIS of Training Module

6.1 Module

6.2 Uses

1. PyTorch Dataloader (PyTorch Documentation)
2. PyTorch Optimizer (PyTorch Documentation)
3. PyTorch Loss (PyTorch Documentation)
4. Model
5. Checkpoint
6. Logger

6.3 Syntax

6.3.1 Exported Constants

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
train	PyTorch Dataloader (Eval), PyTorch Dataloader (Training), PyTorch Optimizer, Model, Loss	-	-

6.4 Semantics

6.4.1 State Variables

6.4.2 Environment Variables

6.4.3 Assumptions

6.4.4 Access Routine Semantics

train():

- transition: N/A
- output: N/A
- exception: N/A

6.4.5 Local Functions

1. **trainEpoch**(trainDataloader, optimizer, model, loss):
   (a) **Input:** The dataloader of training data, the model to be optimized, the ADAM optimizer, and the loss function for backpropagation.
   (b) **Method Description:** This method optimizes the model over the input training data to lower the given loss that the model incurs. Statistics describing the performance of the model (loss) over the training set are logged using the global logger.
   (c) **Output:** N/A
2. **evalModel**(evalDataloader, model):
   (a) **Input:** The dataloader of evaluation data and the model to be evaluated.
   (b) **Method Description:** This method collects the mAP metrics obtained from applying the current model to the evaluation dataset. These metrics are logged using the global logger. The current state of the model is additionally saved using the checkpointing system.
   (c) **Output:** N/A

7 MIS of Inference Module

7.1 Module

7.2 Uses

- PyTorch Dataloader (PyTorch Documentation)
- Checkpoint
- Model
- Logger

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
inference	Images, LiDAR, Bounding Boxes, Model, Checkpoint	-	-

7.4 Semantics

7.4.1 State Variables

7.4.2 Environment Variables

7.4.3 Assumptions

7.4.4 Access Routine Semantics

inference():

- transition: N/A
- output: N/A
- exception: N/A

7.4.5 Local Functions

1. **getPredBoundingBoxes**(Images, LiDAR, Model):
   (a) **Input:** The model to be used for inference in addition to the input data (multi-view camera images and a LiDAR pointcloud).

(b) **Method Description:** This method uses the model to get a set of predicted bounding boxes for the given input data.

(c) **Output:** The predicted bounding boxes.

2. **plotBoundingBoxes**(LiDAR, gtBoundingBoxes, predBoundingBoxes):

   (a) **Input:** The input LiDAR pointcloud alongside the ground truth and predicted bounding boxes for this scene.

   (b) **Method Description:** This method displays a 3D plot of the ground truth and predicted bounding boxes on the input LiDAR pointcloud to allow for visual inspection (using the plotting module).

   (c) **Output:** N/A

8 MIS of Evaluation Module

8.1 Module

8.2 Uses

- PyTorch Dataloader (PyTorch Documentation)
- Model
- Checkpoint

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
eval	PyTorch Dataloader (Eval), Model, Checkpoint (Optional)	Evaluation Metrics	-

8.4 Semantics

8.4.1 State Variables

8.4.2 Environment Variables

8.4.3 Assumptions

8.4.4 Access Routine Semantics

eval():

- transition: N/A
- output: This function will run the input model over the evaluation loader and collect the mAP statistics over this set of data. These statistics will then be returned as output. If a checkpoint is passed in, the model will first load the given checkpoint.
- exception: N/A

8.4.5 Local Functions

9 MIS of Model Module

9.1 Module

9.2 Uses

- Is a subclass of PyTorch Module (PyTorch Documentation)
- Equivariant Layers
- OpenPCDet Layers
- Configuration

9.3 Syntax

9.3.1 Exported Constants

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
forward	Images, LiDAR	Predicted Bounding Boxes	-

9.4 Semantics

9.4.1 State Variables

- pcdetLayers
- eqLayers

9.4.2 Environment Variables

9.4.3 Assumptions

9.4.4 Access Routine Semantics

init():

- transition: This function will load the model-related configuration from the global configuration module and will then create the OpenPCDet and Equivariant Layers accordingly. References to these layers will be stored in the pcdetLayers and eqLayers state variables.
- output: N/A

- exception: N/A

forward():

- transition: N/A
- output: This function will run the input data (LiDAR and Images) through the OpenPCDet and Equivariant layers to produce a set of predicted bounding boxes. These predicted bounding boxes will then be returned.
- exception: N/A

9.4.5 Local Functions

10 MIS of Loss Module

10.1 Module

10.2 Uses

- Is a subclass of PyTorch Loss (PyTorch Documentation)
- Configuration

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
forward	Predicted Bounding Boxes, Ground Truth Bounding Boxes	Loss Value	-

10.4 Semantics

10.4.1 State Variables

- hyperparameters

10.4.2 Environment Variables

10.4.3 Assumptions

10.4.4 Access Routine Semantics

init():

- transition: This method will set the loss module's hyperparamters according to the loss-related configuration from the global configuration module.
- output: N/A
- exception: N/A

forward():

- transition: N/A
- output: This method will use the predicted and ground truth bounding boxes to regress and output a loss value.
- exception: N/A

10.4.5 Local Functions

11 MIS of Optimizer Module

11.1 Module

11.2 Uses

- Is a wrapper around the PyTorch ADAM Optimizer (PyTorch Documentation)
- PyTorch Parameter (PyTorch Documentation)
- Configuration

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	Model Parameters	-	-

11.4 Semantics

11.4.1 State Variables

- hyperparameters
- modelParameters

11.4.2 Environment Variables

11.4.3 Assumptions

11.4.4 Access Routine Semantics

init():

- transition: This method will set the loss module's hyperparameters according to the optimizer-related configuration from the global configuration module. It will also save a reference to passed in model parameters in the modelParameters state variable.
- output: N/A
- exception: N/A

11.4.5 Local Functions

12 MIS of Plotting Module

12.1 Module

12.2 Uses

- Open3D Oriented Bounding Boxes (Open3D Documentation)
- Open3D Pointclouds (Open3D Documentation)
- Open3D in general (Open3D Documentation)

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
plot	Ground Truth Bounding Boxes, Predicted Bounding Boxes, LiDAR	-	-

12.4 Semantics

12.4.1 State Variables

12.4.2 Environment Variables

- Computer Screen

12.4.3 Assumptions

12.4.4 Access Routine Semantics

plot():

- transition: The LiDAR pointcloud is first converted into an Open3D Pointcloud. Then, the bounding boxes are converted into the Open3d OrientedBoundingBox type. Finally, these are all visualized together in a single 3D plot using Open3D's draw_geometries function. This outputs a plot and thus updates the computer screen environment variable.
- output: N/A
- exception: N/A

12.4.5 Local Functions

13 MIS of Checkpoint Module

13.1 Module

13.2 Uses

- PyTorch Model (PyTorch Documentation)
- PyTorch State Dictionary (PyTorch Documentation)

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
saveCheckpoint	Model, Checkpoint Path	-	-
loadCheckpoint	Model, Checkpoint Path	-	-

13.4 Semantics

13.4.1 State Variables

13.4.2 Environment Variables

- File System

13.4.3 Assumptions

13.4.4 Access Routine Semantics

saveCheckpoint():

- transition: The input model's state dictionary is saved to a pytorch tensor file at the input path.
- output: N/A
- exception: N/A

loadCheckpoint():

- transition: The input model's parameters are set using the state dictionary saved at the input path.
- output: N/A
- exception: N/A

13.4.5 Local Functions

14 MIS of Data Module

14.1 Module

14.2 Uses

1. PyTorch Dataset and Dataloader (PyTorch Documentation)
2. Configuration

14.3 Syntax

14.3.1 Exported Constants

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	isEval Boolean	-	-
getDataloader	-	PyTorch Dataloader	-

14.4 Semantics

14.4.1 State Variables

- dataset

14.4.2 Environment Variables

1. File System

14.4.3 Assumptions

14.4.4 Access Routine Semantics

init():

- transition: This function will load the data-related configuration from the global configuration module and will then create the dataset accordingly. This dataset will either be the NuScenes (Caesar et al. (2020)) or the Waymo (Sun et al. (2020)) dataset for now and will be stored in the dataset state variable. If isEval is false, the training portion of the dataset will be loaded. If isEval is true, the evaluation portion of the dataset will be loaded.
- output: N/A
- exception: N/A

getDataloader():

- transition: N/A
- output: This function will use the dataset state variable to create a PyTorch Dataloader which will then be returned.
- exception: N/A

14.4.5 Local Functions

15 MIS of Equivariant Layers Module

This module is meant to represent the equivariant layers I will be adding into the OpenPCDet repository for the novel part of my project. To provide context for the reader, equivariant layers refer to neural network layers that preserve input transformations when generating output features. For example, a rotated image of a cat will result in similarly rotated output features (when compared to features generated from an upright image of a cat). This work by Cohen et Al. (Cohen and Welling (2016a)) more clearly explains this concept.

This module will encapsulate two different types of equivariant layers. The first of these is steerable kernel equivariant CNNs (Cohen and Welling (2016b)) extended to include both 2D and 3D symmetries using the escnn repository (Cesa et al. (2022)). The second of these is equivariant transformer networks (Tai et al. (2019)). Existing OpenPCDet layers can essentially then be directly swapped out for these equivariant alternatives.

15.1 Module

15.2 Uses

- PyTorch Parameter (PyTorch Documentation)
- Steerable CNNs (Cesa et al. (2022))
- Equivariant Transformers (Tai et al. (2019))

15.3 Syntax

15.3.1 Exported Constants

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
forward	-	-	-

15.4 Semantics

15.4.1 State Variables

- layerParameters

15.4.2 Environment Variables

15.4.3 Assumptions

15.4.4 Access Routine Semantics

init():

- transition: The parameters for the layer are randomly initialized according to a uniform distribution around 0 (saved in the layerParameters state variable).
- output: N/A
- exception: N/A

forward():

- output:
  - **For the CNN case:** Since the CNN case is simpler, the math relating to it can be shown here itself. The result y of this formula is returned:

    $y[m,n](k,x) = \sum_{i=-w}^{w} \sum_{j=-h}^{h} k[w+i, h+j]x[m+i, n+j] \rightarrow y(k,x) = k * x$ such that if $x_2 = gx_1$ for some g in the group of considered transformations, $y(x_2) = \rho(g)y(x_1)$ for a single function ρ.

  - **For the Transformer case:** The math here is more complicated and so I will simply refer the reader to the paper by Tai et. al (Tai et al. (2019)). The output of this layer is returned. Generally though for attention function f, the following can be written similar to the CNN case: $x_2 = gx_1 \rightarrow y(k, x_2) = \rho(g)y(k, x_1)$

15.4.5 Local Functions

16 MIS of OpenPCDet Layers Module

This module is simply used to represent the existing layers in the OpenPCDet library (Team (2020)). Rather than highlight specific relevant layer types and modules, I would encourage the reader to read through the BEVFusion (Liang et al. (2022)) configuration file (BEVFusion Config File) and associated code sections (Model-Related OpenPCDet Code).

17 MIS of PyTorch Module

This module is simply used to represent the PyTorch library referenced in the other modules. The main list of components that are used from this library is as follows:

- PyTorch Module (PyTorch Documentation)
- PyTorch Dataloader (PyTorch Documentation)
- PyTorch Optimizer (PyTorch Documentation)
- PyTorch Loss (PyTorch Documentation)
- PyTorch State Dictionary (PyTorch Documentation)
- PyTorch Parameter (PyTorch Documentation)

18 MIS of Config Module

18.1 Module

18.2 Uses

18.3 Syntax

18.3.1 Exported Constants

- CONFIG_PATH

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
getModelConfig	-	-	-
getDataConfig	-	-	-
getOptimConfig	-	-	-
getLossConfig	-	-	-

18.4 Semantics

18.4.1 State Variables

- modelConfig
- dataConfig
- optimConfig
- lossConfig

18.4.2 Environment Variables

- File System

18.4.3 Assumptions

18.4.4 Access Routine Semantics

init():

- transition: This method loads the yaml configuration file stored at the predefined constant path CONFIG_PATH. Then, the dictionary loaded from this JSON file is split into model, data, optimizer, and loss sections. Each of these is saved to the corresponding state variable for this module.

- output: N/A
- exception: N/A

getModelConfig():

- transition: N/A
- output: This method returns the configuration parameters for the model module.
- exception: N/A

getDataConfig():

- transition: N/A
- output: This method returns the configuration parameters for the data module.
- exception: N/A

getOptimConfig():

- transition: N/A
- output: This method returns the configuration parameters for the optimizer module.
- exception: N/A

getLossConfig():

- transition: N/A
- output: This method returns the configuration parameters for the loss module.
- exception: N/A

18.4.5 Local Functions

19 MIS of Data Processing Module

19.1 Module

19.2 Uses

-

19.3 Syntax

19.3.1 Exported Constants

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
process	Image Path, LiDAR Path	Images, LiDAR	-

19.4 Semantics

19.4.1 State Variables

19.4.2 Environment Variables

19.4.3 Assumptions

19.4.4 Access Routine Semantics

process():

- transition: N/A
- output: Loads the multiview images and LiDAR pointcloud from their respective files and converts them into a standard PyTorch Tensor format that can be used by the model. The Tensor form of the multiview images and the LiDAR pointcloud are returned.
- exception: N/A

19.4.5 Local Functions

20 References

References

Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

Gabriele Cesa, Leon Lang, and Maurice Weiler. A program to build E(N)-equivariant steerable CNNs. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=WE4qe9xlnQw.

Taco Cohen and Max Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016a.

Taco S Cohen and Max Welling. Steerable cnns. *arXiv preprint arXiv:1612.08498*, 2016b.

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

Tingting Liang, Hongwei Xie, Kaicheng Yu, Zhongyu Xia, Zhiwei Lin, Yongtao Wang, Tao Tang, Bing Wang, and Zhi Tang. Bevfusion: A simple and robust lidar-camera fusion framework. *Advances in Neural Information Processing Systems*, 35:10421–10434, 2022.

Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020.

Kai Sheng Tai, Peter Bailis, and Gregory Valiant. Equivariant Transformer Networks. In *International Conference on Machine Learning*, 2019.

OpenPCDet Development Team. Openpcdet: An open-source toolbox for 3d object detection from point clouds. https://github.com/open-mmlab/OpenPCDet, 2020.

21 Appendix

[Extra information if required —SS]

Appendix — Reflection

[Not required for CAS 741 projects —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. Which of your design decisions stemmed from speaking to your client(s) or a proxy (e.g. your peers, stakeholders, potential users)? For those that were not, why, and where did they come from?

4. While creating the design doc, what parts of your other documents (e.g. requirements, hazard analysis, etc), it any, needed to be changed, and why?

5. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

6. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)