LiDAR and Deep Learning for Object Detection and Environment Description for the Visually Impaired

Sooa Han, *(301595561)*, Alex Cao, *(301339414)*

***Abstract*—The objective of this project is to develop a LiDAR based system that assists individuals who are visually impaired. Through this, it allows for enhanced navigation and environmental awareness through real time object and surrounding detection. With the advancement of AI and sensors based technologies, the limitations of traditional mobility aids, like minimum spatial awareness and lack of environmental descriptions can be vastly reduced. This project attempts to propose an assistive system that integrates LiDAR sensors with a deep learning based visual aid to offer real time obstacle detection and scene interpretation. Unlike most existing solutions that solely rely on camera-based vision and large language models, LiDAR is capable of producing high-speed and depth precision capabilities to ensure an accurate 3D representation of the surroundings. Along with the supplementary visual module, even complex scenarios where colours and textures play an important role can also be taken care of. We integrated an audio feature that outputs additional labels such as object direction, confidence score, and distance alongside predictions. This enhancement allows users to navigate with greater confidence through audio feedback.**

***Index Terms*—Assistive technology, object detection, LiDAR, computer vision, deep learning, YOLO, CoreML, TensorFlow Lite, real-time processing, spatial awareness, ARKit, object tracking, accessibility, mobile computing, iOS applications**

I. Introduction

THERE exist fundamental challenges for visually impaired individuals to navigate around their surroundings. According to the World Health Organization (WHO), over 2.2 billion people globally have some sort of vision impairment where many of which have difficulties to mobilize independently [4]. Traditional navigation aids, such as white cane or guide dogs, have been assisting individuals with visual impairments but exist limitations to provide real-time spatial information.

With the recent advancements in AI and embedded systems, there has been an increase in the number of developments in intelligent navigation aids. For example, recently at CES 2025, a company with the name WeWalk introduced WeWalk Smart Cane 2, a white cane that is integrated with a multimodal large language model to provide feedback of their surroundings [5]. Although systems like these have shown promise, they suffer from several key limitations. To start off, camera based systems perform poorly in low light or overexposed conditions leading to inaccurate results. Secondly, with only one camera setup, it is difficult to determine depth perception with accuracy. Lastly, real time video processing and large language models require a large amount of computational power to run. This often leads to expensive products, slow feedback from llm and increased latency created by cloud services.

Manuscript received April 5, 2025

Apple has been including LiDAR technology in certain models of their iPhone and iPad since 2020 [6]. Unlike cameras, LiDAR technology is able to capture precise 3D depth information in real-time. This ensures different lighting conditions will not affect LiDAR sensors, providing a more reliable detection. Furthermore, LiDAR sensors can scan an entire scene with millions of depth data points in milliseconds, eliminating the need for complex computational steps like image segmentation, feature extraction, and depth estimation that are required for vision based systems.

Despite LiDAR potential, the adoption of LiDAR technology in mainstream apps has been relatively slow. Many apps finds difficult to utilizing such technology, unable to take advantage of its precision and real-time depth mapping features. This slow adoption is likely due to its complexity to integrate into their existing workflows.

The goal of this paper is to understand PointPillar based architecture and attempts to improve its performance through finetuning process over different datasets [3].

II. Related Work

Currently, there exist several LiDAR detectors methods that are categorized based on their way of processing point clouds. Each paradigm has its own advantages and disadvantages when it comes to accuracy and efficiency. Through researching, PointPillars, a method that begins with discretizing point clouds into pillars on the ground plane and utilizing a 2D CNN backbones on the bird's eye view seems to be an ideal architecture for our project [3]. This is because, in their paper, they got 62 Hz inference without sacrificing accuracy on the KITTI dataset [1]. Furthermore, researchers had been able to run it on an embedded GPU at $\sim$40 LiDAR frames/s [3].

Voxel Grid (3D Convolutions): A type of method that first divides space up into a 3D grid of voxels followed by training its voxel wise features with 3D CNNs [5]. Examples of this are VoxelNet , and SECOND [5] [8]. Unlike the PointPillar architecture we are using, this method uses 3D grid instead of 2D convolutions method, therefore, the general performance of Voxel Grid Encoders based model only achieves an average FPS of 4-5 on high end GPUs [5].

Point-Based Networks: took an approach where it operates directly on raw point sets through point wise MLPs, meaning instead of a grid strategy, it uses a sample/aggregate strategy [9]. Some examples of this method are PointRCNN and 3SDDS [9] [10]. Due to the point based operations approach, especially during grouping and sampling points, when comparing to the PointPillar architecture we are using, it requires

high computational power and memory resulting in an only ∼10 FPS inference on GPU [9].

You Only Look Once (YOLO3D) is an extension of a 2D detector that lacks native support for LiDAR, requiring significant adaptation (e.g., depth projection, calibration, and BEV encoding) to handle point clouds [11]. While YOLO3D may be faster in image-based scenarios, it struggles with depth accuracy, small objects, and long-range detection [12]. PointPillars, with its pillar-based voxelization and BEV representation, provides a more reliable and scalable solution for safety-critical applications.

III. Methodology

Our 3D object detection pipeline leverages the **PointPillars** architecture to efficiently process LiDAR point clouds by transforming them into a 2D pseudo-image representation. The method consists of four key stages: (1) pillar encoding, (2) pseudo-image generation, (3) feature extraction with a 2D CNN backbone, and (4) multi-task detection head.

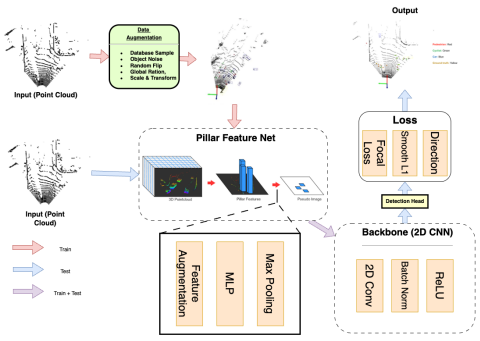


Fig. 1: PointPillar workflow

PointPillars Pipeline Overview

A. Pillar Encoding

Given a LiDAR point cloud with N points represented as $\mathbf{P} \in \mathbb{R}^{N\times 4}$ (where each point has coordinates (x, y, z) and reflectance r), we first discretize the 3D space into a grid of pillars (vertical voxels). Each pillar aggregates all points within its (x, y) bounds.

Feature Augmentation: PointPillars enhances each LiDAR point by computing its spatial offsets relative to the pillar's center and mean position. These offsets provide crucial local context about point distributions within each pillar, helping the model better understand geometric relationships while maintaining computational efficiency. The augmented features combine raw coordinates with these spatial relationships for richer representation.

MLP Processing: The augmented features are processed through a lightweight MLP consisting of linear layers, ReLU activations, and batch normalization. Implemented using efficient 1D convolutions, this step projects low-level point features into a higher-dimensional space suitable for object detection while ensuring stable training through normalization.

Max Pooling: To handle variable point densities, PointPillars applies max pooling across all points in each pillar, extracting only the most significant features. This operation produces fixed-size pillar representations that are invariant to point order and count, while empty pillars are zero-padded to maintain consistent dimensionality for subsequent CNN processing. The result is an efficient pseudo-image representation preserving the most relevant spatial information.

B. Pseudo-Image Generation

Pillar features $\mathbf{F}$ are scattered back to their (x, y) grid locations to form a 2D pseudo-image $\mathbf{I} \in \mathbb{R}^{H\times W\times D}$, where H and W are the grid dimensions, and D is the feature dimension. Empty pillars are zero-padded.

C. Backbone Network

The pseudo-image is processed by a lightweight 2D CNN backbone to extract multi-scale features. The architecture consists of sequential convolutional blocks, where each block contains:

- A 3×3 convolution with stride 2 for downsampling
- Batch normalization for stable gradient flow
- ReLU activation for nonlinear feature learning

The output feature map encodes object locations and shapes. The final output is a high-level feature representation of the scene, optimized for detection.

D. Detection Head and Loss

The detection head comprises three parallel branches, each responsible for a specific task:

Classification Branch: Predicts the probability of object classes for each location. It uses **Focal Loss** to address class imbalance by downweighting well-classified examples. This ensures the model focuses on harder-to-classify instances.

Regression Branch: Estimates 3D bounding box parameters (center coordinates, dimensions, and yaw angle) for detected objects. It employs **Smooth L1 Loss** to minimize the difference between predicted and ground-truth box parameters, ensuring smooth and accurate regression.

Direction Branch: Predicts discrete orientation bins (e.g., 0° or 90°) for objects. It uses **Cross-Entropy Loss** to measure the discrepancy between predicted and true orientations.

E. Post-Processing

The final outputs are filtered using a non-maximum suppression (NMS) algorithm to remove duplicate detections. Confidence thresholds are applied to select the most probable objects.

Training and Finetuning

F. Dataset Preparation

We used two primary datasets:

- KITTI: Downloaded from *The KITTI Vision Benchmark Suite*. Contains 29GB of Velodyne LiDAR scans (point clouds with x, y, z, r coordinates), 12GB of camera images, and 16MB of calibration files (including $P2$ for

3D→ 2D projection and $Tr_velo_to_cam$ for LiDAR-to-camera coordinate transformation). Annotations provide 3D bounding boxes for cars, pedestrians, and cyclists.
- Lyft: Downloaded from Kaggle (125.79GB in nuScenes format) and converted to KITTI format for compatibility. This included reformatting LiDAR data, annotations, and calibration matrices to match KITTI's structure.

We converted Lyft (nuScenes format) dataset to KITTI format to maintain consistency with pretrained weights. The Lyft class mapping is as follows:

$$\begin{aligned} \text{car} &\rightarrow \text{Car} \\ \text{truck} &\rightarrow \text{Car} \\ \text{bus} &\rightarrow \text{Car} \\ \text{bicycle} &\rightarrow \text{Cyclist} \\ \text{motorcycle} &\rightarrow \text{Cyclist} \\ \text{pedestrian} &\rightarrow \text{Pedestrian} \end{aligned}$$

Why LiDAR Outperforms Images for 3D Detection LiDAR provides unambiguous depth measurements and robustness to lighting/occlusions, while cameras struggle with depth estimation and low-light conditions:

LiDAR vs. Camera for 3D Object Detection

Aspect	LiDAR	Camera
Depth Accuracy	Precise (x, y, z)	Estimated
Occlusion Handling	Robust	Fragile
Low-Light Performance	Unaffected	Degraded
Data Efficiency	Sparse points	Dense pixels

G. Data Augmentation

To improve generalization and robustness to real-world variations, we apply LiDAR-specific data augmentation to the raw point cloud before pillar encoding. Global transformations (rotation, scaling, and translation) simulate diverse sensor viewpoints, while object-level augmentations (ground-truth-aware sampling and noise injection) ensure balanced exposure to rare objects and occlusions. These operations artificially expand the effective training set while preserving physical plausibility in 3D space. These augmentations are applied before pillar encoding to preserve geometric coherence.

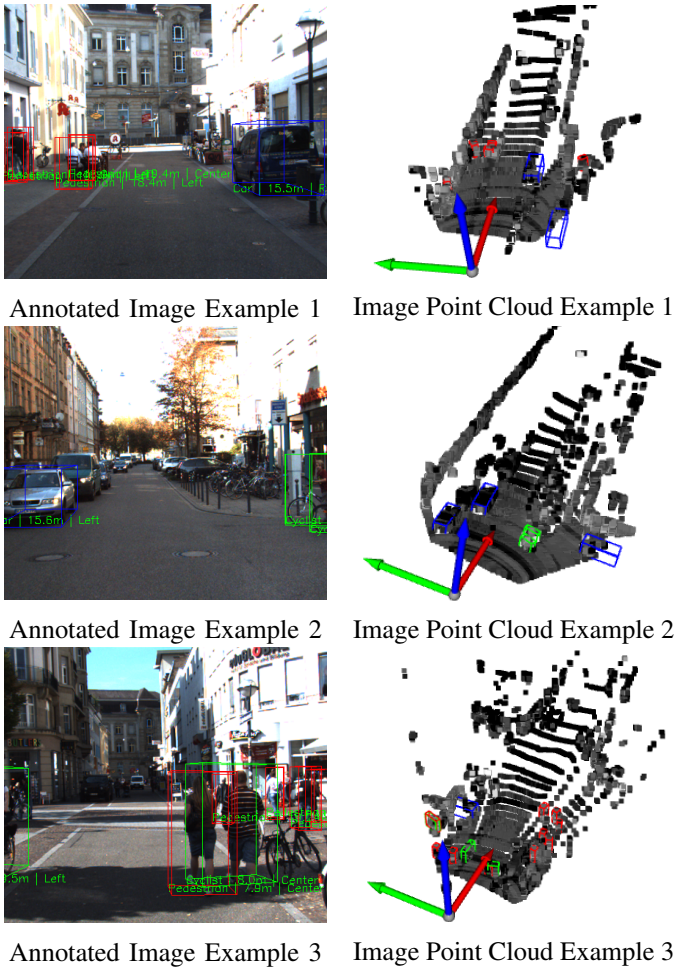
H. Training and Finetuning Protocol

We first trained the model on KITTI, then to improve accuracy and improve generalization of the model, we fine-tuned on the converted Lyft dataset with learning rate of $lr = 1 \times 10^{-7}$ and epoch of 100. We applied LiDAR-specific augmentations: global rotations $(\pm 20°)$, scaling $(0.95–1.05\times)$, and ground-truth-aware object sampling. Training used AdamW ($lr = 2.5 \times 10^{-4}$, weight decay$= 0.01$) with OneCycleLR scheduling over 160 epochs (batch size$= 6$). Gradient clipping (max norm$= 35$) stabilized training. Mixed-precision acceleration and GPU parallelization reduced runtime.

Evaluation Metrics Performance was measured via Average Precision (AP) at IoU thresholds of 0.5 and 0.7 for cars and pedestrians. Lyft classes (trucks, buses) were mapped to KITTI equivalents (e.g., truck→car) for consistency. Fine-tuning on Lyft data yielded slight improvements (detailed in Results).

I. Final Output

Sample outputs



Annotated Image Example 1 | Image Point Cloud Example 1
Annotated Image Example 2 | Image Point Cloud Example 2
Annotated Image Example 3 | Image Point Cloud Example 3

Fig. 2: Simulation results for the output limiting distance at 20m.

Audio Feedback Integration
To enhance the interpretability of 3D object detection results, we incorporated an audio feedback system that converts model predictions into spoken descriptions using gTTS (Google Text-to-Speech). For each detected object, the system announces its class label (e.g., car), estimated distance from the ego vehicle (e.g., 12.5 meters), spatial direction (left, center, or right), and the model's confidence score. These attributes are formatted into a sentence such as: "Pedestrian, 8.3 meters, right, 87

IV. Results

We compared three models:
- The baseline PointPillars implementation [1],
- Our re-trained model with updated KITTI data, and
- Our fine-tuned model using Lyft data converted to KITTI format

To analyze the performance of our finetuned model for PointPillar architecture, we used metrics like Average Orientation Similarity (AOV TABLE I), 2D bounding box detection performance (BBOX_2D TABLE II), 2D bounding box detection in the Bird's Eye View (BBOX_BEV TABLE III), 3D bounding box detection performance (BBOX_3D TABLE IV), and a Overall Matrics Table (TABLE V).

Out of all tables, we used two different type of metric to measure the performance of our model. For AOS table, the metric used was AOS@0.5, meaning it measures how closely the predicted orientation of each object matches the ground truth with a 50% IoU threshold. For all other tables, AP@0.5 metrics is used, where it measures the overlapping of detected bounding boxes with ground truth boxes at 50% IoU threshold.

Each results analysis the model performances on 3 labels, Pedestrian, Cyclist, and Car, where each labels are further split into 3 difficulty, Easy, Moderate, and Hard. The easy level include objects that are clearly visible with minimal occlusion or truncation. The moderate level include objects with some occlusion or partial truncation. As for the hard level, it include objects that are heavily occluded, truncated, or are very small. The AOS measures the detection performance

TABLE I: Results for AOS

Label	Difficulty	Trained	Fine-Tuned	Baseline
Pedestrian AOS@0.5	Easy	49.3713	**49.4335**	49.3777
Pedestrian AOS@0.5	Moderate	46.7359	**46.8695**	46.7284
Pedestrian AOS@0.5	Hard	**43.8572**	43.8217	43.8352
Cyclist AOS@0.5	Easy	**85.0412**	84.5293	**85.0412**
Cyclist AOS@0.5	Moderate	69.0867	67.2033	**69.1024**
Cyclist AOS@0.5	Hard	**66.2872**	63.5806	66.2801
Car AOS@0.7	Easy	90.4754	**90.4895**	90.4752
Car AOS@0.7	Moderate	88.6842	**88.7199**	88.6828
Car AOS@0.7	Hard	85.7265	85.6991	**85.7298**

that additionally considers the accuracy of object orientation estimation using AOS@0.5 metric. For the easy level, the finetuned model shows an improvement in Car and Pedeestrian labels than trained model and baseline. This indicates the fine tuned model is able to handle orientation estimation well when objects are clearly visible. As for the Moderate difficulties, finetuning is able to increase the result by a small yet consistent performance boost, likely due to improved depth estimation and orientation refinement. Lastly, For the Hard difficulty, finetuned model is not able to bring any performance over trained and baseline model.

TABLE II: Results for BBOX_2D

Label	Difficulty	Trained	Fine-Tuned	Baseline
Pedestrian AP@0.5	Easy	64.6159	**64.6463**	64.6249
Pedestrian AP@0.5	Moderate	61.3699	61.3873	**61.4201**
Pedestrian AP@0.5	Hard	**57.6202**	57.6005	57.5965
Cyclist AP@0.5	Easy	86.2569	**86.4016**	86.2569
Cyclist AP@0.5	Moderate	73.0655	72.9704	**73.0828**
Cyclist AP@0.5	Hard	**70.1906**	70.0547	70.1726
Car AP@0.7	Easy	90.6471	**90.6610**	90.6471
Car AP@0.7	Moderate	89.3344	**89.3664**	89.3330
Car AP@0.7	Hard	86.6545	86.6403	**86.6583**

As for BBOX_2D, BBOX_BEV, and BBOX_3D, finetuned model is able to obtain an overall improvement over the trained and baseline model in most of the labels and difficulty. Although the improvements are marginal, it still suggests that with the additional Lyft dataset, fine tuning refines the model's ability to detect and localize object.

TABLE III: Results for BBOX_BEV

Label	Difficulty	Trained	Fine-Tuned	Baseline
Pedestrian AP@0.5	Easy	59.1692	**59.4158**	59.1687
Pedestrian AP@0.5	Moderate	54.3533	**54.5086**	54.3456
Pedestrian AP@0.5	Hard	50.4978	**50.6886**	50.5023
Cyclist AP@0.5	Easy	84.4268	**84.5293**	84.4268
Cyclist AP@0.5	Moderate	67.1167	**67.2033**	67.1409
Cyclist AP@0.5	Hard	**63.7409**	63.5806	**63.7409**
Car AP@0.7	Easy	89.9664	**89.9724**	89.9664
Car AP@0.7	Moderate	87.8855	87.8774	**87.9145**
Car AP@0.7	Hard	85.7658	**85.7688**	85.7664

TABLE IV: Results for BBOX_3D

Label	Difficulty	Trained	Fine-Tuned	Baseline
Pedestrian AP@0.5	Easy	51.4462	**51.7014**	51.4642
Pedestrian AP@0.5	Moderate	47.9711	**48.0932**	47.9446
Pedestrian AP@0.5	Hard	43.8442	**43.9367**	43.8040
Cyclist AP@0.5	Easy	**81.8821**	81.4928	81.8677
Cyclist AP@0.5	Moderate	63.6617	**63.7385**	63.6617
Cyclist AP@0.5	Hard	60.8990	**60.9588**	60.9126
Car AP@0.7	Easy	86.6200	**86.8753**	86.6456
Car AP@0.7	Moderate	76.7294	76.7382	**76.7439**
Car AP@0.7	Hard	74.1545	**74.1836**	74.1668

TABLE V: Results for Overall Metrics

Label	Difficulty	Trained	Fine-Tuned	Baseline
bbox_2d AP	Easy	80.5067	**80.5696**	80.5097
bbox_2d AP	Moderate	74.5899	74.5747	**74.6120**
bbox_2d AP	Hard	**71.4884**	71.4319	71.4758
AOS AP	Easy	74.9627	**75.0134**	74.9647
AOS AP	Moderate	68.1690	68.1438	**68.1712**
AOS AP	Hard	**65.2903**	65.2010	65.2817
bbox_bev AP	Easy	77.8541	**77.9725**	77.8540
bbox_bev AP	Moderate	69.7851	**69.8631**	69.8003
bbox_bev AP	Hard	66.6681	**66.6793**	66.6699
bbox_3d AP	Easy	73.3161	**73.3565**	73.3259
bbox_3d AP	Moderate	62.7874	**62.8566**	62.7834
bbox_3d AP	Hard	59.6326	**59.6930**	59.6278

This improvements can further be visualized on the Overall Matrics Table (TABLE V) as finetuning model achieved a slight increase in most of the categories. From the aggregated performance measures, it indicate that fine-tuning brings a small but consistent improvement across all evaluated aspects: 2D detection, BEV localization, 3D detection, and orientation estimation (AOS).

Metric	Diff.	Baseline	FT	Δ (Imp.)
2D Bbox	Easy	80.5097	80.5696	+0.0599 (+0.07%)
2D Bbox	Mod.	74.6120	74.5747	-0.0373 (-0.05%)
2D Bbox	Hard	71.4758	71.4319	-0.0439 (-0.06%)
AOS	Easy	74.9647	75.0134	+0.0487 (+0.06%)
AOS	Mod.	68.1712	68.1438	-0.0274 (-0.04%)
AOS	Hard	65.2817	65.2010	-0.0807 (-0.12%)
BEV	Easy	77.8540	77.9725	+0.1185 (+0.15%)
BEV	Mod.	69.8003	69.8631	+0.0628 (+0.09%)
BEV	Hard	66.6699	66.6793	+0.0094 (+0.01%)
3D	Easy	73.3259	73.3565	+0.0306 (+0.04%)
3D	Mod.	62.7834	62.8566	+0.0732 (+0.12%)
3D	Hard	59.6278	59.6930	+0.0652 (+0.11%)

Note: Positive Δ values indicate improvement. Relative improvement calculated as $\frac{\text{Fine-Tuned} - \text{Baseline}}{\text{Baseline}} \times 100\%$.

The fine-tuned model showed best improvements in BEV Bbox (Easy) with +0.1185 AP (largest gain), 3D Bbox (Moderate/Hard) with +0.07 AP, and AOS (Easy) with +0.0487 AP, while exhibiting minor regressions in 2D Bbox (Moderate/Hard) with -0.04 AP drops and AOS (Hard) with -0.0807 AP (worst regression). Overall, the model achieved modest gains (mostly ¡ +0.1 AP) across metrics, demonstrating improved 3D/BEV detection capability at a slight cost to 2D/AOS performance.

V. Conclusion

In conclusion, this project demonstrated the potential of integrating LiDAR-based technology with deep learning methods has the possibility of enhancing object detection and environmental awareness for visually impaired users. When employing the PointPIllars architecture, it is able to take advantage of LiDAR in precise depth measurement, robustness to lighting conditions, and reliable performance in challenging scenarios to improve the overall accuracy. Our experiment begins with training the model on the most updated KITTI dataset, followed by fine-tuning it using an additional Lyft dataset. This resulted in consistent improvements in several key performance metrics including orientation estimation, 2D and 3D bounding box detection, and Bird's Eye View localization. Although some improvements were incremental, we do observe consistency across results suggesting the potential of expanding LiDAR training sets for real world applications.

With fine-tuning, the model trained showed modest yet meaningful improvements over the original and baseline model. This improvement is particularly evident in Easy and Medium challenging datasets. Through those results, it validates the efficacy and the possibilities of fine tuning with additional data to improve the robustness and generalization capability of the PointPillars model.

Beyond the numerical evaluation increase, this project included the addition of annotations through visual and text-to-speed to provide insights when applied in a real world scenario, where the train system can become practical where clarity in labeling and voice driven feedback are paramount. Furthermore, by integrating these annotation modules, it provides the system with the ability to not only enhance user comprehension during testing phases but to address different accessibility needs.

Overall, this study validates the practicality and benefits when it comes to training a LiDAR based object detection model with combined dataset approach. With the addition of including visual annotations and text-to-speech functionality, it provides a promising potential for improving the interpretability and usability of 3D object detection outputs in assistive technology scenarios.

Although this research demonstrated the viability of training and fine-tuning the PointPillars architecture using multiple datasets, there remain several opportunities to further enhance the system.

To start off, further research could look into adding more LiDAR datasets from diverse environments to further enhance the generalization and robustness of the model. By leveraging data from various scenarios like urban, and rural, it could further improve the accuracy, allowing it to better handle real world circumstances. Additionally, a focus on adding additional labels would also be beneficial to the project. Currently, the model only predicts 3 different label objects, and with addition of more labels would help the model to provide more detailed feedback.

Moreover, the model would benefit from gaining a more advanced interpretable output. By including richer contextual information like dynamic object trajectories or predictive paths, it would provide the user more actionable insights.

Lastly, future iterations of this work can consider analyzing the potential of domain adaptation and transfer learning techniques. From our results, fine tuning on a large size dataset only provides a marginal increase, domain adaptation and transfer learning, allowing the dependency on an extensive dataset to be reduced.

References

[1] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 3354–3361. [Online]. Available: https://www.cvlibs.net/datasets/kitti/

[2] M. Christy, N. Nikatos, P. Culliton, V. Shet, and V. Iglovikov, "Lyft 3D Object Detection for Autonomous Vehicles," Kaggle, 2019. [Online]. Available: https://kaggle.com/competitions/3d-object-detection-for-autonomous-vehicles

[3] Z. Hu, "PointPillars: Fast Encoders for Object Detection from Point Clouds," GitHub repository, 2021. [Online]. Available: https://github.com/zhulf0804/PointPillars

[4] World Health Organization: WHO, "Blindness and vision impairment," Aug. 10, 2023. https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment

[5] "WeWALK Smart Cane – Smart cane for the visually impaired." https://www.wewalk.io/

[6] Apple, "IPhone," *Apple*. https://www.apple.com/iphone-12-pro/

[7] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End learning for Point Cloud based 3D object Detection," *arXiv.org*, Nov. 17, 2017. https://arxiv.org/abs/1711.06396

[8] Y. Yan, Y. Mao, and B. Li, "SECOND: sparsely embedded convolutional detection," Sensors, vol. 18, no. 10, p. 3337, Oct. 2018, doi: 10.3390/s18103337.

[9] S. Shi, X. Wang, and H. Li, "PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud," arXiv.org, Dec. 11, 2018. https://arxiv.org/abs/1812.04244

[10] Z. Yang, Y. Sun, S. Liu, and J. Jia, "3DSSD: Point-based 3D Single stage object detector," arXiv.org, Feb. 24, 2020. https://arxiv.org/abs/2002.10187

[11] W. Ali, S. Abdelkarim, M. Zahran, M. Zidan, and A. E. Sallab, "YOLO3D: End-to-end real-time 3D Oriented Object Bounding Box Detection from LiDAR Point Cloud," arXiv.org, Aug. 07, 2018. https://arxiv.org/abs/1808.02350

[12] M. Hu, Z. Li, J. Yu, X. Wan, H. Tan, and Z. Lin, "Efficient-Lightweight YOLO: Improving small object detection in YOLO for aerial images," Sensors, vol. 23, no. 14, p. 6423, Jul. 2023, doi: 10.3390/s23146423.