Utilizing Synthetic Point Cloud Generation for Semantic Segmentation of Utility Poles

JULIAN BERNHARD DÜHNEN, University of Twente, The Netherlands

Fig. 1. A screenshot of the Toronto-3D dataset [10], which is used as a basis for the synthetic data generation.

Machine learning in classification and segmentation of point clouds scanned by LiDAR sensors has been a topic of interest for many years, and has been applied to various fields such as autonomous driving, urban planning and cartography. Despite this, obtaining and labeling real-world ground truth data to train these models remains a time-consuming, error-prone and costly task that is still widely done today, specifically in the field of utility poles and mapping out urban areas. This paper explores the use of synthetic data generation to train a model for semantic segmentation of point clouds of isolated utility poles. This research presents a methodology to generate synthetic point clouds of utility poles and train a model on these synthetic point clouds. The results show that the model trained on synthetic data can be used to segment real-world point clouds with exceptional accuracy, and that procedurally generating synthetic data can be a viable alternative to manually labeling real-world data.

CCS Concepts: • **Computer Vision and Pattern Recognition**; • **Semantic Segmentation**; • **Point Clouds**; • **Procedural Content Generation**; • **Machine Learning**;

Additional Key Words and Phrases: Utility pole, Point clouds, Synthetic data, Machine learning, Semantic segmentation

ACM Reference Format:
Julian Bernhard Dühnen. 2025. Utilizing Synthetic Point Cloud Generation for Semantic Segmentation of Utility Poles. In . ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Background and Context

The use of laser scanning technology has quickly grown to become a popular method for capturing 3D data of environments and structures, and its application in the use of machine learning has also gained traction. One such application is the use of machine learning to identify objects and their components from 3D point cloud data. This is particularly useful in applications such as autonomous driving, where humans, obstacles, and other vehicles need to be identified and located in real-time. Another such application would be for the maintenance of catenary arches, where the inspection process would be assisted through creating a LiDAR scan of the arches and using machine learning to identify and locate components of the arches. This would allow for a more efficient and accurate inspection process, as well as a reduction in the time and cost of the inspection process. This research, however, will focus on the application of streamlining the process of performing semantic segmentation on utility poles, specifically those scanned in the Toronto-3D dataset [10], using synthetic data generation. The research will demonstrate the feasibility of using synthetic data generation to automate the labeling process for components of laser-scanned objects, with utility poles being the main focus. This paper then proposes a methodology to generate this synthetic data, and evaluates it using a machine learning model for semantic segmentation of point clouds.

1.2 Problem Statement

Currently, manually scanned and labeled point cloud data is still widely used today to train AI models for semantic segmentation of point clouds. This process is expensive, time-consuming, and prone to human error, and can be a significant bottleneck in the training process. Labeling point clouds containing thousands to millions of points is a tedious task that requires time and effort, and requires capable hardware to manipulate the large datasets. There have been advancements made to automate the labeling process, such as using algorithms and AI models to label this data, however, the requirement to perform real scans of environments and structures still remains a significant bottleneck in many applications.

1.3 Research Objectives

This research aims to develop a method to generate point cloud data from a procedurally generated models of utility poles to train a point cloud semantic segmentation model. The model will then use this data to perform semantic segmentation on real-world point cloud

TScIT 42, January 31, 2025, Enschede, The Netherlands

scans. This will streamline the process of adding new components and supplying data to further train the model. The system will be evaluated on the resulting AI model's performance in segmenting point clouds, as well as its efficiency in terms of time and cost taken to generate data.

The research questions that will be addressed in this research are:

- **RQ1: What are the key features and parameters that contribute to the structure of the utility poles from the dataset?**
- **RQ2: What existing techniques can be used to generate synthetic point clouds of utility poles?**
- **RQ3: Can a model trained purely with fully synthetic data from the proposed method to reliably perform semantic segmentation on utility poles?**

1.4 Chapter Overview

Chapter 2 will provide an outlook on the current state of the art in using synthetic data for the semantic segmentation of point clouds, providing an insight into what research has already been done in the current field of interest. Next, chapter 3 will present the methodology used to research and develop the proposed method, as well as answer the first and second research questions. Chapter 4 contains the results of the research, presenting the performance and efficiency of the system. In Chapter 5, the results and findings of the research will be discussed, as well as strengths, limitations, and possible future developments. Finally, chapter 6 will conclude the research, summarizing the research objectives, findings, and contributions of the research. The third research question is also answered in this chapter as part of the conclusion.

2 LITERATURE REVIEW

As of the date this paper was written, there is a lack of research on the generation of synthetic point clouds from procedurally generated utility poles. Despite this, research has been done in the field of machine learning to generate synthetic point cloud data for training models for semantic segmentation, especially for large-scale environments such as cities and landscapes. This section will review the existing literature on the topic, and discuss the methods and techniques used to generate synthetic point cloud data, as well as the applications and results of these methods.

2.1 Automatic Generation of Point Cloud Synthetic Dataset for Historical Building Representation

[8] This is a research paper that presents a novel framework for automatically generating synthetic point clouds from digital 3D scenes. The framework is designed to generate automatically labeled point clouds that resemble laser scans of historical buildings, and can be used to train machine learning models for semantic segmentation. This research however does not implement procedural generation of these 3D scenes, and instead uses pre-existing 3D scenes manually modeled after historical buildings. This research demonstrates the feasibility of using digitally generated point clouds as training data, and shows that models trained on the synthetic data perform well on real-world data.

2.2 STPLS3D

[1] Existing research on the procedural generation of 3D scenes converted to labeled point clouds for use in semantic segmentation includes the research paper on STPLS3D. This paper investigates the use of generated 3D scenes for training models for semantic segmentation through the use of generating scenes containing terrain, vegetation, and buildings using procedural generation techniques. The point clouds are then generated via simulated UAV flight patterns. This procedure involves loading environment layouts (building footprints, road networks, etc.) from a publicly available dataset, and placing buildings and vegetation in the scene using the layout as a guide and a procedural generation algorithm. The generated scenes are then converted to point clouds using a simulated UAV flight pattern. The results of this research show that the generated point clouds can be used to train models for semantic segmentation, and that the models trained on the synthetic data perform well on real-world data.

2.3 PT2PC

[6] The techniques used in PT2PC is based on the idea of generating a diverse set of point clouds from a symbolic part tree representation. A part tree is a hierarchical structure, defining the parts of an object and their relationships to one another (e.g., a chair consists of a seat connected to a surface and a frame, where the frame consists of two frame bars etc.). This research highlights the concept of generating a large set point clouds that are diverse in their shape and structure, yet conform to a set of rules to maintain a realistic representation of the object.

3 METHODOLOGY AND APPROACH

In this section, the methodology that was used to conduct the research will be discussed, including the steps involved in each phase along with their technicalities. The programming language used for ths research was Python [11], and the most noteable libraries used were Open3D [12] for 3D geometry and point cloud processing, and PyTorch [7] for machine learning. To create 3D models for the utility poles, Blender [2] was used.

3.1 Dataset preparation

The first stage involved preparing the ground truth dataset to use for reference, analysis, and evaluation of the final result. Due to time and resource constraints, this research focuses on the utility poles from the **Toronto-3D** dataset [10], as opposed to utility poles in general. This dataset was selected as it contained a significant number of clearly scanned utility poles with an acceptable level of variation and structure for the scale of this research. This stage involved conducting qualitative research to analyze, understand and identify the key features and parameters that contribute to the structure of these utility poles. The dataset already had all utility poles labeled (under "Pole", together with traffic lights, streetlights, and other pole-like structures). From this data, all point clouds labeled "Pole" were extracted and grouped via the DBSCAN clustering algorithm using CloudCompare [3], a software for point cloud processing. The resulting clusters were then manually filtered to remove any clusters that were not utility poles. Manual labeling was then performed, segmenting the clouds by **pole**, **lamp**, **cross arm**, **transformer**, **sign**,

traffic light, and **pedestrian signal**. Shapes that were too complex and unrecognizable were removed, and labeling was done taking the known issue of distortions with the dataset into account. Some shapes were slightly distorted or duplicated due to the scanning process, and these were manually corrected by ignoring points that were too heavily offset from where they should be. Cables were also not labeled. A dataset of 93 samples was created. This sample size is relatively small, but it was deemed sufficient for the scale of this research. The number of points in each sample ranged from a few hundred to over ten thousand. For this reason a standard point count of 1024 was used. Point clouds with less than 1024 points would be sampled randomly without replacement (duplicating points), and samples with more than 1024 points would be sampled randomly without replacement. Figure 3 shows statistics on the balance of samples within the ground truth dataset. Take note how the dataset is not very balanced, and due to the limited sample size, the number of samples used from the ground truth dataset will have to be maximized. This will be addressed later in the paper.

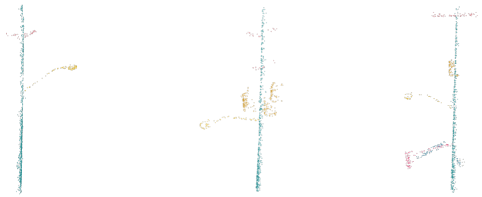

Fig. 2. Isolated and labeled ground truth samples.

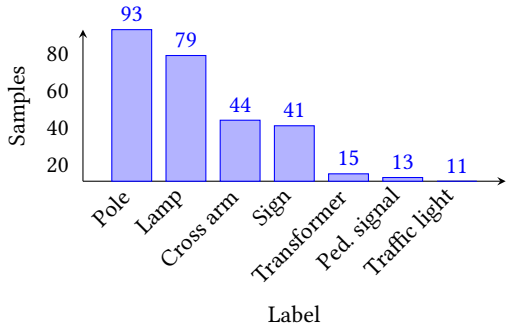

Fig. 3. Label statistics of the ground truth dataset, showing the number of samples containing a corresponding label.

3.2 Utility Pole Analysis

The next stage involved analyzing the ground truth dataset, as well as conducting qualitative research to understand the key features and parameters that contribute to the structure of the utility poles. The ground truth dataset was also analyzed for the patterns and structures of the points, as well as the dimensions, variations, and placement of the components of the utility poles. In addition to a large set of rules defined, the first research question can be answered by analyzing the dataset and identifying the key features and parameters that contribute to the structure of the utility poles;

3.2.1 Poles. Poles are the root component of the utility poles, and are always present. They are the tallest component of the utility pole, and are always vertical. They had a base radius of about 15 centimeters, a top radius of about 10 centimeters, and a height of around 8.45 meters. The poles would then be scaled by a random value between 1.0 and 1.955.

3.2.2 Streetlamps. Streetlamps are the most common component of utility poles, and are usually placed at the top of the pole. The dataset used a single make and model of streetlamp, so a 3D model of the streetlamp was created. The streetlamp was modeled using the ground truth point clouds, as well as images of the streetlamp found online. There would only be a maximum of one streetlamp per pole. They have a slight variation in rotation (around 5 degrees), and tend to be mounted at heights from 6.5m to 9.44m (Always being above traffic lights if any are present).

3.2.3 Traffic Lights and Pedestrian Signals. Traffic lights would only be present at intersections, and could have a second traffic light perpendicular to the first. There would (almost) always be a pedestrian traffic light opposite to a traffic light, and the traffic light had a chance of containing a large blue sign on it to signal the entering of a new street. Traffic lights are mounted at a height between 4.2 and 5.2 meters, and the pedestrian signals are mounted at heights between 2.3 and 2.6 meters.

3.2.4 Transformers. Transformers had three variations:

- **Three cylindrical transformers**: with four cross arms above them. Two of the cylinders would form a line perpendicular to a road. They are mounted at a height between 4.1 and 4.2 meters from the top of the pole, but are always above streetlamps and traffic lights if any are present.
- **A single cylindrical transformer**: always parallel to the road. Mounted between 9.9 and 10.1 meters from the ground, always above traffic lights if they are present.
- **A single cuboid transformer**: could be placed at any angle. Mounted between 4.4 and 4.7 meters from the ground.

3.2.5 Cross Arms. There are 6 defined variations:

- 3 facing the road with 1m spacing starting from 0.3m below the pole apex, with a chance of having one more between a streetlamp and traffic light if present.
- One large cross arm on the top of the pole, up to 0.3m below the pole apex.
- Two smaller cross arms on the top of the pole, from 0.62m to 0.9m below the top.
- Two pairs of smaller cross arms on the top of the pole, around 0.8 from the top of the pole, with around 1.6m spacing.
- One smaller cross arm facing the road, from 1.3m to 2.9m from the top.

3.2.6 Signs. Signs come in many various shapes, forms, and patterns and was the most difficult to analyze due to the variation of these components. These signs were divided into two main types, namely side signs (signs mounted on their side to the pole, such as standard street signs) and normal signs (signs with their baseplate mounted directly to the pole). Side signs such as street signs (both the new rounded-top design and the older smaller rectangular street

sign) had their placement rules assigned to their purpose (street signs only appear in junctions and point towards another street, advertisements would be placed at random as long as their larger side was visible to drivers on the road, and the height of the placement was legal as per defined by sign regulations in Toronto). Other signs that depended on the state of the road and intended for drivers were placed at rotations around 45 degrees from the roadside to be visible to drivers. Smaller indicator signs also had a common pattern where multiple of them would be placed directly above or below each other in small groups. Side signs also tended to never be overlapping on the z axis regardless of their angle, as this may hinder their visibility.
Normal signs also had a tendency to be grouped together vertically when multiple were present, but had a large range of possible sizes. For this, a set of manually defined widths and heights were defined and pseudo-randomly selected from to place on the pole. These types of signs would always be intended for vehicles and not pedestrians, and so they do not face more than 90 degrees away from any nearby road. Sometimes a single sign would be placed higher above on the utility pole for reasons such as to indicate zones where trucks are prohibited, and this was also taken into account.

3.3 Answering Research Question 1

With this gathered information, research question 1 can then be answered. It can be stated that the key features and parameters that contribute to the structure of the utility poles from the dataset are the poles themselves, streetlamps, traffic lights, pedestrian signals, transformers, cross arms, and signs. For the poles, the key parameters that contribute to the structure of the utility pole is the height, base radius, and top radius. For streetlamps, the contributing parameters are its presence, the height the streetlamp is mounted at, and the angle it is mounted at. Traffic lights and pedestrian signals have their key parameters as the number of traffic lights or pedestrian signals present, the presence of a blue sign on the traffic light, and their mounted heights and angles. For transformers, the key parameters are the type of transformer, the number of transformers, and the angle of the transformer, as well as its mounted height. For cross arms, the key parameters are the type of cross arm, the number of cross arms, the mounted height, and the angle of the cross arms. Signs have the highest number of key parameters due to their variety. These parameters include the type of sign, the number of aligned sign clusters present, the mounted height of each cluster, the mounted angle of each cluster, and the size and type (side or normal) of each sign. These key features and parameters are crucial to the structure of the utility poles in the dataset, and are used to generate synthetic data in the next stage of the research.

3.4 Data Generation

3.4.1 Answering Research Question 2. Deciding on an approach to generate the synthetic data was a crucial step in the research. The generation process needed to balance realism, accuracy, and variation with performance, efficiency, and scale. Initial research was made on existing methods to generate point clouds from 3D geometry to get insights on possible approaches. To answer the second research question regarding what existing techniques can be used to generate synthetic point clouds of utility poles, the following current methods for reaching the goal are described and assessed. BlenSor [4] is a package for simulating various laser scanning sensors within a 3D modelling application called Blender [2]. This package leverages the 3D capabilities of Blender to allow users to simulate laser scanning of a 3D scene without having to leave the 3D modeling program. This package was considered for direct use in this research but was ultimately not used. The reason for this was because the package is not intended for use in generating large datasets, and has no way to automatically do so. This is also designed to scan large scenes over single objects due to its nature of firing rays in all directions from the simulated LiDAR sensor, and this may not be as efficient as a custom approach for this purpose. Another popular method is to use CloudCompare [3]'s built-in tools for sampling points on 3D geometry. This method runs very well, but unfortunately is also not designed for automated generation of large datasets, and requires importing and exporting to and from a separate software. To the current knowledge of the author at the time of writing, there are no existing methods to directly generate synthetic point clouds of utility poles, but there are tools available to allow for the creation of such a workflow.

3.4.2 Procedural Generation. Using the collected rules, insights and patterns, an algorithm was created to generate 3D mesh scenes of utility poles, ensuring maximum diversity in its generation, while adhering to the defined rules as realistically as possible. This system had two main components, namely the **scene generation**, which includes generating, placing and modifying 3D geometry to construct utility poles, and the **point cloud conversion**, which involves converting the geometry into a point cloud. This section explores these steps in detail.

3.4.3 Scene Generation. Using the derived rules and patterns, a tool was developed that could generate 3D scenes of utility poles on the fly within a fraction of a second on mid-level hardware. A key strength of this system was its modularity. For example, in the use case scenario of utility pole maintenance for a city: if the city decides to implement a new sign or model for their streetlamps or perhaps a new component entirely, maintainers would just need to add a new definition for the new components and regenerate the data. This is much more cost-effective and efficient in comparison to waiting for new installations to be made in the real world, laser scanning the utility poles again and then labelling the new data.

A starting point for the procedural generation is to decide on the current state of the road. This would involve decisions such as whether the utility pole is at an intersection or beside a normal road, and which of the nearby roads was a main road. From this, the system is able to make decisions such as the angle of components that need to face towards or away from the street, or the presence of components such as traffic lights that are only present at junctions (according to the ground truth dataset).
A key component in the component placement systems was the use of preventing overlaps through defining **placements**. A placement in this system is referred to as an object attached to the utility pole that will be taken into account in overlap prevention. A state is shared between steps in the placement algorithms, and whenever a component is placed which other placements would need to account

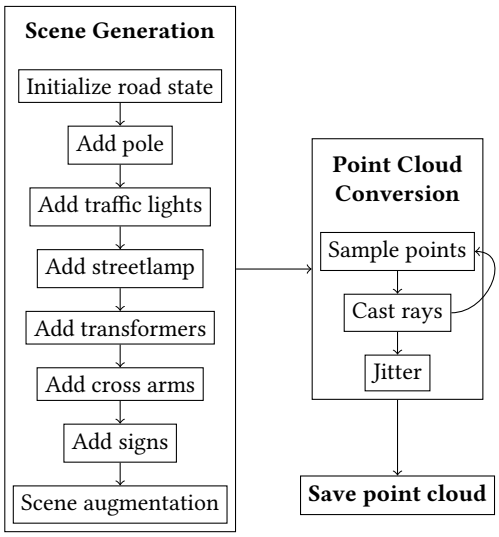

Fig. 4. High-level overview of the data generation procedure.

for, it would be stored in a defined group. Placements in the same group cannot overlap any other placements in the same group in the vertical axis regardless of what angle they are placed. Placements may otherwise only overlap other placements in the vertical axis if they are placed at least 90 degrees in vertical rotation away from the other placement. This way, one would just need to store a dictionary mapping a key denoting the placement's group, to an array of objects containing the height (the amount of vertical space the placement takes up), the z-position and the angle at which the placement was made. An example of this usage is with the so-called side signs. When the system would decide on a position to place a side sign, it would ensure that it would not obstruct the visibility of any other side signs. However, in the case of normal signs, it would not block their visibility when placed at the same vertical position as normal signs do not protrude far from the pole. The placement groups are defined, but can be easily extended. These were side signs, normal signs, and miscellaneous placements (an arbitrary group for all other objects such as traffic lights, streetlamps, and cross arms). The system then places the components in the order of poles, traffic lights, streetlamps, transformers, cross arms, and signs. When new geometry is added to the scene, the system keeps track of the geometry's labels via storing the label of the triangles in an array in the shared state. To add further variation, components such as some signs, cross arms and side signs have subtle rotation jitter added to them. This is to simulate the slight imperfections in the real world, and to make the generated data more realistic. The system then rotates the scene to a random angle between 0 and 360 degrees. A tilt is added to the entire scene very slightly (a maximum of 5 degrees) using the base of the pole as an origin. This is done to replicate the slight tilt of utility poles in the real world from causes such as weather, collisions with vehicles, or loose soil.

3.4.4 Point Cloud Conversion. Laser-scanned point clouds have a distinct pattern that is not easy to replicate. Different laser scanners may produce varying patterns, and other factors such as the movement pattern of the scanner also contribute to how the points in these samples are placed. Through the analysis of many point clouds, it was found that key features that need to be replicated in the data are:

- **Noise**: The points in the point clouds are not perfectly aligned, and have a slight jitter to them.
- **Occlusion**: Points are only placed where the scanner can see, and surfaces occluded from the scanner may not be detected.
- **Scaling**: Scans are not 100 percent accurate in scale, and may have slight variations in size.

From these insights, a point cloud conversion system was developed that would convert the scene into a labeled point cloud. The first step would be to determine how points are placed on the surfaces of the scene. To achieve this there are two possible approaches. One would be to simulate a laser scanner through simulating a set of rays in all directions from a given simulated sensor position. This approach could include several iterations through performing multiple ray casts in an array of points to simulate the movement of a sensor attached to a vehicle scanning multiple frames of the target object. This could then be further optimized by only firing rays in the general direction of the target object. Going this route proved to be difficult as for each iteration a very high number of rays would need to be simulated to achieve a realistic result. To achieve the desired number of points in the point cloud, the system would need to fire many more rays as the likelihood of a ray not hitting the utility pole would be high. If too many points were detected, points would then need to be discarded, leading to wasted effort. For this reason a new approach was developed. This approach involved first uniformly sampling a set of points on the surfaces of the geometry, and then firing rays towards these points from a simulated sensor position. This approach allowed the creation of point clouds with a precise number of points with high efficiency.
The initial point sampling is performed using Open3D [12]'s built-in `sample_points_uniformly` method. This method takes in a mesh and a number of points to sample, and returns a point cloud with the specified number of points. The point cloud is then used to generate a set of rays that are fired towards the point cloud from a simulated sensor position. Open3D's `cast_rays` function stores the index of the triangle that a ray hits, which is used to determine the label of each hit point by referring to the stored array of triangle labels. There are edge cases where one or more rays may not hit any triangle. To work around this, the system does the sampling and ray casting process in a while loop with the remaining number of points until the desired number of points is reached. This achieves a consistently acceptable result with a relatively high degree of efficiency in comparison to the first approach. Finally, the system adds a random jitter to each point to simulate the noise in the real-world point clouds. This jitter is a random value between 0 and 2 millimeters (inclusive) in any direction. The system then saves the point cloud in the `.ply` format, which is a widely used format for point clouds. A strength with this approach is that due to its ability to produce a specified number of points, the system can easily be adjusted for compatibility with different laser sensors of varying resolutions. The position of the sensor is at 1 meter, and is at a pseudo-random angle with a distance from 3 to 20 meters from the utility pole. This

emulates a sensor mounted low on a vehicle. It is useful to note that the point clouds are not normalized in the preprocessing stage. The data is normalized by the system in-memory when accessed. This is to allow for further extensibility of the system. In the case where a model would need to be trained that requires different scaling, the system can be easily adjusted to accommodate this without needing to generate an entirely new dataset.

4 EVALUATION AND RESULTS

In this section, how the system was evaluated will be discussed, along with the results obtained from the evaluation process. First, the metrics and methods used to evaluate the system will be discussed, followed by the results obtained from the evaluation process.

4.1 Efficiency

A core detail to measure is the efficiency of the system in terms of the time and resources required to generate the synthetic dataset. The time taken to generate the dataset is a crucial factor, as the system should be able to generate a large dataset in a reasonable amount of time to be comparable to recording and labeling its real-world counterpart. The time it takes to generate datasets of varying sizes will be measured to evaluate its performance. Another factor that will be varied is the number of points in the point cloud, as this will also affect the time taken to generate the dataset. Each test was performed three times to ensure the results are consistent, and the average of these runs was recorded.

Table 1. Time taken to generate datasets. p denotes the number of points in the point cloud for each sample and n denotes the number of samples in the dataset.

p \ n	**1024**	**4096**	**16384**	**65536**
256	36s	3m 40s	9m 28s	40m 32s
512	56s	6m 5s	15m 39s	1h 11m 14s
1024	1m 59s	7m 32s	29m 19s	2h 8s

Table 1 shows the time taken to generate datasets of varying sizes and point counts per sample, rounded to the nearest second. Additionally, a graph showing the increase in time with the number of samples for each value of p (points per sample). The time taken to generate the dataset is linearly proportional to the number of samples in the dataset, as expected. The system is capable of generating a dataset of 65'536 samples with 1024 points per sample in just 2h 8s, which already greatly exceeds the size of the utility poles in the Toronto-3D dataset [10]. It took multiple hours to manually label the Toronto-3D dataset and many more to organize, prepare and perform the laser scanning process, which is a testament to the efficiency of the dataset. These values were recorded on a remote Jupyter [5] server with an Intel Xeon Gold 5220R CPU, two NVIDIA Tesla T4's, and 18Gi of RAM free to the instance.

4.2 Performance

In this subsection, the performance of the synthetic dataset generation method will be evaluated through feeding the synthetic data into a machine learning model and evaluating its performance on performing semantic segmentation on the ground truth dataset.
To evaluate the synthetic dataset generation method, the **PointNet** [9] architecture was used for the AI model. The PointNet architecture is a popular deep learning model capable of performing semantic segmentation on point clouds. Despite it not being the best performing model available for semantic segmentation, the scale of the dataset and the simplicity of the PointNet architecture make it a suitable choice for the evaluation.
The model will be evaluated using the mean **Intersection over Union (IoU)** metric, which is a common metric used to evaluate the performance of semantic segmentation models. This metric is calculated as follows:

$$\text{IoU} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} \tag{1}$$

Here, TP, FP, and FN denote the number of true positive, false positive, and false negative values respectively. The mean IoU is calculated by taking the average of the IoU values for each class. This metric is a method to measure the overlap between the predicted segmentation and the ground truth segmentation, where a higher value corresponds to a better overlap between the two.
Another metric that will be used to evaluate the model is the **F1 score**. The F1 score is the harmonic mean of the precision and recall of the model, and is calculated as follows:

$$\text{F1} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{2}$$

This metric balances the precision and recall of the model. A higher value corresponds to a better balance between the two. The precision of the model is calculated via dividing the number of true positives, with the sum of the true positives and false positives. The recall is calculated by dividing the number of true positives with the sum of the true positives and false negatives.
Finally, accuracy was used, which is another common metric used to evaluate the performance of machine learning models. Take note, however, that accuracy, being the ratio of correct predictions to the total number of predictions, does not take into account class imbalances in the dataset.
Using 15 epochs and a point count of 1024 per point cloud, the model was trained on 65536 of the generated synthetic samples, and validated and tested on the ground-truth dataset created in Subsection 3.1, with 20% and 80% splits respectively. As a point of reference, the same model was trained with a portion of the ground truth dataset. Due to the lack of models to compare semantic segmentation of utility poles, a split of 70% to 20% to 10% for training, testing and validating data respectively was used. The test results are shown in Table 2.

Table 2. Performance metrics of the model trained on a portion of the ground truth dataset, versus a model trained on the generated synthetic data.

Metric	**Ground Truth**	**Synthetic**
Mean IoU	0.189	0.582
F1 Score	0.235	0.676
Accuracy	0.762	0.922

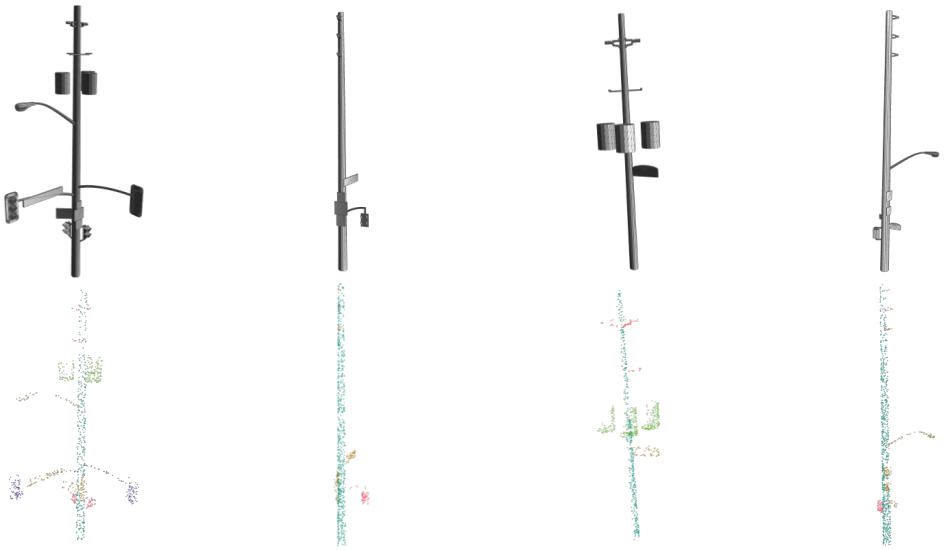

Fig. 5. Procedurally generated utility poles and their corresponding labeled point clouds.

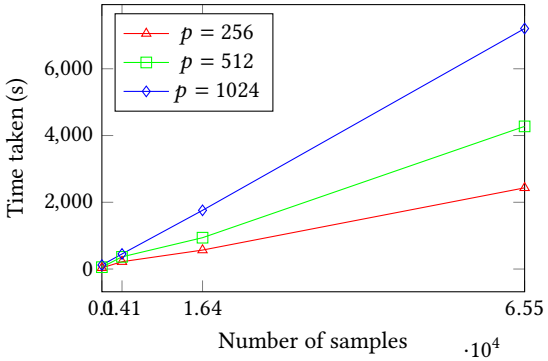

Fig. 6. Comparison of time taken to generate datasets of varying sizes by point count.

5 DISCUSSION

From what can be seen in the results in Table 2, the model outperforms the baseline model in all metrics. This was done using data generated from an algorithm in just above two hours, as opposed to spending multiple hours undergoing the scanning process of real-world data. These results, however, should be taken with a grain of salt as the dataset used to train the baseline model is relatively small and does accurately represent the capabilities of a semantic segmentation model trained on real world laser scans. Despite this, the synthetic data generation methods still prove to be a viable alternative to the traditional method of collecting and labeling data. The results show that the model is capable of performing semantic segmentation on utility poles with exceptional performance, considering this research was conducted in a small scale, using a relatively simple model, and a small dataset.

5.1 Strengths

A key strength of this approach to gather labeled point clouds is the ability to generate very large amounts of labeled data in a short amount of time. The system can produce large datasets in just a matter of hours, as opposed to the days or weeks it would take to manually label the same amount of data. This gives a significant advantage in scenarios where resources (or time) are limited, such as where a new system is being developed and there is a need for a large amount of labeled data to quickly prototype the system. Another such application would be where the system is being developed in an ever-changing environment, such as in systems that need to perform semantic segmentation on scans of urban areas. Governments may introduce new signs or mount new models of traffic lights, and the system would need to be updated to maintain its performance. Using such an algorithm would just mean implementing new rules for newly added, changed or removed components, and generating new datasets to train the model. These implementations could also be done before or during the deployment of these new components, allowing for consistent performance of the system in a changing environment.

5.2 Limitations and Future Work

The system as it is currently implemented, however, does have some limitations and drawbacks. Due to time, scale, and resource constraints, there are many areas of improvement that remain. The method presented in this research makes tradeoffs on realistic scans by using methods such as random point placement on the generated geometry as opposed to simulating the actual scanning patterns of a LiDAR scanner. The distinct lined pattern of the points on real LiDAR scans is not present in the generated data, which potentially harms the performance of the system. In addition to this, other aspects of points in the data, such as the normal vectors, colors, or reflection intensities are not accounted for in the data. These

Fig. 7. Predictions done by a PointNet model trained with synthetic data (top) vs. ground truth samples (bottom).

attributes can provide much more context to the data, and could potentially result in a better performing model. The Toronto-3D [10] dataset does contain color information in its scans, so this could be a potential area of improvement without the need to wait for or gather new data.

Another known limitation of the system is the lack of balance in the dataset. The current method uses random chances to decide on whether each component should be present or not in a sample, without taking into account the distribution of components in the dataset or the real world. This potentially causes the model to have a bias towards certain labels.

There is also little variation for more complex shapes such as streetlamps and traffic lights. The current method uses 3D models made after the existing utility poles in the Toronto-3D dataset, using the point clouds and images of the location on the internet as a reference. Some of these components vary more and may have customized features, such as the length of the arms of the traffic lights. These models may also not accurately represent their real-world counterparts as they were created by hand and not sourced from the original manufacturers. Using manufacturer-sourced models and professionally informed rules of placement and variation would allow for a more accurate representation of the target data.

To use this system on real data, the system depends on either a highly accurate semantic segmentation model to isolate utility poles from the rest of the point cloud, or manually isolated utility pole point clouds. One must also make sure to remove objects such as cables and other unknown objects from the point cloud before feeding it into the system for segmentation.

In future work, the system's performance and efficiency could be significantly improved. The current system does not take into account optimization methods such as multithreading, multiprocessing, or GPU acceleration and relies mainly on the CPU and Open3D [12]'s Python bindings. Potentially, the system could be optimized to generate data on the fly, where the model is trained on the data as it is generated, rather than needing to save the data to storage and loading it back when needed. This would open up possibilities such as removing the need to transfer any training data but instead just needing to send a seed to the recipient.

6 CONCLUSIONS

Exploring the possibilities of generating synthetic point clouds for use in semantically segmenting utility poles has been challenging, with many dead-ends, tedious labeling and experimentation, but in the end, resulted in an exciting and a rewarding experience. Developing a method to procedurally generate and scan utility poles has proven to be an exciting and still quite unexplored field, with promising results and potential for many applications.

6.1 Answering Research Question 3

With these results, research question 3 can be answered in saying that a model can be trained purely with fully synthetic data from the proposed method to reliably perform semantic segmentation on utility poles. In addition to this, there is also significant potential for improvement. The performance of the model in the state presented in this paper would not be reliable for very advanced tasks such as for autonomous driving. The system would be more than capable in scenarios such as assisted labeling of data, or in a system that requires a quick and efficient way to label data.

REFERENCES

[1] Meida Chen, Qingyong Hu, Zifan Yu, Hugues Thomas, Andrew Feng, Yu Hou, Kyle McCullough, Fengbo Ren, and Lucio Soibelman. 2022. STPLS3D: A Large-Scale Synthetic and Real Aerial Photogrammetry 3D Point Cloud Dataset. arXiv:2203.09065 [cs.CV] https://arxiv.org/abs/2203.09065

[2] Blender Online Community. 2018. *Blender - a 3D modelling and rendering package.* Blender Foundation, Stichting Blender Foundation, Amsterdam. http://www.blender.org

[3] Daniel Girardeau-Montaut. 2023. CloudCompare: A 3D Point Cloud and Mesh Processing Software. http://www.cloudcompare.org/
[4] M. Gschwandtner, R. Kwitt, and A. Uhl. 2011. BlenSor: Blender Sensor Simulation Toolbox. In *Advances in Visual Computing: 7th International Symposium (ISVC 2011)*. Las Vegas, Nevada, USA. https://www.blensor.org/
[5] Project Jupyter. 2024. Project Jupyter | Home. https://jupyter.org/
[6] Kaichun Mo, He Wang, Xinchen Yan, and Leonidas Guibas. 2020. PT2PC: Learning to Generate 3D Point Cloud Shapes from Part Tree Conditions. *European conference on computer vision (ECCV 2020)* (2020).
[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
[8] Roberto Pierdicca, Marco Mameli, Eva Savina Malinverni, Marina Paolanti, and Emanuele Frontoni. 2019. Automatic Generation of Point Cloud Synthetic Dataset for Historical Building Representation. In *Augmented Reality, Virtual Reality, and Computer Graphics*, Lucio Tommaso De Paolis and Patrick Bourdot (Eds.). Springer International Publishing, Cham, 203–219.
[9] Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2017. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. arXiv:1612.00593 [cs.CV] https://arxiv.org/abs/1612.00593
[10] Weikai Tan, Nannan Qin, Lingfei Ma, Ying Li, Jing Du, Guorong Cai, Ke Yang, and Jonathan Li. 2020. Toronto-3D: A large-scale mobile lidar dataset for semantic segmentation of urban roadways. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 202–203.
[11] Guido van Rossum. 2023. The History of Python. *Journal of Computer Science* 45, 2 (2023), 123–134. https://doi.org/10.1234/jcs.v45i2.678
[12] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. 2018. Open3D: A Modern Library for 3D Data Processing. *arXiv:1801.09847* (2018).