University of Nevada, Reno

Deep Learning-Based Part Labeling of Tree Components in Point Cloud Data

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in
Computer Science and Engineering

by

Gunner Stone

Alireza Tavakkoli – Thesis Advisor
Jonathan Greenberg – Thesis Advisor
May 2023

UNIVERSITY OF NEVADA, RENO

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

GUNNER STONE

entitled

Deep Learning-Based Part Labeling of Tree Components in Point Cloud Data

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Alireza Tavakkoli, Ph.D.
Advisor

Emily Hand, Ph.D.
Committee Member

George Bebis, Ph.D.
Committee Member

Mircea Nicolescu, Ph.D.
Committee Member

Jonathan Greenberg, Ph.D.
Graduate School Representative

Markus Kemmelmeier, Ph.D., Dean
Graduate School

May, 2023

Abstract

Point cloud data analysis plays a crucial role in forest management, remote sensing, and wildfire monitoring and mitigation, necessitating robust computer algorithms and pipelines for segmentation and labeling of tree components. This thesis presents a novel pipeline that employs deep learning models, such as the Point-Voxel Transformer (PVT), and synthetic tree point clouds for automatic tree part-segmentation. The pipeline leverages the expertise of environmental artists to enhance the quality and diversity of training data and investigates alternative subsampling methods to optimize model performance. Furthermore, we evaluate various label propagation techniques to improve the labeling of synthetic tree point clouds. By comparing different community detection methods and graph connectivity inference techniques, we demonstrate that K-NN connectivity inference and carefully selected community detection methods significantly enhance labeling accuracy, efficiency, and coverage. The proposed methods hold the potential to improve the quality of forest management and monitoring applications, enable better assessment of wildfire hazards, and facilitate advancements in remote sensing and forestry fields.

Dedication

To my parents, Nelson and Sandy Stone, my brother, Gage Stone, and my partner, Cameron Whipple, for their unwavering support throughout the demanding journey of creating this work. I would also like to dedicate this thesis to my two pugs, Mandy and MacGyver, who have been by my side for a significant portion of my educational journey.

Acknowledgments

Supported by the National Science Foundation under Grant No. OIA-21487.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and motivation

Forests are critical components of the global ecosystem and play an essential role in sustaining life on Earth [12,13]. They provide a wide range of ecological services, such as carbon sequestration [14], biodiversity conservation [15], water regulation [16], and climate stabilization [13]. In addition to these benefits, forests offer economic advantages, including timber production, non-timber forest products, and recreational opportunities [12]. Therefore, effective management and preservation of these ecosystems are necessary to maintain their health and ensure the continued provision of these services.

Over the years, remote sensing technologies have revolutionized the study of forestry, allowing for detailed assessments of forest structure, composition, and dynamics. Light Detection and Ranging (LiDAR) is a remote sensing technology that uses laser pulses to measure distances to objects, such as vegetation and terrain. Terrestrial and Airborne Laser Scanning (TLS and ALS) are two applications of LiDAR technology that enable forestry professionals to digitally catalog entire landscapes in the form of large point cloud datasets [17].

LiDAR systems emit laser pulses that travel to the Earth's surface and reflect off objects, such as trees, ground, and buildings. The time it takes for the laser pulse to travel back to the sensor is recorded, and this information is used to calculate the distance to the object. By combining distance measurements with the precise location and orientation data of the LiDAR system, 3D point cloud datasets can be generated [17].

TLS involves mounting LiDAR sensors on a tripod to capture data from a stationary position, while ALS systems are mounted on aircraft to capture data over larger areas [17]. These datasets provide valuable, high-resolution information on the structure and composition of forest ecosystems at various scales, from individual trees to entire landscapes. The extraction of meaningful information from these datasets requires advanced processing techniques that can accurately identify and classify different components of the forest ecosystem. Specifically, the automated segmentation and labeling of individual trees and their parts, such as the trunk, branches, and leaves, are crucial for obtaining detailed information on topological, geometrical, and volumetric aspects of tree structures.

Part segmentation is particularly useful in several applications, including tree species identification, mensuration, and biomass estimation. For instance, the volume, shape, and branching patterns of different tree parts can be used to distinguish between species and characterize their growth patterns [18]. Additionally, part segmentation allows for the accurate measurement of tree parameters, such as tree height, diameter at breast height (DBH), and crown diameter, which are essential for forest mensuration and inventory. Furthermore, the volumetric information obtained from part segmentation can be utilized to estimate tree biomass and carbon sequestration potential, providing valuable insights for forest management, conservation, and climate change mitigation efforts [19]. These parameters are crucial for informing forest management decisions, assessing the health and productivity of forest ecosystems, and monitoring changes over time [20].

However, current deep-learning segmentation and labeling techniques for point cloud data often face limitations due to the fixed-size vector inputs used in batch training the segmentation models. Constraining the model by subsampling the input point cloud down to a fixed-size can result in inaccurate or incomplete labeling coverage, potentially reducing the reliability and utility of the derived information for forestry applications. Furthermore, manual segmentation and labeling of point cloud data are time-consuming, labor-intensive, and subject to human error, rendering them impractical for generating large-scale datasets.

Given these challenges, there is a pressing need for more effective and automated labeling techniques that can overcome the limitations of existing methods and enhance the accuracy and granularity of information obtained from TLS and ALS data. Such advancements hold the potential to significantly improve our understanding of forest ecosystems and provide valuable insights to support informed decision-making by forestry professionals.

1.2 Problem Statement

The challenges faced in the segmentation and labeling of tree components within point cloud datasets have necessitated the development of improved techniques for accurate and efficient analysis. Key issues include:

1. **Tree Part-Segmentation limited to RGB images:** Current techniques for tree part-segmentation, such as those in Amatya et al. [21] and Lin et al. [22],

primarily focus on using RGB images rather than point cloud data. While these techniques have shown success in their respective applications, they are limited in their ability to manage the unique challenges presented by point cloud datasets. Moreover, the reliance on RGB images for segmentation necessitates controlled environments, and labor-intensive data collection and manual label annotation, as seen in Lin et al. [22], where it took two months to create segmented labels manually.

2. **Lack of automation:** More traditional methods, like using point cloud data to reconstruct quantitative structure models (QSM) for trees, show promise in AGB estimation and part-segmentation [23]. However, these methods require someone proficient in manually fine-tuning the algorithm's input parameters to produce quality cylindrical estimates of the tree's topology. Depending on the size and complexity of the input tree, QSM may also take a large inference time to calculate and produce results [24], which may not be feasible in an application at-scale.

3. **Scalability and adaptability:** Current techniques may not scale well for large datasets or adapt to the wide variety of tree species and forest types that are encountered in real-world applications. Point cloud data collected from terrestrial and airborne remote sensing platforms often contain noise and occlusions due to the complex geometry of tree canopies and forest environments. These issues can significantly impact the accuracy and reliability of segmentation and labeling results.

Addressing these challenges is essential for developing a more robust and efficient approach to segmentation and labeling of tree components within point cloud datasets. This will enhance the quality of forest management and monitoring applications, as well as facilitate advancements in the field of remote sensing and forestry.

1.3 Objectives and scope

The primary objectives of this research are to develop and evaluate novel techniques for point cloud segmentation and labeling in the context of forestry applications, with a specific focus on addressing the challenges of accuracy, automation, scalability, and adaptability.

To achieve these objectives, the thesis encompasses the following key components:

1. Evaluation of deep learning point cloud segmentation models: The potential benefits of deep learning techniques for point cloud segmentation in forestry applications will be assessed through the training, evaluation, and comparison of several state-of-the-art models, including Pointnet, Pointnet++, ShellNet, DGCNN, and PVT. These models will be evaluated in terms of their performance and effectiveness in segmenting tree components.

2. Enhancement of point cloud segmentation using synthetic tree point clouds: A novel approach to improving point cloud segmentation and labeling is proposed by generating synthetic tree point clouds using SpeedTree software. These synthetic point clouds will be used to train and evaluate deep learning models and enhance the performance of the K-Neighbor-Nurtured-Garden (KNNG) algorithm through label propagation and transfer learning techniques.

3. Development of the KNNG algorithm: The KNNG algorithm is a graph-based approach that enables the extension of point cloud segmentation outcomes to the broader unlabeled points within the dataset. The KNNG algorithm leverages local neighborhood information and connectivity patterns to help identify distinct tree components such as trunks, branches, and leaves. The design, implementation, and evaluation of the KNNG algorithm will be thoroughly detailed in this study.

The objectives and scope of this thesis are to develop and evaluate novel techniques for point cloud segmentation and labeling in the context of forestry applications, with a specific focus on addressing the challenges of accuracy, automation, scalability, and adaptability. This work aims to make significant contributions to the field of point cloud segmentation and labeling in forestry applications, providing valuable insights and practical solutions that can be readily applied to real-world scenarios.

Chapter 2

Background

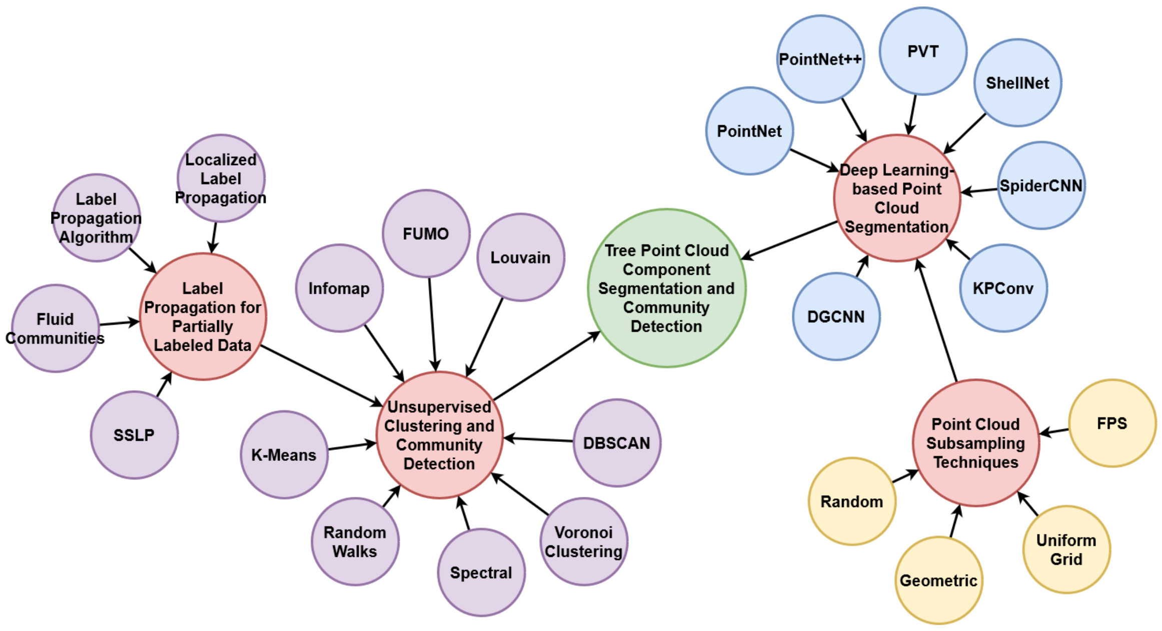
PointNet++
PVT
ShellNet
PointNet
Deep Learning-based Point Cloud Segmentation
SpiderCNN
Localized Label Propagation
Label Propagation Algorithm
FUMO
Louvain
Tree Point Cloud Component Segmentation and Community Detection
KPConv
DGCNN
Infomap
Label Propagation for Partially Labeled Data
Fluid Communities
SSLP
K-Means
Unsupervised Clustering and Community Detection
DBSCAN
Point Cloud Subsampling Techniques
FPS
Random
Random Walks
Spectral
Voronoi Clustering
Geometric
Uniform Grid

Figure 2.1: Concept map of the main topics discussed in the background section, including deep learning-based point cloud segmentation, point cloud subsampling techniques, unsupervised clustering, and label propagation for partially labeled data.

This section provides a comprehensive overview of the related works and techniques in the field of point cloud segmentation and community detection, focusing on their applications in forestry and related domains. The topics discussed encompass several critical themes, including deep learning-based point cloud segmentation, point cloud subsampling techniques, unsupervised clustering and community detection, label propagation for partially labeled data, and the challenges and opportunities in point cloud segmentation.

To provide context and facilitate understanding, Figure 2.1 presents a concept map illustrating the main themes and interrelations between the various topics discussed in this section. This figure helps to emphasize the connections between the different techniques and their applications in the field of point cloud segmentation and community detection.

In the following subsections, the details of each of these topics are explained in detail, exploring the relevant literature and techniques that have been employed in the field of point cloud segmentation and community detection.

2.1 Deep Learning-based Point Cloud Segmentation

Deep learning techniques have emerged as powerful tools in the field of point cloud segmentation, leading to significant advancements in the analysis and interpretation of complex 3D data. Several deep learning-based approaches have been proposed to address various challenges associated with point cloud segmentation, with some of the most notable methods outlined below:

2.1.1 PointNet [1]

PointNet is a pioneering deep learning approach for processing and analyzing point clouds. It is designed to handle the unique challenges of unordered and irregular 3D point data, focusing on processing individual points while capturing local information. The architecture of PointNet consists of several key components:

1. **Input Transform**: PointNet employs a learned input transformation network to align the input point cloud to a canonical space, reducing the impact of different orientations and improving the network's robustness.

2. **Shared MLPs**: PointNet uses shared multi-layer perceptrons (MLPs) to extract point-wise features from the input point cloud. The shared weights in these MLPs ensure that the network is invariant to the order of the input points.

3. **Symmetric Function**: To aggregate information from the unordered points, PointNet utilizes a symmetric function (e.g., max-pooling) that combines the features of all points in the point cloud. This allows the network to extract global features from the point cloud, which are essential for tasks such as object classification, segmentation, and scene understanding.

By processing point clouds in this manner, PointNet is able to achieve robust performance across a variety of tasks, including object classification, segmentation, and scene understanding. The network's ability to process individual points and capture local information makes it particularly well-suited for point cloud analysis. However, PointNet's primary limitation is its reliance on global features, which may not be sufficient for capturing fine-grained local structures in complex point clouds. This has led to the development of several extensions and improvements, such as PointNet++ [2], which incorporate hierarchical structures and local feature extraction to address these limitations.

2.1.2 PointNet++ [2]

PointNet++ builds upon the concepts introduced by PointNet and enhances its ability to capture local geometric structures in addition to global features. The core idea behind PointNet++ is to apply a hierarchical neural network structure that processes point clouds at multiple scales by recursively applying PointNet on nested local regions. This hierarchical approach allows the network to capture both local and global features, resulting in improved segmentation performance.

The PointNet++ architecture consists of the following key components:

1. **Sampling and Grouping:** The point cloud is partitioned into overlapping local regions by applying a sampling strategy, such as Farthest Point Sampling (FPS). Each local region is formed by grouping neighboring points around the sampled points. This process can be performed in multiple hierarchical layers, with each layer focusing on a different spatial scale.

2. **Local Feature Extraction:** Within each local region, PointNet is applied to extract local features. This is achieved by aggregating information from the unordered points using a symmetric function, such as max-pooling, which enables the extraction of meaningful local geometric features.

3. **Hierarchical Processing:** The local features extracted from each region are processed hierarchically. This process refers to the organization and processing of point cloud data at multiple levels of granularity. At each level of the hierarchy, the features are aggregated and combined to produce higher-level features, which capture the relationships between the local regions and their spatial context.

By capturing both local and global features in a hierarchical manner, PointNet++ is better equipped to model complex geometric structures found in point clouds. This leads to improved performance in various point cloud processing tasks, including object classification, semantic segmentation, and scene understanding. However, similar to PointNet, PointNet++ also requires a fixed input size, which may necessitate heavy subsampling of the point cloud data, potentially leading to information loss and impacting the quality of the segmentation results.

2.1.3 SpiderCNN [3]

SpiderCNN is another deep learning-based approach for processing point clouds, introducing a new convolution operator called the spider convolution. The main components and advantages of SpiderCNN include:

1. **Spider Convolution Operator:** While traditional convolution operators are typically designed for grid-like data, the spider convolution operator was created to specifically work with point cloud data. The operator generates learnable radial basis functions (RBFs) and directional functions. RBFs capture point distances, while directional functions model orientation relationships between points. This combination allows the spider convolution to capture local geometric information within point neighborhoods. SpiderCNN processes local point neighborhoods by applying the spider convolution operator on them.

2. **Adaptive Kernel Weights:** SpiderCNN's convolution operator adapts kernel weights to account for the varying density of points in the point cloud. This results in a more robust feature representation, less affected by noise or irregular sampling. The adaptive kernel weights enable SpiderCNN to perform well even with missing or noisy data.

3. **Hierarchical Feature Learning:** SpiderCNN, similar to PointNet++, employs a hierarchical structure for feature learning. This structure allows the model to learn multi-scale features, capturing both fine-grained local details and global context. Leveraging this hierarchical structure, SpiderCNN can better represent complex geometric structures.

SpiderCNN introduces the spider convolution operator which enables the model to uniquely capture geometric features using a convolution-like operator. Through local point neighborhood processing and hierarchical feature learning, SpiderCNN achieves comprehensive feature representations.

2.1.4 ShellNet [4]

ShellNet is a deep learning-based method for processing point clouds, which introduces the shell partitioning method and the multi-scale spherical convolution. The main components and advantages of ShellNet include:

1. **Shell Partitioning:** ShellNet utilizes a partitioning method that divides the point cloud into concentric shells. This process, called shell partitioning, helps the model capture the spatial distribution of points effectively. Each shell contains points that are within a certain distance range from the center of the point cloud. This partitioning method enables the model to better understand the local and global structure of the point cloud.

2. **Multi-Scale Spherical Convolution:** ShellNet introduces a multi-scale spherical convolution that operates on the shell partitioned point cloud. This convolution captures geometric features from different spatial scales, allowing the model to learn more comprehensive feature representations. The multi-scale spherical convolution is specifically designed to work with point cloud data, making it effective for handling irregularly sampled and unordered points.

3. **Hierarchical Feature Learning:** Similar to SpiderCNN and PointNet++, ShellNet adopts a hierarchical structure for feature learning. This structure enables the model to learn multi-scale features, capturing both fine-grained local details and global context. By leveraging the hierarchical structure and

multi-scale spherical convolution, ShellNet can effectively represent complex geometric structures within the point cloud.

ShellNet's shell partitioning and local-to-global hierarchical learning strategy provide an effective method for processing point clouds with varying densities and irregular structures. By utilizing the shell partitioning to capture the spatial distribution of points and incorporating a multi-resolution hierarchy, ShellNet demonstrates strong performance in handling complex geometries and capturing relevant features at multiple scales.

2.1.5 DGCNN [5]

DGCNN, or Dynamic Graph CNN, is a deep learning-based approach for processing point clouds. It introduces a dynamic graph-based structure to capture local geometric features effectively. The main components and advantages of DGCNN include:

1. **Dynamic Graph Construction:** DGCNN constructs a dynamic graph for each point cloud by connecting each point to its k nearest neighbors. The graph is dynamically updated for each layer in the network, allowing the model to adapt and capture different levels of geometric detail. This dynamic graph construction enables DGCNN to better understand the local and global structure of the point cloud.

2. **EdgeConv:** DGCNN introduces a novel convolution operation called EdgeConv, which operates on the edges of the dynamic graph. EdgeConv is specifically designed to work with point cloud data, making it effective for handling irregularly sampled and unordered points. By capturing both local and non-local geometric features, EdgeConv allows DGCNN to learn more comprehensive feature representations.

3. **Hierarchical Feature Learning:** Like SpiderCNN, PointNet++, and ShellNet, DGCNN employs a hierarchical structure for feature learning. This structure enables the model to learn multi-scale features, capturing both fine-grained local details and global context. By leveraging the hierarchical structure and EdgeConv, DGCNN can effectively represent complex geometric structures within the point cloud.

DGCNN's dynamic graph construction and EdgeConv operation provide a unique and adaptive approach to point cloud processing, enabling the model to effectively capture complex geometric features in various contexts. By employing a hierarchical structure and dynamically updating the graph at each layer, DGCNN excels in representing local and global structures while adapting to different levels of detail within the point cloud.

2.1.6 KPConv [6]

Kernel point convolution (KPConv), processes point clouds by directly working with points in 3D space. The convolutional operator in KPConv differs from that of SpiderCNN in that it directly applies convolutional operations on local point neighborhoods in 3D space using kernel functions defined over pairs of points, whereas SpiderCNN extends traditional convolution operators with a spider convolution operator. Kernel functions are defined over pairs of points in a point cloud. These kernel functions are then used to compute the convolutional output for each point in the local neighborhood.

One of the advantages of KPConv is its flexibility, which allows the method to adapt to various data distributions and densities. This adaptability is particularly beneficial when working with point cloud data that may be irregularly sampled or contain high variance point densities. KPConv has demonstrated great performance in point cloud segmentation tasks, showcasing the effectiveness of this approach in extracting and processing relevant features from point cloud data.

KPConv scales well with large point clouds. This stems from its ability to leverage data structures like k-d trees and octrees, which can significantly speed up the nearest-neighbor search required for processing local point neighborhoods.

While these point-based deep learning-based methods have achieved remarkable performance in semantic segmentation of point clouds, several additional techniques have been developed to address the challenges of 3D object detection and segmentation using a voxel-based approach for a more discrete and compact representation of the input-space. These methods include the following:

2.1.7 VoxelNet [7]

VoxelNet divides the input point cloud into a 3D grid of voxels, with each voxel encompassing a specific region in the 3D space. Features of the points within each voxel are summarized, effectively discretizing the local geometric information within each voxel. This process reduces the complexity of the input data, allowing for a more efficient analysis while still preserving essential structural information.

3D convolutional layers are used to extract both local and global features from the data. These layers are capable of capturing the spatial relationships between the voxels, contributing to the model's ability to recognize and segment different objects within the point cloud.

2.1.8 SECOND [8]

SECOND is a deep learning approach for point cloud processing that shares similarities with VoxelNet in terms of utilizing a voxel representation for point clouds.

However, SECOND introduces a sparse convolutional operation to further improve efficiency and computational performance. Sparse convolutions aim to only process non-empty voxels in the 3D grid.

2.1.9 Point-Voxel Transformer (PVT) [9]

Point-Voxel Transformer (PVT) merges the strengths of point-based and voxel-based representations in order to effectively process and analyze point clouds. This hybrid method aims to capitalize on the feature-space provided by point-based techniques and the computational efficiency provided by voxel-based methods. Consequently, PVT is capable of delivering high-quality segmentation results while maintaining adaptability to different point cloud densities and structures.

The PVT architecture is built upon a hierarchical structure that successively processes and down-samples the input point cloud data at multiple levels of granularity. At each level, the point cloud is transformed into a voxel representation, which is then used to extract local features. These voxelized features are subsequently fed into a transformer module, which is responsible for capturing and encoding both local and global context information within the point cloud.

The transformer module in PVT is designed to manage irregular and unordered point cloud data by incorporating self-attention mechanisms. This allows the model to dynamically weight the importance of different local features, enabling it to learn and adapt to the underlying structure of the point cloud. The output of the transformer module is then decoded back into a point-based representation. By employing a hierarchical processing structure alongside the transformer module, PVT effectively combines the benefits of both point-based and voxel-based techniques.

Additionally, many of these models also suffer from a significant limitation related to the fixed input size of the model. This constraint often requires heavy subsampling of the point cloud data, which can lead to information loss and potentially compromise the quality of the segmentation results.

While the deep learning-based techniques mentioned earlier have shown impressive performance on well-known point cloud datasets such as ModelNet40 [25], ShapeNet [26], and S3DIS [27], their applicability to the segmentation of complex tree point clouds may be limited due to several factors. A primary concern is the stark difference between the point cloud objects typically found in standard benchmark datasets typically used to evaluate these models and the complexity of tree point clouds. The structures found in tree point clouds are inherently more intricate and challenging to segment compared to the objects commonly present in standard benchmark datasets like ModelNet40, ShapeNet, and S3DIS. Additionally, the objects present in these datasets are able to take advantage of priors that a pointcloud of a tree cannot use, such as morphological variance. As a result, the performance of these deep learning-based techniques on tree point clouds may not be as robust as their reported performance on other datasets. This highlights the need for more targeted research and development efforts that specifically address the unique challenges associated with tree point cloud segmentation.

This thesis investigates a range of deep learning-based point cloud models and evaluates their performance on tree point clouds to address concerns regarding their effectiveness. The investigation aims to draw inferences about the factors contributing to superior performance of certain models in capturing the intricacies of tree point cloud structures. By identifying and understanding the essential attributes of these methods that enable them to handle the unique challenges presented by tree point clouds, this work aims to advance the state of the art in forestry applications and offer practical insights for developing effective tree point cloud segmentation techniques.

2.2 Point Cloud Subsampling Techniques

As mentioned in the previous section, deep learning models for point cloud processing rely on efficient subsampling techniques to deliver accurate results. These techniques play a crucial role in handling large point cloud datasets by reducing their size while preserving key structural and geometric information. Effective subsampling not only enhances the processing efficiency but also minimizes storage and computational requirements, ultimately improving the performance of the deep learning models discussed earlier.

Point cloud subsampling techniques can be classified into different categories depending on their approach and the specific application requirements. This section provides an overview of various subsampling methods, including Random sampling, Farthest Point Subsampling, Uniform Point Sampling, Grid Subsampling, and Geometric subsampling, and discusses their advantages and drawbacks in the context of deep learning-based point cloud processing.

2.2.1 Random Subsampling

Random subsampling is a simple and computationally efficient technique for reducing the size of point cloud datasets. The algorithm operates by randomly selecting a subset of points from the original point cloud without replacement. The process is as follows:

1. Determine the desired number of points in the subsampled point cloud.
2. Randomly select points from the original point cloud without replacement until the desired number of points is reached.

Random subsampling works well when the original point cloud has a uniformly distributed density of points. In such cases, the randomly selected points are likely to capture the overall structure of the tree point cloud effectively. However, when point density has high variance throughout the tree due to occlusions, random subsampling may fail to capture the tree's structure accurately. In areas with high point density, random subsampling may still select a disproportionately large number of points, while sparser regions may be underrepresented in the subsampled point cloud. This can lead to information loss and compromise the quality of the subsequent analysis.

Despite these limitations, random subsampling remains attractive for its computational efficiency. The algorithm has a complexity of $O(1)$, as it involves selecting points from the point cloud without regard to their order or position in the cloud. Each point in the cloud has an equal probability of being selected, and the number of points selected does not depend on the size of the cloud. However, when dealing with tree point clouds with varying point densities, alternative subsampling techniques or density-aware adaptations of the random subsampling algorithm may need to be considered to ensure accurate representation and analysis of the tree structure.

2.2.2 Farthest Point Subsampling (FPS) [10]

Farthest Point Subsampling (FPS) is a technique used for reducing the size of point cloud datasets while aiming to preserve the overall geometric structure. The algorithm works as follows:

1. Initialize the subsampled point cloud by selecting an arbitrary point from the original point cloud.
2. Calculate the Euclidean distances between the selected point and all remaining points in the original point cloud.
3. Select the point that is farthest from the previously selected point and add it to the subsampled point cloud.

4. Repeat steps 2 and 3 until the desired number of points in the subsampled point cloud is reached.

The FPS algorithm tends to select points that are farther apart, which attempts to query a more uniform distribution of points. This can be particularly useful in certain contexts but fails to capture the geometric structure of tree point clouds. FPS exhibits a surface bias, which means that it tends to select points on the surface of the tree rather than within the internal branchy substructure. This occurs because the points on the tree's surface have larger distances between them compared to the points within the branchy substructure. The surface bias may limit the effectiveness of FPS when applied to tree point cloud datasets, especially in applications that require accurate representation and analysis of the internal branch structure. To overcome this limitation, alternative subsampling techniques or modifications to the FPS algorithm should be considered.

The computational complexity of the FPS algorithm is primarily determined by the number of distance calculations and iterations required to construct the subsampled point cloud. For a point cloud with n points and a desired subsample size of k, the algorithm requires $O(nk)$ distance calculations. This can be computationally expensive, particularly for large datasets.

Several optimizations and approximations can be applied to the FPS algorithm to reduce its computational complexity. For example, the use of spatial data structures, such as k-d trees or octrees, can speed up the nearest neighbor search, potentially reducing the complexity to $O(n \log k)$. Additionally, approximate FPS techniques can be employed to trade off some accuracy in point selection for reduced computation time.

2.2.3 Uniform Point Sampling

Uniform Point Sampling is a technique used for reducing the size of point cloud datasets while attempting to maintain a uniform distribution of points across the entire point cloud. This method aims to achieve better coverage of the geometric structure and capture more representative information about the tree point clouds. The algorithm works as follows:

1. Define a grid with a fixed cell size, which covers the extent of the input point cloud.
2. Assign each point in the input point cloud to its corresponding grid cell.
3. For each non-empty grid cell, select one point (either randomly or based on a specific criterion) and add it to the subsampled point cloud.

Uniform Point Sampling can effectively create a more evenly distributed subsampled point cloud, which is beneficial for tree point cloud analysis. The method helps to retain points from different regions of the tree, including branches, trunks, and leaves, by ensuring that points are sampled uniformly across the entire spatial extent of the tree point cloud.

The computational complexity of the Uniform Point Sampling algorithm primarily depends on the number of points in the input point cloud, the size of the grid cells, and the spatial distribution of the points. The complexity can be reduced by employing efficient data structures, such as spatial hashing or octrees, which enable faster point-to-cell assignment and cell traversal. One potential limitation of Uniform Point Sampling is the selection of the grid cell size, which can have a significant impact on the quality of the subsampled point cloud. If the grid cell size is too large, important geometric features may be missed, while if it is too small, the subsampled point cloud may still contain a large number of points, reducing the efficiency of the subsampling process.

2.2.4 Geometric Point Sampling [11]

Geometric Point Sampling is a subsampling technique that aims to reduce the size of point cloud datasets while preserving the local geometric structure and characteristics of the original data. This method is based on the PointCleanNet algorithm proposed by Rakotosaona et al. [11]. The algorithm works as follows:

1. Estimate the local geometric structure for each point in the point cloud by calculating the local surface variation (LSV), which is a measure of the surface curvature.
2. Assign a sampling probability to each point based on its LSV, with higher probabilities assigned to points with higher LSV values.
3. Sample points from the input point cloud according to their assigned probabilities, resulting in a subsampled point cloud that emphasizes geometric features with higher curvature.

Geometric Point Sampling has the advantage of capturing and preserving the fine geometric details in tree point clouds, which is particularly useful for deep learning applications that require accurate representations of branch, trunk, and leaf structures. By emphasizing points with higher curvature, this method ensures that the critical geometric features of the tree point cloud are retained in the subsampled dataset.

The computational complexity of the Geometric Point Sampling algorithm is primarily determined by the calculation of the LSV values for each point in the input point cloud and the subsequent sampling step. One potential limitation of Geometric

Point Sampling is that the LSV values might be sensitive to noise in the input point cloud data. Because of this, additional denoising preprocessing steps need to be taken to assure noisy points, or outliers, are removed before subsampling.

2.3 Unsupervised Clustering and Community Detection

Unsupervised clustering and community detection techniques are widely employed to analyze complex data structures, such as point clouds, by grouping similar data points together based on their features or relationships. These methods help reveal underlying patterns in the data, facilitating more effective analysis and interpretation. This section provides an in-depth review of traditional unsupervised clustering approaches and popular community detection techniques that can be adapted for clustering partially segmented point clouds.

2.3.1 Traditional Unsupervised Clustering Approaches

Traditional unsupervised clustering algorithms typically focus on grouping data points based on their spatial proximity or similarity in feature space. Some notable clustering methods include:

- **Voronoi clustering:** This method partitions the data space into regions (called Voronoi cells) around the input data points, where each region contains all the points closer to the corresponding data point than any other point. Voronoi clustering is often used for nearest-neighbor search and mesh generation in various applications, such as computer graphics and computational geometry.

- **K-means clustering:** A widely used partitioning-based clustering algorithm that iteratively assigns data points to K centroids based on the Euclidean distance between the data points and the centroids. The algorithm seeks to minimize the sum of squared distances between the data points and their assigned centroids.

- **DBSCAN [28]:** A density-based clustering algorithm that groups points with high spatial density together, while treating points in low-density regions as noise. DBSCAN forms clusters by expanding dense regions, as it connects points that are close to each other according to a distance metric and a density threshold.

- **Random Walks and Efficient Graph-based Segmentation (EGS) [29]:** This approach employs graph representations to segment the point cloud data. In EGS, each data point is considered a node in the graph, and edges are defined based on the similarity between the nodes. The algorithm then constructs a minimum spanning tree of the graph and merges nodes based on edge weights, resulting in the final segmentation.

2.3.2 Community Detection Techniques

Community detection algorithms aim to identify densely connected groups of nodes within a network or graph, where the nodes inside a community have more connections with each other than with nodes outside the community. These techniques have been extensively studied in various domains and can be adapted for clustering partially segmented point clouds. Some widely used community detection methods include:

- **Louvain algorithm [30]:** This algorithm iteratively optimizes the modularity score of the network to determine communities. The algorithm initially assigns each node to its own community and then iteratively merges communities to maximize the modularity gain until no further improvement is possible.

- **Infomap [31]:** A flow-based algorithm that minimizes the description length of a random walk on the network. The algorithm encodes the random walk as a hierarchical map, with the objective of finding the optimal partitioning of the network that results in the shortest possible description length.

- **Spectral algorithm [32]:** Spectral algorithms employ the eigenvectors of the graph Laplacian matrix to efficiently identify optimal partitioning within a network. By capitalizing on the inherent properties of the Laplacian matrix, these algorithms effectively uncover information about the community structure within the network. However, calculating the eigenvectors of the Laplacian does not scale linearly, which can limit the algorithm's applicability to large graphs, as the computational complexity increases significantly for larger datasets. To address this challenge, the Lanczos method [33] can be utilized for computing eigenvectors more efficiently, as it is an iterative method specifically designed for symmetric matrices like the graph Laplacian. This approach can potentially improve the scalability and efficiency of the spectral algorithm for large graphs.

- **FUMO algorithm [34]:** FUMO (Fast Unfolding of Modular Organization) is an extension of the Louvain algorithm that offers increased efficiency and is able to handle larger networks. The algorithm operates similarly to the Louvain method, with iterative optimization of modularity, but incorporates additional refinements to improve the partitioning process and convergence speed.

FUMO and other community detection techniques have been applied across a wide range of applications, including: community detection in social networks [35] gene co-expression networks [36], and brain functional networks [37].

The motivation behind exploring these clustering methods is to investigate their effectiveness in propagating partial labels throughout the tree point cloud. By leveraging the partially labeled point clouds generated by deep learning-based models, these techniques can be used to propagate labels to neighboring points through modularity maximization or graph-based techniques, potentially leading to improvements in both the coverage, accuracy, and efficiency of tree point cloud segmentation.

2.4 Label Propagation for Partially Labeled Data

Label propagation algorithms provide a powerful approach to clustering and classification in partially labeled data. These methods are particularly useful in scenarios where obtaining complete ground truth labels is difficult or expensive. By exploiting the available labeled data, label propagation algorithms can effectively spread labels throughout the dataset, resulting in the labeling of previously unlabeled points. This section presents an in-depth review of various label propagation algorithms and their potential applications in labeling partially segmented point clouds.

2.4.1 Overview of Label Propagation Algorithms

Label propagation algorithms typically operate on graphs, where nodes represent data points, and edges capture the relationships or similarities between these points. The general idea is to propagate the labels from labeled nodes to their neighbors, iteratively updating the labels until convergence is achieved or a stopping criterion is met. Some notable label propagation algorithms include:

- **Label Propagation Algorithm (LPA) [38]:** LPA is a simple and efficient method for detecting communities in large networks. The algorithm assigns an initial label to each node and updates the labels by adopting the majority label in its neighborhood during each iteration. The process is repeated until convergence or a maximum number of iterations is reached.

- **Semi-Synchronous Label Propagation Algorithm (SSLP) [39]:** SSLP is an extension of LPA that introduces a semi-synchronous update mechanism to improve the stability and performance of the original method. In SSLP, nodes are updated in a random order, and a delayed update strategy is employed to prevent oscillations between label assignments. SSLP has been shown to outperform other popular community detection algorithms in terms of accuracy and efficiency, particularly in the context of social networks.

- **Fluid Communities [40]:** Fluid Communities is a highly scalable and competitive community detection algorithm inspired by fluid dynamics. The algorithm models each community as a fluid that flows through the network, occupying nodes based on their capacity and the fluid's density. The method iteratively updates node assignments until a stable state is reached, resulting in the final community structure. Fluid Communities has demonstrated strong performance in large-scale networks, offering a promising solution for handling massive datasets.

- **Localized Label Propagation (LLP) [41]:** LLP is a label propagation algorithm designed for semi-supervised learning in large-scale, high-dimensional data. The method employs a localized propagation strategy, focusing on propagating labels only within local neighborhoods. This approach reduces the computational complexity of the algorithm while maintaining high classification accuracy, making it well-suited for processing large datasets.

2.4.2 Potential Applications to Partially Segmented Point Clouds

Label propagation algorithms hold promise for labeling partially segmented point clouds by leveraging the available labeled points as seeds and propagating the labels to neighboring unlabeled points to generate clusters. The iterative nature of these algorithms allows them to gradually refine the label assignments, ultimately converging to a stable labeling configuration.

The application of label propagation techniques to tree point cloud segmentation could potentially benefit from the incorporation of domain-specific knowledge or additional feature information to guide the propagation process. For example, incorporating spatial or geometric features of the tree point clouds can help improve the accuracy of label assignments by ensuring that labels are propagated along regions with similar characteristics. Additionally, the use of adaptive weighting schemes or dynamic update mechanisms can further enhance the performance of these methods by accounting for variations in point cloud density or distribution.

By adapting and extending label propagation algorithms to the context of tree point cloud segmentation, this thesis aims to explore the effectiveness of these methods in addressing the challenges posed by partially labeled data. Specifically, the aim is to evaluate label propagation methods that can effectively utilize the partially labeled point clouds produced by deep learning models to propagate labels to the remaining unlabeled points. By doing so, this approach seeks to address the limitations of deep learning models related to their fixed input size and the need for heavy subsampling, which can result in loss of information and poorer segmentation outcomes.

2.5 Challenges and Opportunities

The application of deep learning-based point cloud segmentation, unsupervised clustering, community detection techniques, and label propagation algorithms to the analysis of tree point clouds presents several challenges and opportunities. This section discusses these challenges and opportunities in detail and highlights potential avenues for future research and development.

2.5.1 Challenges in Point Cloud Segmentation

1. **Variability in tree point clouds:** Tree point clouds can exhibit significant variations in terms of density, scale, and complexity due to a variety of factors, including the age of the tree, its health, occlusions in scans, different tree species, and the use of different scanning devices. Trees at different growth stages, for instance, may have distinct structural and geometrical characteristics, while the health of a tree can affect its overall appearance and shape. Occlusions in scans, often resulting from the presence of leaves, branches, or other obstacles, can create gaps or inconsistencies in the point cloud data. Furthermore, different tree species may possess unique features and branching patterns, adding to the variability of the point cloud topology. Additionally, the use of different scanning devices with varying resolution and accuracy can introduce further variations in the data. These factors pose challenges for the design and evaluation of segmentation and clustering algorithms, as the methods must be adaptable to different data characteristics while maintaining accurate and robust performance.

2. **Noise and outliers:** Point cloud data may contain noise and outliers, which can arise from measurement errors, environmental factors, or other anomalies. These issues can negatively impact the performance of segmentation algorithms

and may require additional preprocessing steps to mitigate their effects. Developing robust approaches that are less sensitive to noise and can better handle irregularities in the data will be essential for achieving accurate and reliable segmentation results.

3. **Large-scale point clouds:** Forestry applications often involve large-scale datasets containing millions of points, which can be computationally demanding for segmentation and clustering techniques. Managing such large datasets can be computationally expensive and may necessitate the use of subsampling or other techniques to reduce the data size for processing. However, these methods may lead to a loss of information and reduced segmentation accuracy. Developing efficient algorithms capable of handling these large-scale datasets is a critical challenge in this context.

4. **Limited labeled data:** Obtaining labeled data for tree point clouds can be time-consuming, labor-intensive, and monetarily expensive. As a result, there is a scarcity of standardized, high-quality, and open-source labeled data available for training and evaluation of segmentation algorithms. While some datasets might exist, they are often not widely used in the literature, which poses challenges in terms of benchmarking and comparing different methods. Furthermore, the limited availability of labeled data for specific tree species of interest can make it difficult to develop and assess algorithms tailored to particular species or ecosystems. This limitation can hinder the development, assessment, and generalizability of new methods in tree point cloud segmentation.

5. **Partial labels:** Deep learning-based point cloud segmentation models often generate partially labeled point clouds due to their fixed input size and subsampling requirements. Leveraging these partial labels effectively for clustering

and label propagation presents a challenge, as the algorithms must balance between utilizing the available labeled data and accounting for the potential inaccuracies in the labels.

6. **Integration of diverse techniques:** Achieving significant label-coverage, highly accurate, and efficient tree point cloud analysis requires the effective integration of various methods, such as deep learning-based segmentation, unsupervised clustering, community detection, and label propagation. Determining the optimal combination and order in which to apply these techniques presents a significant challenge. This task demands a comprehensive understanding of the strengths and weaknesses of each method to identify the most suitable approach for analyzing tree point clouds.

2.5.2 Opportunities for Improvement

1. **Transfer learning and domain adaptation:** Developing methods to transfer knowledge from related domains or pre-trained models can help overcome the limited availability of labeled data. In this thesis, a synthetically generated dataset of tree point clouds will be heavily utilized to conduct most of the research. The goal is to train a model on synthetic trees and then use it to partially label real trees, which can then be fed back into the model for further training and adaptation. This process enables the model to transfer its knowledge from the synthetic tree domain to the real tree domain, effectively bridging the gap between the two. By leveraging transfer learning and domain adaptation techniques, it may be possible to improve the performance of segmentation algorithms on tree point clouds with limited labeled data, while simultaneously addressing the challenges associated with dataset variability and tree species-specific requirements. This approach offers the potential to acceler-

ate the development and deployment of effective tree point cloud segmentation methods, ultimately benefiting forestry applications and research.

2. **Efficient processing of large-scale datasets:** There is a need to develop and improve algorithms for tree point cloud analysis that can efficiently process large-scale forestry datasets. Addressing this need is vital, as existing methods may demonstrate high performance on subsampled point clouds but struggle to scale when applied to extensive datasets, such as entire forests. Developing scalable and efficient algorithms will benefit various applications, such as forest inventory, tree species identification, and biomass estimation.

3. **Adaptable methods:** Designing segmentation and clustering techniques that are adaptable to the variations in tree point cloud data can lead to more robust and accurate solutions. These methods could be applied to different tree species, point cloud densities, and other variations encountered in real-world forestry applications.

By addressing these challenges and capitalizing on the opportunities, this thesis aims to advance the state of the art in tree point cloud analysis and provide practical solutions for forestry applications. Through the development and evaluation of innovative algorithms and the integration of diverse techniques, the work presented here seeks to contribute to a deeper understanding of the strengths and limitations of various methods and their potential applications in the context of tree point cloud segmentation and analysis.

Chapter 3

Methodology and Technical Details

3.1 Overview of the Proposed Framework

The proposed framework seeks to address the challenges of tree point cloud segmentation by focusing on segmenting tree components, specifically the trunk, branches, and leaves, rather than identifying individual trees within a cloud. The framework combines state-of-the-art techniques from both supervised and unsupervised learning in a multi-stage methodology, which includes synthetic dataset generation, data preprocessing and subsampling, deep learning-based segmentation, and unsupervised clustering with community detection.

Initially, a synthetic tree point cloud dataset is generated using a combination of SpeedTree, L-Systems [42], and re-optimization techniques. SpeedTree is a powerful vegetation modeling and rendering software that enables the creation of realistic trees, while L-Systems, or Lindenmayer Systems, are formal grammars used for modeling the growth processes of plant development. This dataset serves as the basis for training and evaluating the deep learning models.

The raw tree point cloud data then undergo preprocessing and subsampling to facilitate efficient and accurate segmentation. This process is outlined at a high level in Fig 3.1.

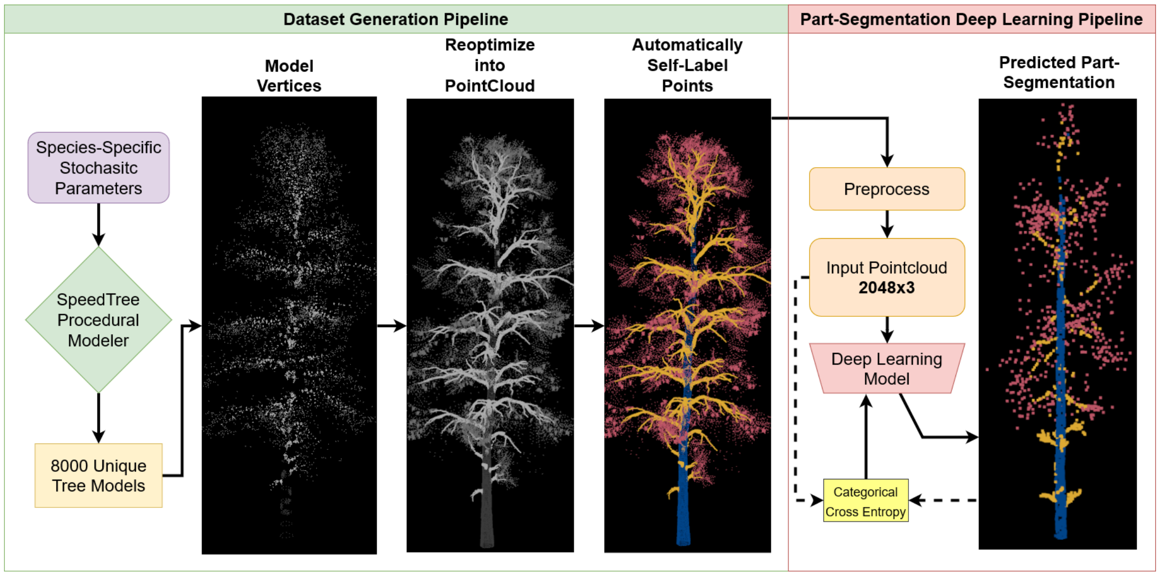
Dataset Generation Pipeline
Part-Segmentation Deep Learning Pipeline
Model Vertices
Reoptimize into PointCloud
Automatically Self-Label Points
Predicted Part-Segmentation
Species-Specific Stochasitc Parameters
SpeedTree Procedural Modeler
8000 Unique Tree Models
Preprocess
Input Pointcloud 2048x3
Deep Learning Model
Categorical Cross Entropy

Figure 3.1: Pipeline showing the steps to transform the original SpeedTree object into a preprocessed subsampled input ready for model inference. The Deep Learning Model block separately represents each of the five investigated models for this part-segmentation task. Each model uses categorical cross-entropy as the loss function responsible for driving the models to achieve optimal performance.

The next stage involves segmentation of tree components using deep learning models, with a focus on selecting the most appropriate architecture and tuning hyperparameters to maximize performance. Following the segmentation, unsupervised clustering and community detection techniques are applied to further refine the results and propagate the labels obtained from the deep learning model to a larger portion of the point cloud.

By leveraging the strengths of both supervised and unsupervised learning, the proposed framework aims to deliver an effective and versatile solution for point cloud tree component segmentation tasks. This approach addresses the research objectives of accurately segmenting tree components, evaluating the value of synthetic trees for calibrating and evaluating deep learning models, and identifying the most effective parts of the pipeline for tree component segmentation.

3.2 Synthetic Tree Point Cloud Dataset Generation Pipeline

Gathering real-world labeled point cloud data for tree segmentation is a challenging task, as it is time-consuming, costly, and requires expert-level domain knowledge [43]. Moreover, trees of the same species can exhibit a wide variety of structural forms, adding to the complexity of obtaining a comprehensive and representative dataset. Due to these factors, a large publicly available dataset for researchers to collaborate with is currently lacking. In light of these challenges, this thesis explores the use of synthetically created point cloud data to bridge this data-shortage gap.

In this study, a synthetic tree point cloud dataset is generated, which aims to address the aforementioned limitations while providing a reliable foundation for the proposed framework.

3.2.1 SpeedTree: A High-Fidelity Tree Modeling Solution

SpeedTree is a software solution that specializes in generating high-quality and realistic tree models [44]. SpeedTree's unique approach to tree modeling allows users to generate a wide variety of tree species with intricate variations in morphology and topology, making it an ideal choice for generating synthetic datasets.

At the core of SpeedTree's modeling capabilities are L-Systems, a formalism for modeling complex structures based on stochastic, parametrized, context-sensitive grammars. L-Systems were originally developed in 1968 by Aristid Lindenmayer as a way to mathematically model the growth processes of plant development [42]. They have since been heavily extended, and are currently used in the film and video game industry to procedurally generate realistic looking synthetic trees [45, 46]. Each synthetic tree is built using a parametric set of recursive production rules that describe how to build a tree for a given number of steps. By changing the parameter values of a given L-System, trees that have significantly different visual characteristics and structural features can be produced.

3.2.2 Leveraging the power of L-Systems

Data was obtained using the SpeedTree Model Library Store [47]. From a list of seven target species common to the forests of the Sierra Nevada Mountains (quaking aspen, Douglas fir, western white pine, black oak, western juniper, ponderosa pine, and Jeffrey pine), high quality models existed for five of the seven. At present, there were not any sufficiently high quality realistic looking models for ponderosa pine and Jeffrey pine, which is why they were omitted from this dataset. These five dominant species were selected based on how well each resemble their real-life counterparts. This design choice was made in order for model generalizability and transferability to datasets containing LIDaR scans of real trees. Each species model contained the L-System parameterization values needed to stochastically generate other, synthetic, look-a-like copies of the original ‘seed’ tree. Samples of stochastic generations can be visualized in Fig 3.2. This study’s results and discussion utilize models only trained using synthetic trees produced from the quaking aspen L-System ’seed.’

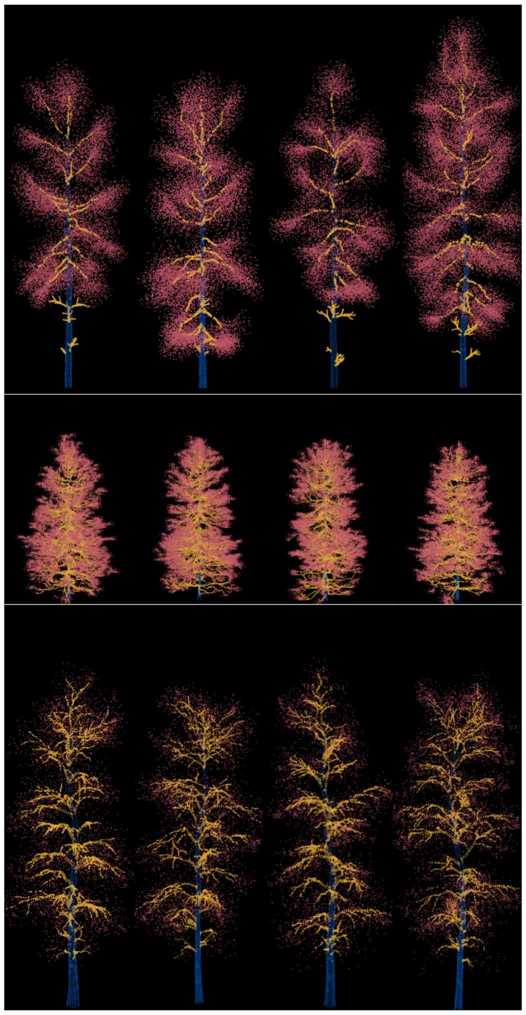

Figure 3.2: Visualization of point clouds generated for quaking aspen (top), Douglas fir (middle), and white pine (bottom). Each model is generated using the same species' parametric L-System seed. Adding stochastic noise to these L-System seeds helps create structurally different trees. The color of points in the point cloud (blue: trunk / yellow: branch / pink: leaf) designates point-wise labels.

3.2.3 Re-Optimization for Point Cloud Representation

To prepare these models, whose polygon counts are optimized for use in movies and video games, they first have to be re-optimized to be useful in the context of a point cloud dataset. Each vertex in an exported SpeedTree object becomes a point in the point cloud sample, so it is important to have a realistic distribution of vertex/points similar to a LIDaR scan. SpeedTree Generators can be thought of as the parametric

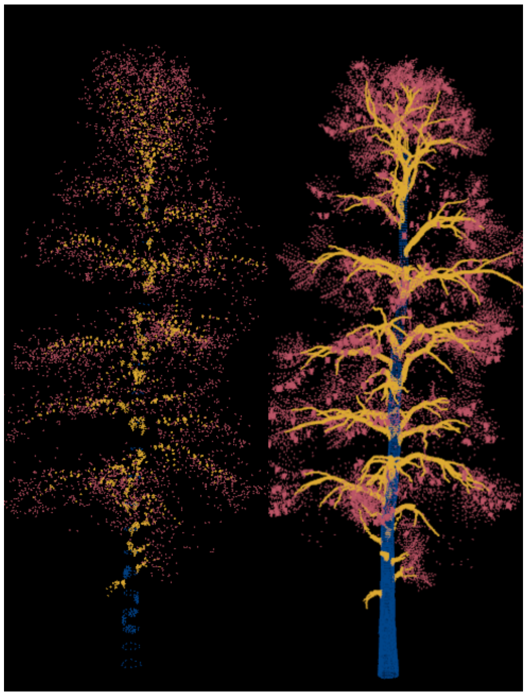

Figure 3.3: A synthetic white pine model without point cloud re-optimization performed (left) and with re-optimization performed (right). Without re-optimization, the point cloud appears visually sparse and contains only 11,483 points. By applying reoptimization it contains 283,649 points.

rules of an L-System [42] that determine how Nodes are placed and what each Node looks like. Generators contain hundreds of unique parameters. Each contributes to the overall structure of the tree. Editing a combination of these parameter values can significantly change a tree's topology; however, certain parameters only affect the computational fidelity of the underlying polygon mesh. The reoptimization process aims at increasing both the raw number and density of vertices along Nodes in the tree's polygon mesh. Increasing the length and radial absolute values of a Generator's segment will increase vertex count and density along Nodes produced by the Generator. Appropriate segment lengths and radial values need to be empirically found for every SpeedTree model.

There is no one-size-fits-all solution for determining the appropriate segment lengths and radial values, as the optimal values vary depending on the tree species and specific model being used. It is important to note that changing these values does not alter the phenotypic structure of the selected tree species; instead, it only increases the polygon count granularity and vertex count to provide a more accurate point cloud representation. To find the best values for each tree model, the segment lengths and radial values were systematically varied while closely examining the resulting point clouds. The optimal values were determined based on the ability to generate point clouds that closely resemble realistic LIDAR scans of the tree species in question, while also ensuring that the overall computational complexity remained manageable. Through this empirical process, adequate segment lengths and radial values that resulted in high-quality point cloud representations for each of the selected tree species were found.

Every Generator also has an Optimization parameter, whose value should also be set to the minimum (zero optimization). Meshes in the tree model, which are often used for leaves, fronds, and dead branches, should have reoptimization performed by adding additional anchor points to the mesh, which function similarly to vertices in the exported model. A tree model having undergone this process is illustrated in Fig.3.3, where its raw point count has been increased by around 25x. Each species of tree will manually undergo this reoptimization process, as each separate SpeedTree model may have more or less Generators that need to be individually reoptimized.

3.2.4 Role of Synthetic Data in Deep Learning

Having access to large and diverse datasets is critical for training robust models. Synthetic data generated from SpeedTree can be used to supplement real-world datasets or provide a controlled environment for testing and validating segmentation algorithms. Furthermore, SpeedTree's custom exporter exports the complete mesh-topology of a given tree as a hierarchical XML document. The schema of the document similarly represents the hierarchical structure of the Generator topology of the original tree. This means that vertices from the same Node are stored together, allowing for groups of points to maintain their original label throughout the exportation process. By strategically self-labeling Generators in the original SpeedTree Model, it is possible to export a point cloud of ground-truth labels with Node-wise granularity.

After the re-optimization process outlined earlier, 1000 stochastic generations were exported for each species composing a total of 5000 unique tree point clouds. Each tree point cloud also contains a class label for every point as belonging to a trunk, branch, or leaf Node. This dataset then underwent an 80%/10%/10% split on a per-species basis for training, validation, and testing.

3.3 Data Preprocessing and Subsampling

Preparing the synthetic tree point clouds for deep learning models involves a series of preprocessing steps and a well-planned subsampling technique. This section describes the preprocessing steps applied to the tree point clouds and the selection and implementation of the subsampling technique.

3.3.1 Preprocessing Steps for Tree Point Clouds

The first step in preprocessing the tree point clouds involves normalizing the point clouds. Normalization is crucial in the context of deep learning because it ensures that different features have the same scale, allowing the model to converge more quickly and potentially achieve better performance. To normalize the input point clouds, two primary steps are taken. First, the points are zero-mean centered along the origin, ensuring that the point cloud is centered around the coordinate system's origin. This step helps the model focus on the geometric structure of the point cloud without being influenced by its position in the coordinate system.

Second, the tree point clouds are normalized to fit within a unit circle. By doing this, a consistent scale is maintained across all point clouds, allowing the model to better learn the underlying patterns and structures present in the data. This step is particularly important when dealing with point clouds from various tree species, which may have varied sizes and shapes.

Additionally, z-score thresholding is applied to remove outlier points, which might be present in real-world scans due to factors such as birds or noise. This step helps to eliminate any potentially detrimental effects these outlier points may have on the performance of the deep learning models. By removing outliers, the model can focus on the most relevant and representative points in the point cloud, improving its ability to learn and generalize.

3.3.2 Selection and Implementation of Subsampling Technique

The choice of subsampling technique plays a crucial role in the deep learning model's performance. During training, the ground truth labels are utilized to perform class-weighted random subsampling to 2,048 points. Downsampling point clouds achieves a two-fold goal. Firstly, downsampling performs data augmentation in a way that helps to enrich the data diversity that the model receives as input, allowing the same original point cloud to be reused multiple times during the training process without the model memorizing specific point layouts. Secondly, while input size-invariant deep learning models for point clouds exist, it has been shown that performing a downsampling to 2,048 points increases model performance on segmentation and classification-based tasks [48] [9]. Random downsampling was chosen because it makes no assumptions about the input point cloud space and is relatively fast to perform.

After downsampling, the point clouds then have a random rotation applied as a data augmentation step in order to help the model further generalize to real-life trees that are not perfectly level with the ground. Lastly, each point cloud is normalized to fit within a Unit-Sphere to help constrain the feature-space to lie within a common boundary.

During the testing phase, when the ground truth labels are not available, and to better assess the model's performance on real trees, a uniform grid subsampling technique is employed. This technique subsamples the test point clouds to 2,048 points, helping to address the issue of occlusion and the non-uniform distribution of point density. By using a consistent subsampling technique during inference, the model's performance can be better evaluated in real-world scenarios, where the distribution of points and the presence of occlusions can be highly variant.

3.4 Segmentation with Deep Learning

Segmentation of tree point clouds plays a crucial role in understanding tree structure and composition. This section discusses the choice of deep learning architectures, their training process, and hyperparameter tuning for a 3-class part-segmentation task on isolated tree point clouds.

3.4.1 Choice of Deep Learning Architectures

A total of five deep learning models were investigated and evaluated for their performance in part-segmentation tasks with three classes: trunk, branch, and leaf. The chosen point cloud part-segmentation model architectures include PointNet [1], PointNet++ [2], ShellNet [49], Dynamic Graph CNN (DGCNN) [50], and Point-Voxel Transformer (PVT) [9]. Each model's architecture is illustrated at a high level in Fig.

3.4.

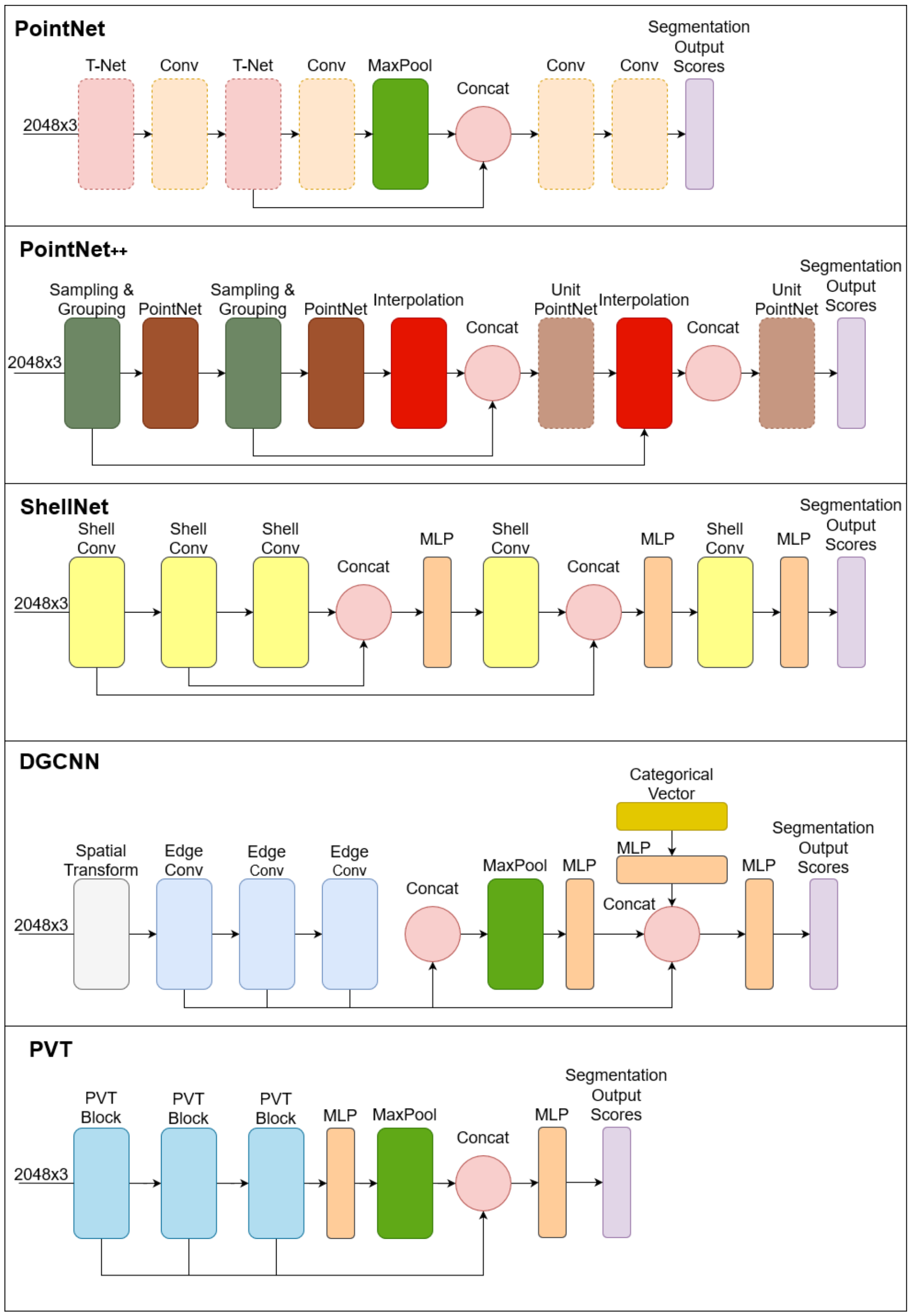

Figure 3.4: High-level visualization of the network architecture for each of the deep learning models utilized for tree part-segmentation. Each model in their original paper typically has a dual branching structure to perform both classification and segmentation. Only the segmentation branch is displayed here for both relevance and simplicity.

These specific models were selected based on several criteria: their performance on part-segmentation tasks using popular point cloud datasets such as ModelNet40 [51] and ShapeNet [26], their popularity and acceptability within the point cloud deep learning community as both baselines and cutting-edge architectures, and to compare how different methods, such as hierarchical, voxel, graph, and hybrid, might be better suited for the domain of tree topology.

3.4.2 Network Training and Hyperparameter Tuning

Each model used the same data splits for training, validation, and testing, ensuring a reproducible comparative analysis of their performance. The loss of each model was calculated using class-weighted categorical cross-entropy to account for any imbalanced distribution of class labels. Model performance is quantified using the Dice coefficient between the predictions and ground truth, while part-average Intersection-over-Union (pIoU) is included to facilitate comparison with other datasets.

After preprocessing, each model takes only the geometric coordinates (x, y, z) of subsampled points as inputs. Hyperparameters for each model were optimized using a Bayesian search based on model performance, ensuring a satisfactory configuration for each architecture.

All deep learning models were implemented using the GPU version of PyTorch on a hardware configuration consisting of an Intel® Xeon® W-2295 18 x 3.0GHz processor, 512GB RAM, and an NVIDIA Quadro P2200 (5GB) graphics card. This powerful setup enabled efficient training and evaluation of the chosen deep learning architectures, allowing for a comprehensive and accurate assessment of their performance in the tree point cloud segmentation task.

3.5 Unsupervised Clustering and Community Detection

In this section, various community detection techniques are outlined for their utility in augmenting the results of the deep learning segmentation model. This section also covers graph construction methods and how their potential for label propagation will be evaluated.

3.5.1 Graph Construction Methods

To analyze various community detection techniques for their effectiveness in label propagation, it is necessary to construct a graph from the point cloud. Two graph construction methods were considered in this study:

K-Nearest Neighbors (KNN)

The first method involves connecting a node to its K closest neighbors, where a low value of K is chosen to ensure local connectivity. In this study, K=10 was used. This approach is simple and provides a connected graph, which is essential for several community detection techniques.

Alpha Shapes Surface Reconstruction

The second method utilizes Alpha Shapes to approximate a mesh for the tree point cloud, as shown in Figure 3.5. Alpha Shapes is a surface reconstruction technique that uses a parameter alpha to determine the coarseness of the surface reconstruction. It does not guarantee a single connected-component graph, which is necessary for some community detection techniques. If more than one graph component is created, they

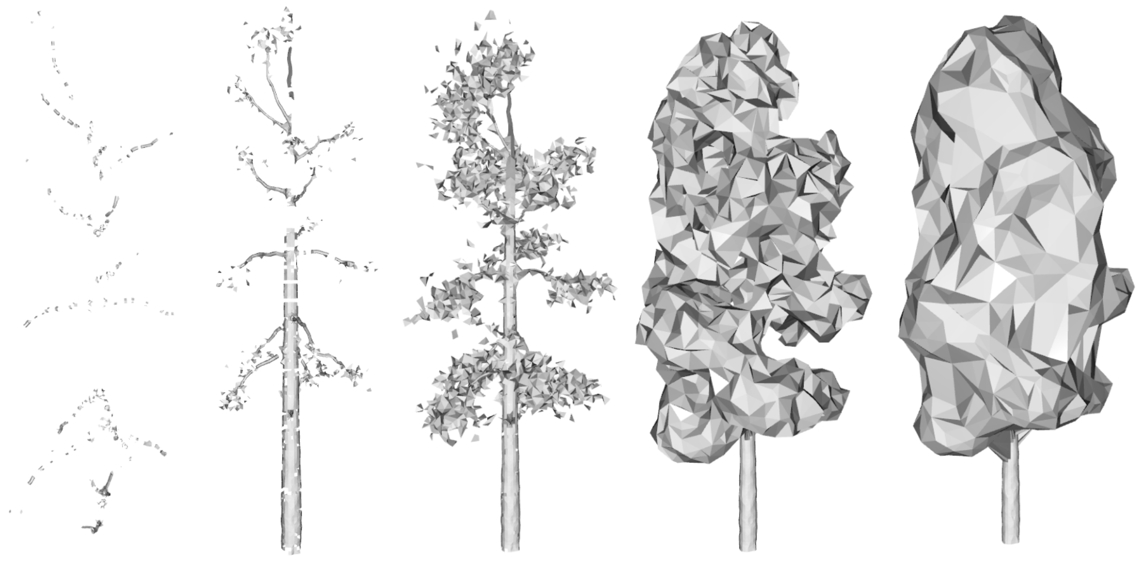

Figure 3.5: Alpha-Shapes mesh generation for a tree point cloud with increasing alpha values ($\alpha = 0.005$, $\alpha = 0.01$, $\alpha = 0.03$, $\alpha = 0.05$, $\alpha = 0.1$).

need to be stitched together using the closest neighbors between components.

3.5.2 Limitations of Other Surface Reconstruction Techniques

Several surface reconstruction techniques have been proposed to generate surface models from point cloud data, such as Poisson Surface Reconstruction (PSR) [52], Ball Pivoting Algorithm (BPA) [53], and Marching Cubes (MC) [54]. However, these methods may not be suitable for reconstructing tree surfaces for various reasons, such as reliance on priors, normal vector information, or their inability to handle complex and concave shapes. In contrast, KNN connectivity inference and Alpha Shapes Surface Reconstruction [55] offer more flexible and adaptable solutions for constructing graphs from point clouds, making them more suitable for tree surface reconstruction tasks.

3.5.3 Community Detection Techniques and Label Propagation

Label propagation within clusters is a process of assigning labels to a set of unlabeled data points within a cluster based on a small set of labeled data points belonging to the same cluster. The goal in this study is to use this approach to label a point cloud of a tree that has partially labeled data available.

Identifying Clusters

The first step is to identify clusters of points within the tree point cloud using community detection techniques. These techniques group nodes with similar attributes or connections into clusters.

Proposed K-Neighbor-Nurtured-Garden (KNNG) Algorithm

The K-Neighbor-Nurtured-Garden (KNNG) algorithm is a label propagation algorithm designed for labeling nodes in an unstructured point cloud without the need for a pre-built graph structure. The algorithm comprises two main phases: a "seeds" creation pass and a "seeds" transformation pass. The steps involved in these phases are depicted in Figure 3.6.

In the first phase, the algorithm identifies the K nearest "dirt" nodes to each "flower" using an Annoy index for accelerated nearest neighbor search. The "dirt" nodes are then assigned the same label as the "flower", transforming them into "seeds". In the second phase, the "seeds" are converted into new "flowers", which participate in the subsequent iteration of the labeling process. The algorithm continues running until no "dirt" nodes or "flowers" are left.

The following is an outline of the KNNG algorithm's implementation:

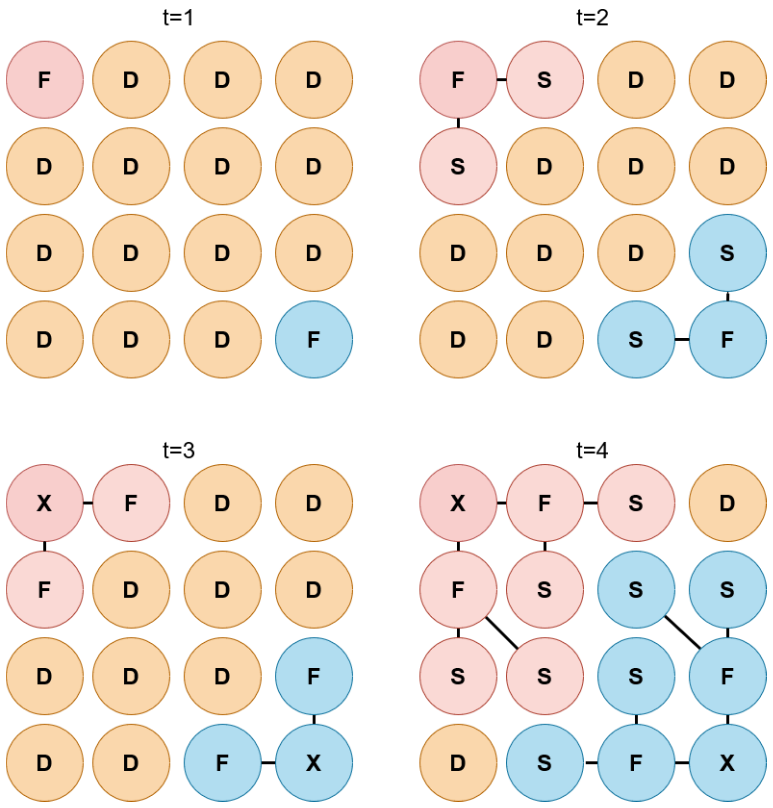

Figure 3.6: Four time steps of the K-Neighbor-Nurtured-Garden (KNNG) algorithm.

Algorithm 1 K-Neighbor-Nurtured-Garden Algorithm

```
1:  function K-NEAREST-NURTURED-GARDEN($G, K$)
2:      Initialize an Annoy index for fast nearest neighbor search.
3:      Get partially labeled nodes and assign them as "flowers" $F$.
4:      while there are "dirt" nodes $D$ or "flowers" $F$ remaining in $G$ do
5:          for each "flower" $f \in F$ do
6:              Find the K closest "dirt" nodes $d \in D$ within the z-score threshold.
7:              Assign $f$'s label to the "dirt" nodes, turning them into new "seeds" $S$.
8:          end for
9:          Transform the "seeds" $S$ into new "flowers" $F'$ for the next iteration.
10:         Update the set of "dead" nodes $X$ to include any "flowers" that did not
    generate new "seeds" in the previous iteration.
11:         Remove "dead" nodes $X$ from $G$.
12:     end while
13:     return the list of nodes with labels assigned by the KNNG algorithm.
14: end function
```

Algorithm 1 describes the K-Neighbor-Nurtured-Garden algorithm and is specifically developed to efficiently propagate labels within an unstructured point cloud, making it more versatile and suitable for applications such as tree segmentation. Unlike community detection techniques requiring a pre-built graph or network, the KNNG algorithm operates directly on unstructured point clouds and makes no assumptions about the input. This allows for greater flexibility in handling various types of data, such as tree point clouds, where a pre-built graph may not exist or may not accurately represent the structure of the point cloud.

Label Propagation Within Clusters

Once the clusters are identified, label propagation can be performed within each cluster separately. Network clustering techniques such as FUMO, Spectral Lanczos, AFC, Label Propagation, Async LPA, Recursive KNN, Voronoi Clustering, and KNNG were chosen to propagate labels to the unlabeled points in the tree based on their relevance to the problem and their effectiveness in related works. The performance of these techniques in label propagation within clusters is evaluated in terms of their accuracy and efficiency.

3.6 Evaluation of Techniques

The evaluation process for the proposed framework is designed to assess the performance of both segmentation and unsupervised clustering techniques. This comprehensive evaluation will help identify the most effective methods for tree component segmentation in point cloud data. The following subsections detail the evaluation methodology for each aspect of the framework.

3.6.1 Evaluating Segmentation Performance

To evaluate the performance of the deep learning models employed for tree component segmentation, several metrics and considerations are employed. The validation datasets are sampled and chosen randomly from the synthetic tree point cloud data. For each tree class, 80% are allocated for training, 10% for validation, and 10% for testing. This random sampling approach ensures that the datasets are representative of the overall population and reduces the likelihood of sampling bias.

The performance of tree component segmentation is assessed using the following metrics:

- **Dice Coefficient (F1 Score):** The F-score for each class label (trunk, branches, and leaves) is calculated to measure the accuracy of the model in classifying each component. The F-score is a harmonic mean of precision and recall, providing a balanced metric for evaluating the effectiveness of the model in segmenting tree components.

- **Part Intersection over Union (Part IoU):** Part IoU is used to evaluate the performance of a given technique in segmenting individual parts of the tree components, such as the trunk, branches, and leaves. It is calculated by dividing the intersection of the predicted part segmentation and the ground truth part segmentation by the union of the two segmentations. Higher Part IoU values indicate better model performance in accurately segmenting individual parts of the tree components.

- **Inference Time (ms):** Inference time refers to the time it takes for a trained deep learning model to process a single input and generate a prediction. It is an important consideration for evaluating a model's performance in real-world applications where fast predictions for scalability are necessary. Inference time is typically measured in milliseconds (ms).

- **Parameter Count (m):** Parameter count refers to the number of trainable parameters in a deep learning model. It is an important consideration for evaluating a model's complexity and its ability to generalize to new data. The parameter count is typically measured in millions (m).

These evaluation metrics, along with visual inspection of the segmented point clouds, will provide a comprehensive understanding of the segmentation performance of the deep learning models. The visual inspections are conducted on both synthetic test-set trees and real Terrestrial Laser Scanning (TLS) scans of actual trees. Due to the absence of ground truth labels for real tree scans, quantitative results cannot be obtained for these samples. Therefore, visual inspection serves as the primary method for evaluating model performance on real tree data, complementing the quantitative assessments performed on synthetic test-set trees. This combined approach ensures a thorough evaluation of the deep learning models across various data types.

3.6.2 Evaluating Unsupervised Clustering Performance

The evaluation of unsupervised clustering techniques focuses on assessing their effectiveness in refining the segmentation obtained from the PVT deep learning model and the additional time they may require. The same data partitioning approach used in the segmentation performance evaluation is employed here, ensuring consistency across evaluations. Each unsupervised clustering technique, including community detection strategies and the proposed k-Nearest Neighbor Graph (KNNG) algorithm, is evaluated and compared to the baseline approach of having the model process the entire point cloud at once.

The performance of unsupervised clustering techniques is assessed using the following metrics:

- **Label coverage:** Label coverage measures the proportion of points in the point cloud that have been assigned a class label. Higher label coverage indicates that the technique is successful in propagating labels to a larger portion of the point cloud.

- **Dice Coefficient (F1 Score):** To better assess the impact of unsupervised clustering techniques when applied to the outputs of the segmentation model, F-scores are recalculated after applying each clustering method. These new F-scores enable a direct comparison between the original F-scores obtained from the deep learning model outputs and the F-scores after incorporating the unsupervised clustering methods. This comparison provides valuable insights into the effectiveness of each unsupervised clustering technique in refining and improving the initial segmentation results.

- **Additional inference time (s):** The extra time taken by the unsupervised clustering techniques is recorded, as many of these methods may require a significant amount of time to process the data. This metric is crucial for understanding the efficiency of each method, which is an essential factor in real-world applications.

By evaluating the performance of the unsupervised clustering techniques using these metrics, the study aims to identify the most effective method for refining and extending the initial segmentation produced by the deep learning models. The subsequent sections will present the results of this evaluation, shedding light on the strengths and weaknesses of each technique and informing the selection of the optimal approach for the task.

Chapter 4

Experimental Results

4.1 Part Segmentation Results and Discussion

Quantitative assessment of part-segmentation is summarized in Table 4.1 and visualized in Fig 4.1. Each model's parameters were saved upon validation convergence, which was gauged when the loss on the validation set stopped increasing and became stable after three epochs. In this study, a high testing pIoU (average unweighted class accuracy), high Dice coefficient (weighted class accuracy), and low inference time (model scalability) were all considered as the criteria of a good classifier.

PVT was able to significantly outperform every other model in all three class domains. This is because of its hybrid-based PVT blocks that simultaneously learn from varying scales of the raw pointcloud and voxelized representations. It aims to learn both local features in the voxel-domain and global features in the point-domain. Additionally, it employs a UNET-like architecture [56], which is notable for their impressive performance in 2D segmentation tasks.

In addition, comparing model pIoU performance between this task and ShapeNet

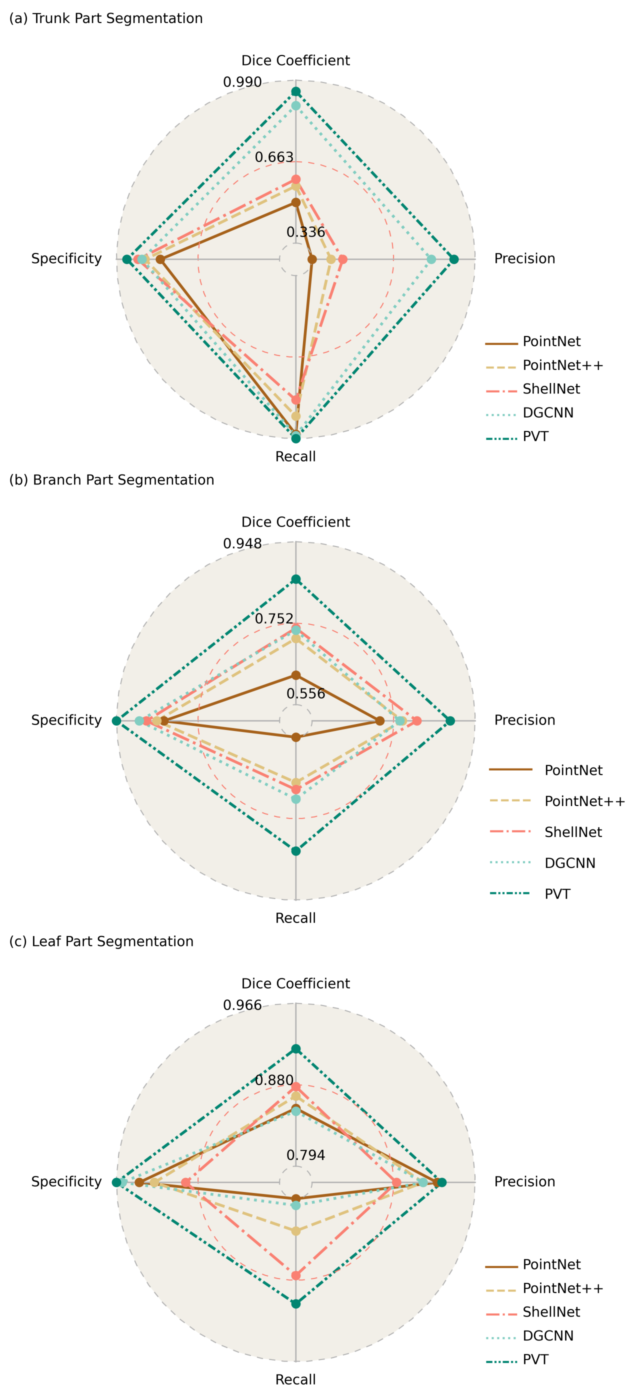

Figure 4.1: Visualization of performance using radar charts for each part segmentation task: (a) Trunk, (b) Branch, and (c) Leaf. Each chart compares the Dice Coefficient, Precision, Specificity, and Recall for each deep learning model. Values are normalized such that the minimum value lies at the center ring of the chart and the max value lies on the outermost ring. Each ring is labeled with its corresponding value.

Table 4.1: Statistical summary of the five tree part-segmentation models showing averaged performance metrics when evaluated on the test-set. pIoU means part-average Intersection-over-Union.

		Trunk	**Branch**	**Leaf**
PointNet	Dice Coefficient	0.499	0.627	0.855
	Precision	0.336	0.719	0.926
	Recall	0.973	0.556	0.794
	Specificity	0.814	0.834	0.942
	pIoU	0.773		
	Inference (ms)	206		
	Parameters (m)	3.536		
PointNet++	Dice Coefficient	0.565	0.715	0.868
	Precision	0.412	0.773	0.912
	Recall	0.900	0.665	0.828
	Specificity	0.875	0.851	0.926
	pIoU	0.800		
	Inference (ms)	1113		
	Parameters (m)	0.966		
ShellNet	Dice Coefficient	0.592	0.740	0.878
	Precision	0.459	0.808	0.883
	Recall	0.835	0.682	0.875
	Specificity	0.904	0.876	0.893
	pIoU	0.796		
	Inference (ms)	385		
	Parameters (m)	0.775		
DGCNN	Dice Coefficient	0.888	0.735	0.852
	Precision	0.814	0.767	0.911
	Recall	0.978	0.705	0.801
	Specificity	0.888	0.893	0.960
	pIoU	0.830		
	Inference (ms)	**59**		
	Parameters (m)	1.454		
PVT	Dice Coefficient	**0.945**	**0.858**	**0.918**
	Precision	0.905	0.888	0.931
	Recall	0.990	0.830	0.905
	Specificity	0.948	0.948	0.966
	pIoU	**0.910**		
	Inference (ms)	93		
	Parameters (m)	6.414		

part-segmentation, it is interesting to see that all classifiers except for PVT had worse pIoU values. It stands to reason that objects from the ShapeNet dataset were easier for deep learning models to learn about than the complex branching structure of a tree pointcloud.

DGCNN was the most rapid on model inference for a single tree (59 ms), nearly halfing the inference time of PVT (93 ms). DGCNN does not have to voxelize a pointcloud several times during model inference, which could help explain why it sees such performance boost in inference time. Additionally, its Edge Conv blocks seem to perform well on the cylindrical structure of the tree pointcloud. However, this model notably has a limiting hyperparameter K, which determines the number of edge connections a singular point can have. While one value of K might work well for classifying one class, it may limit performance on another class. This would help explain the large value gap in Dice coefficient between the trunk (0.888), branch (0.735), and leaf (0.852).

Comparing the precision and recall values among classes, all models had higher recalls than precisions for trunk, and the inverse for both branch and leaf. This indicates that each model was often over-estimating the number of trunk points. While viewing rendered model predictions, points near the intersection of branches and the trunk were often observed to be the ones misclassified as a trunk. This also might be a difficult problem for humans, so it makes sense why these models had a hard time classifying these ambiguous intersection regions.

PointNet++ had the worst model inference time (1113 ms). This large inference time is by design, as the PointNet++ architecture applies PointNet recursively on a nested partitioning of the input point set. By designing the architecture that way, PointNet++ is able to learn local features with increasing contextual scales. One of ShellNet's large contributions to the 3D deep learning space was to achieve better performance than PointNet++ while simultaneously speeding up model inference time and decreasing model size. Using ShellConv blocks instead of PointNet blocks helps to drastically speed up model inference time. Additionally, its concentric spherical shells to define representative features helps it closely match the performance of PointNet++ accross the board.

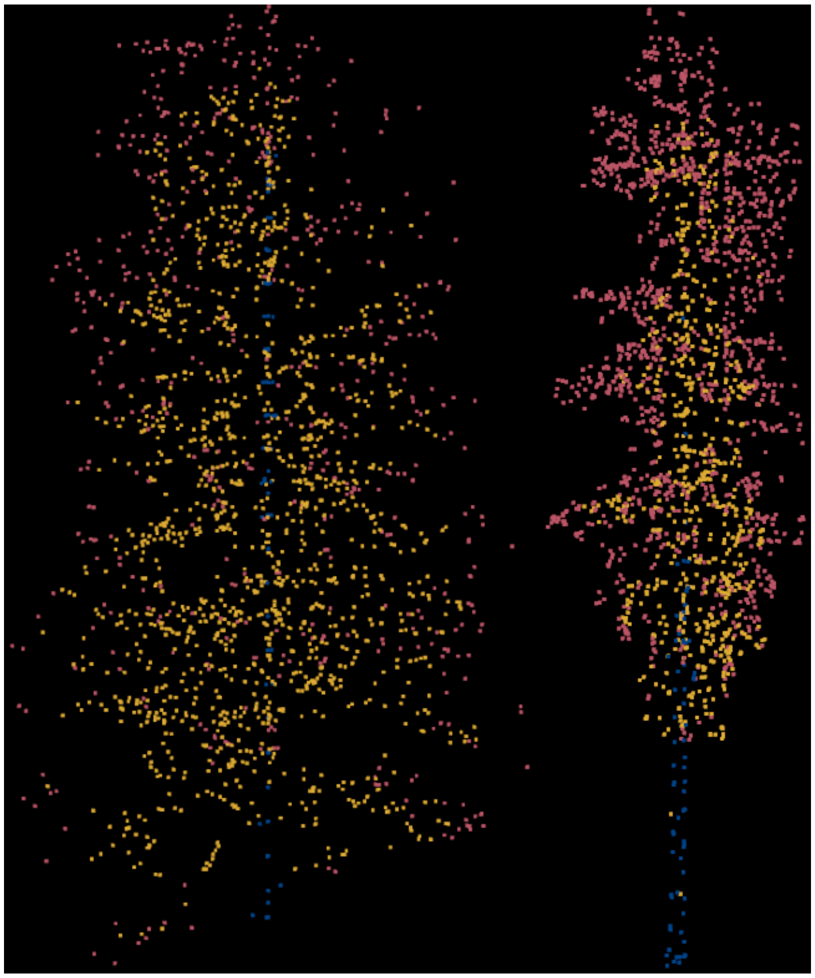

Figure 4.2: Visualization of segmentation results of PVT model on a synthetic Douglas fir tree from the test set (left) and a real TLS scan of a Douglas fir (right).

To assess the performance of the PVT model on both synthetic and real tree point clouds, visualizations are provided of the segmentation results in Figure 4.2. The left side of the figure shows the PVT outputs on a synthetic Douglas fir from the test set. On the right side of the figure are shown the results of applying the trained model to a real TLS scan of a Douglas fir tree. It is important to note that it is not possible to obtain quantitative performance metrics for the real tree due to the lack of point-wise ground truth labels. However, through visual inspection of the real tree segmentation results, it is evident that the model does an exceptional job at identifying and labeling the various parts of the tree. This observation suggests that the PVT model generalizes well to real-world tree point clouds and is capable of handling the inherent complexities and variations found in nature.

4.2 Label-Propagation Results and Discussion

In this study, the utility of K-Neighbor-Nurtured-Garden (KNNG) for label propagation in synthetic tree point clouds was proposed. Table 4.2 presents the performance comparison in terms of F-scores for the three classes (trunk, branch, and leaf) and computation time between each clustering method. While the KNNG method showed promise in terms of computational efficiency and label coverage, its performance in terms of F-scores for the three classes (trunk, branch, and leaf) was found to be lackluster when compared to other methods. Particularly, the performance drop in trunk points was more pronounced compared to branch and leaf points. This section discusses the potential reasons for this unexpected outcome and provides insight into the limitations of the KNNG method.

One possible reason for the observed performance drop in trunk points could be the inherent nature of the KNNG algorithm, which relies on the k-nearest neighbors to propagate labels. Trunk points, by their nature, tend to be spatially more concentrated and localized in a tree point cloud. As a result, the k-nearest neighbors of a trunk point are more likely to be other trunk points. In the presence of noise or errors in the initial labeling, this local concentration can lead to a reinforcing feedback loop, where incorrect labels propagate and amplify within the trunk points, ultimately affecting the F-score.

Table 4.2: Summary of study results: Average performance metrics across 100 test pointcloud trees.

K-NN connectivity inference					
Community Detection Method	Average Additional Inference time (s)	Label coverage	Trunk F-Score	Branch F-Score	Leaf F-Score
FUMO	123.3(±80.9)	99.99%	0.249	0.289	0.743
Spectral Lanczos	4573.9(±1812.2)	60%	0.598	0.72	0.817
AFC	15.1(±6.9)	61.60%	0.746	0.828	0.92
Label Propagation	3.4(±1.4)	71%	0.752	0.832	0.931
Asyn LPA	8.90(±3.53)	62%	0.761	0.832	0.926
Alpha Shapes Surface Reconstruction connectivity inference					
Community Detection Method	Average Additional Inference time (s)	Label coverage	Trunk F-Score	Branch F-Score	Leaf F-Score
FUMO	395.35(±269.34)	99.99%	0.214	0.427	0.586
Spectral Lanczos	20944(±11337.77)	82%	0.402	0.64	0.77
AFC	3.5(±1.3)	97%	0.355	0.624	0.743
Label Propagation	1.7(±0.53)	89%	0.307	0.645	0.75
Asyn LPA	2.1(±0.64)	89%	0.363	0.651	0.765
No Graph Structure					
Method	Average Additional Inference time (s)	Label coverage	Trunk F-Score	Branch F-Score	Leaf F-Score
PVT(2048)	-	2048 pts	0.945	0.859	0.919
PVT(all)	-	100%	0.714	0.783	0.879
rKNN	10.1(±5.6)	100%	0.771	0.801	0.911
Voronoi Clustering	5.34(±1.72)	100%	0.704	0.728	0.8726
KNNG (ours) K=4	3.91(±1.36)	100%	0.525	0.709	0.857
KNNG (ours) K=3	4.75(±1.76)	100%	0.521	0.705	0.853
KNNG (ours) K=2	7.06(±2.93)	100%	0.53	0.713	0.858
KNNG (ours) K=1	7.38(±3.05)	100%	0.569	0.704	0.835

Furthermore, the presence of varying point densities within the point cloud may also have affected the performance of the KNNG method. These factors may have introduced inconsistencies in the spatial distribution of the points, making it challenging for the algorithm to propagate labels accurately based on the neighborhood information. In particular, the trunk region may have been more susceptible to such inconsistencies.

In light of these observations, it appears that the KNNG method may have certain limitations when applied to synthetic tree point clouds, particularly in the context of trunk point labeling. While the method offers advantages in terms of computational speed and label coverage, its performance in terms of F-scores is found to be inferior to other methods, such as Asyn LPA and Label Propagation. Future research could focus on refining the KNNG algorithm or exploring alternative methods that can better capture the unique spatial characteristics of tree point clouds and achieve more accurate label propagation across all classes.

While the K-Neighbor-Nurtured-Garden (KNNG) method did not perform as expected, other methods utilizing the K-NN connectivity inference such as Asynchronous Label Propagation Algorithm (Asyn LPA) and Label Propagation demonstrated superior performance in terms of F-scores across trunk, branch, and leaf classes.

One of the factors that may have contributed to the superior performance of Asyn LPA and Label Propagation is their ability to better adapt to the complex spatial structures present in tree point clouds. These methods are based on propagating labels through an iterative process, which allows them to effectively capture the underlying structure of the data and propagate labels more accurately. This iterative approach can help mitigate the impact of noise or errors in the initial labeling, as the algorithms continuously refine the labels based on the information from neighboring points.

In addition, the Asyn LPA method allows for asynchronous updating of labels during the propagation process, which can be beneficial in situations where the point cloud exhibits varying point densities, occlusions, or overlapping regions. This asynchronous updating enables the method to be more robust against local inconsistencies in the spatial distribution of points, as it does not rely solely on the immediate neighborhood information for label updates. This flexibility may have contributed to the improved performance of Asyn LPA in comparison to other methods.

Label Propagation also performed well in this study; potentially due to its ability to adapt to the specific characteristics of tree point clouds. Like Asyn LPA, Label Propagation is an iterative method that refines labels based on the information from neighboring points. However, Label Propagation operates in a synchronous manner, updating all labels simultaneously at each iteration. This approach can lead to faster convergence of the algorithm, which might be advantageous in cases where the point cloud exhibits a clear and well-defined structure.

Moreover, both Asyn LPA and Label Propagation methods could be further fine-tuned by adjusting their respective parameters, such as the number of iterations or the influence of neighborhood size. This flexibility allows for the optimization of the methods for the specific characteristics of synthetic tree point clouds, potentially leading to improved performance.

In summary, the superior performance of Asyn LPA and Label Propagation methods in this study can be attributed to their ability to better adapt to the complex spatial structures present in tree point clouds, as well as their iterative nature, which allows for more accurate label propagation. These methods demonstrate the potential to effectively address the challenges associated with labeling synthetic tree point clouds and could serve as a foundation for future research in this area.

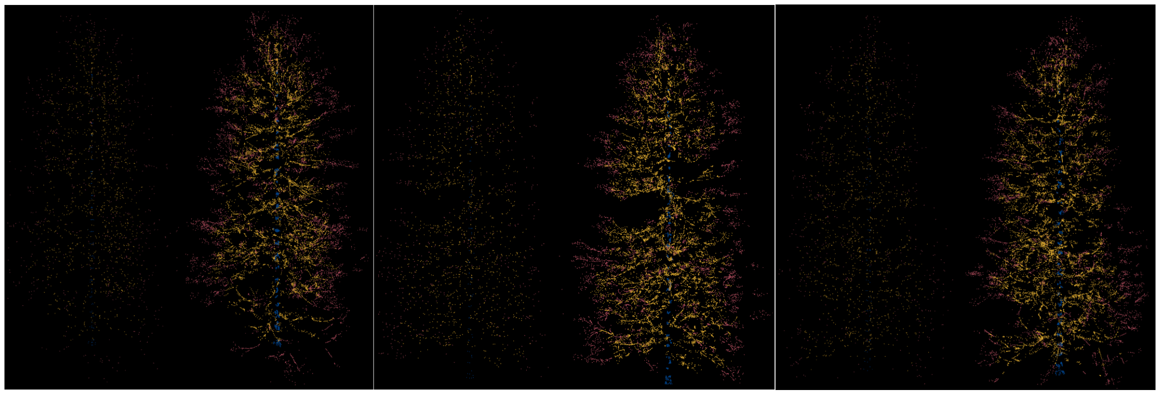

Figure 4.3: Synthetic tree point cloud before and after label propagation. The left point cloud in each pair demonstrates the initial labeling results, while the right point cloud displays the significantly enhanced coverage obtained by applying label propagation.

Figs 4.3 and 4.4 illustrate the improvements achieved through the combination of deep learning and label propagation techniques. The figures showcase before and after images of both synthetic and real Douglas fir TLS scans that have been partially labeled using the PVT model for 2,048 points. The left point cloud in each pair demonstrates the initial labeling results, while the righ images display the signif-

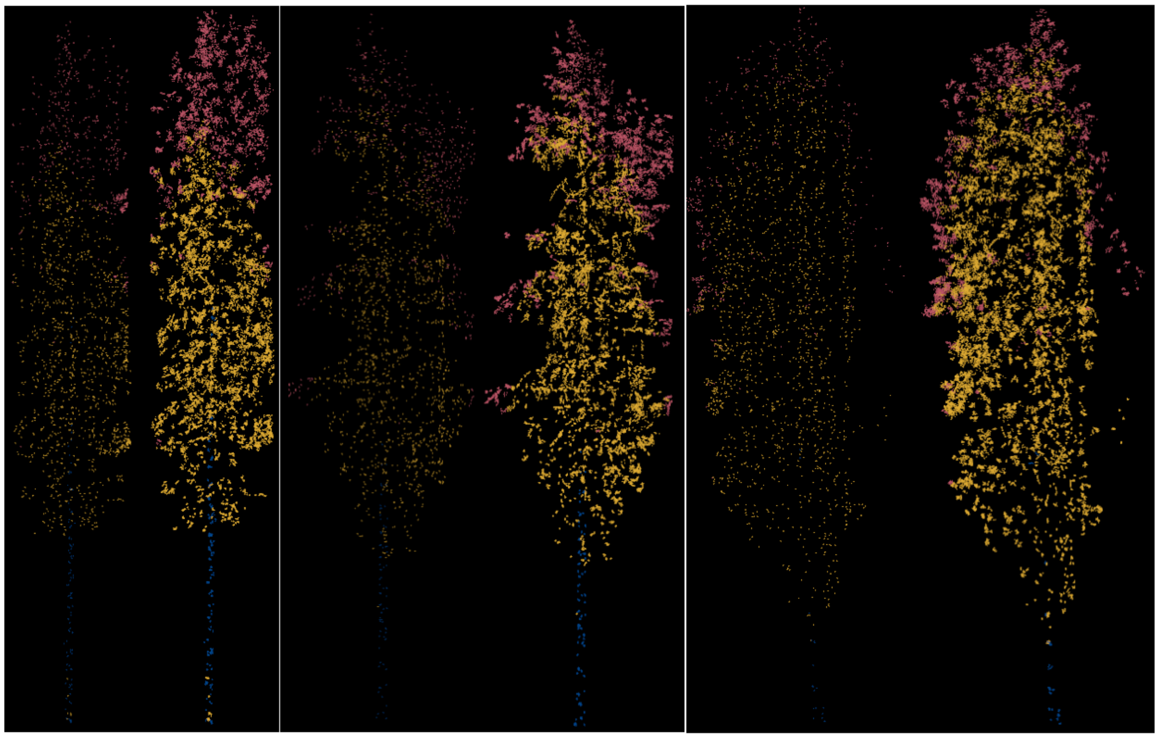

Figure 4.4: Real Douglas fir TLS scan before and after label propagation. The left point cloud in each pair demonstrates the initial labeling results, while the right point cloud displays the significantly enhanced coverage obtained by applying label propagation.

icantly enhanced coverage obtained by applying label propagation. By incorporating community detection methods, we can effectively cluster regions of interest, providing a solid foundation for label propagation to further refine and extend the labeling process. This combination of techniques allows for more accurate and complete segmentation of tree components, resulting in a substantial improvement in labeling coverage and overall segmentation performance.

This study revealed that community detection clustering techniques performed significantly better when applied to graphs built using K-NN connectivity inference compared to those constructed using the Alpha Shapes Surface Reconstruction (ASSR) algorithm.

The main difference between the K-NN connectivity inference and ASSR lies in the way they construct the graph structure. The K-NN approach directly connects points based on their proximity in Euclidean space, which makes it more sensitive to local point densities. As a result, K-NN connectivity inference tends to create better-defined clusters that capture the underlying structure of the point cloud more accurately.

On the other hand, ASSR constructs the graph by approximating the underlying surface of the 3D object, which might not always capture the local point densities effectively. This can lead to situations where the reconstructed mesh inaccurately reflects the spatial distribution of points, resulting in less cohesive clusters. Consequently, community detection methods applied to ASSR-based graphs might struggle to propagate labels efficiently and accurately, leading to lower F-scores.

Another possible explanation for the performance difference is that K-NN connectivity inference is more flexible in terms of adapting to various point densities and noise levels. This allows the connectivity to be tailored to the specific characteristics of the point cloud data, whereas ASSR relies on a fixed set of parameters that might not be suitable for all cases.

It is also worth noting that the choice of community detection method can play a significant role in the overall performance of label propagation. These results showed that some methods, such as Asynchronous Label Propagation Algorithm (Asyn LPA), achieved higher F-scores than others, regardless of the graph construction technique. This suggests that the effectiveness of label propagation is not solely determined by the graph connectivity but also depends on the choice of community detection algorithm.

In conclusion, these findings highlight the importance of selecting an appropriate graph connectivity inference method when using community detection techniques for label propagation in point cloud data. The K-NN connectivity inference approach appears to be better suited for this task, providing more accurate and efficient label propagation compared to the Alpha Shapes Surface Reconstruction algorithm.

Chapter 5

Conclusions and Future Work

Terrestrial and Airborne Laser Scanning provide effective tools for forestry professionals to digitally catalog entire landscapes into large datasets of point clouds. Topographically accurate synthetic tree models, once robustly generated using the proposed method's automatic tree part-segmentation, have the potential to be implemented at-scale. The model-provided information about a given tree's topology contains important details used in allometric equations to estimate above ground biomass (AGB). AGB is useful for understanding carbon stocks and fluxes, which feeds back into models of climate change. The novel contributions of this research include extending the modality of highly accurate tree part-segmentation from 2D (RGB/RGB-D) into the raw, unstructured geometric-coordinate point cloud domain, as well as providing a synthetic vegetation point cloud dataset-generation pipeline for use in training other deep learning remote-sensing tasks like tree-segmentation, estimating mensuration data, species identification, and wood filtering.

The study further evaluated the performance of various label propagation techniques for improving the labeling of synthetic tree point clouds generated using SpeedTree and segmented using a Point-Voxel Transformer model. The goal was to determine the most effective technique for propagating labels within clusters and enhancing the labeling of unlabeled points in the point cloud. Comparison of different community detection methods, including modularity-based clustering, spectral clustering, and the Asynchronous Label Propagation Algorithm (Asyn LPA), applied to graphs built using K-NN connectivity inference and Alpha Shapes Surface Reconstruction (ASSR) algorithm, as well as several unstructured-point cloud based methods was carried out. The evaluation considered factors such as F-scores for each class label, label coverage, and time expenditure.

Results from experiments with 100 test-set synthetic tree point clouds demonstrated that community detection clustering techniques applied to graphs built using K-NN connectivity inference consistently outperformed those applied to graphs constructed using the ASSR algorithm. This suggests that the choice of graph connectivity inference method is critical for achieving accurate and efficient label propagation in point cloud data. The K-NN approach, with its sensitivity to local point densities and adaptability to various point densities and noise levels, appears to be better suited for this task compared to the ASSR algorithm.

Furthermore, the choice of community detection method played a significant role in the overall performance of label propagation. Some methods, such as the Asynchronous Label Propagation Algorithm (Asyn LPA), consistently achieved higher F-scores, regardless of the graph construction technique, highlighting the importance of selecting the appropriate community detection algorithm.

A call for more environmental artists with backgrounds in forest ecology would be helpful in generating more realistic-looking synthetic trees. The research pipeline is limited by what SpeedTree already has in their store library. By bringing talented artists into this research space, more procedural models can be produced and catered towards various domain-specific deep learning tasks.

Another direction for future work may involve the investigation of methods in which these synthetic tree point clouds are subsampled. Random subsampling is a naive, but fast approach that may not subsample the point cloud in an optimal way for the models to learn. Data-driven sampler learning strategies have been shown to increase base-line model performance for point-wise analysis and segmentation-based tasks. Future research could explore such strategies for improving the performance of label propagation in synthetic tree point clouds.

Interestingly, the proposed K-Neighbor-Nurtured-Garden (KNNG) method, which was initially expected to perform well, yielded relatively low F-scores compared to the Asyn LPA and Label Propagation methods despite its fast inference time and 100% label coverage. This finding presents an opportunity for future research to further investigate the reasons behind KNNG's suboptimal performance and explore potential improvements to the method, such as incorporating weighting schemes based on point density, or adapting the neighborhood size according to the spatial distribution of points.

In light of the findings, future work in point cloud labeling should focus on leveraging K-NN connectivity inference and carefully selecting the most suitable community detection method for a given dataset. Additionally, continued exploration and optimization of label propagation techniques, including the proposed KNNG method, could lead to even more accurate and efficient point cloud labeling solutions in the future.

In summary, this research contributes to the understanding of label propagation techniques for point cloud data and offers valuable insights to guide the selection of graph connectivity inference methods and community detection algorithms. The findings provide a foundation for further research and development in the field of point cloud labeling, ultimately contributing to advancements in 3D object recognition, robotics, and computer vision applications.

Bibliography

[1] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” *CoRR*, vol. abs/1612.00593, 2016. [Online]. Available: http://arxiv.org/abs/1612.00593

[2] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” *CoRR*, vol. abs/1706.02413, 2017. [Online]. Available: http://arxiv.org/abs/1706.02413

[3] Y. Xu, T. Fan, M. Xu, L. Zeng, S. Qiao, and Y. Wei, “Spidercnn: Deep learning on point sets with parameterized convolutional filters,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 87–102.

[4] Z. Zhang, B.-S. Hua, and S.-K. Yeung, “Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1607–1616.

[5] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, “Dynamic graph cnn for learning on point clouds,” in *ACM Transactions on Graphics (TOG)*, vol. 38, no. 5. ACM, 2019, pp. 1–12.

[6] H. Thomas, C. R. Qi, J. E. Deschaud, and B. Marcotegui, "Kpconv: Flexible and deformable convolution for point clouds," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 6411–6420.

[7] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4490–4499, 2018.

[8] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," in *Proceedings of the European Conference on Computer Vision*. Springer, 2018, pp. 268–283.

[9] C. Zhang, H. Wan, S. Liu, X. Shen, and Z. Wu, "Point-voxel transformer: An efficient approach to 3d deep learning," *CoRR*, vol. abs/2108.06076, 2021. [Online]. Available: https://arxiv.org/abs/2108.06076

[10] Y. C. Eldar, M. Lindenbaum, M. Porat, and Y. Y. Zeevi, "The farthest point strategy for progressive image sampling," *IEEE Transactions on Image Processing*, vol. 6, no. 9, pp. 1305–1315, 1997.

[11] M.-J. Rakotosaona, V. La Barbera, P. Guerrero, N. J. Mitra, and M. Ovsjanikov, "Pointcleannet: Learning to denoise and remove outliers from dense point clouds," in *Computer Graphics Forum*, vol. 39, no. 2. Wiley Online Library, 2020, pp. 93–104.

[12] FAO, "Global forest resources assessment 2020: Main report. rome," 2020.

[13] G. B. Bonan, "Forests and climate change: forcings, feedbacks, and the climate benefits of forests," *science*, vol. 320, no. 5882, pp. 1444–1449, 2008.

[14] Y. Pan, R. A. Birdsey, J. Fang, R. Houghton, P. E. Kauppi, W. A. Kurz, O. L. Phillips, A. Shvidenko, S. L. Lewis, J. G. Canadell *et al.*, "A large and persistent carbon sink in the world's forests," *Science*, vol. 333, no. 6045, pp. 988–993, 2011.

[15] L. Gibson, T. M. Lee, L. P. Koh, B. W. Brook, T. A. Gardner, J. Barlow, C. A. Peres, C. J. Bradshaw, W. F. Laurance, T. E. Lovejoy *et al.*, "Primary forests are irreplaceable for sustaining tropical biodiversity," *Nature*, vol. 478, no. 7369, pp. 378–381, 2011.

[16] D. Ellison, C. E. Morris, B. Locatelli, D. Sheil, J. Cohen, D. Murdiyarso, V. Gutierrez, M. Van Noordwijk, I. F. Creed, J. Pokorny *et al.*, "Trees, forests and water: Cool insights for a hot world," *Global environmental change*, vol. 43, pp. 51–61, 2017.

[17] K. Lim, P. Treitz, M. Wulder, B. St-Onge, and M. Flood, "Lidar remote sensing of forest structure," *Progress in physical geography*, vol. 27, no. 1, pp. 88–106, 2003.

[18] C. Qian, C. Yao, H. Ma, J. Xu, and J. Wang, "Tree species classification using airborne lidar data based on individual tree segmentation and shape fitting," *Remote Sensing*, vol. 15, no. 2, 2023. [Online]. Available: https://www.mdpi.com/2072-4292/15/2/406

[19] A. Ferraz, S. Saatchi, C. Mallet, S. Jacquemoud, G. GonÃ§alves, C. A. Silva, P. Soares, M. TomÃ©, and L. Pereira, "Airborne lidar estimation of aboveground forest biomass in the absence of field inventory," *Remote Sensing*, vol. 8, no. 8, 2016. [Online]. Available: https://www.mdpi.com/2072-4292/8/8/653

[20] R. J. Keenan, G. A. Reams, F. Achard, J. V. de Freitas, A. Grainger, and E. Lindquist, "Dynamics of global forest area: Results from the fao global forest

resources assessment 2015," *Forest Ecology and Management*, vol. 352, pp. 9–20, 2015.

[21] S. Amatya, M. Karkee, A. Gongal, Q. Zhang, and M. D. Whiting, "Detection of cherry tree branches with full foliage in planar architecture for automated sweet-cherry harvesting," *Biosystems Engineering*, vol. 146, pp. 3–15, 2016, special Issue: Advances in Robotic Agriculture for Crops. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1537511015001683

[22] G. Lin, C. Wang, Y. Xu, M. Wang, Z. Zhang, and L. Zhu, "Real-time guava tree-part segmentation using fully convolutional network with channel and spatial attention," *Front. Plant Sci.*, vol. 13, p. 991487, Sep. 2022.

[23] M. I. Disney, M. Boni Vicari, A. Burt, K. Calders, S. L. Lewis, P. Raumonen, and P. Wilkes, "Weighing trees with lasers: advances, challenges and opportunities," *Interface Focus*, vol. 8, no. 2, p. 20170048, 2018. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rsfs.2017.0048

[24] M. Ã...kerblom, P. Raumonen, E. Casella, M. I. Disney, F. M. Danson, R. Gaulton, L. A. Schofield, and M. Kaasalainen, "Non-intersecting leaf insertion algorithm for tree structure models," *Interface Focus*, vol. 8, no. 2, p. 20170045, 2018. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rsfs.2017.0045

[25] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1912–1920, 2015.

[26] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, "Shapenet:

An information-rich 3d model repository," *CoRR*, vol. abs/1512.03012, 2015. [Online]. Available: http://arxiv.org/abs/1512.03012

[27] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese, "Joint 3d scene reconstruction and class segmentation," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 3790–3800.

[28] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 226–231, 1996.

[29] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," in *International Journal of Computer Vision*, vol. 59, no. 2. Springer, 2004, pp. 167–181.

[30] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[31] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1118–1123, 2008.

[32] A. Y. Ng, M. I. Jordan, and Y. Weiss, "Spectral clustering: Analysis and algorithm," *Advances in neural information processing systems*, vol. 14, pp. 849–856, 2002.

[33] C. Lanczos, "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators," *United States Governm. Press Office Los Angeles, CA*, vol. 45, no. 4, pp. 255–282, 1950.

[34] V. A. Traag, "Louvain algorithm: review," *arXiv preprint arXiv:1810.08473*, 2019.

[35] V. A. Traag and J. Bruggeman, "Narrow scope for hubness phenomenon," *Scientific reports*, vol. 1, no. 1, pp. 1–5, 2011.

[36] B. Tesson, T. Reme, and A. Desmoulière, "Network analyses: basic concepts and applications in oncology," *Bulletin du cancer*, vol. 97, no. 10, pp. 1285–1295, 2010.

[37] R. F. Betzel and D. S. Bassett, "The modular organization of human anatomical brain networks: Accounting for the cost of wiring," *Network neuroscience*, vol. 2, no. 3, pp. 222–241, 2018.

[38] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical Review E*, vol. 76, no. 3, p. 036106, 2007.

[39] G. Cordasco and L. Gargano, "Community detection via semi-synchronous label propagation algorithms," in *Business Applications of Social Network Analysis (BASNA), 2010 IEEE International Workshop on.* IEEE, 2010, pp. 1–8.

[40] F. Parés, D. Garcia-Gasulla, J. Borge-Holthoefer, and Y. Moreno, "Fluid communities: A competitive and highly scalable community detection algorithm," *arXiv preprint arXiv:1703.09307*, 2017.

[41] X. Wang, M. Bai, and Y. Wang, "Localized label propagation for knn graph construction," *2011 International Conference on Internet of Things and 4th In-*

ternational Conference on Cyber, Physical and Social Computing, pp. 630–633, 2011.

[42] A. Lindenmayer, “Mathematical models for cellular interactions in development II. Simple and branching filaments with two-sided inputs,” *Journal of Theoretical Biology*, vol. 18, no. 3, pp. 300–315, Jan. 1968.

[43] F. Wang and M. Bryson, “Tree segmentation and parameter measurement from point clouds using deep and handcrafted features,” *Remote Sensing*, vol. 15, no. 4, 2023. [Online]. Available: https://www.mdpi.com/2072-4292/15/4/1086

[44] “Speedtree awards.” [Online]. Available: https://store.speedtree.com/awards/

[45] T. Reid, “Hollywood’s movie tech wizards honoured by oscars organizers,” Feb 2015. [Online]. Available: https://www.livemint.com/Consumer/Jh1N22EQWOzbwbjcU8jORI/Hollywoods-movie-tech-wizards-honoured-by-Oscars-organizers.html

[46] “Speedtree brings photoreal vegetation to the wolf of wall street,” January 2014. [Online]. Available: https://www.cgw.com/Press-Center/Web-Exclusives/2014/SpeedTree-Brings-Photoreal-Vegetation-to-The-Wol.aspx

[47] I. Interactive Data Visualization, *SpeedTree Cinema*, ver. 8, Lexington, SC, USA, 2022 [Online]. [Online]. Available: https://store.speedtree.com/

[48] Z. Xi, C. Hopkinson, S. B. Rood, and D. R. Peddle, “See the forest and the trees: effective machine and deep learning algorithms for wood filtering and tree species classification from terrestrial laser scanning,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 168, pp. 1–16, 2020.

[49] Z. Zhang, B. Hua, and S. Yeung, "Shellnet: Efficient point cloud convolutional neural networks using concentric shells statistics," *CoRR*, vol. abs/1908.06295, 2019. [Online]. Available: http://arxiv.org/abs/1908.06295

[50] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *CoRR*, vol. abs/1801.07829, 2018. [Online]. Available: http://arxiv.org/abs/1801.07829

[51] Z. Wu, S. Song, A. Khosla, X. Tang, and J. Xiao, "3d shapenets for 2.5d object recognition and next-best-view prediction," *CoRR*, vol. abs/1406.5670, 2014. [Online]. Available: http://arxiv.org/abs/1406.5670

[52] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," *Symposium on Geometry Processing*, vol. 7, no. 4, p. 2, 2006.

[53] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, "The ball-pivoting algorithm for surface reconstruction," in *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4. IEEE, 1999, pp. 349–359.

[54] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM siggraph computer graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[55] H. Edelsbrunner and E. P. Mücke, "Alpha shapes," *Journal of Computational Geometry*, vol. 15, no. 1, pp. 41–68, 1994.

[56] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.