



Berner Fachhochschule  
Haute école spécialisée bernoise  
Bern University of Applied Sciences



---

# Master's Thesis

TreeGPT: Generative pre-trained transformer for forestry applications with 3D point clouds

Course of study

Master of Science in Engineering - Profile Data Science

Author

Ivo Gasparini

Advisor

Prof. Dr. Souhir Ben Souissi

Expert

Dr. Andrea Cimatoribus (Pix4D)

August 7, 2025



I would like to thank Prof. Dr. Souhir Ben Souissi for her continued support, especially with the project management aspects, and Dr. Andrea Cimatoribus for his sustained interest, technical insights, and friendly in-depth exchanges about model training behavior.

I thank my family for sustaining me through months of laborious work and for their regular check-ins. A big thanks goes also to Lucie, who has been a constant source of support during the office days and beyond.

Finally, I thank my colleagues Katharina and Roman from project UW for taking on work while I was busy with my thesis, and Jean-Baptiste for introducing me to the world of Python and for his patience and accommodating approach as a supervisor.





## Abstract

Swiss forests cover 32% of the national territory and provide essential ecosystem services, yet current inventory practices rely on costly manual field surveys with limited spatial resolution. This study applies **self-supervised learning with Point-GPT for automated tree species classification from LiDAR point clouds** to enable improved forest inventories.

PointGPT extends generative pre-trained transformers to point clouds. The methodology employs a three-stage training approach: **pre-training, post-pre-training, and fine-tuning**. Datasets totaling **202'149 trees** were assembled, combining newly processed data with existing public datasets. Unlabeled pre-training data was generated by processing public airborne laser scanning data through SegmentAnyTree and creating synthetic trees, then complemented with datasets from the literature. Labeled datasets comprised SynForest simulation data (17 species) and the established **FOR-Species20K benchmark** dataset (33 species), on which model performance was assessed.

Comprehensive experimental evaluation across approximately 300 training runs demonstrated that **pre-training achieved 37% top-1 and 72% top-5 linear probing accuracy**, with post-pre-training showing 10% performance improvements over the pre-training baseline. The **final TreeGPT model achieved 76% accuracy, 0.79 precision, 0.67 recall, and 0.67 F1-score** on validation data, matching the performance of Ensemble PointNet++, the leading point-cloud-based method on FOR-Species20K, while **using 4 times lower data resolution**. Parameter-efficient fine-tuning using **spectral adapters proved ineffective** (50% accuracy), while training from scratch failed entirely, confirming benefits from self-supervised learning. Experiments showed that the **inclusion of fully synthetic data improved model performance**.

This work explores the potential of applying generative pre-trained transformers to automated tree species classification. Significant **opportunities for improvement remain**, particularly training high-capacity models on high-resolution LiDAR data and subsequently applying them to lower-resolution data. The self-supervised methodology contributes towards establishing a **foundation model approach** in forestry vision tasks.

*Keywords:* self-supervised learning, tree species classification, LiDAR, point clouds, PointGPT, transformer, forest inventory, 3D computer vision.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context and opportunities . . . . .	1
1.2	Research problem . . . . .	2
1.3	Objectives and structure . . . . .	2
<b>2</b>	<b>Background and related work</b>	<b>5</b>
2.1	Overview of swiss forests . . . . .	5
2.2	LiDAR technology . . . . .	7
2.3	Forest inventory from LiDAR point clouds . . . . .	9
2.4	Tree species classification from LiDAR point clouds . . . . .	10
2.5	Self-supervised learning . . . . .	12
2.6	Parameter-efficient fine-tuning . . . . .	12
<b>3</b>	<b>Methodology</b>	<b>15</b>
3.1	Overall research design . . . . .	15
3.2	Project management . . . . .	17
3.3	Information sources . . . . .	18
3.4	Infrastructure and hardware . . . . .	18
3.5	Models . . . . .	19
3.5.1	PointGPT . . . . .	19
3.5.2	PointGST . . . . .	24
3.6	Data . . . . .	26
3.6.1	Custom datasets . . . . .	26
3.6.2	Literature dataset . . . . .	32
3.6.3	Benchmark dataset . . . . .	33
3.6.4	Dataset summary . . . . .	37
<b>4</b>	<b>Results and discussion</b>	<b>39</b>
4.1	Pipelines implementation . . . . .	39
4.1.1	Preprocessing . . . . .	39
4.1.2	Training . . . . .	42
4.2	Training runs . . . . .	49
4.2.1	Overview . . . . .	49
4.2.2	Pre-training . . . . .	50
4.2.3	Post-pre-training . . . . .	58
4.2.4	Fine-tuning . . . . .	60

4.3 Best model . . . . .	68
<b>5 Conclusions</b>	<b>71</b>
<b>Bibliography</b>	<b>75</b>
<b>List of Figures</b>	<b>85</b>
<b>List of Tables</b>	<b>87</b>
<b>Listings</b>	<b>89</b>
<b>Glossary</b>	<b>91</b>
Glossary . . . . .	94

# 1 Introduction

## 1.1 Context and opportunities

Switzerland's forests cover approximately 1.32 million hectares, representing 32% of the total land area as of 2023 [1], [2]. Swiss forests provide numerous ecosystem services [3]: they produce timber, offer protection against natural hazards, contribute significantly to biodiversity and carbon storage, and serve as valuable recreational spaces. The Swiss forestry sector operates within a framework of sustainable management principles, anchored in multifunctional forest management [4]. Systematic monitoring of this valuable resource through instruments such as the Swiss National Forest Inventory (Landesforstinventar, [1]) provides essential data for evidence-based management decisions at federal and cantonal levels, including the allocation of federal subsidies totaling 451 million CHF for the period 2025-2028 [5]. Despite increasing integration of remote sensing approaches [6], forest inventories still largely rely on costly manual field surveys conducted on sparse grids of sample plots, providing limited spatial detail. At the stand and individual tree level, management decisions are often based on incomplete or absent data, leading to suboptimal outcomes that can impact both the fulfillment of forest functions and the economic viability of Swiss forest enterprises — 41% operated at a deficit in 2022 [7].

Under the paradigm of Forestry 5.0 [8], the forestry sector can use more advanced technologies to improve forest management and operations. This transformation becomes increasingly relevant given the challenges posed by climate change. Advances in remote sensing technologies and data analysis techniques offer the possibility to extract comprehensive information from forest ecosystems [9]. By applying 3D computer vision to LiDAR (Light Detection and Ranging) data, numerous applications become feasible, ranging from silvicultural metrics extraction (including tree height, diameter at breast height, crown and trunk volumes, and tree species identification) to the creation of digital forest twins [10]. Detailed forest inventories could potentially be automated [11]–[14], providing comprehensive data at high spatial resolution. This capability enables more precise and targeted management across scales, from individual trees (precision forestry, [15]) and stands to entire national forest extents. Existing instruments such as the Swiss National Forest Inventory could be significantly enhanced by higher-precision inventories, improving both public subsidy allocation and forestry operation effectiveness. The declining costs of LiDAR data acquisition, driven partly by the

expanding autonomous vehicle industry, further support this technological shift. Public entities such as Swisstopo in Switzerland are implementing comprehensive national LiDAR acquisition campaigns while providing free data access [16], creating favorable conditions for widespread further adoption of these techniques.

### 1.2 Research problem

Despite promising advances, automated extraction of silvicultural metrics from LiDAR point clouds and automated full forest inventorying have not yet achieved large-scale practical implementation. Tree species identification represents one of the primary challenges in this domain. Multiple approaches have been explored in the literature (see [17] for a comprehensive overview). These methods utilize either LiDAR data alone or combine LiDAR with RGB or hyperspectral photogrammetry [18]. Methodological approaches include classical machine learning algorithms such as random forests and support vector machines based on extracted features, 2D vision models applied to point cloud slices [19], and deep learning models applied directly to point clouds (see section 2.4).

Self-Supervised Learning (SSL) is an established technique in deep learning that enables training models on large quantities of unlabeled data through various pretext tasks [20]. Through these pretext tasks, models acquire general knowledge about the data that can subsequently be transferred to downstream tasks via fine-tuning or transfer learning. Recent developments have extended this technique to point clouds [21]–[24]. The abundance of available unlabeled data facilitates the application of transformers [25], which have become the dominant architecture in deep learning, including computer vision [26]. This dominance has been driven by the success of models such as the Generative Pretrained Transformer (GPT) [27], [28] in natural language processing. Transformers exhibit superior scaling behavior [29], enabling them to outperform other architectures by increasing dataset and model size.

To the best of the author’s knowledge, SSL has not yet been explored for tree species classification using purely point clouds. While existing approaches have applied SSL to individual tree segmentation [30], others rely on combinations of data sources for species classification [31], [32]. Implementing SSL using a transformer architecture specifically for tree species classification while leveraging the large amounts of freely available unlabeled LiDAR data represents a significant research gap.

### 1.3 Objectives and structure

Considering what has been outlined above, the objectives of this work are:

- ▶ Implement a transformer architecture for tree species classification on LiDAR point clouds as downstream task.
- ▶ Collect and create forest point cloud datasets for the pre-training and the downstream task.
- ▶ Develop pipelines for point cloud data processing, model SSL pre-training, fine-tuning, and inference.
- ▶ Evaluate and compare the performance of the models against baseline approaches.
- ▶ Conduct ablation studies to understand the impact of different pipeline implementations, training strategies, and architectural choices on computational efficiency and model performance.

This work aims to advance SSL in 3D computer vision on point clouds for forestry applications such as tree species recognition.

The thesis is organized as follows. Section 2 reviews the relevant literature on LiDAR technology, forest inventory applications, and point-based deep learning methods. Chapter 3 presents the research design, the chosen SSL architectures, dataset development, and practical implementation. Chapter 4 reports and discusses the pipeline implementations and the experimental results across pre-training, post-pre-training, and fine-tuning stages. Chapter 5 provides a summary and concluding remarks on the potential of transformer-based SSL for automated tree species classification.





## 2 Background and related work

### 2.1 Overview of swiss forests

The swiss National Forest Inventory [1], on which the following data is based, currently employs around 6,600 sampling areas arranged at intervals of 1.4 km. To compensate for the sampling error inherent in this sampling density, the data are evaluated using estimation procedures based on aerial images and vegetation height models derived from LiDAR data.

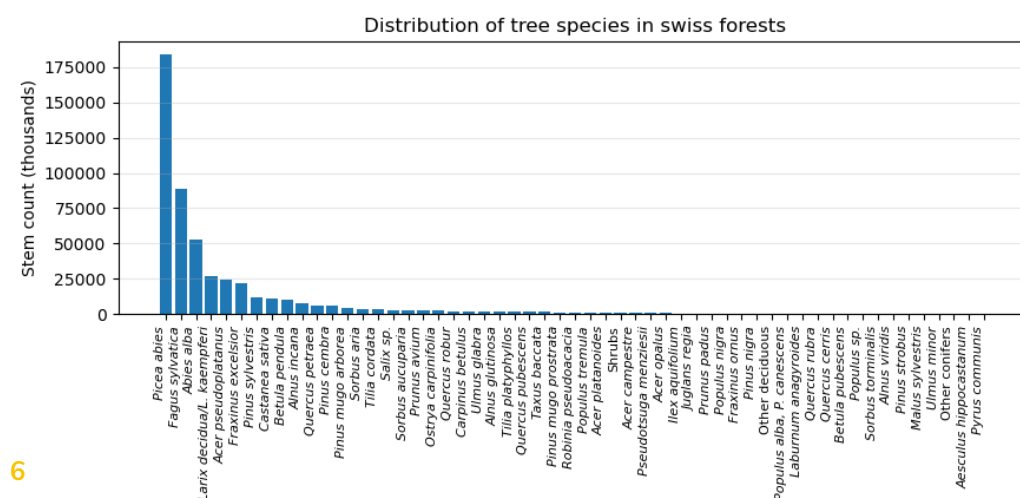
swiss forests extend over around one third of the national surface. The forest area has been increasing for the last 150 years, with a 2.4% increase between the third and fourth National Forest Inventories (LFI3: 2004 - 2006; and LFI4: 2009 - 2017). The distribution of forest coverage varies across the country's geographical regions, ranging from 41% coverage in the Jura mountains to 24% in the Mittelland, with the Voralpen, Alps, and Alpensüdseite regions maintaining 35%, 27%, and 54% forest coverage respectively. swiss forests contain substantial timber resources, with a growing stock that amounts to 450 million m<sup>3</sup> or 374 m<sup>3</sup> per hectare, representing one of the highest forest densities in Europe.

The forest landscape consists primarily of coniferous species, which constitute 69% of the total forest volume, while deciduous species account for the remaining 31%. *Picea abies* is the dominant species, representing 44% of the total forest volume, followed by *Fagus sylvatica* at 17% and *Abies alba* at 15%. Approximately 42% of swiss forest stands are essentially dominated by coniferous species, while 24% are dominated by deciduous species. Forest stand diversity is increasing: mixed stands are becoming more prevalent, with monospecific pure stands now representing only 17% of forest area, while 48% of stands contain two to three tree species and 34% contain more than three species.

When considering stem count (number of individual tree trunks), the species distribution is heavily skewed toward a few dominant species (fig. 2.1 and table 2.1). The top 3 species account for 65.4% of all stems, the top 5 species represent 75.7%, and the top 10 species comprise 88.2% of the total stem count. *Picea abies* remains the dominant species with 37.0% of all stems, followed by *Fagus sylvatica* at 17.8%, but *Abies alba* drops to 10.6% of total stem count. Discrepancies like these indicate, for example, that *Abies alba* trees are on average bigger compared to *Picea abies* and *Fagus sylvatica*, suggesting the presence of older, more mature *Abies alba* stands.

Species	Count	%	Species	Count	%
<i>Picea abies</i>	183 902	37.00	<i>Populus tremula</i>	1250	0.25
<i>Fagus sylvatica</i>	88 523	17.81	<i>Acer platanoides</i>	1217	0.24
<i>Abies alba</i>	52 794	10.62	Shrubs	1185	0.24
<i>Larix decidua</i> /L. <i>kaempferi</i>	27 100	5.45	<i>Acer campestre</i>	1042	0.21
<i>Acer pseudoplatanus</i>	24 092	4.85	<i>Pseudotsuga menziesii</i>	868	0.17
<i>Fraxinus excelsior</i>	21 760	4.38	<i>Acer opalus</i>	820	0.16
<i>Pinus sylvestris</i>	11 519	2.32	<i>Ilex aquifolium</i>	460	0.09
<i>Castanea sativa</i>	10 937	2.20	<i>Juglans regia</i>	304	0.06
<i>Betula pendula</i>	10 102	2.03	<i>Prunus padus</i>	226	0.05
<i>Alnus incana</i>	7601	1.53	<i>Populus nigra</i>	202	0.04
<i>Quercus petraea</i>	5788	1.16	<i>Fraxinus ornus</i>	198	0.04
<i>Pinus cembra</i>	5720	1.15	<i>Pinus nigra</i>	185	0.04
<i>Pinus mugo arborea</i>	4606	0.93	Other deciduous	171	0.03
<i>Sorbus aria</i>	3784	0.76	<i>Populus alba</i> , <i>P. canescens</i>	161	0.03
<i>Tilia cordata</i>	3624	0.73	<i>Laburnum anagyroides</i>	107	0.02
<i>Salix</i> sp.	2772	0.56	<i>Quercus rubra</i>	96	0.02
<i>Sorbus aucuparia</i>	2732	0.55	<i>Quercus cerris</i>	84	0.02
<i>Prunus avium</i>	2707	0.54	<i>Betula pubescens</i>	82	0.02
<i>Ostrya carpinifolia</i>	2363	0.48	<i>Populus</i> sp.	79	0.02
<i>Quercus robur</i>	2196	0.44	<i>Sorbus torminalis</i>	76	0.02
<i>Carpinus betulus</i>	2129	0.43	<i>Alnus viridis</i>	70	0.01
<i>Ulmus glabra</i>	1903	0.38	<i>Pinus strobus</i>	56	0.01
<i>Alnus glutinosa</i>	1863	0.37	<i>Malus sylvestris</i>	50	0.01
<i>Tilia platyphyllos</i>	1663	0.33	<i>Ulmus minor</i>	40	0.01
<i>Quercus pubescens</i>	1651	0.33	Other conifers	34	0.01
<i>Taxus baccata</i>	1415	0.28	<i>Aesculus hippocastanum</i>	25	0.01
<i>Pinus mugo prostrata</i>	1375	0.28	<i>Pyrus communis</i>	14	0.00
<i>Robinia pseudoacacia</i>	1256	0.25			

**Table 2.1:** Tree species distribution of swiss forests based on stem count (thousands) [1].



**Figure 2.1:** Tree species distribution of swiss forests based on stem count (thousands) [1].

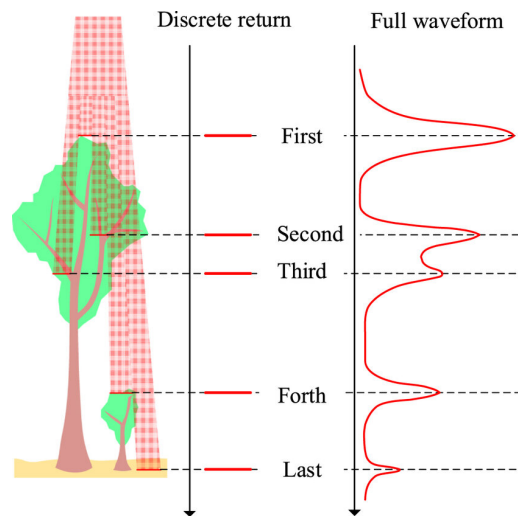
## 2.2 LiDAR technology

Light Detection and Ranging (LiDAR) is an active remote sensing technology that measures distances by emitting laser pulses and calculating the time-of-flight for reflected signals to return to the sensor [33]. The system operates on the principle that distance equals the speed of light multiplied by half the round-trip travel time:

$$d = \frac{c \cdot t}{2} \quad (2.1)$$

where  $d$  is the distance to the target,  $c$  is the speed of light ( $3 \times 10^8$  m/s), and  $t$  is the round-trip travel time. Modern LiDAR systems typically employ near-infrared wavelengths (1064 nm or 1550 nm) that are eye-safe and provide optimal atmospheric transmission.

The core components include a laser transmitter, receiver optics, photodetector, high-precision timing electronics, and positioning systems (GPS/IMU). When a laser pulse encounters a surface, a portion of the energy reflects back to the sensor. As visible in fig. 2.2, the intensity and timing of multiple returns from a single pulse enable detailed characterization of surface features and vegetation structure [34]. Full-waveform systems additionally record the complete continuous return signal, providing even more detailed information [35], [36].



**Figure 2.2:** Principle of multiple return LiDAR systems. Figure taken from [37].

Point positioning accuracy depends on precise sensor location and orientation. GPS provides absolute positioning while Inertial Measurement Units (IMU) record pitch, roll, and yaw angles. Post-processing algorithms integrate these data streams to calculate 3D coordinates for each laser return, typically achieving centimeter to millimeter-level precision under optimal conditions [38].

Each point in a LiDAR point cloud contains multiple attributes. Beyond the 3D spatial location (X, Y, Z), most points will have an intensity value, representing the amount of light energy recorded by the sensor, and return information (fig. 2.2), including return number (first, second, etc.) and total number of returns per pulse. Additional attributes may include RGB color values when LiDAR is combined with optical sensors, GPS timestamps, edge-of-flight line flags, and user-defined fields for specific applications.

LiDAR systems can be deployed on various platforms, each with distinct characteristics and applications (table 2.2):

- ▶ **Airborne Laser Scanning (ALS):** LiDAR systems that operate above the forest canopy, typically mounted on manned aircraft flying at altitudes of hundreds of meters to several kilometers. They allow for efficient coverage of extensive territories with homogeneous point clouds.
- ▶ **Unmanned Laser Scanning (ULS):** LiDAR systems carried by unmanned aerial vehicles (UAVs/drones) that can be categorized as either MLS or ALS depending on whether the UAV flies under or above the forest canopy during data collection [39].
- ▶ **Mobile Laser Scanning (MLS):** LiDAR systems that move under the forest canopy, including ground-based systems mounted on vehicles or handheld (Personal Laser Scanners, PLS). These systems capture high-density point clouds with significant variation in range data due to proximity to targets.
- ▶ **Terrestrial Laser Scanning (TLS):** Stationary ground-based LiDAR systems that provide the highest point densities and precision. Operating from fixed positions under the canopy, TLS systems excel at detailed characterization of small areas with minimal distance variation.

Parameter	Airborne Laser Scanning (ALS)	Unmanned Laser Scanning (ULS)	Mobile Laser Scanning (MLS)	Terrestrial Laser Scanning (TLS)
Point density	Up to few hundreds pts/m <sup>2</sup>	Up to few thousands pts/m <sup>2</sup>	Up to few thousands pts/m <sup>2</sup>	Up to tens or hundred of thousands pts/m <sup>2</sup>
Accuracy	5–15 cm vertical, 10–25 cm horizontal	1–5 cm vertical, 2–10 cm horizontal	5–10 mm	1–5 mm
Operating range	Hundreds of m to few km	Tens to hundreds of m	Up to hundreds of m	Up to hundreds of m
Coverage rate	Up to hundreds of km <sup>2</sup> /h	Up to a few km <sup>2</sup> /h	A few ha/h	Single scans

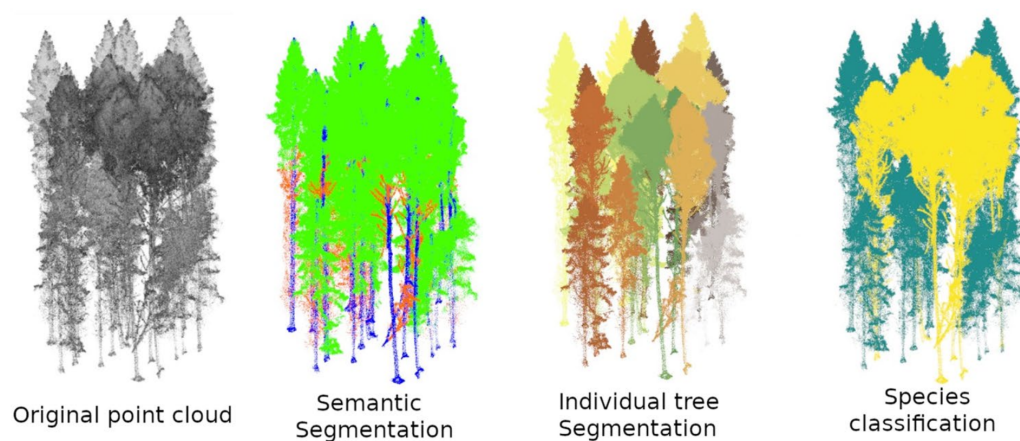
**Table 2.2:** Comparison of LiDAR platform characteristics. From [40], p.54, adapted.

## 2.3 Forest inventory from LiDAR point clouds

In the context of forest inventories, three main tasks applied to point clouds are particularly relevant (fig. 2.3):

- ▶ **Instance segmentation:** this task divides the point cloud into distinct individual object instances. In forestry applications, this approach enables the delineation of individual trees within larger forest point clouds [30], [41]–[43].
- ▶ **Semantic segmentation:** this process assigns semantic labels to each individual point within the cloud. In forest environments, this technique distinguishes between different vegetation components, enabling the separation of leaves, branches, and trunks [44], [45].
- ▶ **Point cloud classification:** this task assigns a single label to an entire point cloud based on its overall characteristics. In forestry, this can be applied to identify tree species or to assess their health status (see section 2.4).

Instance and semantic segmentation are frequently addressed jointly due to their complementary nature [11], [46], [47]. Several downstream applications build upon these foundational tasks. Point cloud reconstruction and completion techniques [48] infer missing portions of the dataset to create complete tree models. Trunk Diameter at Breast Height (DBH) estimation [49], [50] uses semantically and instance-segmented stems, typically employing ellipse fitting algorithms for precise measurements. Additional regression tasks encompass volume and biomass estimations, which predict continuous variables directly from point cloud features. The integration of these extracted metrics enables the construction of comprehensive forest inventories with detailed tree-level information.



**Figure 2.3:** Overview of the main tasks applied to point clouds for forest inventory. Figure taken from [15].

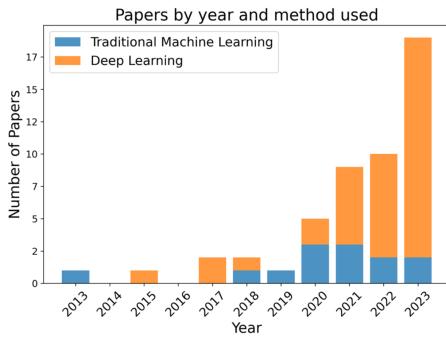
## 2.4 Tree species classification from LiDAR point clouds

Various prediction techniques have been applied to tree species recognition from LiDAR point clouds (fig. 2.4 and fig. 2.5). Machine Learning (ML) models rely heavily on feature engineering, the process of creating descriptive features that capture relevant information while reducing data complexity. Common ML approaches include Random Forest (RF) [51], Support Vector Machine (SVM) [52], Multi-layer Perceptron (MLP) [53], and XGBoost [54]. Compared to deep neural networks, these models offer faster computation and can learn effectively from small datasets. However, their performance depends critically on feature selection. Features are typically derived from geometric descriptions of point neighborhoods, defined either by fixed distance or k-nearest neighbors. For a neighborhood  $N_R$  around a point, a covariance matrix  $C$  is computed as:

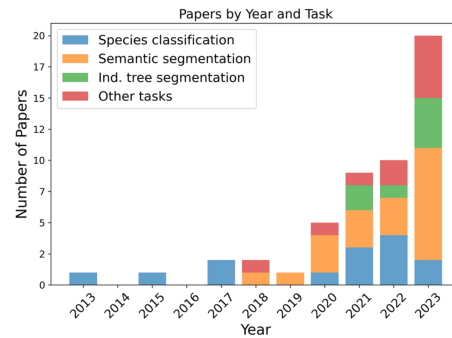
$$C(N_R) = \frac{1}{N} \sum_{p \in N_R} (p - \bar{p})(p - \bar{p})^T \quad (2.2)$$

where  $\bar{p}$  represents the neighborhood centroid. The eigenvectors  $e_1, e_2, e_3 \in \mathbb{R}^3$  and eigenvalues  $\lambda_1 \geq \lambda_2 \geq \lambda_3 \in \mathbb{R}$  of this covariance matrix are used to calculate shape-descriptive features. Typical features include linearity, planarity, sphericity, verticality, and anisotropy [55] at single or multiple scales [56]. Tree-based features are also commonly employed, including aggregations of geometric features and measurements such as height and convex hull volume [31]. More specialized approaches include Qualitative Structure Model (QSM), which represents stems and branches as a hierarchical set of cylinders to approximate their actual geometry [57].

Deep Learning (DL) approaches have emerged as an alternative to traditional machine learning methods (fig. 2.4). The promising performance of deep learning methods has driven increased research activity in tree species classification and other forest inventory tasks (fig. 2.5).



**Figure 2.4:** Number of papers using machine learning and deep learning methods on TLS forest point clouds over time. Figure from [15].



**Figure 2.5:** Number of papers per task (section 2.3) on TLS forest point clouds over time. Figure from [15].



DL models automatically learn feature representations from data, eliminating the need for manual feature engineering. However, this capability comes with increased computational complexity and typically requires larger datasets for effective training. Processing point cloud data presents challenges for DL architectures due to several characteristics of this data format. Point clouds lack inherent ordering, requiring all processing operations to be permutation-invariant, meaning that identical point sets in different arrangements must produce consistent outputs. This is different from other data modalities such as sequential text data or structured image grids. Additionally, point clouds exhibit irregular sampling densities and variable point counts per sample, further complicating neural network processing.

To address these challenges, several methodological approaches have been developed [58]. Many of them involve converting the point cloud to other more suitable formats. These include 2D projections of multiple viewpoints of the point cloud, and voxel grids that discretize 3D space into regular cubic cells (fig. 2.6). These methods typically rely on different flavors of Convolutional Neural Network (CNN) architectures including YOLO [59] and ResNet [60], and have been adapted for tree analysis [19]. For voxel-based representations, 3D CNNs such as VoxNet [61] and MinkowskiNet [62] have been developed and used in forestry [42]. MinkowskiNet addresses the sparsity in voxelized point clouds, where the majority of voxels remain empty, through sparse convolution operations. Alternatively, point-based methods process point clouds directly without intermediate transformations [21], [63]–[66]. While this approach remains less developed compared to other modalities, the field is in expansion.

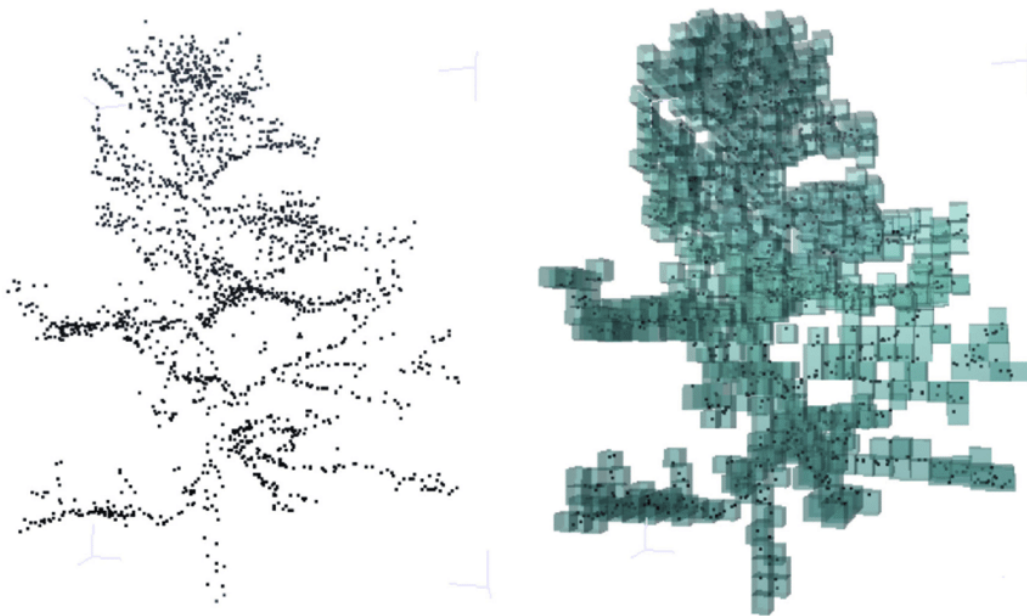


Figure 2.6: A voxelized point cloud of a tree. Figure taken from [67].

## 2.5 Self-supervised learning

Self-supervised learning (SSL) is a training paradigm in which models learn useful representations from unlabeled data by solving auxiliary tasks, known as pretext tasks, that do not require manual annotations [68]. SSL has proven very important for scaling deep learning in domains like vision and language by leveraging the abundance of raw data. Unlike supervised learning, which optimizes performance on a fixed labeled dataset, SSL focuses on learning general-purpose features that transfer well to a variety of downstream tasks.

Modern SSL methods are commonly divided into four families, each defined by a different learning objective.

- ▶ In contrastive learning, samples are contrasted against each other, and those belonging to the same distribution are pushed towards each other in the embedding space. In contrast, those belonging to different distributions are pulled against each other.
- ▶ Self-distillation methods train a model to match the output of a second copy of itself, which helps stabilize learning.
- ▶ Correlation-based approaches involve manipulating feature dimensions to be as uncorrelated as possible, so that the learned representations are more diverse and useful for downstream tasks.
- ▶ Masked modeling methods, inspired by BERT [21], involve hiding parts of the input data and training the model to predict the missing content, such as reconstructing masked point patches in a point cloud [23].

GPT-style architectures are a type of masked modeling SSL, since they use masked token prediction to train the model and to learn information-rich geometric features. The pre-trained models can then be fine-tuned (which usually involves re-training the model) or used as backbone to do transfer learning (which mostly keeps the bigger part of the parameters frozen as feature extractors).

## 2.6 Parameter-efficient fine-tuning

Parameter-Efficient fine-Tuning (PEFT) refers to adaptation techniques that modify only a small subset of parameters in pre-trained models rather than updating all model weights during downstream task training [69]. Traditional fine-tuning requires updating all parameters of large pre-trained models, leading to big computational costs and storage requirements, particularly when adapting to multiple downstream tasks. PEFT methods address these limitations by introducing lightweight, trainable modules or selecting specific parameter subsets for optimization while keeping the majority of pre-trained weights frozen.



Common PEFT approaches include adapter modules that insert small neural networks between existing layers [70], low-rank adaptation techniques that decompose weight updates into low-rank matrices [71], and prompt-based methods that prepend learnable tokens to model inputs [72], like a Classification (CLS) token for classification tasks. These techniques typically achieve comparable performance to full fine-tuning while requiring orders of magnitude fewer trainable parameters, making them particularly useful for large-scale model deployment and multi-task scenarios.



## 3 Methodology

### 3.1 Overall research design

This work investigated SSL with transformer architectures for tree species classification from LiDAR point clouds through five principal phases (fig. 3.1): architecture selection and analysis, dataset development and selection, pipeline implementation, training optimization, and performance evaluation.

The initial phase involved identifying state-of-the-art SSL approaches for point clouds, leading to the selection of PointGPT [24] (section 3.5.1) as the foundation architecture, as it represents one of the leading architectures on 3D point cloud classification benchmarks according to Papers with Code [73]. Its development into PointGST [74] (section 3.5.2) was also included, given that this architecture promises superior fine-tuning performance while being considerably more efficient through adapter layers. This selection was followed by comprehensive analysis and experimentation with both codebases to build the understanding and familiarity necessary for experimental investigation.

The dataset development phase addressed the challenge of collecting sufficient labeled and unlabeled point cloud data for effective SSL training and fine-tuning. Three datasets were assembled to support the training pipelines:

- ▶ **Custom dataset** (section 3.6.1): This dataset comprises two sub-datasets with distinct purposes. First, an unlabeled pre-training dataset was generated by segmenting public ALS point clouds from Canton Neuchâtel using the SegmentAnyTree architecture [42] and expanded with synthetic tree data created through the SimpleSynthTree framework [75]. Second, a labeled post-pre-training dataset was developed using SynForest [76].
- ▶ **Literature dataset** (section 3.6.2): This compilation consists of six datasets obtained from the literature, acquired from different LiDAR platforms and lacking tree species annotations. The dataset was used to expand the custom pre-training dataset.
- ▶ **Benchmark dataset** (section 3.6.3): FOR-Species 20K [77] was selected as the fine-tuning and benchmark dataset due to its explicit focus on tree species classification tasks.

The pipeline implementation phase involved developing training loops for distinct training stages and implementing performance optimizations.

The specific implementations and optimizations are detailed in section 4.1. The training stages were:

- ▶ **Pre-training:** SSL pre-training through a generative task, specifically next patch prediction of point cloud sequences (section 3.5.1).
- ▶ **Post-pre-training:** A combined training approach where the model simultaneously continues generative next patch prediction and performs tree species classification on labeled data.
- ▶ **Fine-tuning:** Both full fine-tuning and adapter-based fine-tuning (section 3.5.2) for tree species classification. Experiments also included maintaining the parallel generative task during fine-tuning.

Training optimizations included grid searches to identify optimal training parameters and dataset and model configurations. Additional optimizations encompassed weighted loss functions, weighted sampling strategies, and validation schemas incorporating linear probing for pre-training feature evaluation to ensure appropriate adaptation to the downstream classification task.

Performance evaluation was conducted through experiments across different model scales, enabling assessment of both the effectiveness of the SSL approach and the scalability of the proposed methodology (section 4.2).

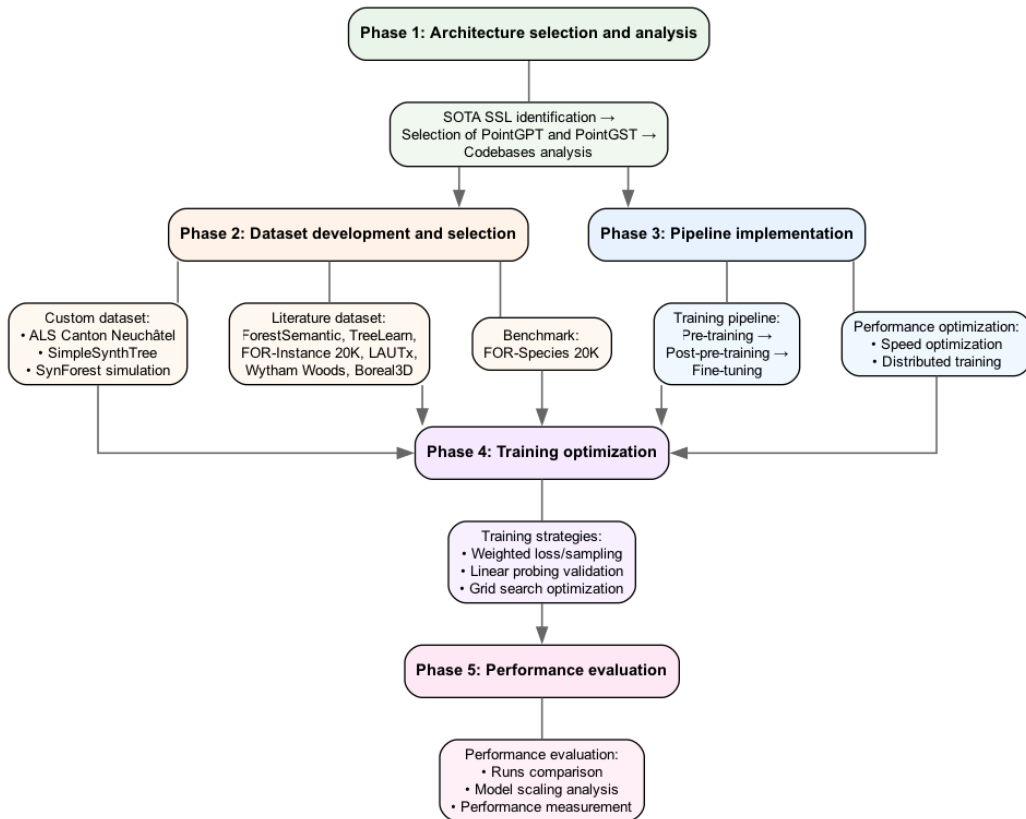


Figure 3.1: Flowchart of the research process.

## 3.2 Project management

The project management organized the phases outlined in fig. 3.1 into main activities (work packages) distributed across 9 months (fig. 3.2). As a part-time student with 50% work obligations, the author required more than one semester to complete the project. Over the project duration, it was possible to reduce the work percentage, allowing July to be fully dedicated to the thesis.

	Nov.24	Dec.24	Jan.25	Feb.25	Mar.25	Apr.25	May.25	Jun.25	Jul.25	Aug.25	
Activities	Month 1	Month 2	Month 3	Month 4	Month 5	Month 6	Month 7	Month 8	Month 9	Month 10	Duration [Months]
Phase 1: Architecture selection and analysis											3
Phase 2: Dataset development and selection											6
Phase 3: Pipeline implementation											6
Phase 4: Training optimization											3
Phase 5: Performance evaluation											2
Report writing											2
Milestones							M1			M2	
Progress report											

Figure 3.2: Project schedule of this master's thesis.

Phases 2 and 3 proceeded largely in parallel due to their high interdependence. These phases also consumed the majority of the project timeline. Technical challenges encountered in both phases, along with problems related to diverging training runs during pre-training (see section 4.2.2), extended phases 2 and 3 by approximately 2 months beyond the original plan. The main solutions adopted for the technical challenges are outlined in section 3.4. Consequently, phase 5 and report writing were delayed accordingly.

This delay was hardly avoidable, as significant results could not have been obtained without resolving these technical challenges. This represented the primary project management challenge, as the feasibility of completing the entire project within the given timeframe became uncertain at a certain point.

Key milestones included an expert meeting to assess preliminary results in month 7 and the final delivery and presentation in early August. There was ongoing exchange with the advisor throughout the project, while the expert became more involved from month 6 onward.

### 3.3 Information sources

Background literature was gathered from Web of Science, Google Scholar, Semantic Scholar, and arXiv to establish understanding of forestry applications of 3D computer vision on point clouds, point-based deep learning methodologies, transformer architectures, and self-supervised learning techniques. Reference tracing was employed extensively to identify additional relevant sources. Web searches were conducted to locate industry reports and technical documentation. Papers with Code and GitHub repositories were consulted to identify state-of-the-art implementations and available codebases.

### 3.4 Infrastructure and hardware

The experimental infrastructure comprised multiple computing environments for different phases of the research. Initial experimentation and research were conducted on a personal laptop, with the first pipelines implemented to support CPU operation to enable preliminary testing and development.

The primary computational work was performed on two Linux servers provided by Bern University of Applied Sciences (BFH). The newer server, designated "Apex", featured two NVIDIA A100 GPUs with 82 GB RAM each, though institutional policies restricted usage to a single GPU per session. The older server, "DGX Station", housed four NVIDIA V100 GPUs with 32 GB RAM each and was utilized extensively in distributed mode, achieving higher training speeds despite the older hardware.

Server access was managed through SSH connections and the JupyterLab IDE interface, with all operations executed within Docker containers. Environment configuration utilized Conda package management running through pyenv for Python version control. Resource management presented challenges throughout the project, primarily manifesting as CUDA out-of-memory errors and server disk space limitations. Command-line monitoring tools including `htop` and `ncdu` were employed extensively to address these bottlenecks. A notable issue involved JupyterLab's hidden trash folder accumulating over 700 GB of data within containers due to persistence across IDE sessions.

Frameworks used for custom dataset creation required custom implementation solutions due to system constraints. SegmentAnyTree [42] (section 3.6.1) necessitated execution within a nested container using `udocker` [78] to accommodate the multi-user environment without root privileges. The SynForest [76] (section 3.6.1) source code required modifications to convert Slurm [79] job scheduling for High Performance Computing (HPC) operations to standard execution and to remove containerized execution of Helios++ [80], which was subsequently installed directly on the machine to circumvent privilege restrictions.

The main codebase utilizes PyTorch as the deep learning framework, with external

dependencies evolving throughout the project duration. Initial implementations employed PyTorch3D [81] for Farthest Point Sampling (FPS), but compatibility issues with newer Python versions and dependencies necessitated compilation from source due to outdated community-maintained wheels. PyTorch3D was eventually replaced with CUDA kernels from the PyTorch PointNet++ codebase [64] to resolve dependency conflicts. Additional dependencies, including accelerated Chamfer Distance implementations, required source compilation and encountered compatibility issues with Apex’s system environment. These challenges were resolved by establishing a complete environment with all compilations on DGX Station and replicating it on Apex using Conda-Pack [82], which creates portable archives of conda environments for installation on other systems.

Version control and backup were managed through Git with a private repository hosted on GitHub. Training run logging initially utilized offline TensorBoard, the default logging solution in the PointGPT codebase, but was subsequently migrated to Weights and Biases.

## 3.5 Models

### 3.5.1 PointGPT

PointGPT [24] extends the concept of Generative Pre-trained Transformers (GPT) to point clouds, addressing three fundamental challenges:

1. Disorder properties of point clouds
2. Low information density compared to natural language
3. Gaps between generation and downstream tasks

The architecture employs an auto-regressive generation task to pre-train a transformer on point cloud sequences. PointGPT consists of three main components: a point cloud sequencer that converts unordered point clouds into ordered sequences, an extractor-generator based transformer decoder with dual masking strategy, and a prediction head for auto-regressive generation. The overall pipeline processes a point cloud  $X = \{x_1, x_2, \dots, x_M\} \subseteq \mathbb{R}^3$  through sequential patch prediction.

In PointGPT, tokens represent embedded point groups (or patches). Each token encodes the geometric properties of a local neighborhood of group size  $k$ , typically 32 points. The sequencer converts the unordered point cloud into ordered token sequences through a three-stage process (fig. 3.3). First, point group partitioning samples  $n$  center points ( $C$ ) using Farthest Point Sampling (FPS) and constructs point groups using K-Nearest Neighbors (KNN):

$$C = \text{FPS}(X), \quad C \in \mathbb{R}^{n \times 3} \quad (3.1)$$

$$P = \text{KNN}(C, X), \quad P \in \mathbb{R}^{n \times k \times 3} \quad (3.2)$$

FPS is a deterministic downsampling algorithm that iteratively selects points to maximize spatial coverage. Beginning with an initial seed point, the algorithm progressively selects points that maintain maximum distance from all previously chosen points:

$$\mathbf{c}_{j+1} = \arg \max_{\mathbf{x} \in X \setminus S} \min_{\mathbf{s} \in S} \|\mathbf{x} - \mathbf{s}\|_2 \quad (3.3)$$

where  $S = \{\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_j\}$  represents the set of already selected center points,  $\mathbf{c}_{j+1}$  denotes the next center point to be selected,  $X$  denotes the original point cloud, and  $\|\cdot\|_2$  indicates the Euclidean distance.

Following center point selection, KNN constructs local point groups around each center. For each center point  $\mathbf{c}_i$ , KNN identifies the  $k$  closest points from the original point cloud  $X$  based on Euclidean distance.

This creates  $n$  groups, where each group  $P_i$  contains  $k$  points representing a local neighborhood around center point  $\mathbf{c}_i$ . To establish sequential order, center points are encoded into one-dimensional space using Morton code and sorted along order  $O$  accordingly, obtaining sorted centers  $C^s$  and sorted point groups  $P^s$  (patches):

$$O = \arg \max(\text{MortonCode}(C)), \quad O \in \mathbb{R}^{n \times 1} \quad (3.4)$$

$$C^s, P^s = C[O], P[O], \quad C^s \in \mathbb{R}^{n \times 3}, P^s \in \mathbb{R}^{n \times k \times 3} \quad (3.5)$$

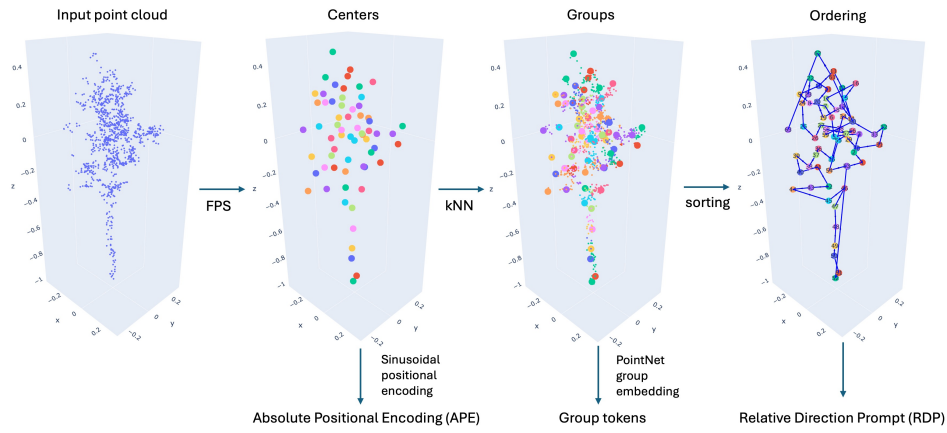
Morton codes convert 3D point coordinates into a single number that encodes spatial position. The code is generated by combining the binary representations of the  $(x, y, z)$  coordinates.

Binary numbers have bit positions, where the leftmost bit is the most significant (contributing the largest value) and the rightmost bit is the least significant (contributing the smallest value). For example, in the binary number  $101_2$ , the leftmost bit (position 2) contributes  $1 \times 2^2 = 4$ , the middle bit (position 1) contributes  $0 \times 2^1 = 0$ , and the rightmost bit (position 0) contributes  $1 \times 2^0 = 1$ , yielding the decimal value 5. For an example point at coordinates  $(2, 1, 3)$ , each coordinate is first converted to binary with equal bit lengths:  $x = 2 = 010_2$ ,  $y = 1 = 001_2$ ,  $z = 3 = 011_2$ . The Morton code is constructed by combining bits from all three coordinates:



- ▶ Position 2 (most significant):  $z_2 = 0, y_2 = 0, x_2 = 0$
- ▶ Position 1 (middle):  $z_1 = 1, y_1 = 0, x_1 = 1$
- ▶ Position 0 (least significant):  $z_0 = 1, y_0 = 1, x_0 = 0$

The resulting Morton code becomes  $z_2y_2x_2z_1y_1x_1z_0y_0x_0 = 000101110_2$ , which converts to the decimal value 46. Each center point  $c_i$  is assigned a Morton code through this process, and the centers are subsequently sorted by these values to create ordered sequences  $C^s$  and  $P^s$ . In the actual PointGPT codebase [83], Morton codes are replaced by a nearest-neighbor selection algorithm that iteratively selects the closest unvisited center point at each step, though this substitution is not disclosed in the published paper.



**Figure 3.3:** Processing of an input point cloud of a tree of the pre-training dataset in PointGPT.

Each point group is processed by a PointNet [63] network to produce a  $D$ -dimensional feature vector, which is the token  $T$ :

$$T = \text{PointNet}(P^s), \quad T \in \mathbb{R}^{n \times D} \quad (3.6)$$

with coordinates normalized relative to center points. The resulting tokens  $T = \{T_1, T_2, \dots, T_n\}$  represent embedded local geometric features, with each token  $T_i \in \mathbb{R}^D$  encoding the geometric properties of a point group  $P_i^s$ . With this, the disorder challenge is addressed.

To deal with the low information density of point clouds, the transformer decoder incorporates a dual masking strategy. This strategy additionally randomly masks

a proportion of attending preceding tokens during pre-training. The self-attention mechanism with dual masking operates on tokens, where each token can attend to a subset of preceding tokens:

$$\text{SelfAttention}(T) = \text{softmax} \left( \frac{QK^T}{\sqrt{D}} - (1 - M^d) \cdot \infty \right) V \quad (3.7)$$

where  $Q, K, V$  are query, key, and value matrices derived from  $T$ , and  $M^d$  is the dual mask with masked locations set to 0 and unmasked locations set to 1.

When masking attending tokens, the system controls which other tokens each token can attend to. In practical terms, when token 3 (representing patch 3) would normally attend to token 1 and token 2, dual masking might force it to attend only to token 1, thereby reducing the available contextual information and encouraging the model to learn more discriminative features.

The architecture separates representation learning from generation through distinct extractor and generator modules (fig. 3.4). The extractor is composed of transformer decoder blocks, learning latent representations. To learn global structure information, sinusoidal positional encodings are added to the group centers [25], obtaining the Absolute Positional Encodings (APE). The generator contains fewer transformer blocks and incorporates Relative Direction Prompts (RDP) to give the model information about group order and direction:

$$\text{RDP}_i = \text{PE} \left( \frac{C_{i+1}^s - C_i^s}{\|C_{i+1}^s - C_i^s\|_2} \right), \quad i \in \{1, \dots, n'\} \quad (3.8)$$

where  $n' = n - 1$  and PE denotes positional encoding. The extractor-generator procedure is formulated as:

$$\mathcal{T} = \text{Extractor}(T + \text{APE}), \quad \mathcal{T} \in \mathbb{R}^{n \times D} \quad (3.9)$$

$$T^g = \text{Generator}(\mathcal{T}_{1:n'} + \text{RDP}), \quad T^g \in \mathbb{R}^{n' \times D} \quad (3.10)$$

where  $\mathcal{T}_{1:n'}$  represents the encoded feature vectors output by the extractor, and  $T^g$  the predicted encoded point group from the generator.

The input sequences are constructed differently for pre-training and fine-tuning phases (fig. 3.4). During pre-training, the extractor uses a Start Of Sequence (SOS) token with APE, while the generator uses RDP instead. Extractor sequence:

$$\mathbf{T}_{\text{ext pre}} = [\text{SOS} + \text{SOS}_{\text{PE}}, \mathbf{T}_1 + \text{APE}_1, \mathbf{T}_2 + \text{APE}_2, \dots, \mathbf{T}_{N-1} + \text{APE}_{N-1}] \quad (3.11)$$

For downstream classification tasks, learnable Classification (CLS) tokens (section 2.6) are introduced to the pre-trained model. Extractor sequence:

$$\mathbf{T}_{\text{ext cls}} = [\text{CLS} + \text{CLS}_{\text{PE}}, \mathbf{T}_1 + \text{APE}_1, \mathbf{T}_2 + \text{APE}_2, \dots, \mathbf{T}_N + \text{APE}_N] \quad (3.12)$$

The SOS token provides generative context during pre-training, while CLS tokens encapsulate global point cloud properties useful for classification. The group tokens  $\mathbf{T}_i$  contain geometric features from PointNet, APE encodes spatial ordering from Morton sequencing, and RDP provides directional information for autoregressive generation.

During generation, the prediction head consists of a two-layer MLP that projects generator tokens to coordinate space, predicting the content of subsequent point groups  $P^{pd}$ :

$$P^{pd} = \text{Reshape}(\text{MLP}(T^g)), \quad P^{pd} \in \mathbb{R}^{n' \times k \times 3} \quad (3.13)$$

When predicting "the next token", the model predicts the features of the next local point group in the spatial sequence, rather than individual points. This enables the model to learn meaningful spatial relationships and geometric patterns across local neighborhoods.

The generation loss combines  $\ell_1$  and  $\ell_2$  forms of Chamfer Distance (CD). CD measures similarity between two point clouds by quantifying how well they align with each other. The distance operates on a simple principle: for every point in one cloud, it finds the closest point in the other cloud and measures their separation. With  $n \in \{1, 2\}$ :

$$\mathcal{L}_n^g = \frac{1}{|P^{pd}|} \sum_{a \in P^{pd}} \min_{b \in P^{gt}} \|a - b\|_n^n + \frac{1}{|P^{gt}|} \sum_{b \in P^{gt}} \min_{a \in P^{pd}} \|a - b\|_n^n \quad (3.14)$$

where  $|P^{pd}|$  is the cardinality of the set  $P$  and  $\|a - b\|_n^n$  is the  $L_n$  distance between  $a$  and  $b$ . The total generation loss is  $\mathcal{L}^g = \mathcal{L}_1^g + \mathcal{L}_2^g$ .

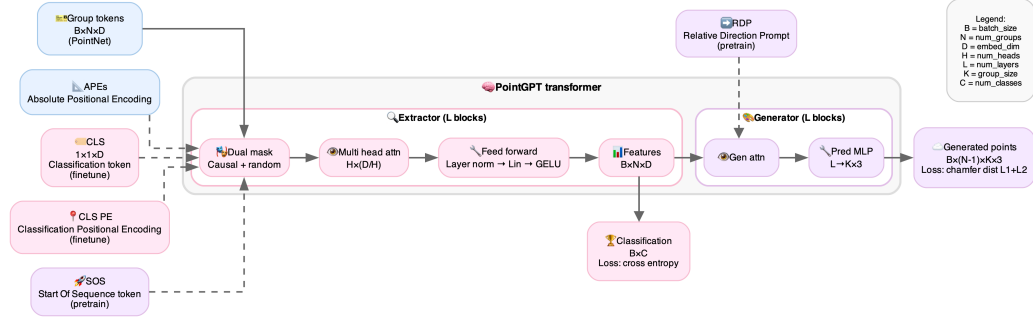


Figure 3.4: Flow of the input tokens in the PointGPT transformer.

To address the third fundamental challenge outlined in the beginning of this section, PointGPT incorporates a post-pre-training stage using labeled datasets. This intermediate fine-tuning strategy enables the model to incorporate semantic information before task-specific fine-tuning, facilitating the training of high-capacity models while mitigating overfitting concerns.

During fine-tuning, the generation task can be included as an auxiliary objective with coefficient  $\lambda$ :  $\mathcal{L}^f = \mathcal{L}^d + \lambda \times \mathcal{L}^g$ , where  $\mathcal{L}^d$  represents the downstream task loss. The PointGPT authors assessed that this provides regularization benefits and improved generalization.

### 3.5.2 PointGST

PointGST (Point cloud Graph Spectral Tuning) [74] is a PEFT (Parameter-Efficient Fine-Tuning) method (section 2.6) to fine-tune pre-trained models to new point cloud tasks. Existing PEFT (Parameter-Efficient Fine-Tuning) methods such as IDPT [84] and DAPT [85] struggle with so-called "token confusion" within frozen models, and PointGST sets out to address this. Token confusion occurs when pre-trained models produce similar internal representations for points that should be distinguished in the downstream task. Since pre-trained models learn general features without knowledge of specific task requirements, their output tokens can fail to capture the fine-grained distinctions necessary for good task performance. For instance, tokens representing different tree species may appear too similar in the feature space, making classification difficult. Existing spatial domain methods

attempt to resolve this confusion by adding learnable parameters that modify these confused tokens directly.

PointGST addresses this limitation by shifting the adaptation process from spatial coordinates to the spectral domain. The framework integrates lightweight Point Cloud Spectral Adapters (PCSA) into each transformer layer while keeping the entire pre-trained backbone frozen. The spectral domain transformation decomposes point cloud features into orthogonal frequency components, where each component captures a specific pattern of variation across the point cloud structure. This orthogonality provides a mathematical guarantee that different frequency components remain independent, creating natural separation channels for distinguishing confused tokens. Rather than attempting to separate similar spatial features directly, the method can selectively enhance or suppress specific frequency patterns that correspond to task-relevant geometric characteristics.

The method operates through three sequential steps. PointGST constructs point graphs  $\mathcal{G} = \{V, E, W\}$  where vertices  $V$  represent the point centers from FPS, edges  $E$  encode relationships, and the adjacency matrix  $W \in \mathbb{R}^{n \times n}$  weights the relationships. The adjacency matrix elements are computed using:

$$w_{i,j} = \frac{1}{\delta_{i,j} / \min(\Delta) + I_{i,j}} \quad (3.15)$$

where  $\delta_{i,j}$  represents the Euclidean distance between points  $i$  and  $j$ ,  $\min(\Delta)$  denotes the minimum non-zero distance, and  $I$  is the identity matrix. This way, nearby points receive stronger connections. The method constructs both global graphs capturing overall structure and local graphs capturing small-scale patterns.

Second, these graphs undergo spectral transformation via Graph Fourier Transform (GFT). The graph Laplacian matrix  $L = D - W$ , where  $D$  represents a diagonal matrix with each element  $d_{i,i} = \sum_{j=0}^{n-1} w_{i,j}$  representing the degree of vertex  $i$  (i.e. the cumulative strength of all connections), undergoes eigenvalue decomposition:

$$L = U \Lambda U^T \quad (3.16)$$

The eigenvectors in  $U$  form an orthogonal basis representing different variation patterns across the graph structure. Low-frequency eigenvectors correspond to smooth, gradual changes across the point cloud, while high-frequency eigenvectors correspond to sharp, localized variations.

Third, the Point Cloud Spectral Adapter (PCSA) performs fine-tuning entirely within this spectral domain. Input tokens  $T_{in}$  undergo down-projection via  $T_s = T_{in} W_d^T$ , then get transformed to the spectral domain using the pre-computed spectral basis:  $T^f = U^T T_s$ . A shared linear layer adapts these frequency coefficients:

$$T^{f'} = T^f + \text{act}(\text{Linear}(T^f)) \quad (3.17)$$

This adaptation process selectively adjusts the magnitude of different frequency components, controlling which geometric patterns receive emphasis. Low-frequency components correspond to smooth, global variations while high-frequency components represent sharp, local details. The inverse transformation  $\hat{T}^f = UT^{f'}$  returns adapted features to the spatial domain and then features get up-projected to the original shape.

The spectral approach provides better performance across multiple benchmarks while requiring significantly fewer (about 1%) trainable parameters than fully fine-tuned counterparts. The spectral basis computation is also efficient because it occurs only once per dataset, being shared across all transformer layers.

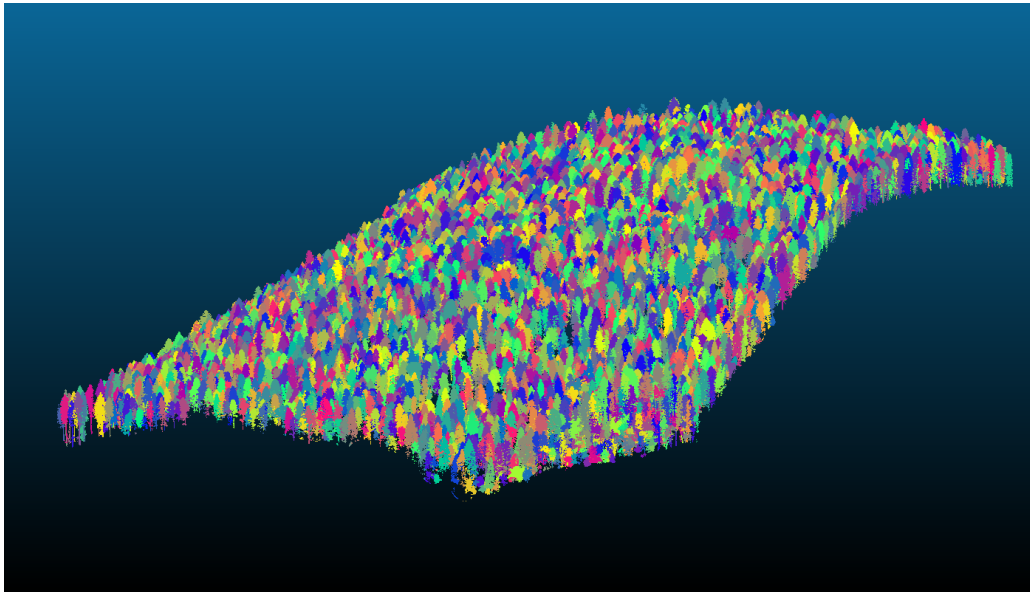
## 3.6 Data

### 3.6.1 Custom datasets

#### Pre-training dataset

The unlabeled pre-training dataset comprises two components: segmented individual trees extracted from public ALS point clouds from Canton Neuchâtel using the SegmentAnyTree architecture [42] (fig. 3.5), and synthetic tree data generated through the SimpleSynthTree framework [75]. Since pre-training aims to enable the model to learn fundamental tree structures and patterns without requiring species labels, forest areas in Neuchâtel were selected without regard to species composition, given that comprehensive inventory data is unavailable. The dataset can be assumed to roughly follow the mean Swiss species distribution (table 2.1), with this assumption being most reliable for the five most abundant species, which are commonly found at the latitudes of Canton Neuchâtel. Similarly, the synthetic trees lack species-specific attributes.

The public ALS point clouds from Canton Neuchâtel provide coverage of the entire cantonal surface in 1km<sup>2</sup> tiles and can be viewed at [86]. The SITN (Neuchâtel Territorial Information System) conducted the LiDAR survey in spring 2022. These point clouds achieve an average density exceeding 100 pts/m<sup>2</sup> across the canton and were acquired using a Riegl VQ-Q1560II sensor operating at 4 MHz across 81 flight lines, with a mean flight height of around 1000 m. The survey attained altimetric (vertical) precision below 5 cm and planimetric precision below 10 cm, representing the upper quality threshold achievable with contemporary ALS systems (table 2.2). The point classifications include ground, low vegetation, medium vegetation, high vegetation, building, and road categories.



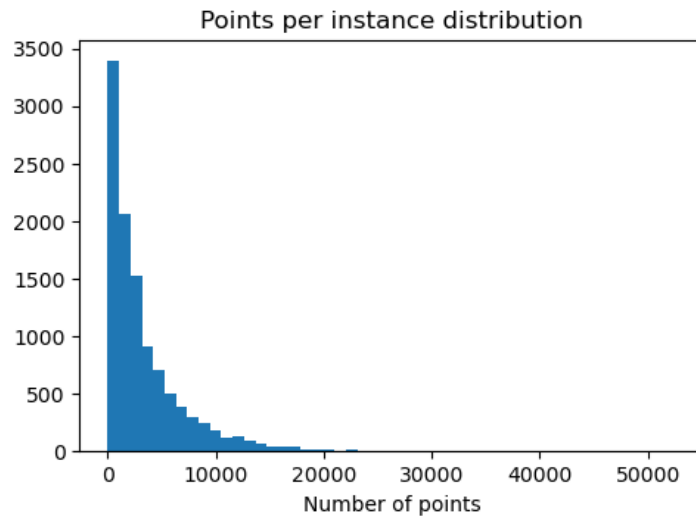
**Figure 3.5:** An example of a segmented ALS point cloud from Canton Neuchâtel used for the pre-training dataset.

Thirteen  $1\text{km}^2$  tiles from selected forested areas were processed using SegmentAnyTree [42], a platform-agnostic deep learning model designed for individual tree instance segmentation across varying point densities. The model employs a 3D convolutional network based on the PointGroup architecture [87] with a U-Net backbone for feature extraction. Input point clouds undergo voxelization using the Minkowski Engine [88] to enable 3D convolutions. The architecture incorporates three parallel prediction heads that operate on extracted features: semantic segmentation for tree versus non-tree classification, 3D offset prediction that generates vectors directing each point toward its respective tree center to facilitate spatial clustering, and 5-dimensional embedding that maps points into a learned feature space where intra-tree points cluster together while inter-tree points remain separated. Point clustering utilizes region-growing [89] and mean-shift [90] methods applied to these embeddings, with final refinement through ScoreNet [91]. ScoreNet scores and filters tree candidates based on their intersection-over-union with ground truth data, followed by non-maximum suppression to eliminate redundant detections.

The point cloud tiles were processed directly without any point classification filtering. Only resulting instances with more than 1024 points were retained, preserving approximately 60% of all instances (table 3.1). As a representative example, tile 2565500\_1212500 exhibited the following distribution: mean of 3405 points per instance, median of 2096 points, minimum of 10 points, and maximum of 52554 points. Tree instances in every tile demonstrated significant skewness toward lower point counts (fig. 3.6).

Tile ID	Total instances	Filtered instances ( $\geq$ 1024 pts)	Ratio
2536000_1194000	9600	5439	57%
2536500_1193500	11574	7697	67%
2536500_1194000	11754	6996	60%
2537000_1194000	11354	7841	69%
2537500_1194000	10687	7011	66%
2537000_1194500	10370	5965	58%
2564000_1217000	6761	3698	55%
2564000_1216500	8827	3639	41%
2564500_1212500	980	552	59%
2565000_1212000	12740	8760	69%
2565000_1212500	11994	7824	65%
2565000_1213000	7565	4657	62%
2565500_1212500	10886	7577	70%
<b>Total</b>	<b>134092</b>	<b>77656</b>	<b>58%</b>

**Table 3.1:** Number of tree instances across point cloud tiles with filtering threshold of 1024 points per instance.



**Figure 3.6:** Distribution of points per instance for tile 25655001212500.

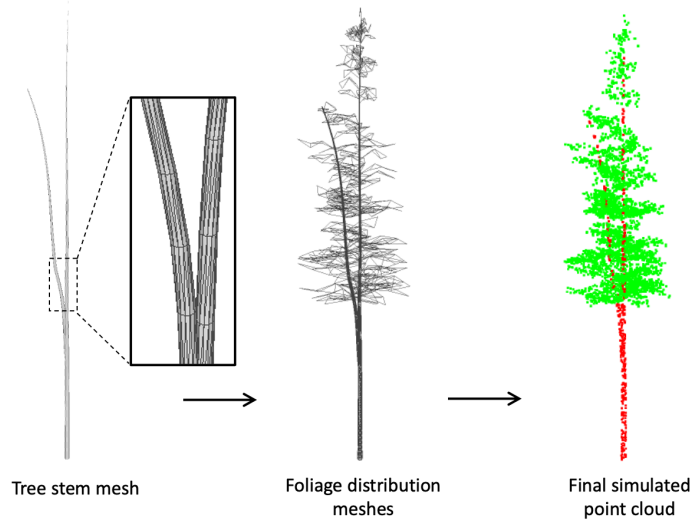
This ALS dataset was complemented with an equal number (77,656, table 3.1) of synthetic point clouds generated with SimpleSynthTree [75]. The framework operates through a two-stage process (fig. 3.7). First, tree structures are modeled with meshes as curved and tapered cylinders using cubic spline interpolation between control points at the tree base, mid-height, and crown. Stem diameter follows a linear taper from a base diameter to zero at the apex. The framework incorpo-



rates stem splitting through additional cylinders branching from the main stem, and simulates foliage distribution through height-dependent branch placement with realistic canopy shapes. Second, the mesh structures undergo point-based sampling to generate final synthetic point clouds. The parameters employed for the generation process are resumed in table 3.2.

Parameter	Values used	Parameter	Values used
Number of points	4096	Maximum canopy width	10.0 m
Tree height ( $h_t$ ) range	20–50 m	Max canopy width height	$0.4\text{--}0.8 h_t$
Base diameter range	0.3–1.0 m	Minimum canopy height	$0.2\text{--}0.5 h_t$
Stem split height range	$0.15\text{--}0.9 h_t$	Tree top horizontal offset	$\pm 2.5$ m
Stem split probability	50%	Tree mid horizontal offset	$\pm 0.5$ m
Number of branches	60–150	Foliage vertical noise	0.5 m

**Table 3.2:** SimpleSynthTree parameters used for synthetic tree generation complementing the Neuchâtel dataset.



**Figure 3.7:** The tree synthesis process in SimpleSynthTree. Figure taken from [75].

### Post-pre-training dataset

The post-pre-training stage addresses the gap between generation and the downstream classification task (section 3.5.1). For this, a labeled dataset was created using the SynForest framework [76]. SynForest generates synthetic forest data by combining ForestFactory [92] for forest stand simulation, PyTreeDB [93] for tree modeling, and Helios++ [80] for LiDAR sensor simulation. Unlike SimpleSynthTree’s individual tree focus, SynForest operates at forest scale, creating complete stands with inter-tree relationships and spatial distribution.

The workflow comprises three stages: map, scene, and simulation (fig. 3.8). In the map stage, ForestFactory generates forest configurations using climate records to simulate tree growth over multiple years, producing tabular data with positional coordinates, height, diameter, and species information for each tree. The scene stage places tree models from PyTreeDB at corresponding positions, selecting models that match the specified species, height, and diameter criteria. Trees in PyTreeDB are represented in GeoJSON files containing tree location, species, properties, and links to one or more point clouds from ALS, ULS, or TLS data. In the simulation stage, Helios++ simulates configurable LiDAR sensors and platforms to generate synthetic point clouds by modeling scanning platform trajectories through the forest environment. Each generated point contains tree instance and species labels, among other attributes.

To create the dataset, a single forest configuration from ForestFactory was used, since inter-tree dynamics are not relevant for this study and individual trees are subsequently isolated during processing. To maximize utilization of tree models in PyTreeDB and ensure diverse representation in the final dataset, a custom algorithm (listing 3.1) distributed available tree models across six identical forest configurations, each containing eight species positions with predetermined tree counts per species for a total of 532 trees. The assignment algorithm iterates through species in reverse order of availability, assigning models to species positions that minimize the difference between required tree count and available model count. While this approach promotes diverse model utilization across both TLS and ULS sensor simulations, the final model selection remains subject to height and diameter matching criteria during the scene stage, meaning not all available models may ultimately be used in the dataset.

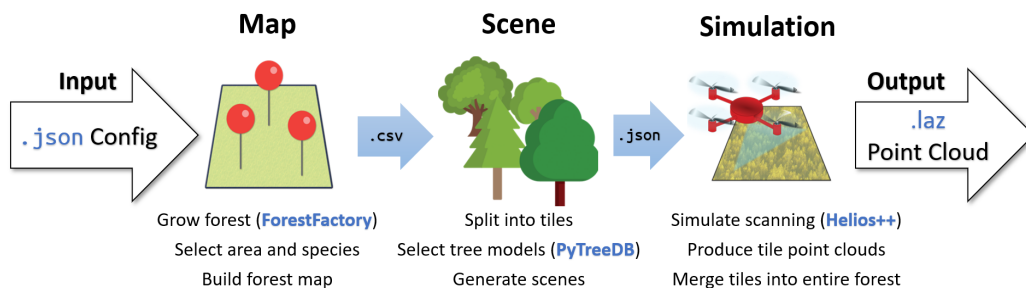


Figure 3.8: The Synforest simulation process. Figure taken from [76].

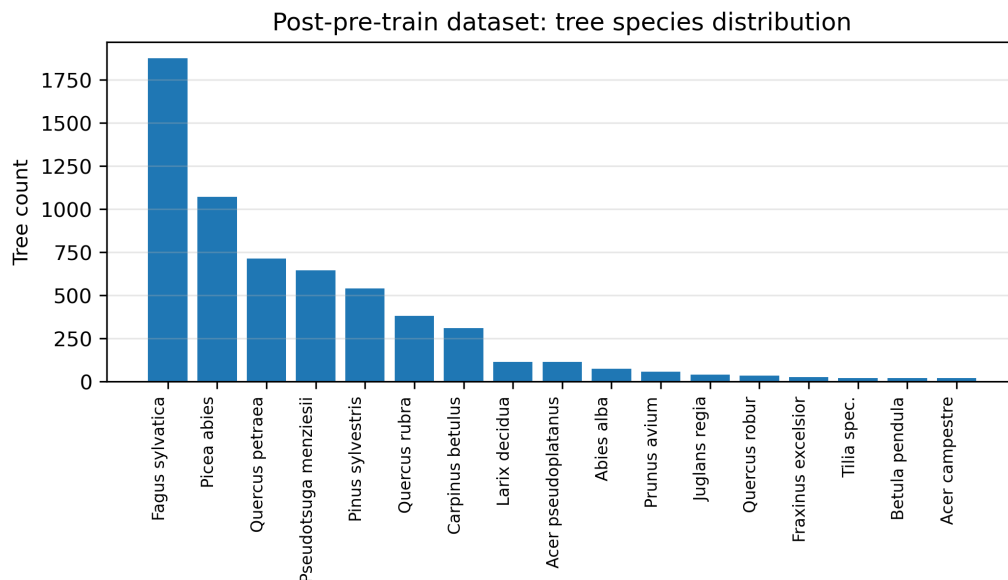
```

1 # goal: use all 3,181 unique tree models from pytreedb without repetition
2 species_models = {'fagus_sylvatica': 930, 'picea_abies': 556, ...} # available models
3 tree_counts = [76, 136, 121, 57, 39, 57, 20, 26] # trees needed per position
4 assignments = empty_grid(6_forests, 8_positions)
5
6 # start with rarest species to ensure they get placed
7 for species in order_by_rarity(species_models):
8     models_left = species_models[species]
9
10     while models_left > 0:
11         # find tree count that best matches remaining models
12         best_fit = find_closest_match(models_left, tree_counts)
13
14         # skip if we would waste too many models (e.g., 7 models for 76-tree position)
15         if models_left < tree_counts[best_fit] and already_placed_somewhere:
16             break
17
18         place_species(species, best_fit)
19         models_left -= tree_counts[best_fit]
20
21 # result: 3,081 models assigned, 6,162 trees total ( 2 sensors)

```

**Listing 3.1:** Tree species assignment algorithm in Python pseudocode for the post-pre-train dataset.

The resulting dataset has a diverse but uneven distribution of 17 tree species (fig. 3.9), reflecting both the availability of models in PyTreeDB and the assignment algorithm’s optimization strategy. *Fagus sylvatica* represents the most abundant species with nearly 1’900 trees, followed by *Picea abies* with approximately 1’100 trees. The remaining species show progressively lower counts, with some species like *Quercus robur* and *Betula pendula* represented by fewer than 100 trees each. This distribution pattern and these species frequencies are reasonably similar to what could be expected in a natural setting (fig. 2.1).



**Figure 3.9:** Tree species distribution in the post-pre-train dataset.

Of the 6,161 generated trees, 106 were filtered out for having fewer than 1024 points, resulting in a final dataset of 6,056 trees. These tree instances exhibit relatively high point densities, attributable to the characteristics of the simulated platforms (fig. 3.10).

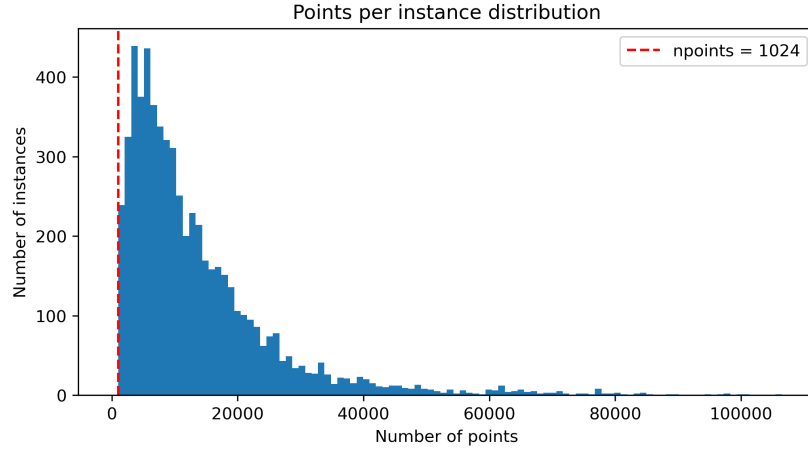


Figure 3.10: Distribution of points per instance for the post-pre-train dataset.

### 3.6.2 Literature dataset

The literature dataset expanded the custom pre-training dataset (section 3.6.1) with diverse unlabeled data from predominantly ground-based sensor platforms (table 3.3). Not all datasets contained the complete data described in their respective publications at the time of this work. For example, the Boreal3D dataset was available only in an incomplete version during data collection. For ForestSemantic, only three out of six forest plots were available.

Dataset	Platform	Type	Raw trees	Trees $\geq$ 1024 pts
Boreal3D [94]	ALS, ULS, TLS, MLS	Synth.	18,976	13,257
ForestSemantic [95]	TLS	Real	308	308
FOR-Instance [96]	ULS	Real	1,130	997
LAUTx [97]	TLS, PLS	Real	1,030	1,030
TreeLearn [43]	MLS	Real	6,884	6,860
Wytham Woods [98]	TLS	Real	876	876
<b>Total</b>			<b>29,204</b>	<b>23,328</b>

Table 3.3: Overview of the literature datasets used to expand the pre-training dataset.

A numerical precision issue emerged during preprocessing of the FOR-Instance

dataset. Converting coordinates from float64 to float32 precision for CUDA-compatible FPS sampling caused large UTM coordinate values to be truncated, resulting in duplicate points during normalization. This truncation distorted point clouds and caused training instabilities. The solution was to perform normalization in float64 precision before converting to float32 for FPS sampling.

### 3.6.3 Benchmark dataset

FOR-Species20K [77] was selected as the benchmark dataset for fine-tuning and evaluation due to its explicit focus on tree species classification tasks. The dataset comprises 20,158 individual tree point clouds representing 33 species (fig. 3.11), making it the largest available point cloud dataset for tree species classification from proximal laser scanning. The data distribution by platform shows TLS constituting 70% of the trees (acquired using 12 different TLS sensors), ULS contributing 22% (1 sensor type), and MLS representing 8% (1 sensor type). Geographically, 90% of the data originates from European forests, covering three main ecoregions: temperate forests (61%), boreal forests (25%), and Mediterranean forests (7%). The remaining data includes temperate and boreal plantation forests from other continents (4%) and tropical savannas (3%).

The species distribution reflects common patterns observed in European forest ecosystems and Switzerland (fig. 2.1). Dominant species such as *Pinus sylvestris* (3'296 individuals), *Fagus sylvatica* (2'482 individuals), and *Picea abies* (1'983 individuals) are well-represented. In contrast, rarer species like *Prunus avium* are represented by only 50 individuals. This imbalance presents challenges for model training but provides a realistic benchmark for evaluating classification performance under real-world conditions where species distributions are naturally uneven. Tree heights range from 0.3 to 56.3 meters and exhibit substantial intra-specific variation, reflecting different developmental stages and growth conditions. Coniferous species show a mean height of 20.4 m (standard deviation = 8.2 m) compared to broadleaved species with a mean of 11.4 m (standard deviation = 8.7 m). Dominant European species cover the full spectrum from young to mature trees, providing good representation across developmental stages.

For benchmarking purposes, the dataset is divided by its authors into development (90%, 17,707 trees) and test sets (10%, 2,254 trees) using stratified random sampling. The stratification involved three levels of balancing. First, species representation was capped at 100 trees per species to prevent dominant species from overwhelming the test set. Second, tree heights were divided into 20 equal bins of 2.8-meter width to ensure representation across developmental stages. Third, each height bin was subdivided by platform type (TLS, MLS, ULS), creating combined strata that represent specific height-platform combinations.

For each species, the number of available height-platform strata was counted as  $j$ , and the target number of trees per stratum was calculated as:

$$n_{\text{stratum}} = \frac{100}{j} \quad (3.18)$$

The actual sampling from each stratum followed the constraint:

$$\text{sampled} = \min(n_{\text{stratum}}, \text{available trees in stratum}) \quad (3.19)$$

This stratified approach produced a test dataset with balanced representation across species, tree sizes, and acquisition platforms, enabling evaluation of model performance under diverse conditions.

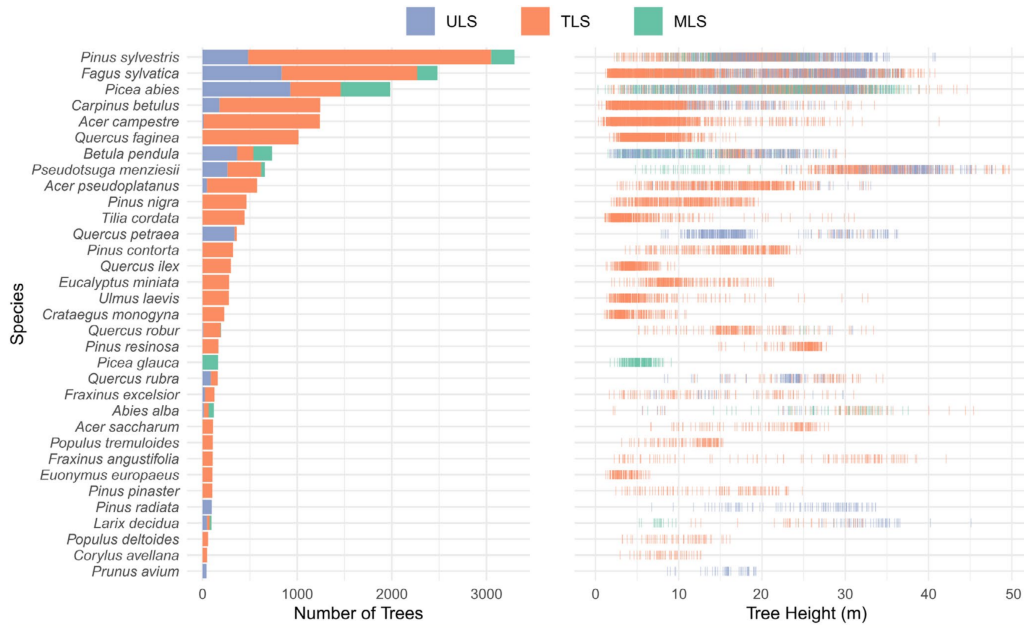


Figure 3.11: Summary chart of the FOR-Species20K dataset. Figure taken from [77].

The distribution of point counts across tree instances has a very broad range (fig. 3.12), with the largest TLS trees containing more than 10 million points and the mean tree having 316'165 points. Trees with fewer than 1'024 points were excluded from the dataset, resulting in 254 filtered instances and a final training split of 17'453 trees.

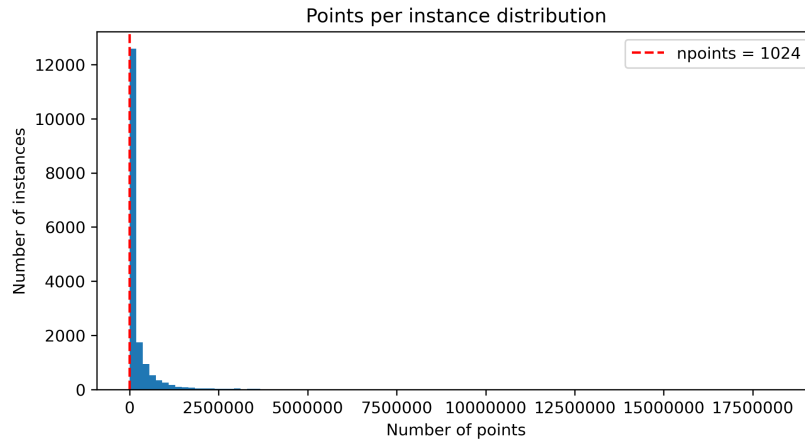


Figure 3.12: Distribution of points per instance for the benchmark dataset.

On the published benchmark evaluation, image-based methods (section 2.4) outperformed point cloud-based approaches across all evaluation metrics. The top three models employed multi-view 2D projection strategies, while the best point cloud method achieved 4th place and performance within 4% overall accuracy of the leading approach.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
DetailView	79.5	82.3	76.7	78.0
YOLOv5	77.9	84.2	75.0	77.3
SimpleView	76.2	76.9	75.5	75.6
Ensemble PointNet++	75.6	78.2	73.5	74.9

Table 3.4: Performance of top-performing models on the FOR-species20K benchmark dataset.

**DetailView** [99] builds upon SimpleView [100] and implements dataset balancing via weighted random sampling across species, tree size, and platform types. The method uses a DenseNet-201 backbone to process seven 2D projected depth images (256×256 pixels), incorporating top and bottom views alongside a trunk projection to capture bark structural features. The model applies augmentation strategies on both point clouds (random subsampling and rotations) and image transformations (flipping), with final predictions derived from averaging 50 different predictions.

**YOLOv5** implements a modified architecture based on the Ultralytics framework [101] that processes four orthographic side-view projections (600×800 pixels) colored by point density. The approach generates final species predictions through weighted averaging of class probabilities across multiple views using a hierarchical weighting scheme that assigns decreasing importance to lower-ranked

predictions.

**SimpleView** [100] is the foundational multi-view approach and processes six orthogonal camera projections through a ResNet-18 backbone. The method generates 512×512 pixel depth-colored images from down-sampled point clouds (16'384 points) and employs balanced accuracy rather than overall accuracy for model selection to address data imbalance.

**Ensemble PointNet++** is the leading point cloud methodology, utilizing PointNet++ [64] to extract features directly from 3D point clouds through three sets of sub-sampling and grouping operations on filtered instances with a minimum of 8'192 points. The approach incorporates rotational augmentation (6-fold around the z-axis) and random point sampling, and performance is enhanced by ensemble prediction of 10 classifiers.

All top-performing models demonstrated consistent performance above 70% accuracy across TLS, MLS, and ULS platforms, with DetailView exhibiting particular strength on MLS and ULS data while YOLOv5 marginally outperformed other methods on TLS acquisitions.

The authors of DetailView published the train and validation loss data on the model's GitHub repository (fig. 3.13). Given that this is the leading modelling approach on the benchmark, this data allows for direct comparison with the approach of this work.

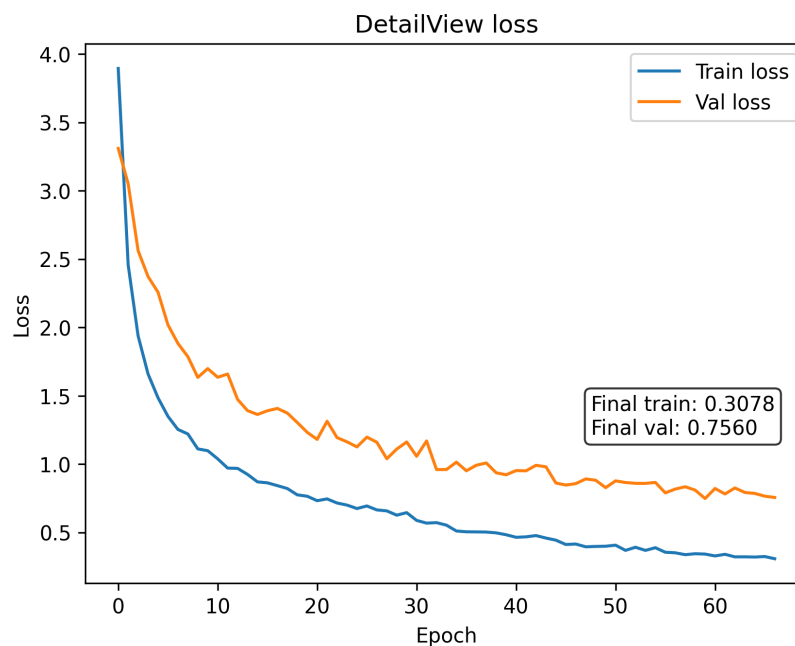


Figure 3.13: Loss curves of DetailView on FOR-SPEcies20K.



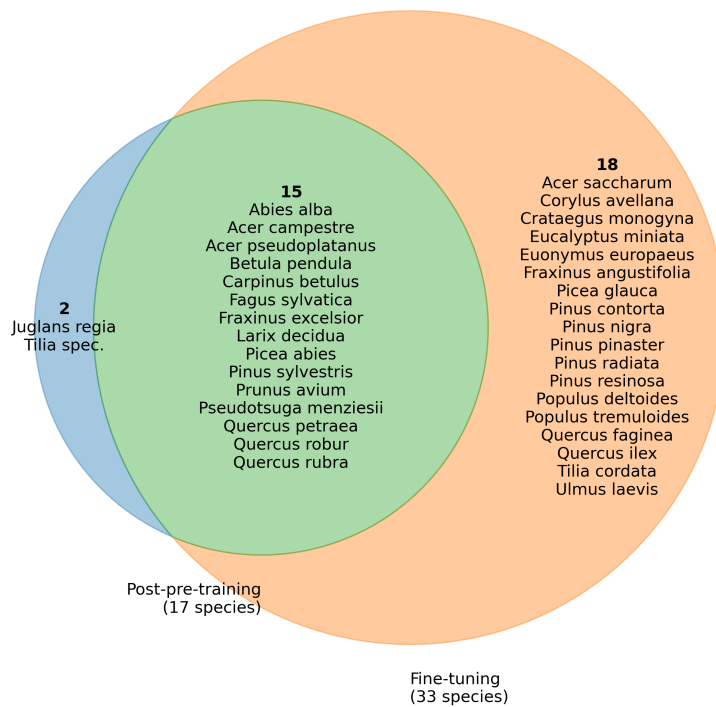
### 3.6.4 Dataset summary

A summary of the datasets per training stage is provided in table 3.5. For the labeled post-pre-training and fine-tuning datasets, the tree species composition and overlap are provided in fig. 3.14. In total, 15 species overlap between the datasets, representing 45.5% of species in the fine-tuning dataset that are already present in the post-pre-training stage.

Dataset	Platform	Type	Trees	Species
Pre-training	ALS, ULS, TLS, MLS, synth.	Mixed	178'640	N/A
Post-pre-training	TLS, ULS (simulated)	Synth.	6'056	17
Fine-tuning	TLS, ULS, MLS	Real	17'453	33
<b>Total</b>			<b>202'149</b>	

**Table 3.5:** Overview of datasets used in this study.

Species overlap between post-pre-training and fine-tuning datasets



**Figure 3.14:** Species overlap between post-pre-training and fine-tuning datasets.



## 4 Results and discussion

### 4.1 Pipelines implementation

#### 4.1.1 Preprocessing

The preprocessing pipeline inherits from PyTorch's Dataset class and operates through sequential stages. Its goal is to transform heterogeneous point clouds of varying sizes into normalized point clouds with uniform point counts through FPS, enabling more efficient and faster processing during training. These point clouds can be further downsampled during training if needed. A code overview is given in listing 4.1. Given the use of GPU operations, batch processing logic is implemented to prevent GPU overload and Out Of Memory (OOM) errors.

The initial stage loads tree instance point cloud data from various file structures and formats, depending on the dataset, and sorts instances by point count. Since point clouds can exhibit vastly different point counts (see fig. 3.12), sorting ensures efficient padding for vectorized GPU operations when creating and processing batches. Without sorting, the largest point cloud in any batch determines the padding size for all other instances in that batch. This approach leads to memory inefficiency, as a point cloud containing millions of points forces all other point clouds in the same batch to be padded to the same length, filling memory with zeros. By sorting data by point count first, padding within batches is applied to similarly sized point clouds, achieving greater efficiency.

A `filter_small_clouds` parameter removes point clouds containing fewer points than the target sampling number `npoints`. No upsampling strategy is implemented because both FPS on fewer than `npoints` points and random sampling with replacement would create point clouds with duplicate points. This duplication is problematic when it propagates through the training pipeline for two reasons:

- ▶ When including the generative task (section 3.5.1), RDPs based on centers are calculated for next group prediction. If two centers are identical, the relative direction calculation becomes undefined, and these NaN values propagate, destabilizing training.
- ▶ When finetuning with spectral adapters (section 3.5.2), Laplacian matrices become rank-deficient, causing eigendecomposition to crash when calculating the spectral basis.

```

1 def preprocess_dataset(file_data, npoints, device):
2     # sort files by point count to make padding more efficient
3     file_data.sort(key=lambda x: x.point_count)
4
5     batch_size = initial_batch_size
6     processed_files = 0
7     all_point_clouds = []
8
9     while processed_files < len(file_data):
10         try:
11             # load batch
12             batch_files = file_data[processed_files:processed_files + batch_size]
13             point_clouds = load_point_clouds(batch_files)
14
15             # filter small clouds:
16             point_clouds = [pc for pc in point_clouds if len(pc) >= npoints]
17
18             # pad tensors for batch processing
19             batch_tensor, mask = create_padded_batch(point_clouds, device)
20
21             # normalization
22             centroids = compute_masked_centroids(batch_tensor, mask)
23             centered_batch = batch_tensor - centroids.unsqueeze(1)
24             max_distances = compute_max_distances(centered_batch, mask)
25             normalized_batch = centered_batch / (max_distances + eps)
26
27             # FPS
28             if device.type == 'cuda':
29                 sampled_batch = sample_farthest_points_cuda(
30                     normalized_batch, npoints)
31             else:
32                 sampled_batch = sample_farthest_points_cpu(
33                     normalized_batch, npoints)
34
35             all_point_clouds.append(sampled_batch.cpu())
36             processed_files += batch_size
37
38             # halve batch size if getting OOM
39             except RuntimeError as e:
40                 print(f'CUDA out of memory: {str(e)}')
41                 batch_size = batch_size // 2
42
43         final_dataset = torch.cat(all_point_clouds, dim=0)
44         return final_dataset

```

**Listing 4.1:** Preprocessing pipeline for point cloud normalization and sampling in Python pseudocode.

The data is then normalized. For each point cloud  $P = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n\}$  where  $\mathbf{p}_i \in \mathbb{R}^3$ , the centroid is computed using validity masks to account for variable-length point clouds within padded batches:

$$\mathbf{c} = \frac{\sum_{i=1}^N \mathbf{p}_i \cdot m_i}{\sum_{i=1}^N m_i} \quad (4.1)$$

where  $\mathbf{c}$  represents the centroid,  $\mathbf{p}_i$  denotes individual points,  $m_i$  indicates the validity mask, and  $N$  represents the padded sequence length. Each point is then

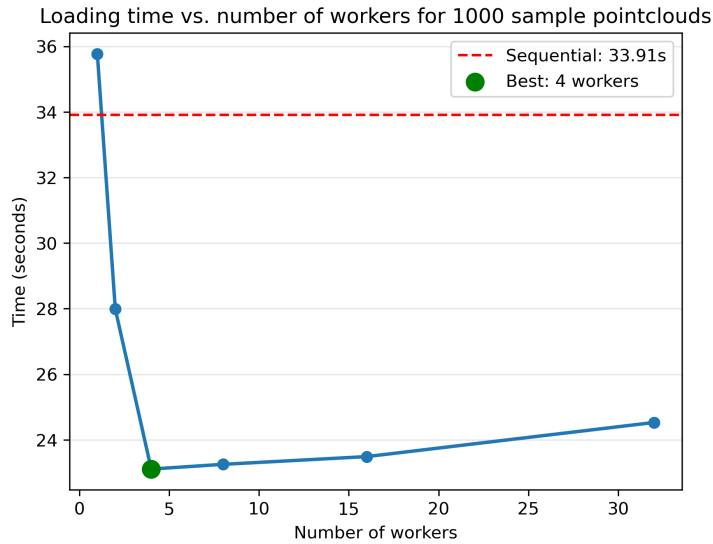
centered by subtracting the centroid:  $P' = \{\mathbf{p}_1 - \mathbf{c}, \mathbf{p}_2 - \mathbf{c}, \dots, \mathbf{p}_n - \mathbf{c}\}$ . Finally, scaling is applied:

$$\mathbf{p}'_i = \frac{\mathbf{p}_i - \mathbf{c}}{d_{\max} + \epsilon} \quad (4.2)$$

where  $d_{\max} = \max_i \|\mathbf{p}_i - \mathbf{c}\|_2$  and  $\epsilon = 10^{-8}$  provides numerical stability when points lie very close to the centroid.

FPS (eq. (3.3)) is applied to reach `npoints`. The preprocessing pipeline begins with an initial batch size of 1'000 point clouds and automatically halves this size whenever an OOM error occurs, ensuring continuous processing. Task-specific point cloud metadata is processed separately. Preprocessed point cloud datasets are stored in PyTorch tensor format (.pt files) to ensure efficient loading and eliminate the need for re-processing during subsequent experiments.

To accelerate file loading for datasets containing numerous individual point cloud files, a parallel loading mechanism is implemented using Python's `concurrent.futures` module with `ThreadPoolExecutor`. This approach maximizes I/O efficiency by minimizing idle times during file operations while avoiding the overhead of spawning multiple processes, unlike the multiprocessing module approach for circumventing Python's Global Interpreter Lock (GIL). Optimal performance is achieved using 4 CPU workers, providing approximately 1.5× speedup compared to sequential loading (fig. 4.1). Preprocessing the FOR-Species20K training split with `npoints` = 8'192 reduces dataset storage from 24 GB to 2.8 GB and requires approximately 70 minutes on an A100 GPU.



**Figure 4.1:** Speedup investigation for parallel vs. sequential loading applied on .laz point cloud files.

### 4.1.2 Training

#### Overview

For each of the training stages (pre-training, post-pre-training, fine-tuning), a separate training loop was developed. Comprehensive experiment configuration is handled through YAML configuration files that are converted into EasyDict dictionaries and passed as arguments to various builder functions throughout the pipeline. Additional command-line arguments are parsed via `argparse` and define higher-level run configurations. All training loops support distributed training via the `torch.distributed` package, leveraging the tensor gathering utilities already present in the PointGPT and PointGST codebases.

To accelerate training, Automatic Mixed Precision (AMP) was implemented through the `torch.amp` package, which automatically scales operations between float32 and float16 precision to reduce computational load. Performance evaluation confirmed that AMP implementation did not degrade model accuracy. The speedup achieved was highly dependent on run configuration parameters, including model size and batch size, as well as whether distributed training was employed, with speedup factors ranging from 1.3× to 2.3× (table 4.1).

Metric	Full precision	Mixed precision
Batch time	3.2 s	1.2 s
Batch speedup	1.0×	<b>2.72×</b>
Epoch 0 time	566 s	231 s
Epoch 1 time	511 s	222 s
Epoch 2 time	511 s	222 s
Epoch speedup	1.0×	<b>2.30×</b>

**Table 4.1:** Training speed comparison between full precision and mixed precision for distributed PointGPT pre-training using 4 GPUs (batch size 128 per GPU, 512 total).

`torch.compile` was additionally implemented to accelerate training through Just-In-Time (JIT) compilation, which compiles PyTorch code into optimized kernels. The performance improvement from this method proved less substantial than AMP, providing speedup factors of approximately 1.10×. However, `torch.compile` could not be utilized in runs involving the generative task due to incompatibility with third-party C++ extensions, particularly the optimized CUDA kernels required for Chamfer Distance calculations.

Monitoring and logging capabilities were expanded and migrated to Weights and Biases for comprehensive experiment tracking. Stage-specific metrics were implemented according to training requirements. Across all training loops, gradient norm, parameter norm, gradient-to-parameter ratio, and AMP scaling factor logging was implemented to identify exploding or vanishing gradient issues and assess AMP scaling influence. Checkpoint management encompassed defining appropriate saving conditions and implementing robust checkpoint saving and resuming functionality.

### Models and training configurations

3 model configurations were experimented with: Small (S), Base (B), Large (L), following the Vision Transformer configurations [102].

Component	Small (S)	Base (B)	Large (L)
Encoder	494K	6.4M	7.0M
Extractor layers	<b>21.3M</b>	<b>85.1M</b>	<b>302.4M</b>
Extractor cls head	272K	469K	600K
Generator layers	7.1M	28.4M	50.5M
<b>Total parameters</b>	<b>29.2M</b>	<b>120.5M</b>	<b>360.5M</b>

**Table 4.2:** Parameter counts across model configurations.

Most experiments were conducted using the S configuration for computational efficiency. Once stable training configurations were established, the best-performing parameters were directly scaled to the L version, as it was expected to outperform the B configuration. The B configuration was evaluated only during pre-training experiments.

The AdamW optimizer was employed across all experiments with a regularizing weight decay of 0.05. Weight decay prevents overfitting by penalizing large parameter values, encouraging the model to learn simpler, more generalizable representations. Following initial experiments with constant learning rates, a cosine annealing learning rate scheduler was implemented with experiment-specific warmup periods. The warmup starting learning rate was consistently set to one-tenth of the peak learning rate, while the minimum learning rate was defined as one-hundredth of the peak value. Additional regularization was provided through drop path, a stochastic regularization technique that randomly drops entire residual transformer blocks during training, forcing the network to learn redundant pathways and improving generalization.

Early stopping was implemented selectively based on experimental requirements, typically configured to trigger after 10% of the total planned epochs when validation metrics ceased improving.

## Transformations

Five different transformations (fig. 4.2) were applied sequentially to point clouds for data augmentation, with each transformation operating on the output of the preceding operation. The transformations are executed within the training loop immediately before model inference. The augmentation strategy addresses rotational invariance, occlusion effects, scale variations, spatial symmetries, and sensor noise:

- ▶ **PointcloudRotate:** Applies random rotations around the y-axis with angles uniformly sampled from  $[0, 2\pi]$ , preserving vertical tree orientation while introducing rotational variance.
- ▶ **PointcloudRandomInputDropout:** Randomly removes a percentage of points, replacing dropped coordinates with the first point's position to simulate occlusion effects common in dense forest environments.
- ▶ **PointcloudScaleAndTranslate:** Applies random scaling factors along all spatial dimensions, combined with translation offsets.
- ▶ **RandomHorizontalFlip:** Randomly flips point clouds along x and y axes while preserving the z-axis as upright, exploiting natural symmetrical properties of tree structures.
- ▶ **PointcloudJitter:** Adds Gaussian noise to simulate LiDAR measurement uncertainty.

Pre-training and post-pre-training stages employed PointcloudRotate, RandomHorizontalFlip, and PointcloudScaleAndTranslate with scaling factors ranging between 0.8 and 1.2, and translation offsets within  $\pm 0.2$  units. This parameter range accommodates natural variability in tree size and spatial positioning while remaining within phenologically realistic bounds for given developmental stages, as the 20% variation was deemed biologically appropriate. For fine-tuning, Point-

cloudRandomInputDropout was added with a maximum dropout ratio of 20% to simulate sensor occlusions characteristic of complex forest environments. PointcloudJitter was ultimately excluded from the fine-tuning augmentation strategy as it was deemed excessively noisy for effective learning.



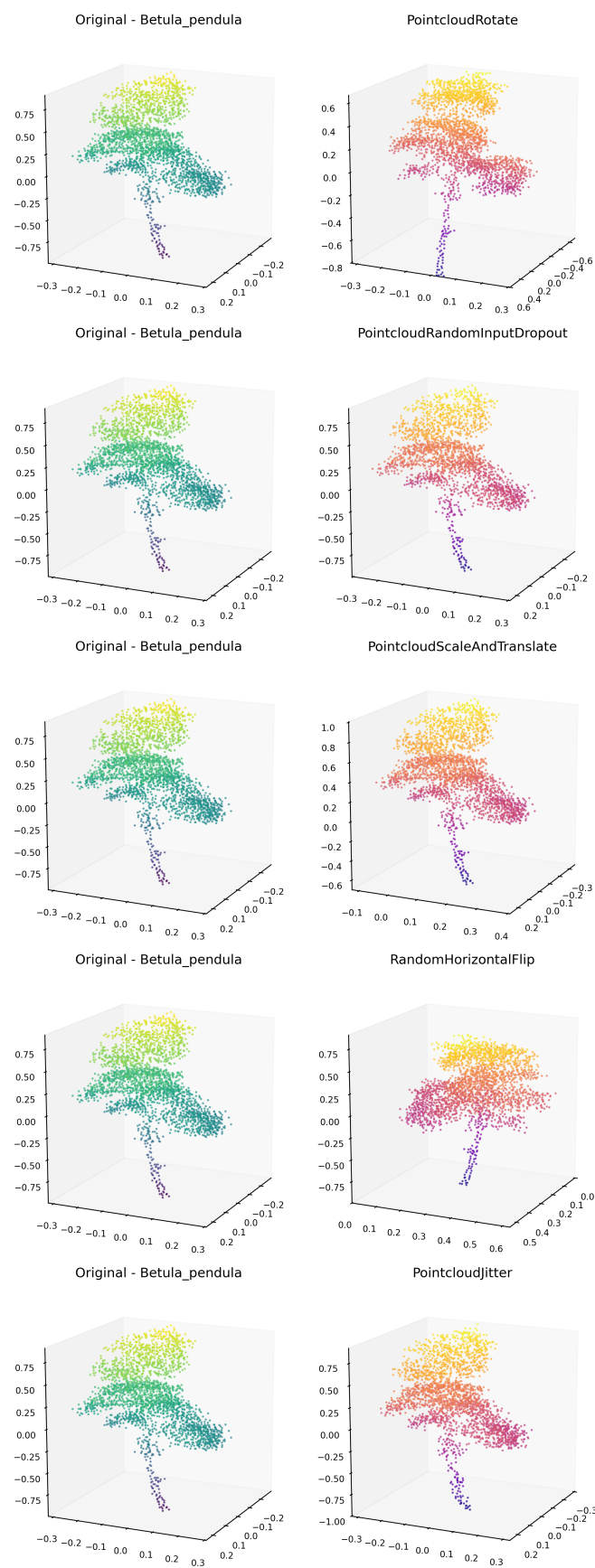


Figure 4.2: The applied point cloud transformations for augmentation purposes.

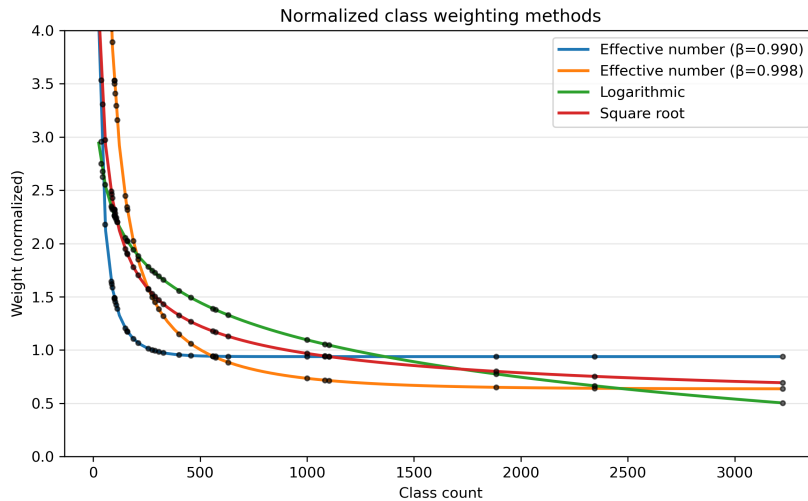
### Dataset balancing

The post-pretraining and finetuning datasets exhibit severe class imbalance, particularly the latter (fig. 3.11), where the ratio between the most common and rarest species reaches 50:1. To mitigate this, different weighting strategies for the loss function were analyzed (table 4.3 and fig. 4.3):

- ▶ Effective numbers [103] ( $\beta = 0.99, 0.998$ ) are based on diminishing information returns as sample size increases. The effective number is calculated as  $E_n = \frac{1-\beta^n}{1-\beta}$ , where  $n$  denotes the class sample count, yielding weights  $w = \frac{1-\beta}{E_n}$ . Higher  $\beta$  values create more aggressive reweighting.
- ▶ Logarithmic weighting employs  $w = 1 + \log\left(\frac{n_{\max}}{n_i}\right)$ , providing moderate reweighting that scales logarithmically with class rarity.
- ▶ Square root weighting utilizes  $w = 1 + \sqrt{\frac{n_{\max}}{n_i}}$  and applies stronger reweighting to rare classes compared to logarithmic weighting.

Weighting method	Min weight	Max weight	Weight ratio
Effective number ( $\beta = 0.990$ )	0.9383	4.0802	4.35
Effective number ( $\beta = 0.998$ )	0.6361	12.5201	<b>19.68</b>
Logarithmic	0.5061	2.9425	5.81
Square root	0.6939	4.1996	6.05

**Table 4.3:** Normalized class weight ranges for different balancing strategies applied to the FOR-Species20K dataset.



**Figure 4.3:** Comparison of class weighting strategies for the FOR-Species20K dataset. The black dots are the effective class counts of the 33 classes in the dataset.

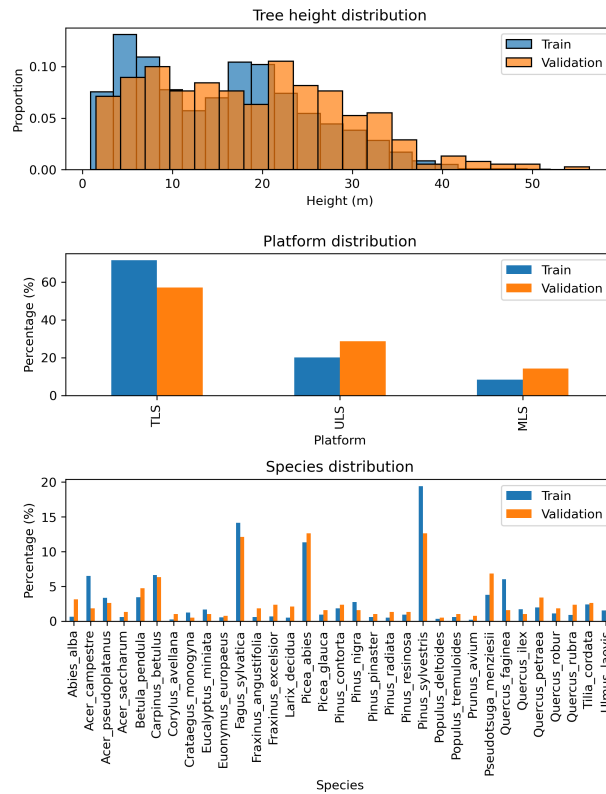
The effective number approach with  $\beta = 0.998$  was selected. This method reduces the effective imbalance ratio to approximately 2.5:1.

## Dataset splits

Dataset splits were applied to the tree counts detailed in table 3.5. The pre-training dataset used a random split of 99% training and 1% validation data to monitor potential overfitting. The post-pre-training dataset employed a similar random split with 95% allocated to training and 5% to validation.

The splitting methodology for FOR-Species20K was more sophisticated, adopting the approach from DetailView [99] to enable direct performance comparison on the validation split. This stratified sampling approach ensures balanced representation across three dimensions in the validation set: species, sensor platform type, and tree height.

The split generation process begins by encoding categorical variables (species and sensor platform type) using label encoders and standardizing all three features to comparable ranges. Starting from a randomly selected tree, the method employs FPS to iteratively select additional trees that maximize distance from previously chosen samples within this 3D parameter space. The sampling process executes 800 iterations, generating a candidate pool twice the size of the target validation set. From this diverse candidate pool, 400 trees are randomly selected to constitute the final validation set. This methodology enables evaluation of model generalization across species diversity, sensor platform types, and tree sizes.



**Figure 4.4:** Comparison of the training and validation splits for the fine-tuning dataset, with values expressed as percentages of the dataset.

### Linear probing

To evaluate the quality of representations learned by self-supervised learning models, Linear Probing (LP) and KNN-probing on labeled datasets are commonly employed [104]. In this approach, downstream task data is processed by the model to extract features, which are subsequently passed to either a logistic regression classifier or a KNN classifier to assess classification performance. Marks et al. [105] demonstrated that linear and kNN probing accuracies are highly correlated and can be used largely interchangeably.

This study implemented a linear probing validation protocol executed every 10 epochs throughout the pre-training and post-pre-training stages. Probing involved extracting features after the extractor blocks (fig. 3.4) from the entire training split of the FOR-Species20K dataset. Accuracy was then evaluated using a LogisticRegression classifier from `scikit-learn` on the validation split. Linear probing was chosen over KNN given the high dimensionality of the feature vector resulting from group sequence concatenation and flattening (24'576 dimensions for model S). While this dimensionality poses challenges for every linear classifier, KNN suffers disproportionately from the curse of dimensionality.

The computational intensity of the probing process required extending the GPU timeout in distributed mode to 3600 seconds, as evaluation is performed by a single rank while remaining ranks wait idle. Both top-1 and top-5 accuracy (correct class among top five predictions) metrics were recorded to assess classification performance and compared against the reference values in table 4.4.

Task category	Top-1 range (%)	Top-5 range (%)	Characteristics
In-domain evaluation	32.5 – 78.9	56.4 – 94.9	Standard SSL benchmark
Object-centric classification	42.4 – 91.2	64.4 – 99.2	Coarse-grained categories
Fine-grained classification	<b>7.8 – 61.1</b>	<b>16.6 – 85.5</b>	<b>Species-level tasks</b>
Cross-domain transfer	22.1 – 75.2	42.1 – 90.8	Style/domain shifts

**Table 4.4:** SSL linear probing performance ranges across vision task categories and datasets. From [105], summarized

## 4.2 Training runs

### 4.2.1 Overview

Excluding runs interrupted at or shortly after launch, approximately 300 partial or complete training runs were conducted. Roughly three-quarters of these runs were exploratory in nature, serving to develop the complete training pipeline and identify appropriate parameter ranges. The remaining quarter aimed to assess the effective model potential, though some still encountered implementation bugs or convergence issues.

The core experimentation focused on the S model configuration before expanding to the L configuration once reliable performance and stable training runs were established. The complete three-stage training process was successfully implemented using 1024 points for both pre-training and post-pre-training phases, mirroring the procedure established in the PointGPT paper. For fine-tuning, both 1024 and 2048 point configurations were thoroughly investigated, with the latter requiring adapted model configurations to accommodate the doubled token sequence length. While pre-training with 2048 points was explored, the 1024-point configuration provided more reliable convergence for the complete training pipeline. Although significantly higher resolutions are available from ground-based platforms (Ensemble PointNet++, the leading point-based method on the benchmark detailed in section 3.6.3, employs 8'192 points), the intention was to establish the approach's limits and develop from that foundation. Moreover, point densities exceeding 2048 points per tree prove challenging to achieve consistently with ALS data, and the objective was to avoid penalizing this platform, given that ALS constitutes a substantial portion of the pre-training dataset. Considering the approach's potential, developing a truly cross-platform model effective on ALS data too would represent a valuable contribution.

Training durations varied considerably across the three stages. Pre-training represented the most intensive phase, with training runs requiring 1.5 to 3.5 days to complete depending on batch and model size, whether distributed training was employed, and whether AMP was implemented. Post-pre-training proved comparatively efficient, requiring 1 to 6 hours to complete. Fine-tuning required intermediate training durations, ranging from 9 hours to 1 day.

### 4.2.2 Pre-training

#### Best runs



Figure 4.5: Pre-training: reconstruction loss on the training split.

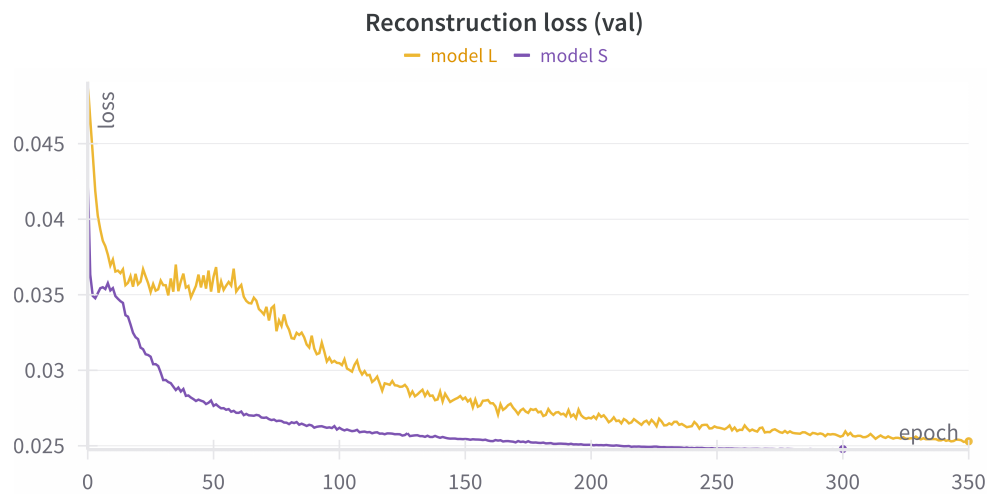


Figure 4.6: Pre-training: reconstruction loss on the validation split.

Model	Epochs	Batch size	LR schedule	Points	Groups
L	350	640	peak $2 \times 10^{-5}$ , warmup 80ep, cycle 600ep	1024	64
S	300	256	peak $1 \times 10^{-4}$ , warmup 10ep, cycle 300ep	1024	64

Table 4.5: Pre-training configurations.

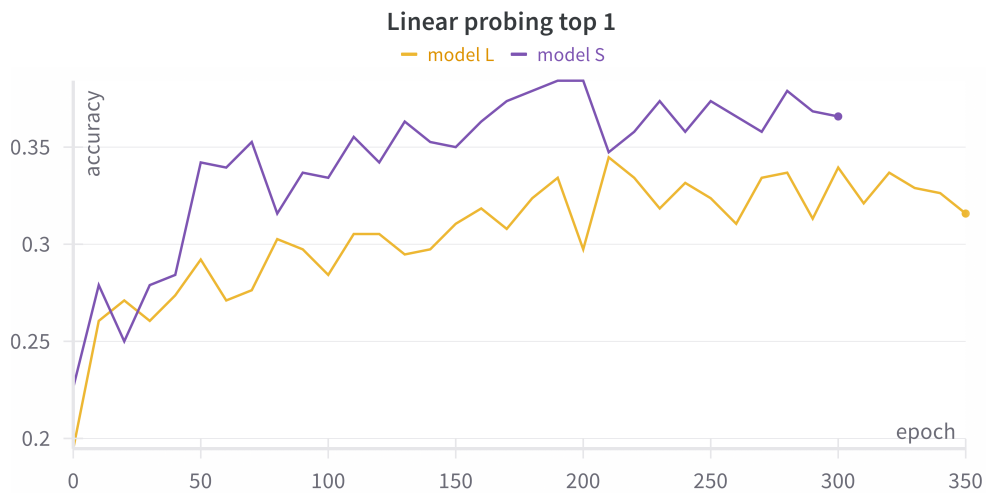


Figure 4.7: Pre-training: linear probing top 1.

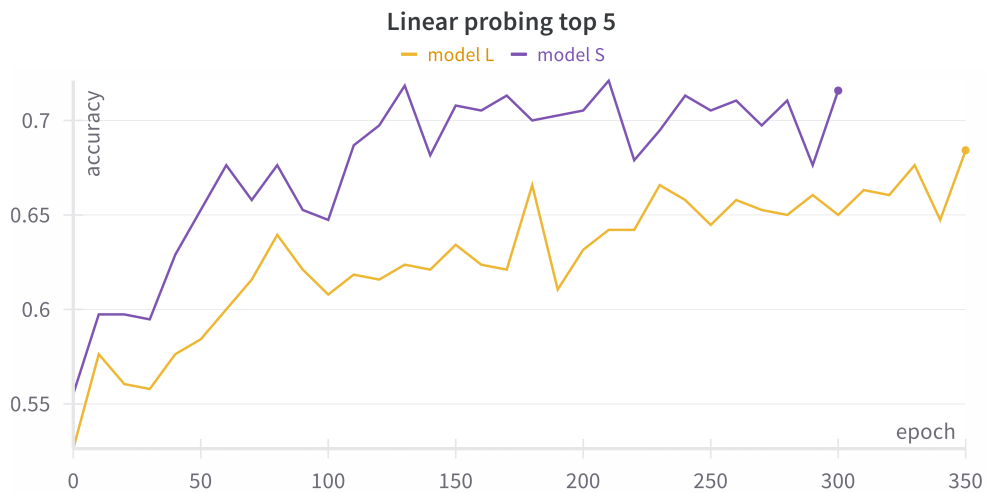


Figure 4.8: Pre-training: linear probing top 5.

The reconstruction loss trajectories (figs. 4.5 and 4.6) show convergence for both model configurations. Both models' losses decrease consistently from approximately 0.05 to 0.025, with model S achieving lower loss values on both training and validation sets throughout the training process. Model L, being approximately 14 times larger than model S, necessitated a more conservative learning rate schedule to avoid divergence (table 4.5). This approach included an extended warmup period of 80 epochs and a peak learning rate five times smaller than Model S. The cosine decay cycle was extended over 600 epochs, resulting in a learning rate that remained relatively high compared to the cycle's potential minimum at

the training’s conclusion. The batch size was also increased to address convergence issues. The close alignment between training and validation reconstruction losses indicates minimal overfitting, showing that both models potentially learned generalizable representations.

In linear probing (figs. 4.7 and 4.8), model S consistently outperforms model L. For top-1 accuracy, model S achieves approximately 37% compared to model L’s 32%. Similarly, top-5 accuracy reaches 72% for model S versus 68% for model L. These performance values align with ranges reported in the literature (table 4.4). Both models demonstrate clear upward trends in classification performance. The gap between top-1 and top-5 accuracies underscores the complexity of tree species classification, where differences between species can be minimal. The inferior linear probing performance of model L can also be attributed to the higher dimensionality of the feature vectors extracted from the model (384 versus 1024 dimensions per token), which penalizes linear classification methods due to the curse of dimensionality.

## Gridsearch

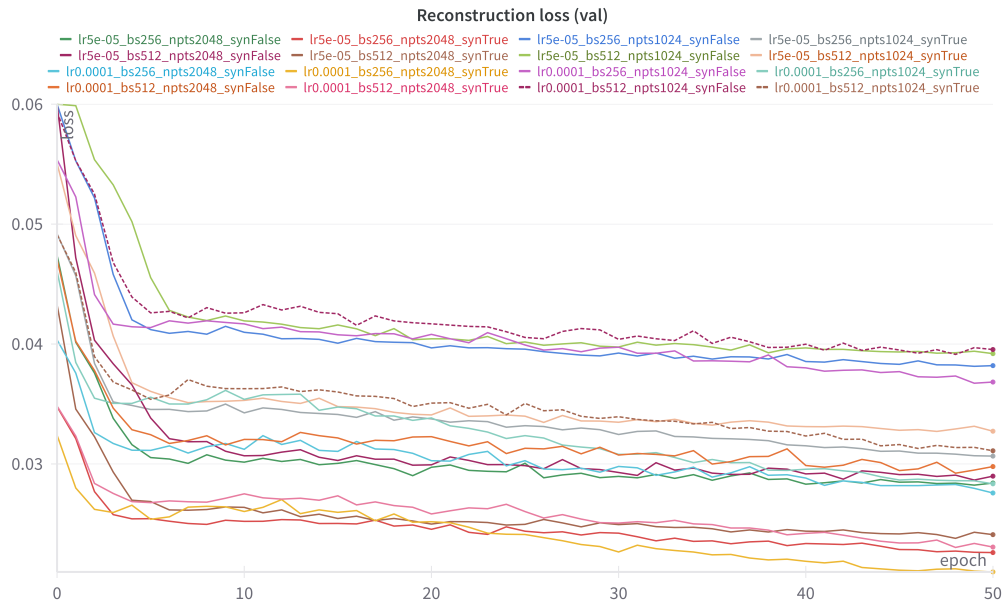


Figure 4.9: Pre-training gridsearch: reconstruction loss on the validation split.



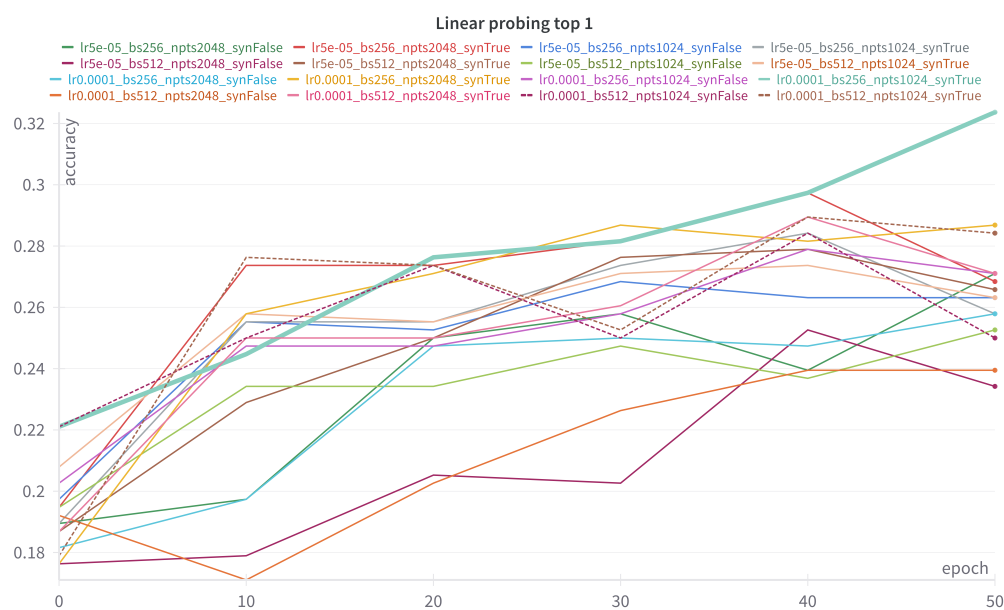


Figure 4.10: Pre-training gridsearch: linear probing top 1.

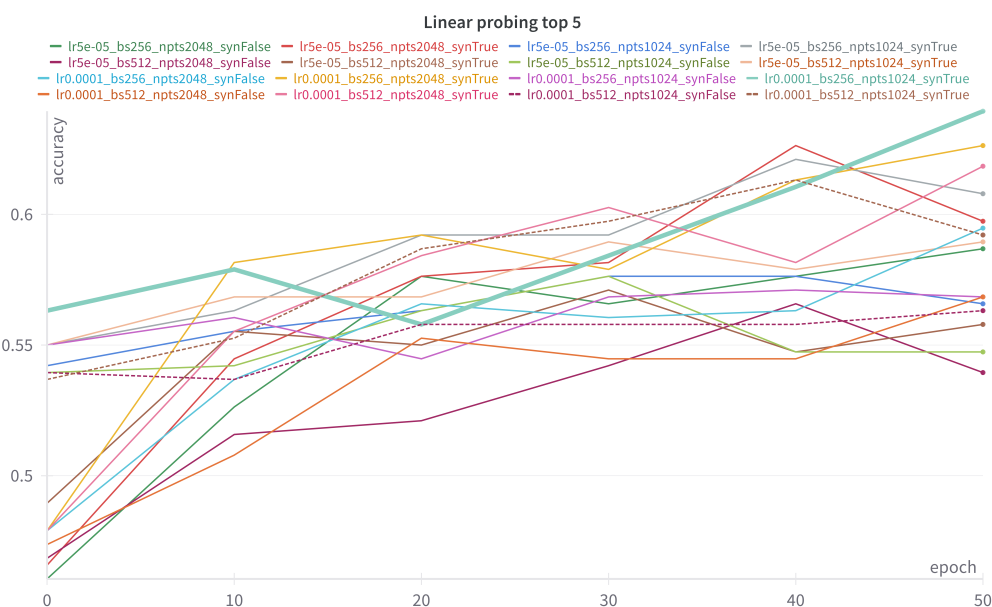


Figure 4.11: Pre-training gridsearch: linear probing top 5.

To determine optimal parameters for a complete model S training run, a grid search experiment was conducted using 50-epoch trials to explore parameter ranges identified from previous exploratory runs (table 4.6). The literature dataset was excluded from this experiment, while a binary parameter controlled the inclusion or exclusion of the synthetic dataset (denoted as `synTrue` and `synFalse` in figs. 4.9 to 4.11). This design enabled assessment of feature transferability when learning exclusively on ALS data and evaluation of the contribution of purely synthetic trees.

Hyperparameter	Values
Learning rate	$1 \times 10^{-4}$ , $5 \times 10^{-5}$
Batch size	512, 256
Number of points	1024, 2048
Synthetic data inclusion	True, False
<b>Total combinations</b>	<b>16</b>

**Table 4.6:** Gridsearch parameter space configuration.

In fig. 4.9, three distinct groups of experiments are identifiable based on their reconstruction loss performance. The upper group comprises exclusively runs without the synthetic dataset component. The inferior linear probing performance of these runs demonstrates that the synthetic component, despite being entirely artificial, provides substantial benefits for representation learning.

The middle group can be subdivided into two categories: the upper subgroup consists of runs using 1024 points without synthetic data, resulting in higher reconstruction losses compared to the lower configurations. The lower subgroup combines two experimental conditions: runs with 1024 points that include synthetic trees, and pure ALS runs using 2048 points.

The lowest group consists exclusively of runs incorporating both the synthetic component and 2048 points, achieving the most favorable reconstruction loss values across all configurations tested.

The linear probing results are likely biased because runs with 2048 points utilize token sequences of double length, resulting in higher feature vector dimensionality and consequently suffering from the curse of dimensionality. Nevertheless, the parameter combination of learning rate  $1 \times 10^{-4}$ , batch size 256, 1024 points, and synthetic dataset inclusion (bold line in figs. 4.10 and 4.11) demonstrated superior performance and was developed further (section 4.2.2).

## Further experiments

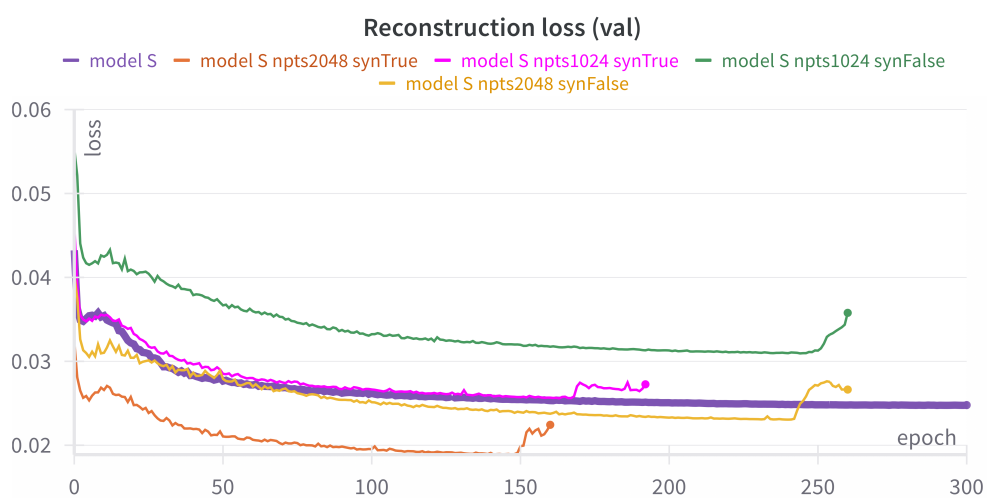


Figure 4.12: Pre-training trials: reconstruction loss on the validation split.

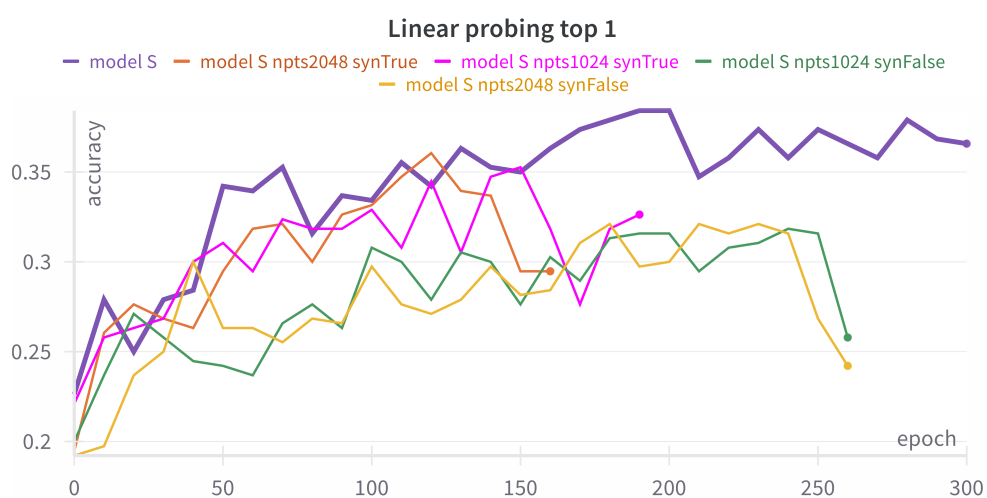


Figure 4.13: Pre-training trials: linear probing top 1.

Selected parameter combinations from the grid search were extended to longer training runs while maintaining the learning rate at  $1 \times 10^{-4}$  and batch size at 256. The literature dataset remained excluded from these experiments. The best performing configuration (section 4.2.2), trained on the complete dataset including literature data, appears in bold for reference. fig. 4.12 demonstrates that all configurations encountered divergence problems at some point, with runs including the synthetic dataset diverging earlier than those without it. Among runs using identical data compositions, configurations with 2048 points diverged before those using 1024 points. As a general consequence, model performance degraded (fig. 4.13).

Divergence issues represented a persistent challenge throughout this work. Analysis identified three most likely causes:

1. Learning rate too high: excessive learning rates can cause gradient updates to overshoot optimal parameter values, leading to unstable training dynamics. This was sometimes the case and was addressed by reducing the learning rate.
2. NaN values in input data or computational operations: this issue affected 3 out of 178'640 point clouds in the pre-training dataset. These point clouds contained duplicate centers after FPS, causing division by zero in the original RDP calculation (eq. (3.8)). These problematic point clouds were removed from the dataset, and a small epsilon value was added to the eq. (3.8) denominator (similar to eq. (4.2)) to enhance numerical stability.
3. Outlier bursts in input data: this represents the most likely explanation for the divergences observed in fig. 4.12. Although batch composition is randomized, the distribution shift between datasets, particularly between ALS and synthetic point clouds, is considerable. Batches with high concentrations of either dataset type could destabilize training, potentially explaining why runs including synthetic data diverge first. Among runs excluding synthetic data, the 2048-point configuration operates at higher detail levels and becomes more susceptible to ALS dataset outliers than the 1024-point configuration.

AMP emerged as a significant contributor to divergence dynamics. Given these models' relatively large size and low learning rates, gradients remain generally small. When NaN values or outlier bursts occur (events 2 and 3 above), the AMP scale factor, which is initially large to compensate for small gradients, must be sharply reduced to prevent gradient explosion. This scale reduction, triggered by only a few problematic samples, destabilizes the entire training process, which subsequently restabilizes at a higher loss baseline. The role of AMP in this issue was evaluated through fig. 4.14, where two identical experiments - one with AMP enabled and one without (corresponding to the best Model S run in section 4.2.2) - demonstrate that the AMP-free configuration avoids divergence. The evolution of gradient norms and AMP scaling factors are displayed in figs. 4.15 and 4.16.

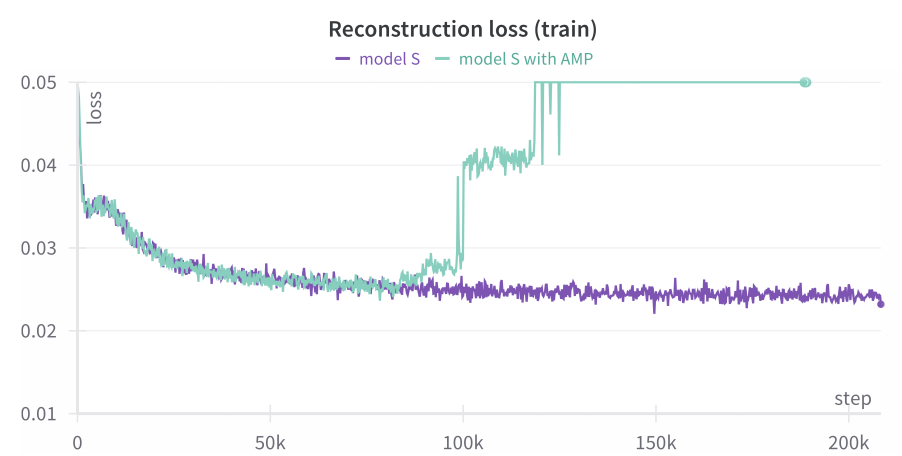


Figure 4.14: Divergence issue: reconstruction loss comparison between identical runs with and without AMP.

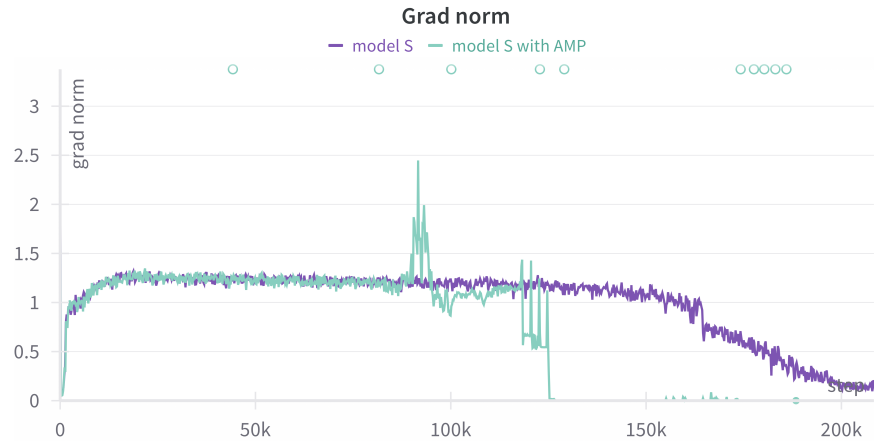


Figure 4.15: Divergence issue: gradients norm.

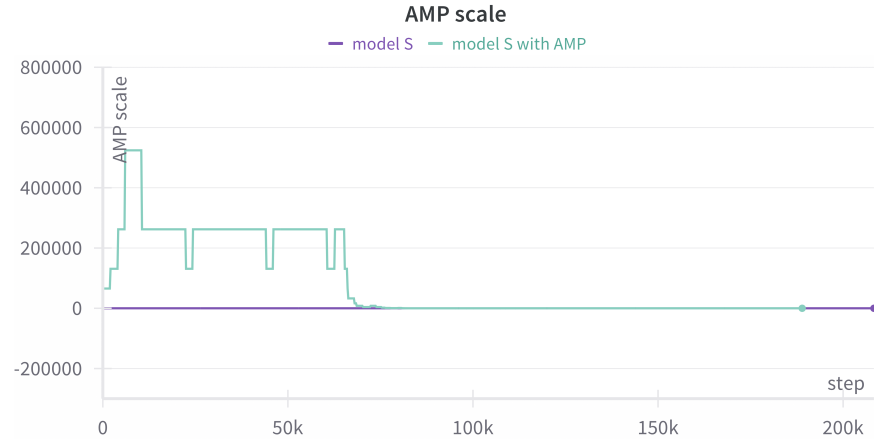


Figure 4.16: Divergence issue: AMP scale.

### 4.2.3 Post-pre-training

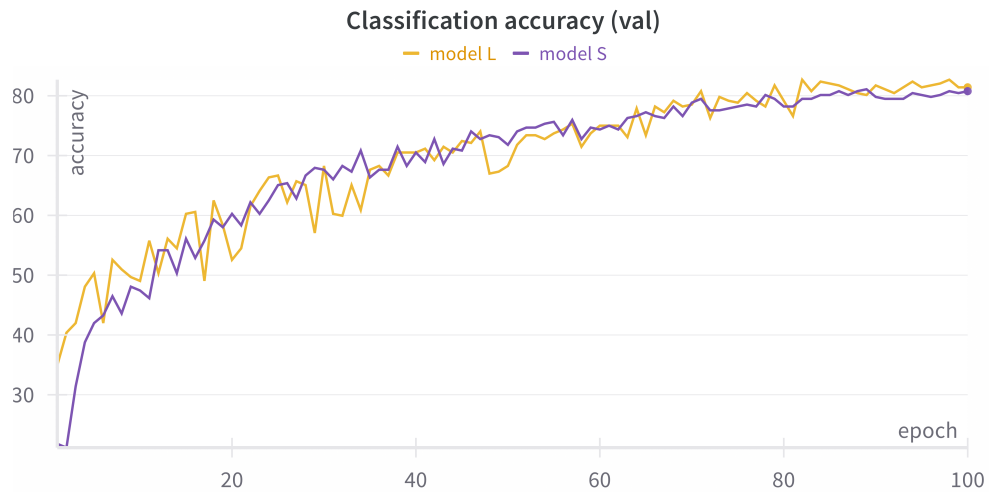


Figure 4.17: Post-pre-training: classification accuracy on the validation split.

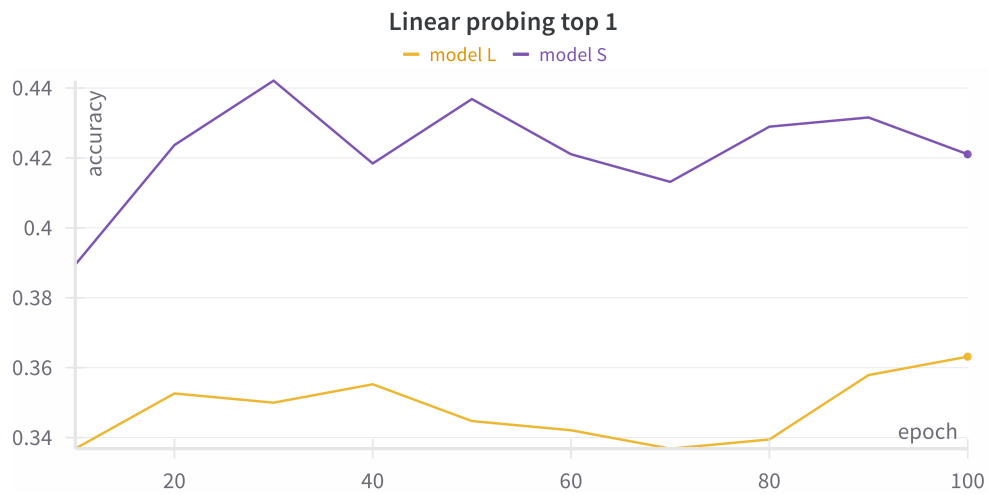


Figure 4.18: Post-pre-training: linear probing top 1.

Model	Epochs	Batch size	LR schedule	Points	Groups
L	100	256	peak $1 \times 10^{-4}$ , warmup 10ep, cycle 100ep	1024	64
S	100	256	peak $1 \times 10^{-4}$ , warmup 10ep, cycle 100ep	1024	64

Table 4.7: Post-pre-training configurations.

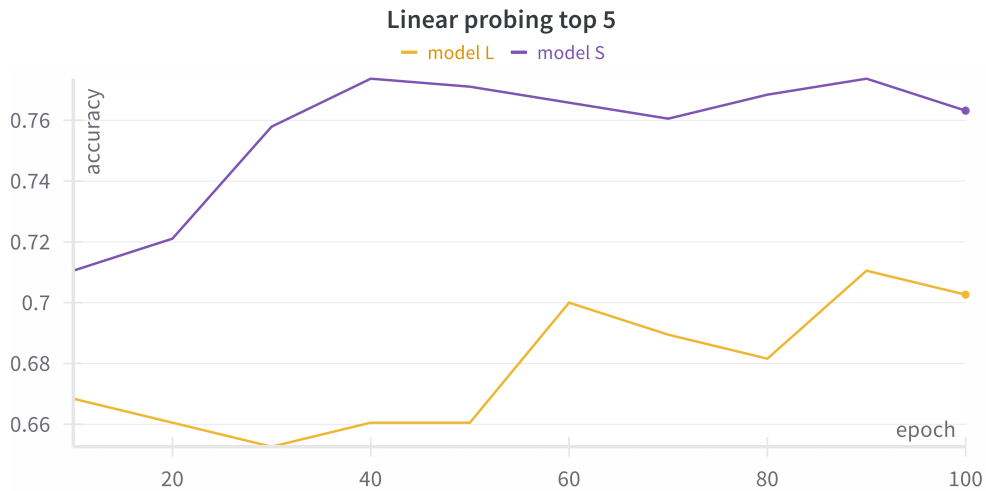


Figure 4.19: Post-pre-training: linear probing top 5.

The post-pre-training stage proceeded smoothly with minimal convergence issues, likely because unstable configurations prone to divergence had already been eliminated during the pre-training phase. Models L and S were trained with identical regimes (table 4.7) and achieved comparable accuracy on the validation split (fig. 4.17). Both configurations demonstrated substantial improvements in linear probing performance, with approximately 10% gains over their pre-training baselines (figs. 4.18 and 4.19). Model L continued to exhibit consistently inferior linear probing performance compared to model S, attributable to the higher dimensionality of its feature representations and the associated curse of dimensionality effects in linear classification.

### 4.2.4 Fine-tuning

#### Best runs

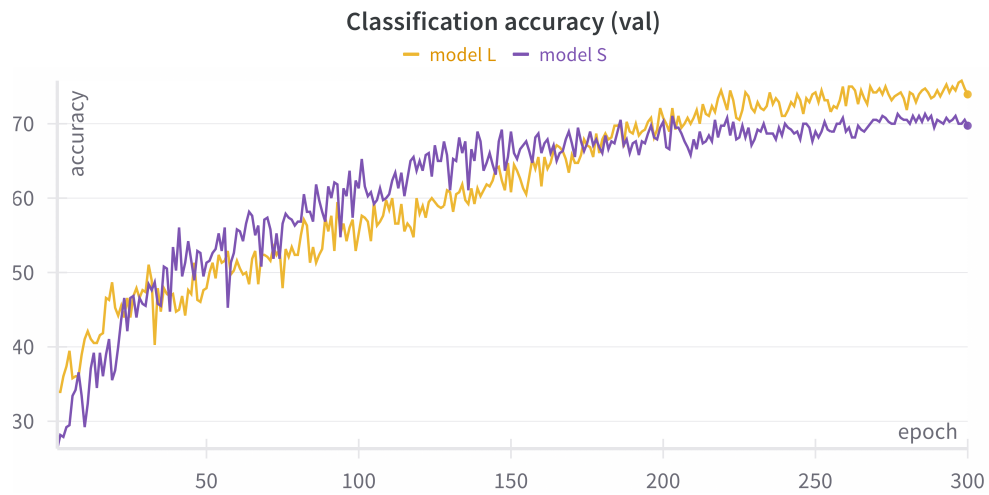


Figure 4.20: Fine-tuning: classification accuracy on the validation split.

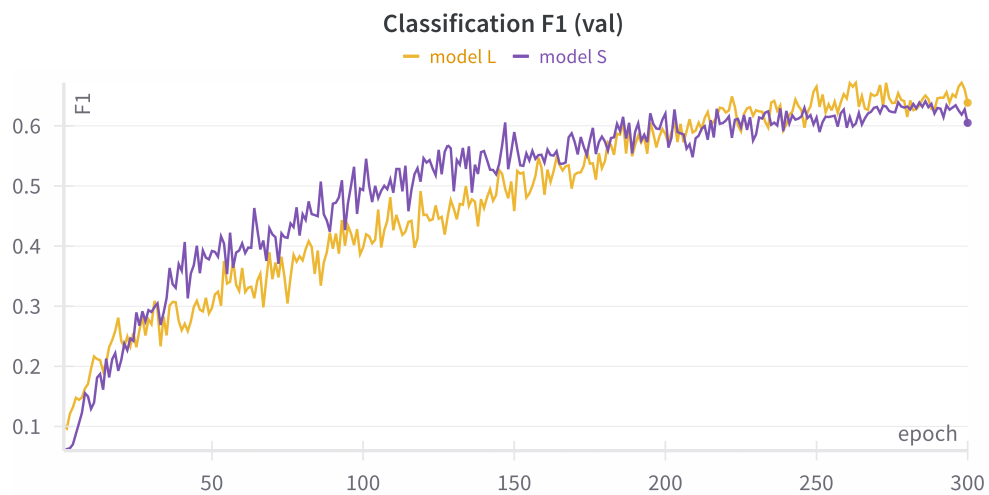


Figure 4.21: Fine-tuning: F1 score on the validation split.



Parameter	Model L	Model S
Epochs	300	300
Batch size	128	32
Peak learning rate	$1 \times 10^{-4}$	$1 \times 10^{-4}$
Warmup epochs	10	10
Cycle length	300	300
Points per sample	2048	2048
Groups	128	128
Loss function	Classification (unweighted) + reconstruction	

**Table 4.8:** Fine-tuning configurations.

During fine-tuning, both model configurations demonstrated substantial performance improvements throughout the 300-epoch training period. The classification accuracy trajectories (fig. 4.20) show slightly different learning dynamics between the two model configurations. Model S exhibits rapid initial improvement, achieving approximately 70% validation accuracy within the first 150 epochs before stabilizing around this performance level. In contrast, model L demonstrates a more gradual but sustained improvement pattern, ultimately surpassing Model S and reaching approximately 75% validation accuracy by the training conclusion. This better performance of the larger model during fine-tuning contrasts with the linear probing results observed in previous stages, suggesting that the additional model capacity becomes beneficial when the full model parameters are updated rather than when only a linear classifier is trained on frozen features. The F1 score evolution (fig. 4.21) follows a comparable pattern.

According to the training configurations (table 4.5), both models utilize 2048 points and 128 groups, doubling the resolution compared to the pre-training and post-pre-training stages. This increased resolution enables the models to leverage the higher point density available in the FOR-Species20K dataset, particularly from ground-based scanning platforms. Following an additional grid search, a reduced BS of 32 proved beneficial for model S, enabling better exploration of the loss landscape. However, this configuration proved too destabilizing for model L, which necessitated a higher BS. The optimal results were achieved by combining both classification and reconstruction objectives, while the weighted loss formulation was not employed.

To assess whether the models retained potential for continued learning, fig. 4.22 reveals that validation loss for both models began increasing, indicating overfitting and suggesting that learning capacity had been reached. Model S exhibits particularly pronounced overfitting characteristics, demonstrating elevated training accuracies (fig. 4.23) coupled with increasing validation loss. The larger capacity of model L appears to provide a regularizing effect, mitigating overfitting behavior.

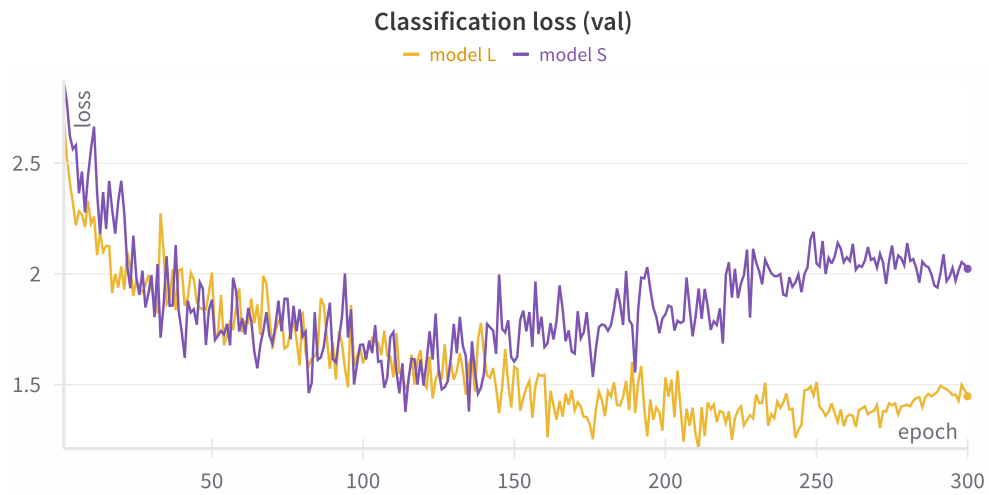


Figure 4.22: Fine-tuning: classification loss on the validation split.

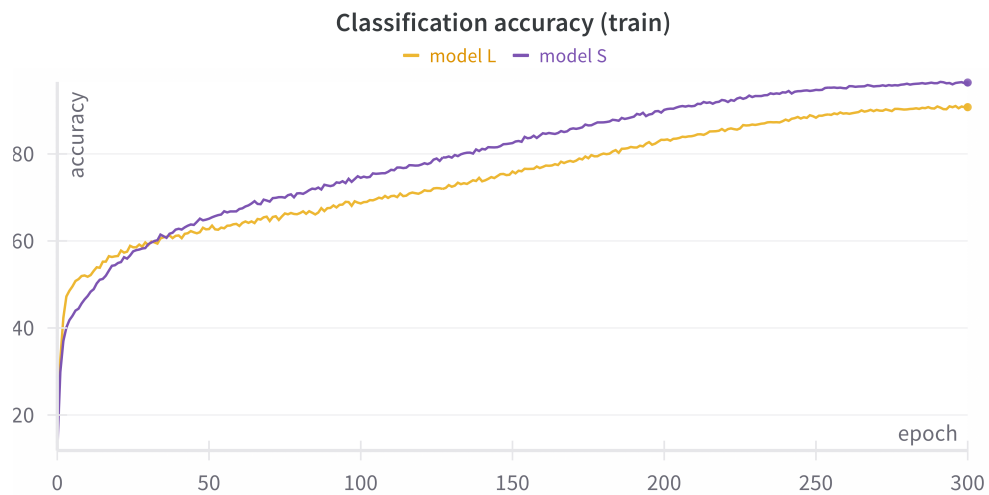


Figure 4.23: Fine-tuning: classification accuracy on the training split.

## Spectral adapters (PEFT)

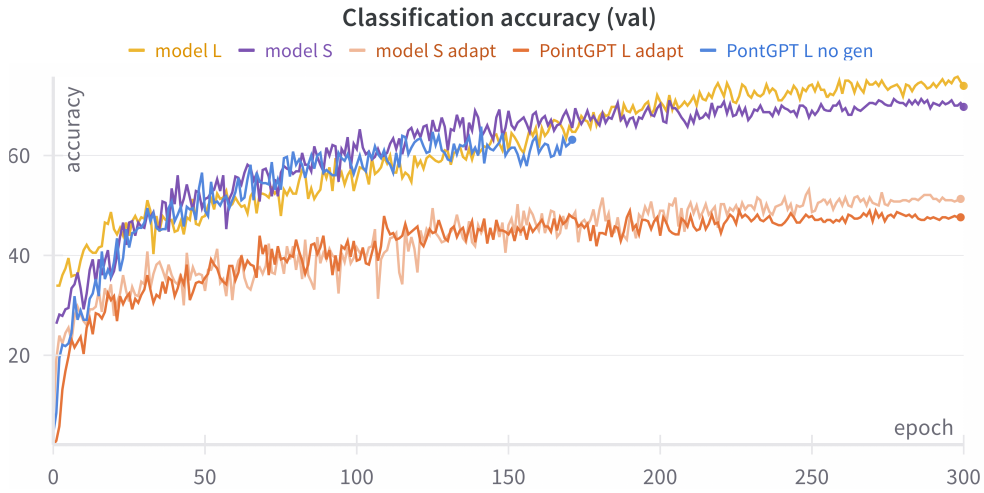


Figure 4.24: Fine-tuning with adapters (PEFT): classification accuracy on the validation split.

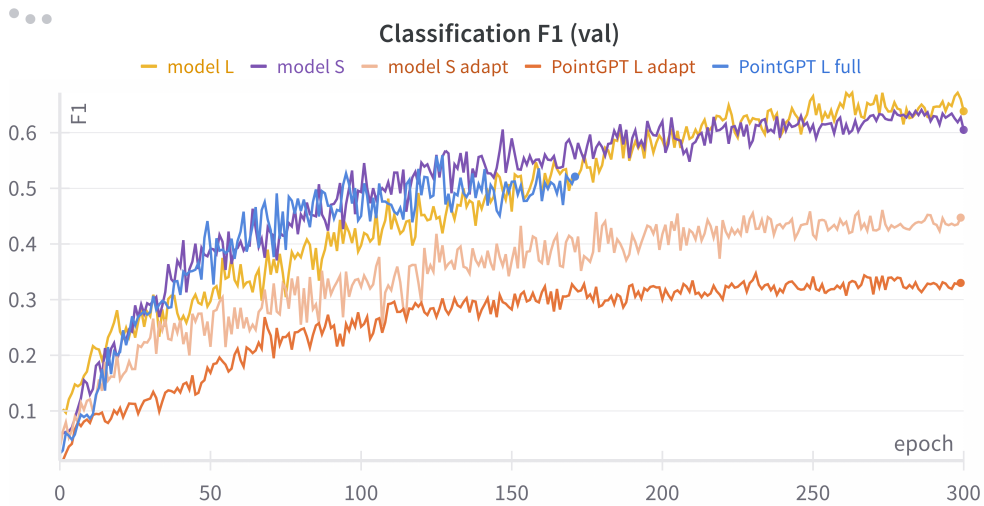


Figure 4.25: Fine-tuning with adapters (PEFT): F1 score on the validation split.

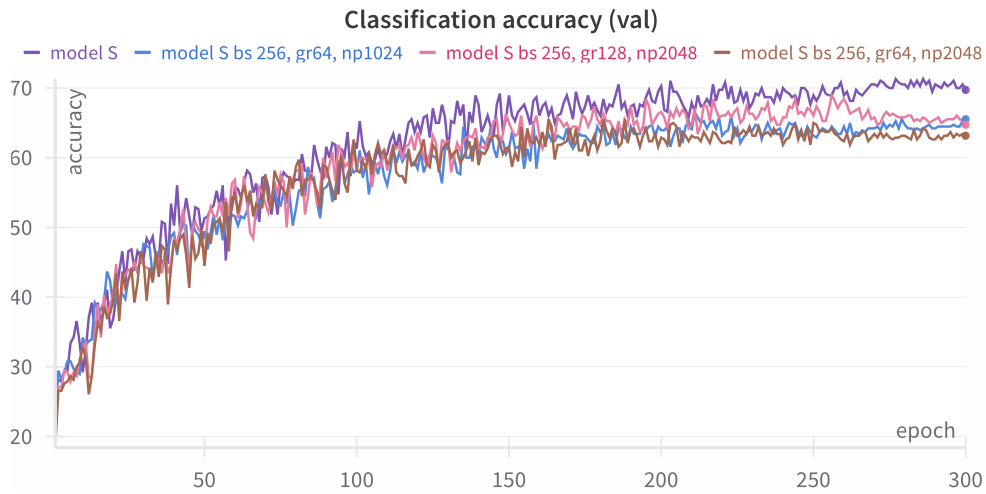
Experiments utilizing PointGST's spectral adapters were conducted across multiple configurations. In figs. 4.24 and 4.25, model S and model L serve as reference baselines from section 4.2.4. "Model S adapt" represents the post-pre-trained model from section 4.2.3, while "PointGPT L adapt" corresponds to the post-pre-trained model from the PointGPT paper. Both configurations employed fewer than 1% of total model parameters as tunable parameters.

While poor performance from "PointGPT L adapt" was anticipated due to its training on objects and indoor/outdoor urban scenes - representing a substantial domain gap from trees - the suboptimal performance of "model S adapt" was unexpected. Model S adapt achieved approximately 50% accuracy but with substantially better F1 scores. The poor performance can be attributed to several factors:

1. Spectral domain patterns may be absent or insufficiently discriminative. Given the high degree of noisiness in subsampled tree point clouds, spectral decomposition likely fails to produce meaningful patterns. Trees at this resolution lack the simpler geometric surfaces (planar, curved, sharp, thin, etc.) that spectral methods typically capture. Instead, tree crown features at these point densities likely generate similar spectral components across species, with meaningful differentiation potentially restricted to broad morphological categories such as deciduous versus coniferous trees.
2. The inherent capacity of adapter layers inserted into the model may be insufficient, causing underfitting through inadequate learning capability.
3. The pre-training and post-pre-training stages may contribute limited transferable representations, resulting in insufficient frozen knowledge for adapter layers to leverage effectively. However, this hypothesis receives partial contradiction from the experimental evidence presented in figs. 4.27 and 4.28.

"PointGPT no gen" represents a post-pre-trained model from the PointGPT paper fine-tuned without the generative task. This configuration demonstrated rapid improvement during the initial 100 epochs before stagnating and triggering early stopping after 30 additional epochs. Nevertheless, this result suggests that pre-training functions primarily as model initialization in this context, consistent with point 3 above, indicating only marginal tree-specific knowledge acquisition by the fine-tuning stage.

## Sequence length



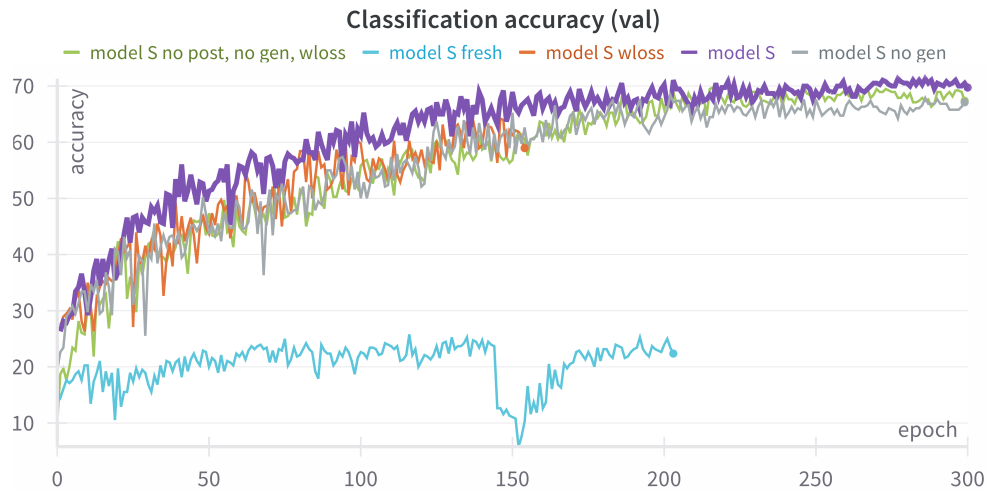
**Figure 4.26:** Fine-tuning: classification accuracy on the validation split for models with different token sequence lengths.

This series of experiments was designed to assess model behavior when confronted with changes in token sequence length. During pre-training, the model is trained on 1024 points using 64 groups of 32 points each. Since  $64 \times 32 = 2048$ , point groups overlap and reconstruction in the generative task remains predominantly partial. This configuration means that attention patterns become fitted to this specific sequence length, as the attention mask size in transformer blocks corresponds to this length. When sequence length doubles, these learned patterns face disruption since the attention mask itself doubles in size.

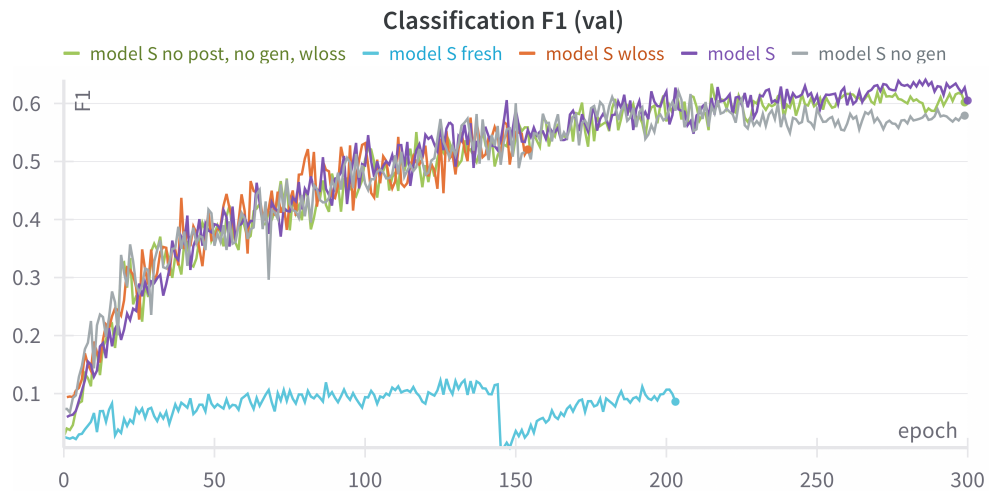
To evaluate this effect, fig. 4.26 presents experiments varying both the number of groups and point count. Despite attention mask disruption, longer sequence lengths prove beneficial to performance. Conversely, reducing group overlap (64 groups with 2048 points, where  $64 \times 32 = 2048$  means minimal group overlap) detracts performance. Investigation reveals that although absolute positional relationships become disrupted, feature representations and relative attention patterns remain preserved. Consequently, doubling sequence length provides more advantages through higher resolution information than disadvantages through pattern disruption.

However, the extent to which sequence length can safely vary remains limited, as sequence length itself can become overfitted [106]. An alternative approach for incorporating additional information could involve increasing group size rather than group number. Experimental evidence suggests this represents a failing strategy [107].

## Further experiments



**Figure 4.27:** Fine-tuning: classification accuracy on the validation split for different model S configurations.



**Figure 4.28:** Fine-tuning: F1 score on the validation split for different model S configurations.

The final series of experiments evaluated miscellaneous configurations (figs. 4.27 and 4.28), with model S serving as the reference model from section 4.2.3. An uninitialized "model S fresh" configuration with parameters set to negative infinity was trained from scratch directly on the FOR-Species20K dataset. Considering the analysis presented in section 4.2.4, although knowledge acquired after pre-training and post-pre-training may be somewhat limited, training from scratch

fails to achieve meaningful performance and encounters significant divergence issues. This confirms that SSL provides substantial benefits, although the precise extent of these benefits remains unclear.

Introducing the weighted loss formulation outlined in section 4.1.1 proved ineffective, as demonstrated by the "model S wloss" run, which was terminated due to inferior performance compared to baseline model S. This finding was, however, partially contradicted by the "model S no post, no gen, wloss" run. This configuration utilized model S from the pre-training stage without post-pre-training and was fine-tuned without the generative task, achieving final accuracy and F1 scores not far from model S. This result requires closer analysis, since model S exhibited overfitting during later training stages (section 4.2.2). Excluding these overfitting periods, its performance remained consistently superior and is therefore considered the preferred approach.

The final comparison examines training with (model S) and without ("model S no gen") the auxiliary generative task. As demonstrated in figs. 4.27 and 4.28, omitting the generative task is an inferior approach, as the generative component appears to provide regularization benefits. This finding is interesting, since knowledge acquired for the generative task could potentially interfere with classification learning when gradients from the generator propagate back into the extractor. Indeed, the PointGPT paper reports improved classification performance with deeper generator architectures. This gradient backpropagation may paradoxically serve as the source of the regularization benefits identified above, suggesting that the conflict between generative and discriminative objectives actually improves model generalization.

### 4.3 Best model

Model L from section 4.2.4, now designated as TreeGPT, represents the best-performing configuration and undergoes detailed performance analysis. Since inference results on the FOR-Species20K test split have not been submitted to the official benchmark, the following analyses are conducted on the validation split described in section 4.1.1.

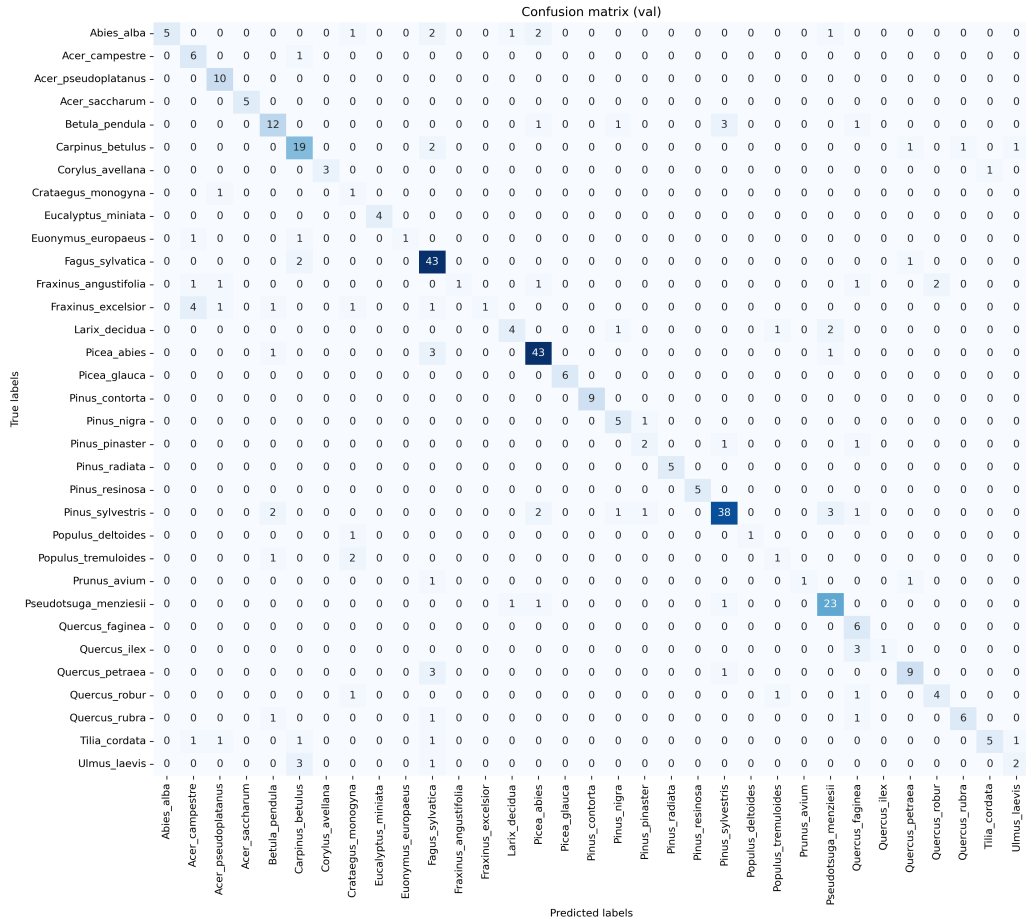


Figure 4.29: Best model: confusion matrix.

Method	TreeGPT (this work)	DetailView [99]	Ensemble PointNet++
Dataset	Validation split	FOR-Species20K test	FOR-Species20K test
Accuracy	76%	79%	76%
Precision	0.79	0.81	0.77
Recall	0.67	0.79	0.76
F1-score	0.67	0.79	0.75
Note	This work	Best overall	Best point-based method

**Table 4.9:** Performance comparison between TreeGPT and leading methods. TreeGPT results are reported on the validation split of this work’s dataset, while benchmark results are on the official FOR-Species20K test split, limiting direct comparability.



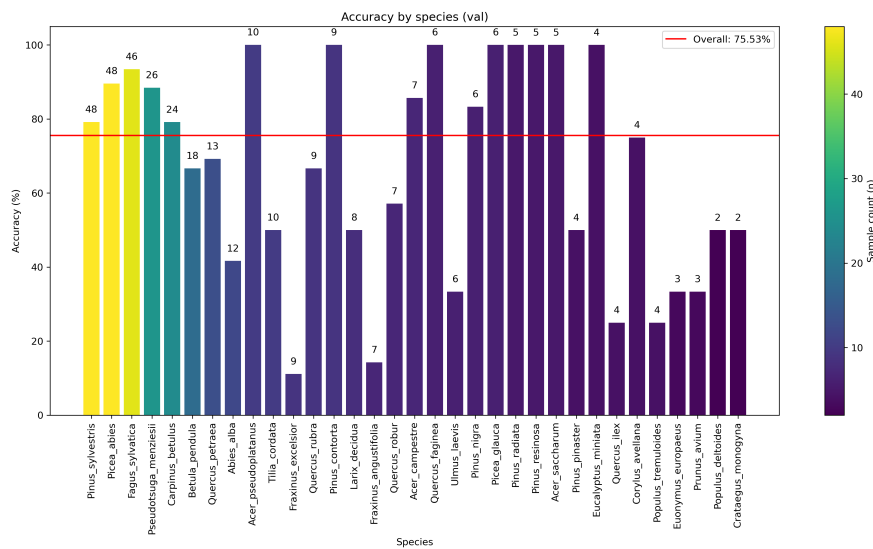


Figure 4.30: Best model: accuracy by species.

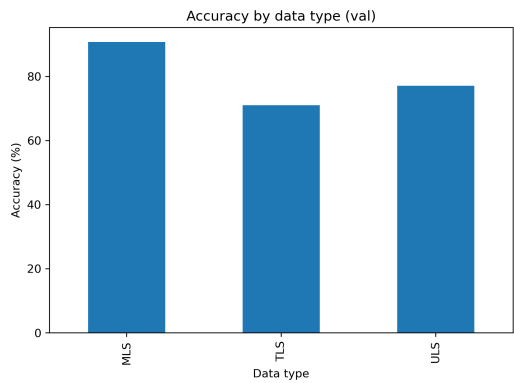


Figure 4.31: Best model: accuracy by data type.

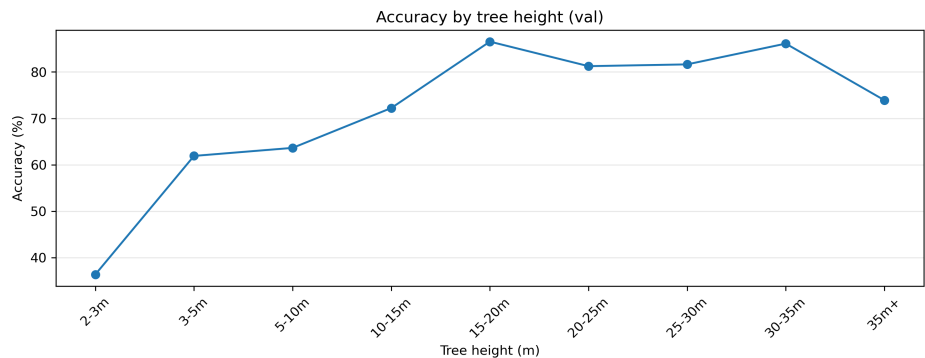


Figure 4.32: Best model: accuracy by tree height.

As an additional baseline, a random forest classifier tuned for this task achieves 40% accuracy when trained on the training split and evaluated on the validation split.

Despite training on a heavily imbalanced dataset without balancing measures, TreeGPT does not excessively penalize species with lower representation (fig. 4.30). Good accuracy is achieved even for species with counts 10 times lower than the most numerous ones.

Performance across platform types (fig. 4.31) and tree heights (fig. 4.32) follows the same trends as benchmark models in the dataset paper [77]. TLS performs worst despite its high scanning fidelity, while MLS achieves the best performance. Accuracy increases with tree height, which is expected since taller trees are typically more mature and have fully developed the distinctive characteristics that define their species.

According to table 4.9, TreeGPT achieves performance comparable to Ensemble PointNet++, the leading point-cloud-based method on the benchmark. TreeGPT achieves final training and validation classification loss values of approximately 0.29 and 1.21, respectively. DetailView, which employs a similar dataset split methodology as this study, achieves corresponding loss values of 0.30 and 0.76 (fig. 3.13). This comparison reveals that while TreeGPT achieves competitive classification accuracy, it exhibits higher validation loss than DetailView, suggesting potential overfitting when compared to the latter.

The TreeGPT approach shows potential for further optimization to surpass this performance and rival image-based baselines. TreeGPT has not yet been trained or evaluated on higher point densities. Training on higher densities could enable the model to learn patterns imperceptible at lower densities, potentially improving robust knowledge transfer to lower-density scenarios—a direction not explored in the current work, which took the opposite approach. Furthermore, no ensemble or voting mechanism has been implemented, which could augment performance by several percentage points.

From a computational efficiency standpoint, TreeGPT exhibits significant computational demands in its L configuration, despite this being the best-performing variant. This trade-off could pose challenges for downstream applications, depending on computational cost considerations. The large parameter count contributes to model robustness, and few-shot performance is expected to be superior based on corresponding results in the PointGPT paper. This robustness enables further fine-tuning on species with limited data. A hybrid approach combining a large primary model with smaller models fine-tuned on rare species could prove effective.

## 5 Conclusions

This thesis investigated the application of SSL with transformer architectures to tree species classification from LiDAR point clouds. A custom dataset totaling over 200'000 individual tree instances across multiple platforms and geographic regions was assembled. The integration of real ALS data from Canton Neuchâtel with synthetic augmentations, combined with literature datasets from diverse acquisition platforms, created a foundation for SSL pre-training.

A three-stage training approach for tree species classification was implemented using PointGPT. Approximately 300 training runs were conducted to develop the complete pipeline and assess model potential across Small (S), Base (B), and Large (L) configurations.

The pre-training stage demonstrated that model S consistently outperformed model L in linear probing evaluations, achieving 37% top-1 accuracy compared to 32% for Model L, and 72% versus 68% top-5 accuracy respectively. The inferior linear probing performance of model L can be most likely attributed to the higher dimensionality of its feature vectors (1024 versus 384 dimensions per token), which penalizes linear classification methods due to the curse of dimensionality. Grid search experiments revealed that synthetic data inclusion provides substantial benefits for representation learning, while configurations excluding synthetic components showed inferior performance.

Training stability emerged as a constant challenge, with divergence issues affecting multiple configurations. These divergences were attributed to the combination of excessive learning rates, NaN values from duplicate point centers, and outlier bursts made worse by AMP scaling dynamics. The post-pre-training stage achieved approximately 10% improvements in linear probing performance over pre-training baselines for both model configurations.

Fine-tuning results demonstrated that model L ultimately achieved better performance despite its poor linear probing results in earlier stages, reaching 75% validation accuracy compared to 70% for model S. Experiments with spectral adapters achieved only 50% accuracy, indicating limited effectiveness of the PEFT approach in this context. The final TreeGPT model (model L) achieved 75% accuracy, 0.79 precision, 0.67 recall, and 0.67 F1-score on the validation split, demonstrating performance comparable to Ensemble PointNet++, the leading point-cloud-based method on the FOR-Species20K benchmark. Despite training on heavily imbalanced data without explicit balancing measures, the model maintained good

performance across species with varying representation levels. Performance patterns across platform types and tree heights aligned with established benchmark trends, with inference on MLS clouds getting the best performance and accuracy increasing with tree height.

Training from scratch without pre-training failed to achieve meaningful performance, confirming the benefits provided by SSL. Several technical optimizations were implemented to attain faster processing speeds, shorter loading times, and smaller storage requirements.

Future research directions should focus on training on higher point densities, as it is the most immediate opportunity for performance enhancement across all density ranges. The implementation of voting and ensemble methods could also provide several percentage points of performance improvement with relatively modest implementation effort.

The successful adaptation of language model pre-training paradigms to geometric data shows the versatility of transformer architectures. The insights gained regarding training stability, model scaling, and synthetic data integration contribute to the development of further robust point cloud processing systems.

In conclusion, this thesis establishes TreeGPT as a viable approach for tree species classification from LiDAR point clouds. It contributes to the further development and application of SSL techniques on point clouds through practical technical experiences and insights.


## Declaration of Authorship

I hereby declare that I have written this thesis independently and have not used any sources or aids other than those acknowledged.

All statements taken from other writings, either literally or in essence, have been marked as such.

I hereby agree that the present work may be reviewed in electronic form using appropriate software.

August 7, 2025

A handwritten signature in dark ink, reading "Ivo Gasparini", is positioned above a horizontal line.

I. Gasparini



## Bibliography

- [1] U.-B. Brändli, M. Abegg, and B. Allgaier Leuch, *Schweizerisches Landesforstinventar. Ergebnisse der vierten Erhebung 2009–2017*, en, 2020. DOI: 10.16904/ENVIDAT.146.
- [2] Bundesamt für Umwelt, “Jahrbuch Wald und Holz 2024,” Bundesamt für Umwelt BAFU, Bern, Switzerland, Tech. Rep. UZ-2410-D, 2024.
- [3] B. Muys, “Forest Ecosystem Services,” en, in *Life on Land*, W. Leal Filho, A. M. Azul, L. Brandli, A. Lange Salvia, and T. Wall, Eds., Cham: Springer International Publishing, 2021, pp. 386–395, ISBN: 9783319959801 9783319959818. DOI: 10.1007/978-3-319-95981-8\_129.
- [4] Bundesamt für Umwelt (BAFU), “Waldpolitik: Ziele und Massnahmen 2021–2024. Für eine nachhaltige Bewirtschaftung des Schweizer Waldes,” German, Bundesamt für Umwelt, Bern, Umwelt-Info Wald & Holz 2119, 2021, p. 61.
- [5] Bundesversammlung der Schweizerischen Eidgenossenschaft, *Bundesbeschluss zur Finanzierung von Aufgaben im Umweltbereich in den Jahren 2025–2028*, German, 2024.
- [6] C. Fischer and B. Traub, Eds., *Swiss National Forest Inventory – Methods and Models of the Fourth Assessment (Managing Forest Ecosystems)*, en. Cham: Springer International Publishing, 2019, vol. 35, ISBN: 9783030192921 9783030192938. DOI: 10.1007/978-3-030-19293-8.
- [7] P. Bürgi, A. Müller, B. Pauli, and C. Rosset, “Forstwirtschaftliches Testbetriebsnetz der Schweiz: Ergebnisse der Jahre 2020–2022,” Bundesamt für Statistik (BFS), Neuchâtel, Tech. Rep. 1241-2200, 2024, p. 52.
- [8] A. Holzinger, J. Schweizer, C. Gollob, *et al.*, “From Industry 5.0 to Forestry 5.0: Bridging the gap with Human-Centered Artificial Intelligence,” en, *Current Forestry Reports*, vol. 10, no. 6, pp. 442–455, Sep. 2024, ISSN: 2198-6436. DOI: 10.1007/s40725-024-00231-7.
- [9] P. C. Pandey and P. Arellano, Eds., *Advances in Remote Sensing for Forest Monitoring*, en, 1st ed. Wiley, Nov. 2022, ISBN: 9781119788126 9781119788157. DOI: 10.1002/9781119788157.
- [10] T. Zohdi, “A machine-learning enabled digital-twin framework for next generation precision agriculture and forestry,” en, *Computer Methods in Applied Mechanics and Engineering*, vol. 431, p. 117 250, Nov. 2024, ISSN: 00457825. DOI: 10.1016/j.cma.2024.117250.
- [11] B. Xiang, M. Wielgosz, T. Kontogianni, *et al.*, “Automated forest inventory: Analysis of high-density airborne LiDAR point clouds with 3D deep learn-

- ing,” en, *Remote Sensing of Environment*, vol. 305, p. 114 078, May 2024, ISSN: 00344257. DOI: 10.1016/j.rse.2024.114078.
- [12] J. Shao, Y.-C. Lin, C. Wingren, *et al.*, “Large-scale inventory in natural forests with mobile LiDAR point clouds,” en, *Science of Remote Sensing*, vol. 10, p. 100 168, Dec. 2024, ISSN: 26660172. DOI: 10.1016/j.srs.2024.100168.
- [13] D. Laino, C. Cabo, C. Prendes, *et al.*, “3DFin: a software for automated 3D forest inventories from terrestrial point clouds,” en, *Forestry: An International Journal of Forest Research*, vol. 97, no. 4, F. Fassnacht, Ed., pp. 479–496, Aug. 2024, ISSN: 0015-752X, 1464-3626. DOI: 10.1093/forestry/cpae020.
- [14] W. G. Rodrigues, G. S. Vieira, C. D. Cabacinha, R. F. Bulcão-Neto, and F. Soares, “Applications of artificial intelligence and LiDAR in forest inventories: A Systematic Literature Review,” en, *Computers and Electrical Engineering*, vol. 120, p. 109 793, Dec. 2024, ISSN: 00457906. DOI: 10.1016/j.compeleceng.2024.109793.
- [15] M. Kulicki, C. Cabo, T. Trzciński, J. Będkowski, and K. Stereńczak, “Artificial Intelligence and Terrestrial Point Clouds for Forest Monitoring,” en, *Current Forestry Reports*, vol. 11, no. 1, p. 5, Dec. 2024, ISSN: 2198-6436. DOI: 10.1007/s40725-024-00234-4.
- [16] Schweizerische Eidgenossenschaft, “Strategie für offene Verwaltungsdaten in der Schweiz 2019–2023 (Open-Government-Data-Strategie, OGD-Strategie),” German, *Bundesblatt (BBl)*, vol. 2019, no. 2018-2782, pp. 879–894, 2019.
- [17] R. Abreu-Dias, J. M. Santos-Gago, F. Martín-Rodríguez, and L. M. Álvarez-Sabucedo, “Advances in the Automated Identification of Individual Tree Species: A Systematic Review of Drone- and AI-Based Methods in Forest Environments,” en, *Technologies*, vol. 13, no. 5, p. 187, May 2025, ISSN: 2227-7080. DOI: 10.3390/technologies13050187.
- [18] O. Reisi Gahrouei, J.-F. Côté, P. Bournival, P. Giguère, and M. Béland, “Comparison of Deep and Machine Learning Approaches for Quebec Tree Species Classification Using a Combination of Multispectral and LiDAR Data,” en, *Canadian Journal of Remote Sensing*, vol. 50, no. 1, p. 2 359 433, Dec. 2024, ISSN: 0703-8992, 1712-7971. DOI: 10.1080/07038992.2024.2359433.
- [19] D. Seidel, P. Annighöfer, A. Thielman, *et al.*, “Predicting Tree Species From 3D Laser Scanning Point Clouds Using Deep Learning,” *Frontiers in Plant Science*, vol. 12, p. 635 440, Feb. 2021, ISSN: 1664-462X. DOI: 10.3389/fpls.2021.635440.
- [20] J. Gui, T. Chen, J. Zhang, *et al.*, “A Survey on Self-Supervised Learning: Algorithms, Applications, and Future Trends,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 12, pp. 9052–9071, Dec. 2024, ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2024.3415112.
- [21] X. Yu, L. Tang, Y. Rao, T. Huang, J. Zhou, and J. Lu, *Point-BERT: Pre-training 3D Point Cloud Transformers with Masked Point Modeling*, 2021. DOI: 10.48550/ARXIV.2111.14819.



- [22] Y. Pang, W. Wang, F. E. H. Tay, W. Liu, Y. Tian, and L. Yuan, *Masked Autoencoders for Point Cloud Self-supervised Learning*, 2022. DOI: 10.48550/ARXIV.2203.06604.
- [23] R. Zhang, Z. Guo, R. Fang, *et al.*, *Point-M2AE: Multi-scale Masked Autoencoders for Hierarchical Point Cloud Pre-training*, 2022. DOI: 10.48550/ARXIV.2205.14401.
- [24] G. Chen, M. Wang, Y. Yang, K. Yu, L. Yuan, and Y. Yue, *PointGPT: Auto-regressively Generative Pre-training from Point Clouds*, 2023. DOI: 10.48550/ARXIV.2305.11487.
- [25] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention Is All You Need*, 2017. DOI: 10.48550/ARXIV.1706.03762.
- [26] K. Han, Y. Wang, H. Chen, *et al.*, “A Survey on Vision Transformer,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 1, pp. 87–110, Jan. 2023, ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2022.3152247.
- [27] T. B. Brown, B. Mann, N. Ryder, *et al.*, *Language Models are Few-Shot Learners*, 2020. DOI: 10.48550/ARXIV.2005.14165.
- [28] OpenAI, J. Achiam, S. Adler, *et al.*, *GPT-4 Technical Report*, 2023. DOI: 10.48550/ARXIV.2303.08774.
- [29] J. Kaplan, S. McCandlish, T. Henighan, *et al.*, *Scaling Laws for Neural Language Models*, 2020. DOI: 10.48550/ARXIV.2001.08361.
- [30] L. Shaheen, B. Rasheed, and M. Mazzara, “Self-Supervised Learning for Precise Individual Tree Segmentation in Airborne LiDAR Point Clouds,” *IEEE Access*, vol. 13, pp. 70 895–70 908, 2025, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2025.3563363.
- [31] F. Wang and M. Bryson, “Tree Segmentation and Parameter Measurement from Point Clouds Using Deep and Handcrafted Features,” *en, Remote Sensing*, vol. 15, no. 4, p. 1086, Feb. 2023, ISSN: 2072-4292. DOI: 10.3390/rs15041086.
- [32] L. Shaheen, B. Rasheed, and M. Mazzara, “Tree species detection using hyperspectral and Lidar data: A novel self-supervised learning approach,” *Computer Research and Modeling*, vol. 16, no. 7, pp. 1747–1763, Dec. 2024, ISSN: 20767633, 20776853. DOI: 10.20537/2076-7633-2024-16-7-1747-1763.
- [33] A. Wehr and U. Lohr, “Airborne laser scanning—an introduction and overview,” *en, ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 54, no. 2-3, pp. 68–82, Jul. 1999, ISSN: 09242716. DOI: 10.1016/S0924-2716(99)00011-8.
- [34] M. A. Lefsky, W. B. Cohen, G. G. Parker, and D. J. Harding, “Lidar Remote Sensing for Ecosystem Studies,” *en, BioScience*, vol. 52, no. 1, p. 19, 2002, ISSN: 0006-3568. DOI: 10.1641/0006-3568(2002)052[0019:LRSFES]2.0.CO;2.
- [35] C. Mallet and F. Bretar, “Full-waveform topographic lidar: State-of-the-art,” *en, ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 64, no. 1, pp. 1–16, Jan. 2009, ISSN: 09242716. DOI: 10.1016/j.isprsjprs.2008.09.007.

- [36] M. Maltamo, E. Næsset, and J. Vauhkonen, Eds., *Forestry Applications of Airborne Laser Scanning: Concepts and Case Studies* (Managing Forest Ecosystems), en. Dordrecht: Springer Netherlands, 2014, vol. 27, ISBN: 9789401786621 9789401786638. DOI: 10.1007/978-94-017-8663-8.
- [37] X. Li, C. Liu, Z. Wang, X. Xie, D. Li, and L. Xu, “Airborne LiDAR: state-of-the-art of system design, technology and application,” *Measurement Science and Technology*, vol. 32, no. 3, p. 032 002, Mar. 2021, ISSN: 0957-0233, 1361-6501. DOI: 10.1088/1361-6501/abc867.
- [38] J. Shan and C. K. Toth, Eds., *Topographic Laser Ranging and Scanning: Principles and Processing*, en, 2nd ed. Second edition. | Boca Raton : Taylor & Francis, CRC Press, 2018.: CRC Press, Feb. 2018, ISBN: 9781315154381. DOI: 10.1201/9781315154381.
- [39] J. Muhojoki, T. Hakala, A. Kukko, H. Kaartinen, and J. Hyypä, “Comparing positioning accuracy of mobile laser scanning systems under a forest canopy,” en, *Science of Remote Sensing*, vol. 9, p. 100 121, Jun. 2024, ISSN: 26660172. DOI: 10.1016/j.srs.2024.100121.
- [40] P. S. Thenkabail, *Remote Sensing Handbook, Volume IV: Forests, Biodiversity, Ecology, LULC, and Carbon*, en, 2nd ed. Boca Raton: CRC Press, Oct. 2024, ISBN: 9781003541172. DOI: 10.1201/9781003541172.
- [41] P. Wilkes, M. Disney, J. Armston, *et al.*, “*TLS2trees* : A scalable tree segmentation pipeline for <span style="font-variant: small-caps;">TLS</span> data,” en, *Methods in Ecology and Evolution*, vol. 14, no. 12, pp. 3083–3099, Dec. 2023, ISSN: 2041-210X, 2041-210X. DOI: 10.1111/2041-210X.14233.
- [42] M. Wielgosz, S. Puliti, B. Xiang, K. Schindler, and R. Astrup, “SegmentAnyTree: A sensor and platform agnostic deep learning model for tree segmentation using laser scanning data,” en, *Remote Sensing of Environment*, vol. 313, p. 114 367, Nov. 2024, ISSN: 00344257. DOI: 10.1016/j.rse.2024.114367.
- [43] J. Henrich, J. van Delden, D. Seidel, T. Kneib, and A. Ecker, “TreeLearn: A deep learning method for segmenting individual trees from ground-based LiDAR forest point clouds,” 2023. DOI: 10.48550/ARXIV.2309.08471.
- [44] L. Ruoppa, O. Oinonen, J. Taher, *et al.*, *Unsupervised deep learning for semantic segmentation of multispectral LiDAR forest point clouds*, 2025. DOI: 10.48550/ARXIV.2502.06227.
- [45] Y. Bai, J.-B. Durand, G. Vincent, and F. Forbes, “Semantic segmentation of sparse irregular point clouds for leaf/wood discrimination,” 2023. DOI: 10.48550/ARXIV.2305.16963.
- [46] M. Wielgosz, S. Puliti, P. Wilkes, and R. Astrup, “Point2Tree(P2T)—Framework for Parameter Tuning of Semantic and Instance Segmentation Used with Mobile Laser Scanning Data in Coniferous Forest,” en, *Remote Sensing*, vol. 15, no. 15, p. 3737, Jul. 2023, ISSN: 2072-4292. DOI: 10.3390/rs15153737.
- [47] K. Wolk and M. S. Tatara, “A Review of Semantic Segmentation and Instance Segmentation Techniques in Forestry Using LiDAR and Imagery Data,”

- en, *Electronics*, vol. 13, no. 20, p. 4139, Oct. 2024, ISSN: 2079-9292. DOI: 10.3390/electronics13204139.
- [48] A. Bornand, M. Abegg, F. Morsdorf, and N. Rehus, “Completing 3D point clouds of individual trees using deep learning,” en, *Methods in Ecology and Evolution*, vol. 15, no. 11, pp. 2010–2023, Nov. 2024, ISSN: 2041-210X, 2041-210X. DOI: 10.1111/2041-210X.14412.
- [49] S. Ma, Y. Chen, Z. Li, J. Chen, and X. Zhong, “Improved Cylinder-Based Tree Trunk Detection in LiDAR Point Clouds for Forestry Applications,” en, *Sensors*, vol. 25, no. 3, p. 714, Jan. 2025, ISSN: 1424-8220. DOI: 10.3390/s25030714.
- [50] Z. Hui, L. Lin, S. Jin, Y. Xia, and Y. Y. Ziggah, “A Reliable DBH Estimation Method Using Terrestrial LiDAR Points through Polar Coordinate Transformation and Progressive Outlier Removal,” en, *Forests*, vol. 15, no. 6, p. 1031, Jun. 2024, ISSN: 1999-4907. DOI: 10.3390/f15061031.
- [51] L. Breiman, “Random Forests,” en, *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001, ISSN: 1573-0565. DOI: 10.1023/A:1010933404324.
- [52] M. Hearst, S. Dumais, E. Osuna, J. Platt, and B. Scholkopf, “Support vector machines,” *IEEE Intelligent Systems and their Applications*, vol. 13, no. 4, pp. 18–28, Jul. 1998, ISSN: 1094-7167. DOI: 10.1109/5254.708428.
- [53] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, “Multi-layer perceptron and neural networks,” *WSEAS Transactions on Circuits and Systems*, vol. 8, no. 7, pp. 579–588, 2009. DOI: 10.5555/1639537.1639542.
- [54] T. Chen and C. Guestrin, “XGBoost: A Scalable Tree Boosting System,” en, in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco California USA: ACM, Aug. 2016, pp. 785–794, ISBN: 9781450342322. DOI: 10.1145/2939672.2939785.
- [55] T. Hackel, J. D. Wegner, and K. Schindler, “Contour Detection in Unstructured 3D Point Clouds,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 1610–1618, ISBN: 9781467388511. DOI: 10.1109/CVPR.2016.178.
- [56] C. Cabo, C. Ordóñez, F. Sánchez-Lasheras, J. Roca-Pardiñas, and J. De Cos-Juez, “Multiscale Supervised Classification of Point Clouds with Urban and Forest Applications,” en, *Sensors*, vol. 19, no. 20, p. 4523, Oct. 2019, ISSN: 1424-8220. DOI: 10.3390/s19204523.
- [57] P. Raumonon, M. Kaasalainen, M. Åkerblom, *et al.*, “Fast Automatic Precision Tree Models from Terrestrial Laser Scanner Data,” en, *Remote Sensing*, vol. 5, no. 2, pp. 491–520, Jan. 2013, ISSN: 2072-4292. DOI: 10.3390/rs5020491.
- [58] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, “Deep Learning for 3D Point Clouds: A Survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 12, pp. 4338–4364, Dec. 2021, ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2020.3005434.

- [59] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA: IEEE, Jun. 2016, pp. 779–788, ISBN: 9781467388511. DOI: 10.1109/CVPR.2016.91.
- [60] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, 2015. DOI: 10.48550/ARXIV.1512.03385.
- [61] D. Maturana and S. Scherer, “VoxNet: A 3D Convolutional Neural Network for real-time object recognition,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Hamburg, Germany: IEEE, Sep. 2015, pp. 922–928, ISBN: 9781479999941. DOI: 10.1109/IROS.2015.7353481.
- [62] C. Choy, J. Gwak, and S. Savarese, *4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks*, 2019. DOI: 10.48550/ARXIV.1904.08755.
- [63] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*, 2016. DOI: 10.48550/ARXIV.1612.00593.
- [64] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space,” 2017. DOI: 10.48550/ARXIV.1706.02413.
- [65] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, *KPConv: Flexible and Deformable Convolution for Point Clouds*, arXiv:1904.08889, Aug. 2019. DOI: 10.48550/arXiv.1904.08889.
- [66] H. Zhao, L. Jiang, J. Jia, P. Torr, and V. Koltun, “Point Transformer,” in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, QC, Canada: IEEE, Oct. 2021, pp. 16 239–16 248, ISBN: 9781665428125. DOI: 10.1109/ICCV48922.2021.01595.
- [67] K. Zięba-Kulawik, K. Skoczylas, P. Wężyk, J. Teller, A. Mustafa, and H. Omrani, “Monitoring of urban forests using 3D spatial indices based on LiDAR point clouds and voxel approach,” en, *Urban Forestry & Urban Greening*, vol. 65, p. 127 324, Nov. 2021, ISSN: 16188667. DOI: 10.1016/j.ufug.2021.127324.
- [68] R. Balestrieri, M. Ibrahim, V. Sobal, *et al.*, *A Cookbook of Self-Supervised Learning*, 2023. DOI: 10.48550/ARXIV.2304.12210.
- [69] N. Ding, Y. Qin, G. Yang, *et al.*, “Parameter-efficient fine-tuning of large-scale pre-trained language models,” en, *Nature Machine Intelligence*, vol. 5, no. 3, pp. 220–235, Mar. 2023, ISSN: 2522-5839. DOI: 10.1038/s42256-023-00626-4.
- [70] N. Houlsby, A. Giurghi, S. Jastrzebski, *et al.*, *Parameter-Efficient Transfer Learning for NLP*, 2019. DOI: 10.48550/ARXIV.1902.00751.
- [71] E. J. Hu, Y. Shen, P. Wallis, *et al.*, *LoRA: Low-Rank Adaptation of Large Language Models*, 2021. DOI: 10.48550/ARXIV.2106.09685.
- [72] B. Lester, R. Al-Rfou, and N. Constant, *The Power of Scale for Parameter-Efficient Prompt Tuning*, 2021. DOI: 10.48550/ARXIV.2104.08691.
- [73] *Papers with Code - 3D Point Cloud Classification*, <https://paperswithcode.com/task/3d-point-cloud-classification>.

- [74] D. Liang, T. Feng, X. Zhou, Y. Zhang, Z. Zou, and X. Bai, *Parameter-Efficient Fine-Tuning in Spectral Domain for Point Cloud Learning*, 2024. DOI: 10.48550/ARXIV.2410.08114.
- [75] M. Bryson, F. Wang, and J. Allworth, “Using Synthetic Tree Data in Deep Learning-Based Tree Segmentation Using LiDAR Point Clouds,” en, *Remote Sensing*, vol. 15, no. 9, p. 2380, May 2023, ISSN: 2072-4292. DOI: 10.3390/rs15092380.
- [76] A. Doosthosseini, D. Sommer, and H. Kirchner, *SynForest – Synthetic Generation of LiDAR Data in Forests – News*, <https://paperswithcode.com/task/3d-point-cloud-classification>, 2023.
- [77] S. Puliti, E. R. Lines, J. Müllerová, *et al.*, *Benchmarking tree species classification from proximally-sensed laser scanning data: introducing the FOR-species20K dataset*, 2024. DOI: 10.48550/ARXIV.2408.06507.
- [78] J. Gomes, I. Campos, E. Bagnaschi, *et al.*, “Enabling rootless Linux Containers in multi-user environments: the udocker tool,” 2017. DOI: 10.48550/ARXIV.1711.01758.
- [79] SchedMD, *Slurm Workload Manager - Documentation*, <https://slurm.schedmd.com/documentation.html>, 2025.
- [80] L. Winiwarter, A. M. Esmorís Pena, H. Weiser, *et al.*, “Virtual laser scanning with HELIOS++: A novel take on ray tracing-based simulation of topographic full-waveform 3D laser scanning,” *Remote Sensing of Environment*, vol. 269, p. 112 772, Feb. 2022, ISSN: 00344257. DOI: 10.1016/j.rse.2021.112772.
- [81] N. Ravi, J. Reizenstein, D. Novotny, *et al.*, *Accelerating 3D Deep Learning with PyTorch3D*, 2020. DOI: 10.48550/ARXIV.2007.08501.
- [82] C. Jim, *Conda-Pack — conda-pack 0.8.1 documentation*, <https://conda.github.io/conda-pack/>, 2017.
- [83] G. Chen, *PointGPT repository*, <https://github.com/CGuangyan-BIT/PointGPT>, Jun. 2025.
- [84] Y. Zha, J. Wang, T. Dai, B. Chen, Z. Wang, and S.-T. Xia, “Instance-aware Dynamic Prompt Tuning for Pre-trained Point Cloud Models,” in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, Paris, France: IEEE, Oct. 2023, pp. 14 115–14 124, ISBN: 9798350307184. DOI: 10.1109/ICCV51070.2023.01302.
- [85] X. Zhou, D. Liang, W. Xu, *et al.*, “Dynamic Adapter Meets Prompt Tuning: Parameter-Efficient Transfer Learning for Point Cloud Analysis,” in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA: IEEE, Jun. 2024, pp. 14 707–14 717, ISBN: 9798350353006. DOI: 10.1109/CVPR52733.2024.01393.
- [86] *Canton Neuchâtel: LiDAR Potree Viewer*, <https://sitn.ne.ch/lidar/>.
- [87] L. Jiang, H. Zhao, S. Shi, S. Liu, C.-W. Fu, and J. Jia, *PointGroup: Dual-Set Point Grouping for 3D Instance Segmentation*, 2020. DOI: 10.48550/ARXIV.2004.01658.



- [88] NVIDIA, *Minkowski Engine*, <https://github.com/NVIDIA/MinkowskiEngine>, Jul. 2025.
- [89] R. Adams and L. Bischof, “Seeded region growing,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 6, pp. 641–647, Jun. 1994, ISSN: 01628828. DOI: 10.1109/34.295913.
- [90] D. Comaniciu and P. Meer, “Mean shift: a robust approach toward feature space analysis,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, May 2002, ISSN: 01628828. DOI: 10.1109/34.1000236.
- [91] T. Stegmüller, B. Bozorgtabar, A. Spahr, and J.-P. Thiran, *ScoreNet: Learning Non-Uniform Attention and Augmentation for Transformer-Based Histopathological Image Classification*, 2022. DOI: 10.48550/ARXIV.2202.07570.
- [92] H. Henniger, A. Huth, K. Frank, and F. J. Bohn, “Creating virtual forests around the globe and analysing their state space,” en, *Ecological Modelling*, vol. 483, p. 110 404, Sep. 2023, ISSN: 03043800. DOI: 10.1016/j.ecolmodel.2023.110404.
- [93] H. Weiser, J. Schäfer, L. Winiwarter, N. Krašovec, F. E. Fassnacht, and B. Höfle, “Individual tree point clouds and tree measurements from multi-platform laser scanning in German forests,” en, *Earth System Science Data*, vol. 14, no. 7, pp. 2989–3012, Jul. 2022, ISSN: 1866-3516. DOI: 10.5194/essd-14-2989-2022.
- [94] J. Liu, D. Wang, H. Gong, C. Wang, J. Zhu, and D. Wang, *Advancing the Understanding of Fine-Grained 3D Forest Structures using Digital Cousins and Simulation-to-Reality: Methods and Datasets*, 2025. DOI: 10.48550/ARXIV.2501.03637.
- [95] X. Liang, H. Qi, X. Deng, *et al.*, “ForestSemantic: a dataset for semantic learning of forest from close-range sensing,” en, *Geo-spatial Information Science*, vol. 28, no. 1, pp. 185–211, Jan. 2025, ISSN: 1009-5020, 1993-5153. DOI: 10.1080/10095020.2024.2313325.
- [96] S. Puliti, G. Pearse, P. Surový, *et al.*, *FOR-instance: a UAV laser scanning benchmark dataset for semantic and instance segmentation of individual trees*, 2023. DOI: 10.48550/ARXIV.2309.01279.
- [97] Tockner Andreas, J.-M. Burmeister, Gollob Christoph, Ritter Tim, and Nothdurft Arne, *LAUTx - Individual Tree Point Clouds from Austrian forest Inventory plots*, en, May 2022. DOI: 10.5281/ZENODO.6560111.
- [98] K. Calders, H. Verbeeck, A. Burt, *et al.*, *Terrestrial laser scanning data Wytham Woods: individual trees and quantitative structure models (QSMs)*, en, Nov. 2022. DOI: 10.5281/ZENODO.7307955.
- [99] J. Frey and Z. Schindler, *DetailView repository*, <https://github.com/JulFrey/DetailView>, Jun. 2025.
- [100] A. Goyal, H. Law, B. Liu, A. Newell, and J. Deng, *Revisiting Point Cloud Shape Classification with a Simple and Effective Baseline*, 2021. DOI: 10.48550/ARXIV.2106.05304.

- [101] M. Hussain, *YOLOv5, YOLOv8 and YOLOv10: The Go-To Detectors for Real-time Vision*, 2024. DOI: 10.48550/ARXIV.2407.02988.
- [102] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, *Scaling Vision Transformers*, 2021. DOI: 10.48550/ARXIV.2106.04560.
- [103] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, *Class-Balanced Loss Based on Effective Number of Samples*, 2019. DOI: 10.48550/ARXIV.1901.05555.
- [104] J.-H. Lee, D. Yoon, B. Ji, K. Kim, and S. Hwang, *Rethinking Evaluation Protocols of Visual Representations Learned via Self-supervised Learning*, 2023. DOI: 10.48550/ARXIV.2304.03456.
- [105] M. Marks, M. Knott, N. Kondapaneni, *et al.*, *A Closer Look at Benchmarking Self-Supervised Pre-training with Image Classification*, 2024. DOI: 10.48550/ARXIV.2407.12210.
- [106] D. Variš and O. Bojar, “Sequence Length is a Domain: Length-based Overfitting in Transformer Models,” 2021. DOI: 10.48550/ARXIV.2109.07276.
- [107] M. Caron, H. Touvron, I. Misra, *et al.*, *Emerging Properties in Self-Supervised Vision Transformers*, 2021. DOI: 10.48550/ARXIV.2104.14294.





## List of Figures

2.1	Tree species distribution of swiss forests based on stem count (thousands) [1]. . . . .	6
2.2	Principle of multiple return LiDAR systems. Figure taken from [37].	7
2.3	Overview of the main tasks applied to point clouds for forest inventory. Figure taken from [15]. . . . .	9
2.4	Number of papers using machine learning and deep learning methods on TLS forest point clouds over time. Figure from [15]. . . . .	10
2.5	Number of papers per task (section 2.3) on TLS forest point clouds over time. Figure from [15]. . . . .	10
2.6	A voxelized point cloud of a tree. Figure taken from [67]. . . . .	11
3.1	Flowchart of the research process. . . . .	16
3.2	Project schedule of this master's thesis. . . . .	17
3.3	Processing of a input point cloud of a tree of the pre-training dataset in PointGPT. . . . .	21
3.4	Flow of the input tokens in the PointGPT transformer. . . . .	24
3.5	An example of a segmented ALS point cloud from Canton Neuchâtel used for the pre-training dataset. . . . .	27
3.6	Distribution of points per instance for tile 25655001212500. . . . .	28
3.7	The tree synthesis process in SimpleSynthTree. Figure taken from [75]. . . . .	29
3.8	The Synforest simulation process. Figure taken from [76]. . . . .	30
3.9	Tree species distribution in the post-pre-train dataset. . . . .	31
3.10	Distribution of points per instance for the post-pre-train dataset. . . . .	32
3.11	Summary chart of the FOR-Species20K dataset. Figure taken from [77]. . . . .	34
3.12	Distribution of points per instance for the benchmark dataset. . . . .	35
3.13	Loss curves of DetailView on FOR-Species20K. . . . .	36
3.14	Species overlap between post-pre-training and fine-tuning datasets. . . . .	37
4.1	Speedup investigation for parallel vs. sequential loading applied on .laz point cloud files. . . . .	41
4.2	The applied point cloud transformations for augmentation purposes. . . . .	45
4.3	Comparison of class weighting strategies for the FOR-Species20K dataset. The black dots are the effective class counts of the 33 classes in the dataset. . . . .	46

4.4	Comparison of the training and validation splits for the fine-tuning dataset, with values expressed as percentages of the dataset. . . .	47
4.5	Pre-training: reconstruction loss on the training split. . . . .	50
4.6	Pre-training: reconstruction loss on the validation split. . . . .	50
4.7	Pre-training: linear probing top 1. . . . .	51
4.8	Pre-training: linear probing top 5. . . . .	51
4.9	Pre-training gridsearch: reconstruction loss on the validation split.	52
4.10	Pre-training gridsearch: linear probing top 1. . . . .	53
4.11	Pre-training gridsearch: linear probing top 5. . . . .	53
4.12	Pre-training trials: reconstruction loss on the validation split. . . .	55
4.13	Pre-training trials: linear probing top 1. . . . .	55
4.14	Divergence issue: reconstruction loss comparison between identical runs with and without AMP. . . . .	57
4.15	Divergence issue: gradients norm. . . . .	57
4.16	Divergence issue: AMP scale. . . . .	57
4.17	Post-pre-training: classification accuracy on the validation split. .	58
4.18	Post-pre-training: linear probing top 1. . . . .	58
4.19	Post-pre-training: linear probing top 5. . . . .	59
4.20	Fine-tuning: classification accuracy on the validation split. . . . .	60
4.21	Fine-tuning: F1 score on the validation split. . . . .	60
4.22	Fine-tuning: classification loss on the validation split. . . . .	62
4.23	Fine-tuning: classification accuracy on the training split. . . . .	62
4.24	Fine-tuning with adapters (PEFT): classification accuracy on the validation split. . . . .	63
4.25	Fine-tuning with adapters (PEFT): F1 score on the validation split.	63
4.26	Fine-tuning: classification accuracy on the validation split for models with different token sequence lengths. . . . .	65
4.27	Fine-tuning: classification accuracy on the validation split for different model S configurations. . . . .	66
4.28	Fine-tuning: F1 score on the validation split for different model S configurations. . . . .	66
4.29	Best model: confusion matrix. . . . .	68
4.30	Best model: accuracy by species. . . . .	69
4.31	Best model: accuracy by data type. . . . .	69
4.32	Best model: accuracy by tree height. . . . .	69

## List of Tables

2.1	Tree species distribution of swiss forests based on stem count (thousands) [1]. . . . .	6
2.2	Comparison of LiDAR platform characteristics. From [40], p.54, adapted. . . . .	8
3.1	Number of tree instances across point cloud tiles with filtering threshold of 1024 points per instance. . . . .	28
3.2	SimpleSynthTree parameters used for synthetic tree generation complementing the Neuchâtel dataset. . . . .	29
3.3	Overview of the literature datasets used to expand the pre-training dataset. . . . .	32
3.4	Performance of top-performing models on the FOR-species20K benchmark dataset. . . . .	35
3.5	Overview of datasets used in this study. . . . .	37
4.1	Training speed comparison between full precision and mixed precision for distributed PointGPT pre-training using 4 GPUs (batch size 128 per GPU, 512 total). . . . .	42
4.2	Parameter counts across model configurations. . . . .	43
4.3	Normalized class weight ranges for different balancing strategies applied to the FOR-Species20K dataset. . . . .	46
4.4	SSL linear probing performance ranges across vision task categories and datasets. From [105], summarized . . . . .	48
4.5	Pre-training configurations. . . . .	50
4.6	Gridsearch parameter space configuration. . . . .	54
4.7	Post-pre-training configurations. . . . .	58
4.8	Fine-tuning configurations. . . . .	61
4.9	Performance comparison between TreeGPT and leading methods. TreeGPT results are reported on the validation split of this work's dataset, while benchmark results are on the official FOR-Species20K test split, limiting direct comparability. . . . .	68



# Listings

- 3.1 Tree species assignment algorithm in Python pseudocode for the post-pre-train dataset. . . . . 31
- 4.1 Preprocessing pipeline for point cloud normalization and sampling in Python pseudocode. . . . . 40



## Glossary

LiDAR	Light Detection and Ranging. An active remote sensing technology that measures distances by emitting laser pulses and calculating the time-of-flight for reflected signals to return to the sensor
point cloud	A collection of data points in three-dimensional space, where each point represents a specific location defined by X, Y, and Z coordinates, often including additional attributes such as intensity and color
SSL	Self-Supervised Learning. A machine learning technique that enables training models on large quantities of unlabeled data through various pre-text tasks, allowing models to acquire general knowledge that can be transferred to downstream tasks
ALS	Airborne Laser Scanning. LiDAR systems that operate above the forest canopy, typically mounted on manned aircraft flying at altitudes of hundreds of meters to several kilometers
TLS	Terrestrial Laser Scanning. Stationary ground-based LiDAR systems that provide the highest point densities and precision, operating from fixed positions under the forest canopy
ULS	Unmanned Laser Scanning. LiDAR systems carried by unmanned aerial vehicles (UAVs/drones) that can operate either under or above the forest canopy

MLS	Mobile Laser Scanning. LiDAR systems that move under the forest canopy, including ground-based systems mounted on vehicles or handheld Personal Laser Scanners
FPS	Farthest Point Sampling. A deterministic down-sampling algorithm that iteratively selects points to maximize spatial coverage by progressively choosing points that maintain maximum distance from all previously selected points
PointGPT	A transformer architecture that extends Generative Pre-trained Transformers to point clouds, addressing disorder properties, low information density, and gaps between generation and downstream tasks through auto-regressive generation
PointGST	Point cloud Graph Spectral Tuning. A parameter-efficient fine-tuning method that addresses token confusion in pre-trained point cloud models by shifting the adaptation process from spatial coordinates to the spectral domain through lightweight Point Cloud Spectral Adapters
transformer	A neural network architecture based on attention mechanisms that has become dominant in deep learning, particularly successful in natural language processing and increasingly applied to computer vision tasks
AMP	Automatic Mixed Precision. A technique that automatically scales operations between float32 and float16 precision to reduce computational load while maintaining model performance
PEFT	Parameter-Efficient Fine-Tuning. Adaptation techniques that modify only a small subset of parameters in pre-trained models rather than updating all model weights during downstream task training



---

RDP	Relative Direction Prompts. Directional information calculated from spatial relationships between point group centers, used in PointGPT to provide the model with information about group order and direction
APE	Absolute Positional Encodings. Sinusoidal positional encodings added to point group centers to provide the model with global structure information about spatial ordering
CD	Chamfer Distance. A metric that measures similarity between two point clouds by quantifying how well they align, calculated by finding the closest point correspondences between clouds
DBH	Diameter at Breast Height. A standard forestry measurement of tree trunk diameter taken at 1.3 meters above ground level, commonly used for forest inventory and biomass estimation
KNN	K-Nearest Neighbors. An algorithm that identifies the k closest points to a given point based on a distance metric, commonly used for point cloud neighborhood construction
Morton code	A method for converting multi-dimensional coordinates into a single number that preserves spatial locality by interleaving the binary representations of coordinate values
SOS	Start of Sequence. A special token used during pre-training to provide generative context at the beginning of token sequences in transformer architectures
CLS	Classification token. A learnable token introduced to pre-trained models for downstream classification tasks, designed to encapsulate global properties of the input useful for classification

GFT	Graph Fourier Transform. A mathematical transformation that decomposes graph signals into orthogonal frequency components, enabling spectral analysis of data defined on graph structures
PCSA	Point Cloud Spectral Adapters. Lightweight adaptation modules that operate in the spectral domain to fine-tune pre-trained point cloud models while keeping the majority of parameters frozen
TreeGPT	The final model developed in this work, representing the best-performing PointGPT configuration (Model L) specifically adapted for tree species classification from LiDAR point clouds
FOR-Species20K	A benchmark dataset comprising 20,158 individual tree point clouds representing 33 species, designed specifically for tree species classification from proximal laser scanning data
OOM	Out-of-Memory. An error that occurs when a program attempts to use more memory than is available, commonly encountered in deep learning when processing large batches or high-resolution data that exceed GPU memory capacity