OUKA
The University of Osaka
Institutional Knowledge Archive

Title	PlantPose: Plant Skeleton Estimation in 2D and 3D
Author(s)	劉, 新鵬
Citation	大阪大学, 2025, 博士論文
Version Type	VoR
URL	https://doi.org/10.18910/105269
rights	
Note	

The University of Osaka Institutional Knowledge Archive : OUKA

https://ir.library.osaka-u.ac.jp/

The University of Osaka

PlantPose: Plant Skeleton Estimation in 2D and 3D

Submitted to
Graduate School of Information Science and Technology
The University of Osaka

September 2025

Xinpeng LIU

List of Publications

Journal

1. Naoki Asada*, Xinpeng Liu*, Kanyu Xu*, Ryohei Miyakawa, Yang Yang, Hiroaki Santo, Yosuke Toda, and Fumio Okura: "LeafGen: Leaf Instance Segmentation without Manual Annotation via Structure-aware Image Generation," Plant Phenomics.

2. Xinpeng Liu, Kanyu Xu, Risa Shinoda, Hiroaki Santo, and Fumio Okura: "Masks-to-Skeleton: Multi-view mask-based tree skeleton extraction with 3D Gaussian splatting," Sensors.

3. Xinpeng Liu, Hiroaki Santo, Yosuke Toda, and Fumio Okura: "PlantPose: Universal Plant Skeleton Estimation via Tree-constrained Graph Generation," International Journal of Computer Vision (IJCV) (Major Revision).

International conference

1. Xinpeng Liu*, Zeyi Huang*, Fumio Okura, and Yasuyuki Matsushita: "HoGS: Unified Near and Far Object Reconstruction via Homogeneous Gaussian Splatting," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2025.

2. Xinpeng Liu, Hiroaki Santo, Yosuke Toda, and Fumio Okura: "TreeFormer: Single-view Plant Skeleton Estimation via Tree-constrained Graph Generation," IEEE Winter Conference on Applications of Computer Vision (WACV), 2025.

3. Junhao Xing*, Ryohei Miyakawa*, Yang Yang, Xinpeng Liu, Risa Shinoda, Hiroaki Santo, Yosuke Toda, and Fumio Okura: "Zero-shot Hierarchical Plant Segmentation via Foundation Segmentation Models and Text-to-image Attention," IEEE Winter Conference on Applications of Computer Vision (WACV), 2026.

International workshop

1. Xinpeng Liu: "PlantPose: Plant Skeleton Estimation in 2D and 3D," Machine Learning for Cyber-Agricultural Systems (MLCAS), 2025.

Domestic conference

1. Xinpeng Liu, Kanyu Xu, Risa Shinoda, Hiroaki Santo, and Fumio Okura: "Masks-to-Skeleton: Multi-view mask-based tree skeleton extraction with 3D Gaussian splatting," Meeting on Image Recognition and Understanding (MIRU), 2025.

2. Xinpeng Liu, Hiroaki Santo, Yosuke TODA, and Fumio Okura: "TreeFormer: Hard-constrained Image-to-graph Generation," Meeting on Image Recognition and Understanding (MIRU), 2023.

Acknowlegements

My Ph.D. journey started halfway through the COVID-19 period, and it has been a tough time since then. The first year was largely remote, and when campus life resumed, I found that several senior Ph.D. students had already graduated; the lab environment had changed, and I had to grow into new responsibilities. Looking back, I am deeply grateful to those who supported and guided me through these years.

First and foremost, I would like to express my sincere gratitude to my supervisor, Prof. Fumio Okura. He has been patient, kind, and genuinely hands-on in both research and daily life. For a student new to CV, his mentorship was indispensable, from shaping research directions to the smallest details of academic practice. I still remember how he would even help refine my slides and teach me how to align elements properly to make clear figures. His generosity with time and his careful guidance made this thesis possible.

I am also deeply grateful to Prof. Heng Guo, who has been my closest friend in Japan and a constant source of academic support. He pushed me at crucial moments, especially on graduation timelines, and his encouragement and sharp feedback helped me accelerate my research progress. And I would like to thank Prof. Yasuyuki Matsushita for the high standards he set for our laboratory. His presence instilled a spirit of self-discipline and rigor. Although his expectations were strict, they were always principled and valuable. Also, I would like to thank Dr. Xu Cao, Dr. Feiran Li, and Mr. Zhihao Liu, whose advice and support were invaluable in both career development and daily life. They prompted me to think more clearly, work more carefully, and take responsibility for my own growth.

My sincere gratitude extends to my collaborators, and I would like to thank them for their invaluable advice and help on my research projects and this thesis: Mr. Yosuke Toda for my first project (Chapter 3); Prof. Hiroaki Santo for my first two projects (Chapters 3 and 4); Prof. Risa Shinoda for my last project (Chapter 4).

I would also like to express my sincere thanks to the members of my laboratory, Dr. Kensuke Uchida, Dr. Lilika Makabe, Dr. Weng Ian Chan, Dr. Yang Yang, Dr. Zeyi Huang, Dr. Junhao Xing, Dr. Ryouta Fujii, Dr. Siwei Zhou, Mr. Seinosuke Sakata, Mr. Yuki Tsuino, Mr. Taiga Hashida, Mr. Koki Fukai, Mr. Leyang Xu, Ms. Dongni Mao, Mr. Kanyu Xu, Mr. Fubuki Imai, Mr. Keiichiro Iwamoto, Mr. Kouki Nakahama, Mr. Ryoma Bando, Mr. Fumiya Mori, Mr. Naoya Kanzaki, Mr. Jun Ikeda, Mr. Taichi Oyama, Mr. Natsuki Takebayashi, Mr. Naoki Deguchi, Mr. Genki Yoshimura, and Mr. Yuki Kubota.

I would like to express my gratitude to Ms. Mihoko Kaneda, Rie Maeda, and the Graduate School of Information Science and Technology officers for their support in my daily life at The University of Osaka.

Finally, my heartfelt gratitude goes to my family. I especially thank my wife: we married in the final year of my Ph.D., and her companionship, encouragement, and patience sustained me through the most demanding period. I owe the completion of this thesis to her unwavering support. Their unconditional love and trust have always been my foundation, and this thesis is dedicated to them.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration, except where specifically indicated in the text.

Xinpeng Liu

Sep, 2025

Abstract of thesis entitled

"PlantPose: Plant Skeleton Estimation in 2D and 3D"

Submitted by

Xinpeng Liu

for the degree of Doctor of Philosophy

at The University of Osaka

in Sep, 2025

Accurately extracting plant skeletal architecture is a fundamental component for high-throughput phenotyping, growth modelling, and agricultural robotics. A plant skeleton is represented as a graph whose nodes lie on branches and whose edges encode connectivity between node pairs. Plant skeleton estimation is typically approached in two main domains: (i) single-view 2D and (ii) multi-view 3D skeleton estimation. Unlike human poses with a fixed and well-defined joint structure, plant skeletons can contain an unknown number of joints and must follow a tree topology without loops or disconnections. Additionally, in 2D estimation, models trained on one type of plant often fail to generalize to other species or imaging conditions. In 3D, existing methods rely on point clouds from light detection and ranging (LiDAR) or multi-view stereo (MVS), which frequently suffer from uneven density and occlusion, resulting in broken or fragmented connectivity, violating the desired tree structure. To address these challenges, this dissertation introduces **PlantPose**, a unified framework that (1) constructs a diverse and large-scale dataset to enable generalizable 2D skeleton estimation across different plant types, (2) produces tree-constrained skeletons in an end-to-end manner, and (3) proposes a mask-guided 3D optimization method that avoids point cloud fragmentation by directly aligning 3D graphs with multi-view masks.

First, we assemble the *PlantPose dataset*, the largest and most diverse benchmark for plant skeleton estimation, comprising ten subsets including lab seedlings, greenhouse crops, orchard trees, web-crawled photographs, hand-drawn sketches, and

photorealistic synthetics. In the branch-level raw annotations, branch segments are labeled independently as sequences of nodes and edges, without guaranteed cross-branch connectivity, leaving many disconnected components. We therefore propose a graph-recovery pipeline that merges disconnected branch segments into a single tree using a minimum spanning tree (MST). The resulting supervision for each image consists of pixel-level joint locations and an adjacency matrix that encodes a tree topology.

Second, we present *PlantPose2D*, an end-to-end model that predicts graph nodes and edges from a single image while enforcing tree constraints during learning. The central component is a Selective Feature Suppression (SFS) layer. We project the model's unconstrained edge scores at each training step onto an MST defined over the current node set using a non-differentiable solver. SFS layer then suppresses the internal features that create cycles or isolated parts, aligning the network's gradients with the tree structure projection. This reparameterization enforces the constraint directly in the learning loop without requiring a differentiable MST. Leveraging the generalized dataset, PlantPose2D achieves the highest proportion of valid tree-structured skeletons and lower topology-aware error and generalizes well to unseen species, lighting conditions, and drawing styles.

Third, we address the limitations of 3D skeleton estimation from multi-view images. Point-cloud pipelines built on LiDAR or MVS often exhibit uneven density on thin, low-texture branches, which breaks connectivity. We introduce *PlantPose3D*, a mask-guided graph optimization method that reduces reliance on point-cloud quality by aligning a 3D graph's observed masks with refined multi-view masks. We first represent the plant as a graph whose nodes store 3D positions and branch radii, with connectivity encoded by an adjacency matrix. Then, using 3D Gaussian Splatting (3DGS) for differentiable silhouette rendering, we optimize node positions, radii, and adjacency matrix to match the observed masks across views. In addition, we reuse the SFS layer from PlantPose2D to regularize the graph to a tree structure graph during optimization. This method overcomes the limitations of uneven point-cloud density while guaranteeing a tree-structured graph.

Contents

Contents **xi**

List of Figures **xv**

List of Tables **xxiii**

1 Introduction **1**

1.1 Motivation 3

1.2 Thesis Organization 6

1.3 Summary of Contributions 7

2 PlantPose Dataset **9**

2.1 Introduction 9

2.2 Related Work 11

2.2.1 Real-World Datasets 11

2.2.2 Synthetic Datasets 14

2.3 Dataset Construction Method 15

2.3.1 Objectives and Unified Representation 15

2.3.2 Polyline-Based Tree Construction 16

2.3.3 Mask-Based Skeleton Extraction 18

2.4 Datasets 20

2.4.1 PlantPose Real-World Subsets 20

2.4.2 PlantPose Synthetic Subsets 25

2.4.3 Dataset Split 27
2.5 Conclusion 29

3 PlantPose2D **31**
3.1 Introduction 32
3.2 Related Work 34
3.2.1 Plant Skeleton Estimation 35
3.2.2 Graph Generation from Images 35
3.2.3 Constrained Optimization for Neural Networks 36
3.3 Selective Feature Suppression Layer 38
3.3.1 Problem Statement 38
3.3.2 SFS Layer 39
3.4 Analysis of the SFS Layer 42
3.4.1 Motivation 42
3.4.2 Gradient Behavior 42
3.4.3 Formal Derivation 43
3.4.4 Case-by-case Proofs 45
3.5 PlantPose2D Method 48
3.5.1 RelationFormer: A Brief Recap 48
3.5.2 Tree-Constrained Graph Generation 49
3.6 Experiments 50
3.6.1 Baselines 50
3.6.2 Details of the Two-Stage Method 51
3.6.3 Evaluation Metrics 54
3.6.4 Implementation Details 55
3.6.5 Results 58
3.6.6 Performance Analysis 68
3.7 Conclusion 73

4 PlantPose3D **75**

4.1 Introduction . . . 76

4.2 Related Work . . . 78

4.2.1 Three-Dimensional Tree Skeleton Extraction . . . 78

4.2.2 Image-Guided 3D Structure Estimation via Differentiable Rendering . . . 79

4.3 PlantPose3D Method . . . 80

4.3.1 Preprocessing and Initialization . . . 80

4.3.2 Mask-Guided Graph Refinement . . . 84

4.3.3 Structure-Aware Graph Optimization . . . 86

4.3.4 Overall Algorithm . . . 88

4.4 Experiments . . . 88

4.4.1 Implementation Details . . . 88

4.4.2 Datasets . . . 90

4.4.3 Evaluation Metrics . . . 91

4.4.4 Baselines . . . 92

4.4.5 Results on the Synthetic Dataset . . . 94

4.4.6 Results on the Real-World Dataset . . . 96

4.4.7 Ablation Study . . . 99

4.5 Conclusion . . . 101

5 Discussion and Conclusion **103**

5.1 Summary . . . 103

5.2 Future Work . . . 105

References **107**

List of Figures

1.1 **Plant skeleton and its applications.** Left: an example plant with its extracted skeleton. Middle: skeleton represented as a graph, with *nodes* shown in yellow for intermediate positions and in blue for roots, endpoints, or splits, while *edges* are depicted as red lines. Right: representative applications in agriculture, including pruning, robotic crop collection, and plant breeding. 2

1.2 **Limitations of 2D and 3D skeleton estimation.** Top: 2D (left) and 3D (right) skeleton estimation examples. Bottom: representative failure cases: for the 2D case, including non-tree structures, domain-specific bias. For the 3D case, fragmentation is due to sparse point clouds. 3

2.1 **Real-world MIPDB [1] and 3D2cut [2] datasets.** (a) MIPDB [1] maize field images with leaf-wise polylines, enabling leaf-centric phenotyping but lacking an explicit plant-level tree graph. (b) 3D2cut [2] grapevine images with node positions and branch categories; connectivity is inferred via PAF-like affinity fields and weighted shortest paths at inference. 12

2.2 **LVD2021 [3] dataset examples.** From left to right: raw leaf image, binary leaf mask, and binary vein mask. 13

2.3 Illustration of the Mask-to-Skeleton pipeline. From left to right: (a) input binary mask, (b) zoomed region, (c) skeletonized pixel graph, (d) contracted polylines and final graph representation with blue endpoints, yellow intermediate nodes, and green joint nodes. 19

2.4 Atomic structures used for synthetic dataset generation. Three pre-defined initial structures are highlighted in purple. Eight pre-defined rewriting rules are used during the generation. 25

2.5 An example of the rewriting process. Suppose the initial structure is represented as **F0[+A0]F0[-A0]A0**. If a rewrite rule **F→F; A→F[-A]** is applied, *i.e.*, **F** remains unchanged and **A** becomes **F[-A]**, the result of the rewrite process is **F0[+F1[-A1]]F0[-F1[-A1]]F1[-A1]**. The digits in the sequences indicate the number of times the rewrite is applied. 26

2.6 Sample images from the generalized datasets. Annotated graphs are superimposed, where yellow dots represent nodes and red lines represent edges. (a) **Training datasets**: (1) grapevine, (2) LVD2021 leaf image, (3) LVD2021 vein mask, (4) MIPDB, (5) self-captured, (6) web-sourced. (b) **Out-of-domain test datasets**: (7) thickened L-system, (8) LVD with background, (9) tree synthetic, (10) root. The datasets include diverse plant structures and imaging conditions to improve the model's generalization and robustness across in-domain and out-of-domain scenarios. 27

3.1 We propose **PlantPose2D**, a method for single-image plant skeleton estimation combining learning-based graph generators with traditional graph algorithms (*i.e.*, MST). The red lines show the predicted graph edges. Compared to (a) an *unconstrained* graph generator and (b) a naive tree-graph constraint implementation, (c) our method naturally imposes the constraint during the graph generation models' training. Our method can be directly applied to plant science and agricultural applications, such as (d) time-series reconstruction of botanical roots. Furthermore, the PlantPose2D with large-scale training demonstrates strong generalization capabilities across diverse datasets, including significantly different out-of-domain plant species, as shown in (e). 33

3.2 Overview of reparameterization layer that can be easily plugged into off-the-shelf graph generators. Given unconstrained edge predictions by graph generators, our method *projects* it to the closest constrained graph (*i.e.*, tree) using a non-differentiable MST algorithm. Comparing constrained and unconstrained edges, unwanted edge features are selectively suppressed so that the graph becomes a tree. 39

3.3 Network architecture for the first (skeletonization) stage of our two-stage baseline method. 52

3.4 Visual comparisons between a re-implementation of [2] and our two-stage baseline implementation. Our implementation yields better node confidence and edge direction maps, which are the outputs of the first stage of these methods. 53

3.5 Comparisons of graph algorithms (Dijkstra *vs.* MST) and classification labels (binary *vs.* five classes) for the two-stage method. We use the combination of MST and five class labels for the baseline in our experiment. 54

3.6 Visual results for the domain-specific training with the synthetic tree pattern dataset. For the synthetic tree pattern dataset, we translucently overlay the estimated and ground truth edges with red and blue lines, respectively. While all methods accurately detect nodes, only our method accurately predicts the availability of edges from given images compared to the baseline methods. 56

3.7 Visual results for the domain-specific training with the real image datasets: Red lines, yellow dots, and cyan dots indicate the predicted graph edges, nodes, and keypoints (*i.e.*, joints and leaf nodes). Our method accurately estimates the target plant structures compared with baseline methods, demonstrating the applicability of our method for practical uses in plant science and agriculture. 57

3.8 Visual results highlighting generalization ability of domain-specific training using the grapevine dataset for (a) a grapevine tree under a natural background and (b–d) other tree species. 59

3.9 Visual results on the inner-domain datasets. From top to bottom: grapevine, LVD2021 leaf image, LVD2021 vein mask, MIPDB, self-captured, and web-sourced datasets. The two-stage method sometimes produces very few nodes (*e.g.*, on the Self-captured dataset) or even fails to generate any nodes (*e.g.*, on MIPDB and web-sourced dataset). On the grapevine and LVD2021 datasets, it frequently outputs multiple redundant keypoints around actual nodes (visible as dense clusters of blue markers), thereby inflating TOPO scores but also increasing SMD. In contrast, our method achieves a more balanced performance, maintaining stable TOPO scores and low SMD values. 61

3.10 Visual results on the out-of-domain test datasets: thickened L-system, LVD with background, tree synthetic, and root datasets (top to bottom). The unconstrained method generates excessive isolated points and disconnected components, inflating TOPO precision but reducing recall due to poor keypoint coverage. In contrast, our method preserves structural consistency, balancing TOPO precision and recall, and produces coherent tree structures. The root dataset highlights the difficulty of detecting short and thin lateral roots, which remains challenging for all methods. 63

3.11 Visual results on the unlabeled web-sourced dataset, including flowers, rare plants, potted plants, and hand-drawn trees. These results highlight the diversity of test cases and demonstrate our method's generalization ability across different data domains and styles. 66

3.12 Qualitative results on retinal vessel images. PlantPose, trained only on plant skeletons, recovers most major vessel trees after simple preprocessing (contrast adjustment and rotation). 67

3.13 Effect of tree constraint during training and inference. From left to right: (a) ground-truth skeleton, (b) prediction with train-time constraint only (SFS applied during training, inference still uses a 0.5 edge probability threshold), (c) our method with train-time constraint + MST (Ours). The train-time constraint method suffers from instance fragmentation caused by naive thresholding, while our method recovers natural connectivity consistent with the ground truth. 70

3.14 Comparison of our method and test-time MST constraint under generalized training. Both predictions obtain similar TOPO scores ($F1 = 0.94$ vs. 0.96), indicating comparable reachability. However, as highlighted by red arrows, the baseline introduces several spurious or misplaced edges, which are not penalized by TOPO but cause a substantial increase in SMD (almost 2× higher). This example demonstrates that SMD is a more discriminative measure of structural fidelity, capturing global geometric deviations that TOPO may overlook. 71

3.15 Illustration of scalability with respect to node count. From left to right: examples with ∼50, ∼100, ∼200, ∼300, and ∼350 nodes. Simpler graphs (grass-like or single-trunk plants) have sparse nodes, while more complex trees include dense branching structures. 72

4.1 We reconstruct 3D branch structures using refined multi-view masks, visualized as red regions in the adjacent panels of the graph optimization part. Transparent green overlays indicate the predicted tree shape and branch radii. Blue points and red lines represent graph nodes and branch connections, respectively. 77

4.2 Pipeline overview. The provided input images, two complementary mask generators, BiRefNet-HQ for high-resolution foreground masks, and Lang-SAM for language-guided object masks, are fused into refined masks. We use the MST and clustering to obtain an initial graph. Finally, the initial graph is refined through multi-view masks, producing a 3D plant skeleton that is guaranteed to be a valid tree. 81

4.3 Mask fusion pipeline. (**a**) The output of BirefNet-HQ retains fine structures but includes background noise. (**b**) Lang-SAM prediction before (top) and after (bottom) morphological closing, where closing fills small gaps. (**c**) The refined mask is obtained by combining both predictions, applying morphological filtering and connected component analysis to preserve only the main plant region. As a result, small background artifacts (red arrows) and ultra-thin branches (gray arrows) are effectively suppressed. 82

4.4 Datasets used in our experiments. (**a**) Synthetic dataset. Blue boxes highlight L-system-based trees characterized by relatively straight and uniform branches, while pink boxes highlight Mtree-based trees exhibiting more pronounced curvature and significant diameter variation. (**b**) Real-world dataset. Images of urban and roadside trees were captured from the sidewalk side to minimize background clutter and occlusion. . 90

4.5 Effect of duplication removal threshold in AdTree. Visual comparison under varying thresholds illustrates that the default value (0.001) leads to excessive node retention and an over-connected graph, artificially lowering the Chamfer score. A high threshold (0.1) oversimplifies the structure, while a moderate threshold (0.05) yields a result close to the ground truth with better structural fidelity. 93

4.6 **Qualitative results on the L-system-based dataset**. From top to bottom: ground-truth skeletons, our method, AdTree, and Smart-Tree. We overlay green MST-based mask contours to visualize the alignment between predicted radii and ground-truth structure. Our method consistently achieves the most accurate and well-aligned predictions. AdTree tends to overfit straight structures with curved paths, while Smart-Tree suffers from over-segmentation and fails to reconstruct thin branches, despite its high point density. 95

4.7 Qualitative results on the Mtree dataset. From top to bottom: ground-truth skeletons, our method, AdTree, and Smart-Tree. As with the L-system results, green MST-based masks highlight branch regions. While all methods show similar trends, the increased curvature and diameter variation in Mtree trees amplify the weaknesses of AdTree and Smart-Tree. Our method remains the most faithful to the ground truth in both geometry and structure. 96

4.8 Qualitative results on the real-world dataset. From top to bottom: point cloud, refined mask, our method, AdTree, and Smart-Tree. We overlay green MST-based mask contours to visualize the alignment between predicted radii and ground-truth structure. To enhance clarity, the visualization thickness of nodes (blue) and edges (red) is adjusted based on tree species. 97

4.9 Qualitative results on the real-world dataset. From top to bottom: point cloud, refined mask, our method, AdTree, and Smart-Tree. We overlay green MST-based mask contours to visualize the alignment between predicted radii and ground-truth structure. To enhance clarity, the visualization thickness of nodes (blue) and edges (red) is adjusted based on tree species. 98

4.10 Ablation study on loss components. From left to right: refined mask, full model (ours), without $\mathscr{L}_{\text{graph}}$, without the SFS layer, and without both. All results are visualized on the same real-world scene with the predicted skeleton overlaid in green. Removing geometry regularization causes jagged edges and sharp angles, while removing the SFS layer disrupts edge connectivity realism. Removing both leads to severe geometry and topology degradation, confirming the importance of each loss term. 100

List of Tables

2.1 Leaf category distribution of the LVD2021 leaf image and vein mask datasets across train, validation, and test splits. 22

2.2 Plant category distribution of the Web-sourced dataset across train, validation, and test splits. 24

2.3 Generalized dataset composition. Training datasets are marked in white, while out-of-domain test datasets are shaded in gray. Augmentation is applied only to training datasets to improve model generalization. 28

3.1 Case-by-case analyses of our reparameterization layer. For the columns of unconstrained features $\hat{\mathbf{f}}$, constrained prediction $\mathbf{y}$, and the ground truth edge availability $\mathbf{t}$, the table shows the index of the larger element. For example, the column $\hat{\mathbf{f}}$ will be $+$ when the edge feature for the positive edge availability is larger, *i.e.*, $\hat{f^+} > \hat{f^-}$. The column (i, j) displays E^+ or E^- if the MST algorithm modifies the edge availability (in which the rows are also highlighted). For the remaining columns, $\uparrow$ and $\downarrow$ denote each value becoming (relatively) large or small, respectively. 46

3.2 Quantitative comparisons between our re-implementation of [2] and our two-stage baseline implementation. 52

3.3 Quantitative results on domain-specific training. Our method significantly improves both the shape and topology of the predicted graph while enforcing the given constraints. The best scores are highlighted in **bold**. 56

3.4 Results on generalized training. Our method significantly improves both the shape and topology of the predicted graph while enforcing the given constraints, especially for out-of-domain scenarios. 60
3.5 Comparisons of different graph generation models (*i.e.*, RelationFormer [4] and GGT [5]) on the synthetic dataset. 69
3.6 Quantitative results with additional baseline, *train-time constraint*. . . 69

4.1 Quantitative evaluation on synthetic datasets. The Chamfer distance measures geometric similarity. $\Delta|E|$ and $\Delta|N|$ represent the absolute differences in the number of predicted edges and nodes relative to the ground-truth counts (average graph sizes: 29.32 key nodes and 28.32 key edges for the L-system-based dataset; 15.36 key nodes and 14.40 key edges for the Mtree dataset). 94
4.2 Quantitative ablation study on synthetic datasets. Chamfer distance measures geometric similarity, while $\Delta|E|$ and $\Delta|N|$ denote absolute differences in predicted edge and node counts relative to ground-truth. Average ground-truth graph sizes: 29.32 nodes / 28.32 edges for the L-system dataset, 15.36 nodes / 14.40 edges for the Mtree dataset. . . 101

Chapter 1

Introduction

Plant skeleton estimation is defined as the problem of representing the structural skeleton of plants in analogy to human pose [6], while accounting for fundamental differences that make the task more challenging. A plant skeleton is a graph whose nodes lie on the plant's branches, including both internal points and endpoints, and whose edges encode physical connectivity along the plant structure [7, 8] as shown in the left figure in Figure 1.1. Both human and plant skeletons form a single connected tree. The key difference lies in the nature of their joints: human joints are semantically fixed (*e.g.* "left wrist" or "right shoulder") and indicate specific relative positions that guide connectivity. In contrast, plant nodes are non-semantic and variable in number, providing no semantic cues for their connections. Combining an unknown node count and a strict tree graph fundamentally distinguishes plant skeleton estimation from standard image-to-graph prediction [7].

Reliable plant skeletons are valuable in several application areas. In phenotyping [2, 7, 9, 10], they support quantitative traits such as branch count, length, angle, and branching order. In growth modeling, they provide a compact representation of organ structure over time. In agricultural robotics [11–14], plant skeletons guide perception and planning for pruning, thinning, and fruit picking as shown in the right figure in Figure 1.1. For example, pruning has been shown to improve fruit quality in tomato production. Across these scenarios, the central goal is to infer a geometrically faithful

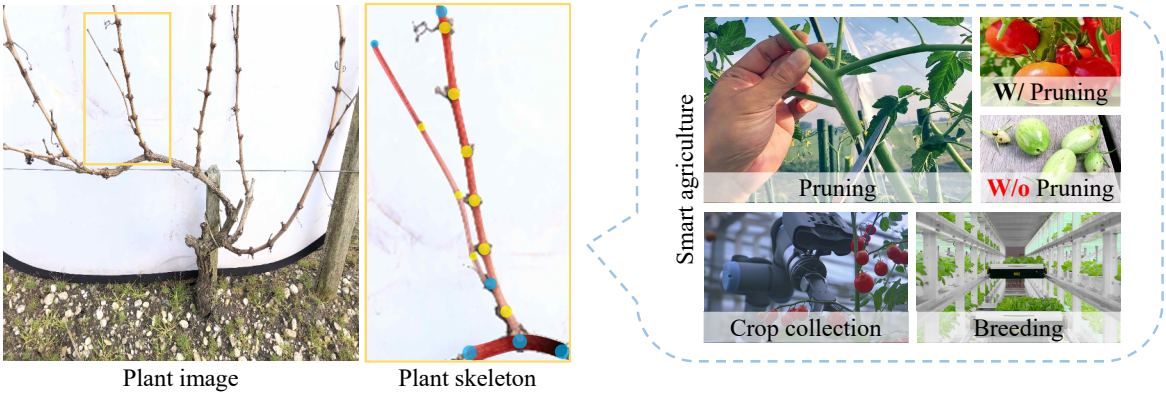


Fig. 1.1 **Plant skeleton and its applications.** Left: an example plant with its extracted skeleton. Middle: skeleton represented as a graph, with *nodes* shown in yellow for intermediate positions and in blue for roots, endpoints, or splits, while *edges* are depicted as red lines. Right: representative applications in agriculture, including pruning, robotic crop collection, and plant breeding.

and topologically consistent tree graph.

We consider two problem settings in computer vision as shown in the top figure in Figure 1.2: 2D single-view skeleton estimation and 3D multi-view skeleton estimation. 2D skeleton estimation [2, 15, 16] aims to detect a tree graph directly from one RGB image. This setting is relevant in narrow or cluttered environments, or when low cost and low latency are critical for real-time operation. Existing 2D methods generally do not guarantee a reasonable tree graph, producing graphs that contain loops or disconnected components. As illustrated in Figure 1.2 (bottom-left), their outputs often include non-tree structures with erroneous loops or disconnected segments. Furthermore, these models suffer from limited generalization due to strict domain-specific constraints. For example, a model trained on grapevine data performs poorly on unseen test species such as Venus flytrap, as shown in Figure 1.2 (bottom-left). 3D skeleton estimation [13, 14, 17, 18] aims to estimate a three-dimensional spatial skeleton with node positions and branch radii. This setting enables richer downstream analysis, including occlusion reasoning and light interception. However, beyond the difficulty of enforcing a tree graph, 3D methods are susceptible to point cloud density, making thin branches and low-texture regions prone to fragmentation (see Figure 1.2,

bottom-right).

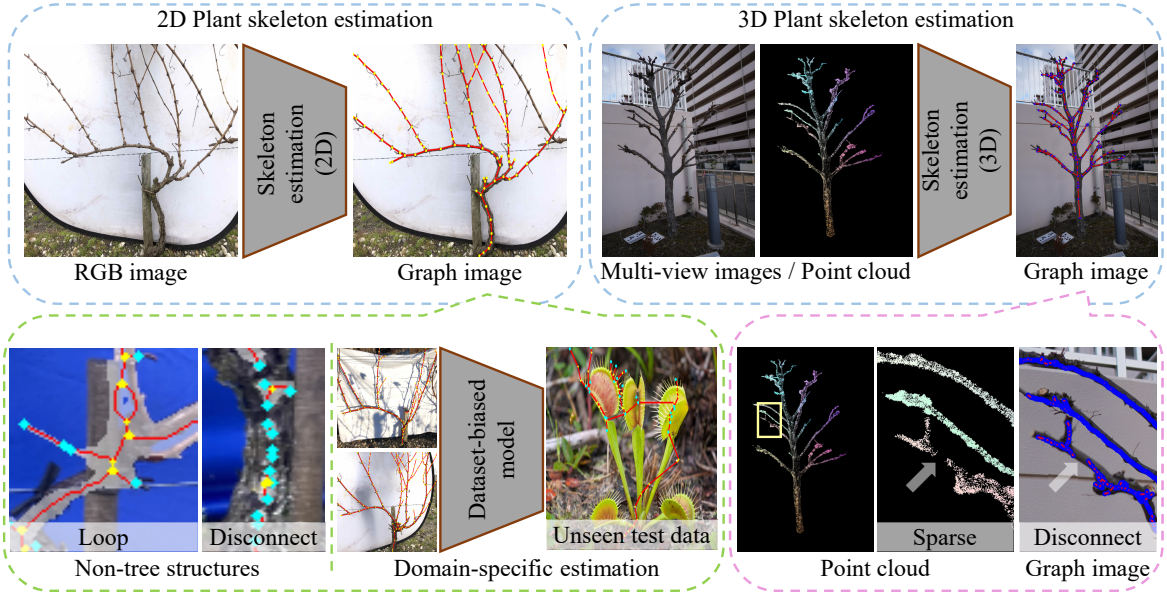


Fig. 1.2 **Limitations of 2D and 3D skeleton estimation.** Top: 2D (left) and 3D (right) skeleton estimation examples. Bottom: representative failure cases: for the 2D case, including non-tree structures, domain-specific bias. For the 3D case, fragmentation is due to sparse point clouds.

1.1 Motivation

The overarching goal of this thesis is to establish a unified framework for plant skeleton estimation across both 2D and 3D settings. To achieve this goal, we proceed step by step: First, in Chapter 2, we construct the PlantPose dataset, focusing on diverse 2D images and unified annotation protocols. Second, in Chapter 3, we leverage this dataset to develop PlantPose2D, an end-to-end generalized model that enforces tree topology during image-based skeleton estimation. Third, in Chapter 4, we extend from 2D to 3D estimation, reusing the constraints and techniques from PlantPose2D while introducing a mask-guided, self-supervised optimization framework. Since 3D data collection is significantly more costly, this chapter relies on multi-view images and self-supervised learning rather than assembling a large 3D dataset. By framing the work in this way, the thesis clarifies how solving the 2D and 3D tasks together

contributes to a larger vision of robust, generalizable, and practically useful skeleton estimation.

Plant skeleton estimation is increasingly studied due to its importance in phenotyping, growth modeling, and agricultural robotics. Various 2D image-to-graph models and 3D estimation pipelines have been proposed. Despite such flourishing, we observe several shared limitations that hinder accuracy and robustness, especially when the output must be a connected, loop-free tree graph. This thesis focuses on three key issues. (i) Lack of tree-level supervision and strong domain shift: existing datasets often provide branch-level annotations rather than full trees, and present significant domain shifts across species, growth stages, and imaging styles, which limit transferability. (ii) Topological violations in 2D and 3D estimation: outputs often contain cycles or disconnected components and enforce a tree graph only via post-processing projections, leading to visually unnatural connections. (iii) Fragility of 3D pipelines under point cloud limitations: thin, low-texture branches cause uneven density, making skeletonization difficult during downstream processing. Respective details of the three problems are presented in the following.

Lack of tree-level supervision and strong domain shift Public plant skeleton datasets remain scarce and domain-specific. Two representative datasets are MIPDB [1], a multi-time maize image–phenotype dataset, and ViNet [2], a grapevine structure dataset. Both datasets' annotations are branch-level: each branch is labeled independently, without a tree graph spanning the entire plant. Existing public annotations are often limited to branch-level (independent sequences of nodes and edges) rather than complete tree graphs, and substantial variation exists across species, growth stages, imaging modalities, and environments. As a result, supervision is locally correct but topologically incomplete, while a complete tree graph is critical for tasks such as automated acquisition, pruning, cut planning, and time-series analysis. In addition, models trained on ViNet generalize poorly to MIPDB and vice versa, due to substantial differences in species morphology, organ type, background clutter,

and capture protocol. These factors highlight a supervision gap: the lack of fully plant-level annotations and pronounced cross-domain shifts hinder learning a single, connected tree graph across diverse plant types.

Topological violations in 2D and 3D estimation Recent 2D methods based on recurrent neural networks (RNNs) [19], graph neural networks (GNNs) [20–22], and transformers [4, 5, 23–25] can recover thin structures from images but decode graphs without explicit tree constraints. In 2D single-view settings, nodes are typically detected through keypoint estimators or heatmaps [2, 15, 16], while edges are selected by local scores. Consequently, predicted graphs often contain cycles or disconnected components, particularly under self-occlusion and branch crossings. Post-processing projections such as MST or shortest-path heuristics can repair connectivity, but they often introduce geometrically implausible shortcuts, leading to visually unnatural structures.

In 3D pipelines, similar errors arise from different sources. Point cloud thinning, clustering, or gap-bridging heuristics can generate small loops around noisy regions; sparse or uneven point cloud density often fragments thin branches; and during multi-view estimation or graph refinement, tree topology is rarely enforced throughout optimization. Taken together, across both 2D and 3D, topology is not enforced during optimization and prediction. As a result, outputs violate the tree constraint (cycles or disconnections) or rely on post-processing fixes that distort geometry and yield structurally implausible skeletons.

Fragility of 3D pipelines under point cloud limitations Most 3D approaches rely on point clouds reconstructed from LiDAR or MVS, which subsequently serve as the basis for skeletonization [18, 26–28]. Structural variations such as low-texture branches and localized swellings often result in uneven point cloud density. Swollen regions like galls are oversampled into dense clusters, whereas low-texture or thin branches are sparsely represented or entirely missing. Preprocessing, such as denoising and thinning, can further erode fragile structures. Downstream skeletonization

methods [29–31], including voxel thinning and clustering with radius- or neighbor-based heuristics, must rely on scene- and species-dependent thresholds to reconnect gaps. In locally dense regions, algorithms tend to overconnect, inserting redundant bridges and small loops. In sparse regions, connectivity breaks persist, leaving fragmentation. Moreover, parameter choices cannot reconcile these regimes: conservative thresholds reduce overconnections in dense clusters but exacerbate disconnections elsewhere, producing gaps that post-processing cannot reliably repair. Together, these challenges render point-cloud–based 3D skeletonization highly weak under realistic field conditions.

1.2 Thesis Organization

This thesis addresses three fundamental challenges outlined in the previous section. The thesis proceeds from dataset construction, through 2D skeleton estimation, to 3D skeleton estimation.

Chapter 2: PlantPose Dataset This chapter introduces a large and diverse plant skeleton estimation dataset comprising ten subsets. It details a unified annotation protocol that pairs pixel-level node locations with a tree-encoded adjacency matrix. Additionally, the chapter introduces an automated graph-construction pipeline that organizes branch-level labels into a single, connected tree graph for each image. It also describes the methods for generating synthetic data and outlines a generalized train/test split that facilitates cross-domain evaluation. Finally, the chapter concludes by analyzing the dataset's biases and limitations.

Chapter 3: PlantPose2D This chapter develops a single-view, end-to-end image-to-graph model that explicitly enforces tree topology during learning [8, 32–34]. Our method incorporates topology directly into training, unlike existing models that produce unconstrained structures and rely on post-processing to impose tree constraints. Specifically, we introduce an SFS layer that projects intermediate edge scores onto

an MST and aligns the internal representations accordingly. This way, cycle-forming and isolated nodes are suppressed throughout optimization, ensuring the predicted skeleton remains a single, connected tree graph.

Chapter 4: PlantPose3D This chapter presents a mask-guided graph optimization framework that estimates 3D plant skeletons directly from refined multi-view masks. Using differentiable silhouette rendering based on 3DGS [35], the method jointly optimizes node positions, branch radii, and connectivity to fit multi-view observations, while enforcing a tree graph throughout optimization. The approach is robust to sparse or fragmented geometry, thereby reducing reliance on dense point clouds and producing skeletons that are both topologically consistent and geometrically faithful.

Chapter 5: Discussion and Conclusion This chapter concludes this dissertation by summarizing the proposed methods and dataset and discussing potential future research directions.

1.3 Summary of Contributions

The main contributions of this thesis are summarized as follows.

- We curate the PlantPose dataset [8, 36], a large and diverse benchmark that includes ten subsets. We have established a unified annotation protocol that pairs pixel-level node locations with a tree-encoded adjacency matrix. To this end, we design an automated graph-construction pipeline that reconciles branch-level annotations into complete tree graphs. Finally, we define generalized train/test splits that explicitly support cross-domain evaluation, allowing for a systematic assessment of generalization beyond species- or dataset-specific training.

- We develop PlantPose2D [8], an end-to-end single-view image-to-graph model that enforces the tree constraint during learning. An SFS layer projects inter-

mediate edge scores onto an MST and aligns internal representations accordingly, suppressing cycle-forming and isolated structures. Experiments show higher tree rates and strong generalization across in-domain and out-of-domain datasets.

- We propose PlantPose3D [37], a mask-guided graph optimization framework that estimates 3D skeletons directly from refined multi-view masks. By leveraging differentiable silhouette rendering with 3DGS, the framework jointly optimizes node positions, branch radii, and connectivity while consistently enforcing a tree topology. This design reduces dependence on dense point clouds and yields more complete and structurally faithful estimations, even under sparse or fragmented geometry.

Chapter 2

PlantPose Dataset

2.1 Introduction

Existing public datasets for plant skeleton estimation are valuable but remain narrow in scope. Most focus on a single species and are thus domain-specific [1–3]. Additionally, annotations are typically branch-level, where each branch is annotated independently rather than assembled into a single connected, loop-free tree graph. However, full plant-level annotations are critical for applications such as automated pruning, cut planning, and temporal growth analysis, where maintaining global connectivity is essential. The absence of such annotations limits topological supervision, since models never observe a complete plant-level tree graph, and it limits cross-domain robustness, as networks trained on one species or capture style transfer poorly to others. To address this limitation, we present **PlantPose dataset**, a large-scale, connectivity-aware dataset that enables structural supervision and cross-domain generalization for plant skeleton estimation.

PlantPose dataset is constructed according to four guiding principles. (i) **Connectivity awareness.** Each image is paired with fully plant-level annotations consisting of pixel-level node locations and an adjacency matrix encoding a single connected, loop-free tree graph. To achieve this consistently across heterogeneous sources, we introduce a graph-recovery pipeline that reconciles independently annotated branches

into a unified tree structure. (ii) **Diversity across biology.** To extend beyond maize and grapevine, we constructed two additional datasets. The self-captured dataset includes several plant types ranging from small grasses to tall trees, while the web-sourced dataset covers ten further categories collected from online sources. Together, these datasets substantially broaden the biological diversity of the PlantPose dataset. (iii) **Diversity across organs.** We incorporate roots (early-stage Arabidopsis) and 29 categories of leaf veins, covering below-ground structures and laminar vasculature. (iv) **Diversity across styles and scenes.** The dataset spans multiple depiction styles, including RGB photographs (field, greenhouse, and lab), binary masks and sketch-like inputs, and realistic composites with natural backgrounds.

Concretely, the PlantPose dataset comprises ten subsets designed to match these principles. To broaden biological diversity, we expand beyond maize [1] and grapevine [2] by constructing two additional datasets: a self-captured collection covering several species of common plants (*e.g.*, cherry blossom, ginkgo, pine, flowers, and grasses), and a web-sourced dataset covering a wide variety of species, including fruit trees and ornamental plants such as pomegranate, coconut, and lemon. Together, these datasets broaden morphological diversity in branch thickness, curvature, branching order, and growth habit, enabling evaluation across a broader biological spectrum. To extend organ coverage, we include roots (lab-captured Arabidopsis) and leaf veins, the latter provided by LVD2021 [3] in three complementary forms: original RGB leaf images, binary vein masks, and vein composites with natural backgrounds to simulate in-the-wild clutter.

For the synthetic datasets, we employ two different frameworks. The first is the Lindenmayer system (L-system) [38, 39], which produces clean black line patterns; straight-line variants are used as test cases that deliberately differ from curved vein masks, probing stylization robustness. The second is modular_tree framework (Mtree)[1], a modular parametric framework that generates complex, irregular shoot structures, which we composite onto diverse natural backgrounds to jointly stress

[1] `https://github.com/MaximeHerpin/modular_tree/wiki`, last accessed September 1, 2025.

shape and appearance generalization. All subsets are normalized to a common graph representation, where nodes are defined at the pixel level and an adjacency matrix encodes edges. This consistent sampling and connectivity protocol enables unified training objectives and fair cross-domain evaluation.

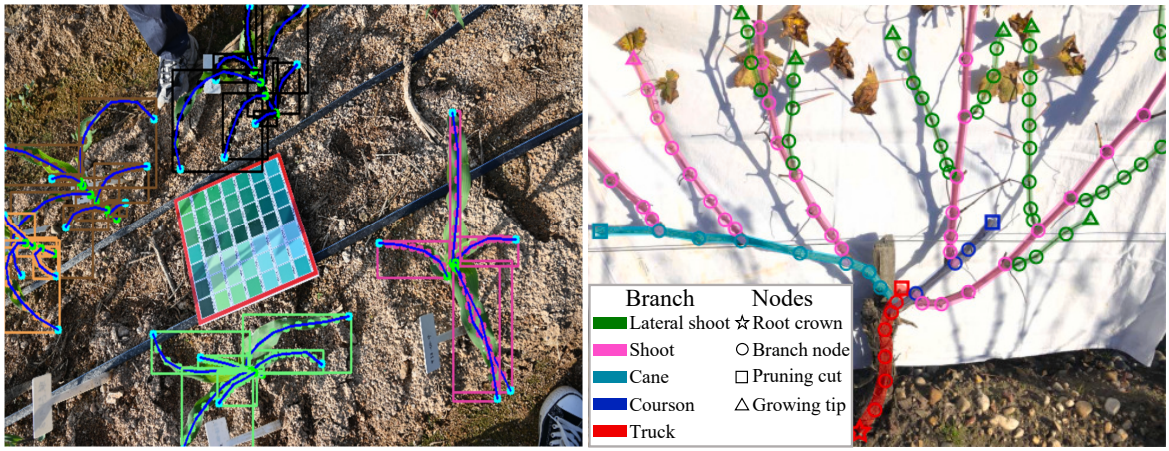
In summary, the PlantPose dataset consolidates ten heterogeneous subsets into a unified benchmark for plant skeleton estimation. Finally, we release standardized training and evaluation protocols that cover domain-specific settings, useful for species-focused analysis, and generalized settings, designed to test cross-domain transfer explicitly. This dual design allows systematic assessment of model robustness under varying species, organs, imaging styles, and environmental conditions.

2.2 Related Work

2.2.1 Real-World Datasets

This subsection summarizes public real-world datasets and highlights their scope and limitations, serving as related work to motivate our unified graph construction in Section 2.3 and the instantiated datasets in Section 2.4.1.

Maize dataset: MIPDB [1] The Maize Image Phenotype Database (MIPDB) comprises more than 30,000 field images, with 17,631 carefully annotated using a point–line protocol. Each maize leaf is traced as an independent polyline, enabling detailed leaf-centric phenotyping tasks such as counting and growth analysis. However, these annotations remain branch-level: individual leaves are traced separately without integration into a complete whole-plant skeleton, as shown in Figure 2.1(a), where different plants are annotated with distinct colors and branch-level labels are highlighted with bounding boxes for clarity. As a result, models trained on MIPDB learn locally correct branch paths but receive no direct supervision of global tree connectivity.

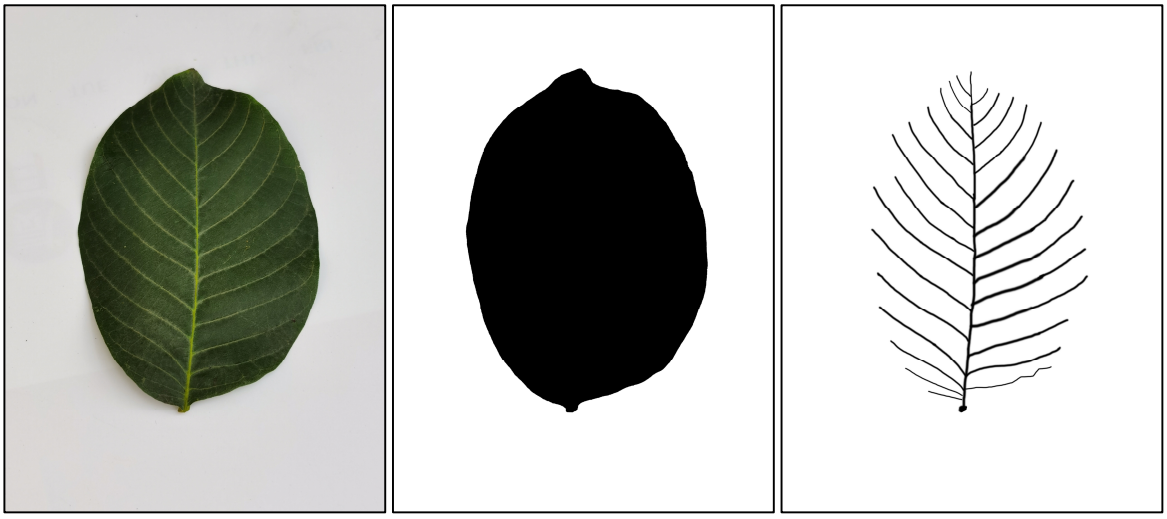


(a) MIPDB dataset (b) Grapevine dataset

Fig. 2.1 **Real-world MIPDB [1] and 3D2cut [2] datasets.** (a) MIPDB [1] maize field images with leaf-wise polylines, enabling leaf-centric phenotyping but lacking an explicit plant-level tree graph. (b) 3D2cut [2] grapevine images with node positions and branch categories; connectivity is inferred via PAF-like affinity fields and weighted shortest paths at inference.

Grapevine dataset: 3D2cut [2] The 3D2cut Single Guyot dataset contains about 1.5k vineyard images annotated for grapevine structure estimation, collected primarily for pruning analysis. The annotation protocol follows a pose-inspired formulation: node positions and branch types are labeled. Connectivity is inferred through vector-field affinity maps, conceptually similar to Part Affinity Fields (PAFs) in human pose estimation. Edges are then reconstructed at inference time via weighted shortest-path optimization, as shown in Figure 2.1(b). As a result, this design provides intense supervision for learning local relations and branch semantics. However, fully plant-level adjacency is not directly annotated, and the global constraint of a single connected, loop-free tree is imposed only post-processing during graph extraction. Similar to the MIPDB dataset, 3D2cut provides intense supervision at the organ level but lacks explicit plant-level tree graphs.

Leaf veins: LVD2021 [3] The Leaf Vein Dataset 2021 (LVD2021) provides 4,977 images from 36 species, consisting of three components: raw leaf images, binary leaf masks, and pixel-wise vein masks, as shown in Figure 2.2. The task is defined as



(a) Leaf image (b) Leaf mask (c) Vein mask

Fig. 2.2 **LVD2021 [3] dataset examples.** From left to right: raw leaf image, binary leaf mask, and binary vein mask.

segmentation rather than graph prediction: annotations are binary masks of venation captured on clean backgrounds. While these masks preserve fine geometry, they lack explicit node–edge labels, and some venation patterns contain loops (areoles), which by definition violate the tree graph. Therefore, converting LVD2021 into a tree graph requires skeletonization followed by graphization and topology repair, such as projecting onto a spanning tree.

Summary Across these real-world datasets, supervision predominantly focuses on local structures such as nodes, per-branch polylines, or pixel masks, rather than providing explicit plant-level connectivity. Grapevine offers node semantics and local affinities but reconstructs connectivity algorithmically. MIPDB supplies detailed per-leaf traces but does not provide whole-plant graphs. LVD2021 delivers segmentation masks that preserve venation geometry but lack adjacency information. Moreover, the three datasets differ substantially in capture conditions, annotation protocols, and biological targets, resulting in significant domain shifts. These limitations collectively highlight the need for a unified dataset that enforces plant-level connectivity across diverse species, organs, and imaging conditions.

2.2.2 Synthetic Datasets

This subsection overviews commonly used synthetic generators (related work) and their characteristics/limitations, setting up how we later utilize them in Section 2.4.2 under a unified graph representation.

L-system Lindenmayer systems [38, 39] are parallel rewriting grammars introduced for modeling growth processes, and are widely used to generate branching forms that resemble plants. An L-system specifies an alphabet, an axiom, and a set of production rules, and the resulting string is interpreted geometrically, for example, through turtle graphics. This mechanism produces self-similar branching patterns with controllable stochasticity and recursion depth, and has become a standard tool for procedural plant geometry.

Mtree *modular_tree* framework ("Mtree") is a modular parametric framework that generates diverse and realistic tree structures. Instead of relying on rewriting grammars, Mtree constructs trees by hierarchically combining parametric modules that control branching order, curvature, thickness, and stochastic variations. This modular design creates complex, irregular shoot architectures that resemble real plants across various scales and species. In practice, generated trees can be rendered onto diverse natural backgrounds, providing synthetic imagery that stresses geometric variability and visual realism, making Mtree a valuable tool for evaluating model robustness under heterogeneous conditions.

Summary Synthetic datasets play a complementary role to real-world collections by providing controlled yet diverse branching structures. L-systems generate clean, mathematically defined patterns as a baseline for evaluating structure recovery under idealized conditions. Mtree, by contrast, produces irregular and realistic tree geometries with controllable complexity and natural appearance. Together, they allow systematic testing of algorithms across a spectrum from abstract generative rules to

photorealistic plant-like structures, supporting robust assessment of skeleton estimation methods.

2.3 Dataset Construction Method

2.3.1 Objectives and Unified Representation

Objectives We aim to establish a unified graph representation that enables consistent training and evaluation across diverse plant datasets. The design follows four principles:

(i) **Topology fidelity**: the representation must form a single connected, loop-free tree graph.

(ii) **Source consistency**: annotations from different datasets are converted into the exact graph-based representation of the plant skeleton, regardless of whether the original labels are polylines, masks, or vector-field relations.

(iii) **Spatial comparability**: graphs are discretized at a uniform arclength resolution to ensure consistent density across species and organs.

(iv) **Minimal sufficiency**: the representation retains only the essential elements for learning and evaluation, namely node coordinates and adjacency.

Unified representation Given a resized input image $I \in \mathbb{R}^{256\times256\times C}$, where C is the number of image channels (*e.g.* C=3 for RGB image), we construct a tree graph

$$G = (V, E), \quad V = \{(x_k, y_k)\}_{k=1}^{|V|}, \tag{2.1}$$

Here $|V|$ denotes the number of vertices. The following conventions apply:

- **Node coordinates.** Each vertex (x_k, y_k) is defined on the resized 256×256 pixel grid, with $x_k, y_k \in [0, 256)$.

- **Adjacency.** The edge set E is encoded as a symmetric binary adjacency matrix $\mathbf{A} \in \{0,1\}^{|V|\times|V|}$ with $\mathbf{A}_{ij} = 1$ if $(i,j) \in E$. By construction, G is a tree graph which by definition satisfies $|E| = |V| - 1$. Here $|E|$ denotes the number of edges.

- **Geometric discretization.** For each polyline segment between anatomical keypoints, intermediate vertices are inserted at uniform arc-length intervals of $\Delta s = 13$ pixels. This enforces consistent spatial density across species and organs.

Scope of applicability. This representation assumes a single plant instance per image and enforces a consistent tree topology across sources. In practice, dataset-specific preprocessing (*e.g.*, skeletonization of vein masks or clustering of field annotations) is applied before conversion, but the resulting graphs always conform to the same canonical form. Quality control checks (*e.g.*, connectivity, cycle count) are applied to guarantee that only valid tree graphs enter training and evaluation.

2.3.2 Polyline-Based Tree Construction

Many public datasets provide branch-level annotations, where each branch is traced independently rather than as a full plant graph. To obtain plant-level tree skeletons consistent with the unified representation defined in Section 2.3, we must merge these disconnected branches into a single connected tree.

Given a set of disconnected branch polylines, our goal is to recover a plant-level tree that satisfies the unified representation in Section 2.3.1. Each input branch is an ordered point sequence $\mathbf{b} = \{p_1, \ldots, p_m\}$, $p_i \in \mathbb{R}^2$. The procedure constructs $G = (V, E)$ by initializing a backbone from the longest branch and attaching the remaining branches with geometry-aware linking while preserving the tree constraint.

Preprocessing and resampling. We first remove duplicate points within each branch and discard very short polylines. To equalize spatial density, every branch

is resampled at a uniform arclength interval $\Delta s = 13$ pixels on the resized canvas, producing sequences with approximately constant step length.

Backbone initialization. Let $\mathscr{B} = \{\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(M)}\}$ be the branch set. We select the longest branch $\mathbf{b}^{(1)}$ as the initial backbone and insert all its samples as vertices V. Consecutive samples are connected to form E along $\mathbf{b}^{(1)}$.

Geometry-aware attachment cost. For any unattached branch $\mathbf{b}$ with endpoints e_{head} and e_{tail}, we use the shorthand $e_\star \in \{e_{\text{head}}, e_{\text{tail}}\}$ to denote either endpoint. We estimate its best attachment to the current graph by minimizing the distance from the branch endpoints to the existing structure. The distance is evaluated against both nodes and edges:

$$d(\mathbf{b}, G) = \min \left\{ \min_{v \in V} \min_{e_\star} \|e_\star - v\|_2, \ \min_{(i,j) \in E} \min_{e_\star} \operatorname{dist}(e_\star, \overline{v_i v_j}) \right\}. \tag{2.2}$$

where $\operatorname{dist}(e, \overline{v_i v_j})$ is the point-to-segment distance. If the closest contact occurs at e_{tail} rather than e_{head}, the branch is reversed to maintain consistent growth direction before attachment. Branches are processed in ascending order of $d(\mathbf{b}, G)$.

Edge splitting and snap-in rules. Let the best contact for $\mathbf{b}$ be either a vertex v_k or an edge (v_i, v_j):

- If the closest contact is a vertex and $\|e_\star - v_k\|_2 \leq r_{\text{merge}}$, we snap the endpoint $e_\star$ onto v_k and append the resampled points of $\mathbf{b}$ to V, connecting consecutive samples to form new edges in E.
- If the closest contact lies on an edge interior, we orthogonally project $e_\star$ onto the segment $\overline{v_i v_j}$. The edge (v_i, v_j) is split by inserting a new vertex v_{ins} at the projection point, replacing (v_i, v_j) with (v_i, v_{ins}) and (v_{ins}, v_j), and then connecting $\mathbf{b}$ to v_{ins}.

Both cases preserve acyclicity because only one attachment is created for each new branch.

Tree constraint and progress order. At each iteration only one attachment is allowed, so $|E| \leftarrow |E| + |\mathbf{b}| - 1$ and $|V| \leftarrow |V| + |\mathbf{b}|$ for the added samples, which maintains $|E| = |V| - 1$ once all branches are attached. If a candidate's attachment would introduce a cycle, it is rejected, and the next best contact is considered. The process repeats until all branches are integrated or no valid attachment remains.

Endpoint consolidation and quality control. After all attachments, nearby endpoints created by discretization are merged within r_{merge}, and incident edges are rewired to the merged vertex. We then verify connectivity and the Euler condition $|E| = |V| - 1$. Samples that fail either check are flagged for manual review or excluded. Finally, coordinates are stored in the resized canvas with recorded scale and padding for inversion to the original image space.

Remarks. The procedure is deterministic given Δs, r_{merge}, and the distance thresholds. It unifies polyline-style annotations from heterogeneous sources into a plant-level tree without relying on dataset-specific heuristics. Optional interactive pruning of erroneous branches can be performed before the attachment stage, but it is not required for the automated conversion.

2.3.3 Mask-Based Skeleton Extraction

Given a binary mask of plant structures, such as the binary vein mask in Figure 2.2 (c), we aim to convert thin branching patterns into a single connected, loop-free tree graph under the unified representation. The procedure follows a pipeline consisting of (i) skeletonization with denoising, (ii) contraction of pixel chains into polylines, and (iii) validation and export to the canonical graph format.

Skeletonization and pruning. Starting from the input masks, we first apply morphological closing to fill small gaps, and connected components below a minimum area threshold are removed; then we perform the standard skeletonization.

Pixel graph contraction and contraction. We treat each pixel on the one-pixel-wide skeleton as a vertex, and edges are established to its 8-neighborhood, yielding an undirected dense pixel-level graph whose vertex set coincides with the foreground pixels. Let $\deg(v)$ denote the graph degree. Vertices with $\deg(v) \neq 2$ are designated as keypoints (junctions or endpoints). Degree-2 chains connecting pairs of keypoints are contracted into polylines, with intermediate points uniformly resampled at a fixed arclength interval of $\Delta s = 13$ pixels. This procedure reduces the pixel graph into a compact set of keypoints and polylines while preserving geometric fidelity.



Fig. 2.3 Illustration of the Mask-to-Skeleton pipeline. From left to right: (a) input binary mask, (b) zoomed region, (c) skeletonized pixel graph, (d) contracted polylines and final graph representation with blue endpoints, yellow intermediate nodes, and green joint nodes.

Tree validation and export. After polyline contraction, to guarantee a valid tree structure, we reuse the branch fusion strategy introduced in Section 2.3.2, which connects polylines by iteratively inserting junction nodes based on nearest-node and nearest-line distances. This ensures that the final graph forms a single connected, loop-free tree graph with $|E| = |V| - 1$. The resulting tree graphs are exported in normalized coordinates as adjacency matrices and node sets. This procedure reduces the pixel graph into a compact set of keypoints and polylines (piecewise linear edges), as illustrated in Figure 2.3.

2.4 Datasets

Building on the existing datasets in Section 2.2 and the conversion pipelines in Section 2.3, this section presents the *PlantPose* dataset subsets actually used in our experiments.

2.4.1 PlantPose Real-World Subsets

This section details how we *leverage* the datasets summarized in Section 2.2.1 and our own collections: we apply data preparation/augmentation and the unified graph-construction pipelines from Section 2.3 to obtain connectivity-aware plant-level tree graphs for the PlantPose dataset.

We first combine three open-source real-world datasets and extend them with two collections to broaden biological coverage. The maize dataset (MIPDB [1]) provides branch-level polylines that we convert into a single plant-level tree graph per image using the module in Section 2.3.2. For the grapevine dataset (3D2cut Single Guyot [2]), connectivity is first inferred from vector-field affinity maps, conceptually similar to Part Affinity Fields, and branch paths are reconstructed via weighted shortest-path optimization; we then link the resulting branch polylines into a full tree graph with the same procedure introduced in Section 2.3.2. The LVD2021 [3] dataset offers leaf vein masks that we transform into tree graphs through the module in Section 2.3.3, and we expose three views: the raw vein-mask view, the RGB leaf-image view, and a composited leaf-image view with natural backgrounds. To expand biological diversity, we include a self-captured dataset spanning several plant types and a web-sourced dataset covering ten additional categories. We include a root dataset of early-stage *Arabidopsis* that complements the above-ground shoot and laminar vasculature to expand organ diversity. All converted graphs follow the unified representation with images resized to 256×256 and uniform arclength discretization at Δs=13 pixels.

MIPDB dataset [1] The MIPDB dataset [1] provides over 28,000 high-resolution field images with independent branch-wise polyline annotations for each leaf. These labels are organ-centric and do not form a plant-level skeleton as shown in Figure 2.1(a). To prepare the dataset for graph-based analysis, we segmented the field images into single-plant views by clustering annotated points, ensuring that each resulting image contained only one complete plant. Then connect branches into a single tree with the procedure in Section 2.3.2: endpoints are snapped or inserted to the nearest node or segment of the evolving tree with geometry-aware distances, and the augmented edge set is possessed as a loop-free tree graph. The final dataset consists of 12,992 training images, 249 validation images, and 250 test images. Due to the dataset's large size and diversity, we did not apply data augmentation.

Grapevine dataset [2] The grapevine dataset [2] provides node locations and branch-type cues for vine shoots collected for pruning analysis. However, connectivity is not provided as an explicit adjacency as shown in Figure 2.1(b). Following the same strategy as the ViNet paper [2], we first reconstruct branch-wise polylines from the released vector-field affinity maps, which act similarly to Part Affinity Fields by indicating local tangent directions and pairwise association strengths between nodes. Minimum-cost paths are traced between admissible node pairs defined by branch-type rules, yielding candidate curves aligned with the affinity field. Redundant or weak alternatives are discarded, resulting in a set of branch-wise polylines. These polylines are then merged into a single connected tree using the module described in Section 2.3.2. Since the native average edge length after resizing ($\approx$ 12.8 px) is already consistent with our discretization step Δs=13, we retain the original sampling to preserve biological joint accuracy. We applied extensive augmentations for the grapevine images, including translation, flipping, rotation, brightness adjustment, aspect ratio modification, and scaling. This process expanded the training set tenfold, resulting in 12,020 training images alongside 150 validation and 151 test images.

Table 2.1 Leaf category distribution of the LVD2021 leaf image and vein mask datasets across train, validation, and test splits.

Leaf Category	**Train**	**Val**	**Test**	**Leaf Category**	**Train**	**Val**	**Test**
Forsythia Suspensa	83	13	7	Sycamores	48	5	8
Fructus Xanthii	32	1	6	Lilac	50	10	12
Grape	70	7	12	Persimmon	7	2	2
Morning Glory	80	5	8	Smoke Tree	105	8	6
Apricot	85	7	8	Vitex Negundo Var	96	13	10
Chenopodium Album	94	14	13	Elm	77	14	12
Callistephus Chinensis	50	6	12	Holly	45	6	2
Walnut	69	9	10	Poplar	144	17	18
Maple Tree	109	16	9	Chinese Redbud	83	16	15
Amaranth	104	10	10	Hackberry	55	6	8
Honeysuckle	52	3	6	Crataegus Pinnatifida	68	8	4
Sweet Potato	87	10	11	Mirabilis Jalapa	47	8	10
Cedar	81	16	13				
Mulberry	26	2	–	Virginia Creeper	1	–	–
Thistle	9	–	–	Hibiscus	1	–	–

LVD2021 leaf image dataset [3] The Leaf Vein Dataset 2021 (LVD2021) [3] provides high–resolution RGB images of leaves captured on a plain white background together with pixel–level vein annotations. We use the mask images as inputs and derive plant–level tree graphs from the vein masks using the Mask–to–skeleton pipeline in Section 2.3.3. Graphs are finally resampled at a uniform arclength step $\Delta s = 13$ pixels and resized to 256×256 under the unified representation. We apply the same geometric and photometric augmentations used for grapevine, expanding the training split tenfold to $18,580$ images, with 232 images for validation and 232 for testing. The species distribution across the 36 categories is summarized in Table 2.1.

LVD2021 vein mask dataset [3] To isolate the value of structural cues, we also use the vein masks as inputs. The masks are converted to graphs with the same Mask–to–skeleton module. The conversion enforces a single connected, loop–free tree graph, then applies uniform arclength resampling with $\Delta s = 13$ pixels on a 256×256

canvas. The vein masks are particularly valuable because real-world tree structures are not limited to captured or synthetic plant images but also exist in simplified forms, such as sketches or abstract drawings. Including vein masks in the training regimen enhances the model's robustness to simplified structural inputs like sketches or abstract drawings. Since this modality already abstracts appearance, we do not apply augmentation. The resulting split contains $1,858$ training images, 232 validation images, and 232 test images, with the same class distribution as the RGB leaf images in Table 2.1.

LVD with background dataset [3] To evaluate robustness to realistic backgrounds, we composite LVD2021 leaf images onto 627 natural scenes sourced from publicly available photographs. Backgrounds span diverse environments and conditions, for example, forest understory, open fields, wetlands, shaded paths, and sunny or overcast illumination. During compositing, we apply random in–plane transforms and brightness or scale adjustments to the leaf while keeping the ground–truth graph derived from its vein mask. Each composite is then resized and paired with the same tree–consistent graph representation used elsewhere, including the $\Delta s = 13$ pixel resampling. This set contains 500 test images and is used exclusively for out–of–domain evaluation.

Self-captured dataset To broaden the PlantPose dataset beyond maize and grapevine, we collected a self-captured dataset covering several plant categories, ranging from small grasses to tall trees. Images were taken in natural outdoor conditions with varying illumination and background clutter, including cherry blossoms, ginkgo, pine, and several flowers and shrubs. We annotate each image with branch-level polylines and then convert it into a plant-level tree graph using the module in Section 2.3.2, ensuring consistency with the unified graph representation. The dataset contains 817 images, split into 652 training, 82 validation, and 83 test samples. After augmentation, the training set expands to $13,040$ images.

Table 2.2 Plant category distribution of the Web-sourced dataset across train, validation, and test splits.

Plant Category	**Train**	**Val**	**Test**	**Plant Category**	**Train**	**Val**	**Test**
Pomegranate	144	16	21	Mango	156	20	19
Plum	125	12	14	Almond	2	–	–
Coconut	525	67	69	Mulberry	1	–	–
Apricot	192	24	25	Cherry	1	–	–
Lemon	173	27	20	Fig	1	–	–

Web-sourced dataset We also extend biological diversity with a web-sourced dataset comprising publicly available images of ornamental and fruit-bearing plants. The set spans ten categories, including pomegranate, mango, plum, apricot, coconut, lemon, and a few rare species. As with the self-captured dataset, branch-level annotations were converted into unified tree graphs with the Branches-to-tree pipeline, ensuring compatibility with the PlantPose dataset format. The dataset comprises $1,654$ images, divided into $1,320$ training, 166 validation, and 168 test images. Data augmentation expanded the training set tenfold to $13,200$ samples. The category distribution is summarized in Table 2.2. Together with the self-captured dataset, this dataset substantially increases the taxonomic breadth of the PlantPose dataset.

Root dataset To extend the PlantPose dataset across organs, we include below-ground structures in addition to above-ground shoots and laminar vasculature. We use early-stage *Arabidopsis* roots, where annotators trace root centerlines as branch-wise polylines. The labels are branch-level and do not provide an explicit plant-level adjacency. We therefore convert each instance into a single tree with the module in Section 2.3.2, which links polylines into a coherent root graph under the unified representation. We standardize geometric density by resampling along continuous segments at a fixed arclength interval of $\Delta s = 13$ pixels. The maximum node count per graph decreases from 117 in the raw annotations to 71 after resampling, which aligns the root graphs with the spacing used for other datasets. This dataset contains 782 test images with no augmentation.

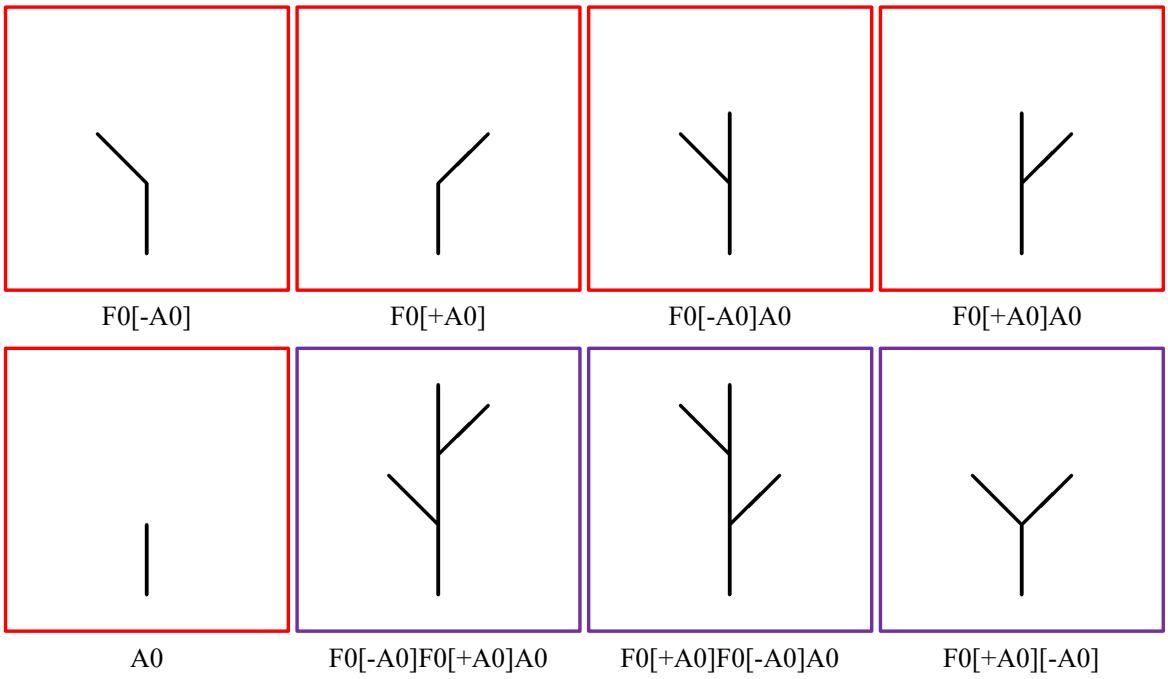
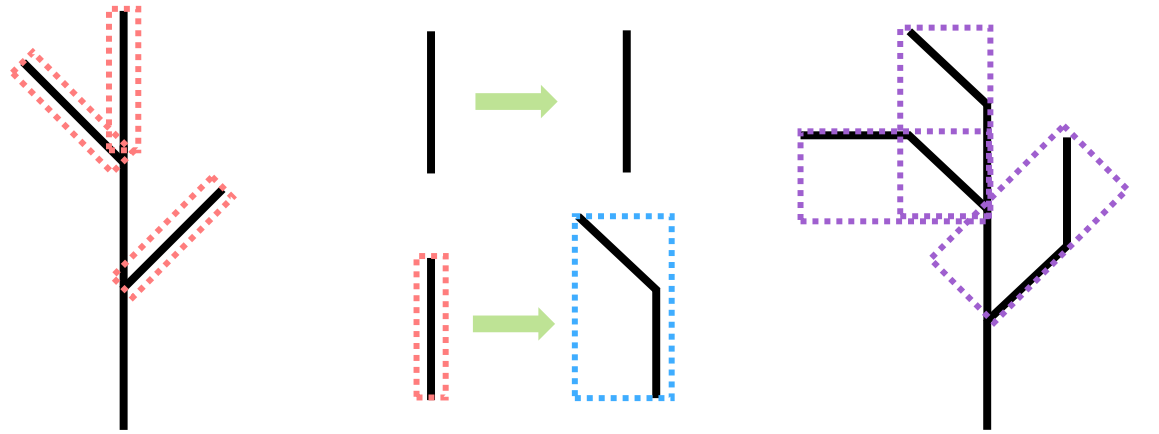


Fig. 2.4 Atomic structures used for synthetic dataset generation. Three pre-defined initial structures are highlighted in purple. Eight pre-defined rewriting rules are used during the generation.

2.4.2 PlantPose Synthetic Subsets

This section explains how we *utilize* the synthetic sources introduced in Section 2.2.2: we generate/compose data and convert them via the unified pipelines in Section 2.3 to form the PlantPose dataset' s subsets.

L-system dataset We generate two-dimensional tree patterns with an L-system [38, 39], which specifies an alphabet, an axiom, and production rules that are iteratively rewritten to produce branching strings. Turtle graphics interprets the strings to obtain planar trees. Figure 2.4 shows the initial structures and the rewriting rules we used. At the beginning of the tree generation, an initial sequence is randomly chosen from the pre-defined sequences marked with a purple frame in the figure. At each iteration during the tree generation, the leaf edges ("**A**" in the sequences) are replaced by a randomly chosen pattern from eight pre-defined ones. A simple example is shown in Figure 2.5. To distinguish this dataset from the fine, curved lines in the LVD2021 vein



(a) Initial structure (b) Rewriting rule (c) After rewriting process

Fig. 2.5 An example of the rewriting process. Suppose the initial structure is represented as **F0**[+**A0**]**F0**[-**A0**]**A0**. If a rewrite rule **F**→**F**; **A**→**F**[-**A**] is applied, *i.e.*, **F** remains unchanged and **A** becomes **F**[-**A**], the result of the rewrite process is **F0**[+**F1**[-**A1**]]**F0**[-**F1**[-**A1**]]**F1**[-**A1**]. The digits in the sequences indicate the number of times the rewrite is applied.

mask dataset, we render the L-system centerlines as straight segments with increased line width on a clean background ("thickened" L-system). Ground-truth graphs are taken directly from the generator, then uniformly resampled to Δs=13 pixels and resized to 256×256 for compatibility with the unified representation. The dataset contains 500 images with no augmentation and a maximum of 211 nodes per graph. Unlike the curved RGB images and vein masks, the L-system dataset can probe robustness for stylization and structural morphology.

Mtree dataset We synthesize photorealistic woody plants with the *modular_tree* framework ("Mtree"), which procedurally assembles trees from hierarchical parametric modules rather than rewriting grammars. A tree is built level by level (trunk, primary, secondary, etc.) with controls for branching order, curvature profiles, thickness taper, and stochastic jitter; species-like presets and tropisms (e.g., gravity or light) allow the same template to realize diverse architectures from conifers to broadleaf trees. For each instance, we randomize branching depth, taper curve, curvature noise, and use foliage to execute the occlusion, then render onto 1,408 natural backgrounds collected

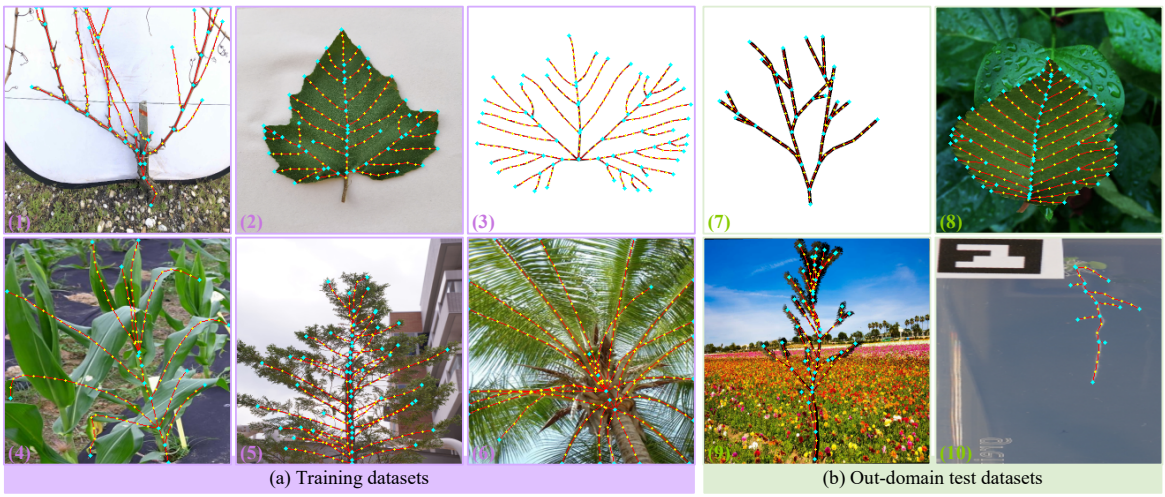


Fig. 2.6 Sample images from the generalized datasets. Annotated graphs are superimposed, where yellow dots represent nodes and red lines represent edges. (a) **Training datasets**: (1) grapevine, (2) LVD2021 leaf image, (3) LVD2021 vein mask, (4) MIPDB, (5) self-captured, (6) web-sourced. (b) **Out-of-domain test datasets**: (7) thickened L-system, (8) LVD with background, (9) tree synthetic, (10) root. The datasets include diverse plant structures and imaging conditions to improve the model's generalization and robustness across in-domain and out-of-domain scenarios.

by environment-based queries (forests, meadows, deserts, wetlands, savannahs) under varied conditions (sunny, shaded, foggy, rainy). Additional variation in camera pose, focal length, scale, and exposure produces silhouettes, occlusions, and contrast regimes that stress both geometric and appearance robustness. The framework outputs ground-truth graphs directly, then the union is resampled along arclength at Δs=13 pixels and resized to 256×256 to match the unified representation. The Mtree dataset generates complex, irregular shoot structures, which we composite onto diverse natural backgrounds to stress shape and appearance generalization jointly.

2.4.3 Dataset Split

We assemble a *generalized* benchmark with six training datasets and four out-of-domain test datasets. An overview of these datasets is provided in Figure 2.6 and Table 2.3. The training datasets are: MIPDB, Grapevine, LVD2021 leaf image, LVD2021 vein mask, self-captured, and web-sourced datasets. The out-of-domain test datasets

Table 2.3 Generalized dataset composition. Training datasets are marked in white, while out-of-domain test datasets are shaded in gray. Augmentation is applied only to training datasets to improve model generalization.

Dataset	Max Nodes	Samples (Train/Val/Test)	Aug.	Total
Grapevine	205	1202 / 150 / 151	10x	1503
LVD2021 leaf image	326	1858 / 232 / 232	10x	2322
LVD2021 vein mask	326	1858 / 232 / 232	1x	2322
MIPDB [1]	210	12992 / 249 / 250	1x	13491
Self-captured	378	652 / 82 / 83	20x	817
Web-sourced	382	1320 / 166 / 168	10x	1654
Thickened L-system	211	– / – / 500	1x	500
LVD w/ background	285	– / – / 500	1x	500
Tree synthetic	384	– / – / 500	1x	500
Root	71	– / – / 782	1x	782

are: Thickened L-system, LVD with background, Mtree, and Root datasets. We increased the maximum node count to 384 and standardized all images to 256×256 pixels to ensure consistent training and evaluation conditions. For all datasets, we constructed tree graphs by identifying key branch points and sampling them at uniform intervals of 13 pixels, thereby establishing a standardized representation of structural patterns.

Rationale for the split. The four out-of-domain test datasets are chosen to probe distinct generalization axes that are under-represented or absent in training:

- **Organ shift.** *Root dataset* introduces transfer across organs, a setting absent from both training and testing.
- **Style shift.** *Thickened L-system dataset* produces straight, thick strokes on clean backgrounds. It is close to, but not the same as, the LVD2021 vein mask, and serves as a proxy for sketches or hand-drawn line art.
- **Background shift.** *LVD with background dataset* composites vein-bearing leaves onto diverse natural scenes, increasing clutter and illumination variability relative to the plain LVD2021 images.

- **Shape and species diversity.** *Mtree dataset* generates irregular shoot architectures across many parameterized species and growth patterns, stressing dataset-level shape diversity and robustness.

2.5 Conclusion

PlantPose dataset establishes the first unified benchmark for plant skeleton estimation that spans species, organs, imaging styles, and environmental contexts. Consolidating ten heterogeneous subsets into a single canonical representation provides domain-specific settings for species-focused analysis and generalized protocols for cross-domain evaluation. Unlike existing datasets that supervise only local structures (per-branch polylines, node cues, or masks), the PlantPose dataset provides complete plant-level tree graphs with a canonical representation with uniform arclength sampling. The PlantPose dataset will be a foundation for advancing structure-aware plant modeling, enabling fair comparison across methods, and fostering new research on cross-domain generalization in graph-based visual understanding.

Chapter 3

PlantPose2D

Accurate estimation of plant skeletal structures (*e.g.*, branching structures) from images is essential for smart agriculture and plant science. Unlike human skeletons with fixed topology, plant skeleton estimation presents a unique challenge, *i.e.*, estimating arbitrary tree graphs from images. To address this problem, we introduce **PlantPose2D**, a universal plant skeleton estimator via tree-constrained graph generation. PlantPose2D combines learning-based graph generation with traditional graph algorithms to enforce tree constraints during the training loop. To enhance the model' s generalization capability, we curate a large and diverse dataset comprising real-world and synthetic plant images, along with simplified representations (*e.g.*, sketches and abstract drawings). This dataset enables the generalized model to adapt to diverse input styles and categories of plant images while preserving topological consistency. Our approach demonstrates robust and accurate plant skeleton estimation across multiple domains, including previously unseen out-of-domain scenarios. Further analyses highlight the method's strengths and limitations in handling complex, heterogeneous data distributions.

3.1 Introduction

Skeletal structures of plants (*e.g.*, branches and roots) are key information for analyzing plant traits in agriculture and plant science. In particular, single-view estimation of plant skeletons has potential benefits for various downstream tasks, such as high-throughput plant phenotyping [2, 9, 10] and plant organ segmentation [40, 41]. As a similar task, single-view estimation of human poses has been widely studied, *e.g.*, OpenPose [6]. However, unlike human skeletons, which have a fixed graph topology, the plant skeleton is not organized because the number of joints and their relationships are unknown, posing a unique problem of estimating an arbitrary tree graph from an image.

The estimation of graph structure from images has been studied to extract thin structures such as road networks in satellite images [42–44]. Recent end-to-end models using recurrent neural networks (RNNs) [19], graph neural networks (GNNs) [20–22], or transformers [4, 5, 23–25] show the ability to extract faithful *unconstrained* graph structures from images. However, inferring tree-constrained graphs with the existing graph generators becomes a non-trivial problem, where the output graph often violates the required constraints, as shown in Figure 3.1a. One reason for this difficulty is that tree graph generation, which requires finding a set of graph edges that satisfy the constraint defined on the entire graph, naturally falls into combinatorial optimization. Although constrained optimization in neural networks has been studied by designing differentiable layers [32], using them to solve our problem is still not straightforward. A simple way to impose constraints on the graph generation is to convert the inferred unconstrained graphs to the closest graph that satisfies the given constraint using traditional graph algorithms such as Dijkstra's shortest path or minimum spanning tree (MST) algorithms. Such post-processing can work; however, because the graph generators are trained without any constraints, the output may be unrealistic, as shown in Figure 3.1b.

For tree graph generation from single images, we propose a simple yet effective way

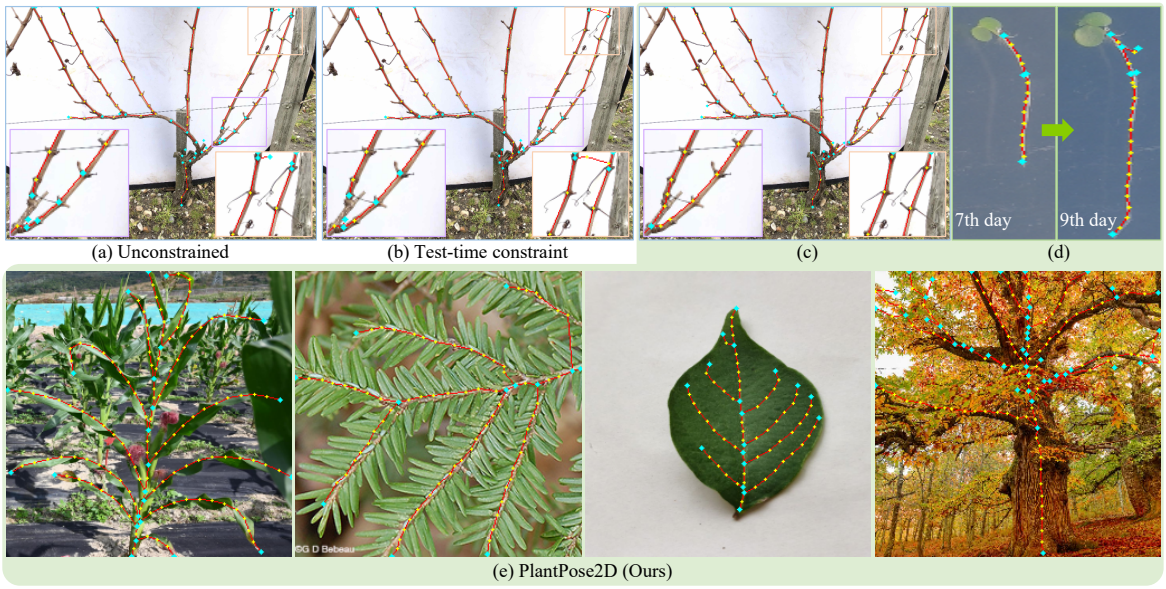


Fig. 3.1 We propose **PlantPose2D**, a method for single-image plant skeleton estimation combining learning-based graph generators with traditional graph algorithms (*i.e.*, MST). The red lines show the predicted graph edges. Compared to (a) an *unconstrained* graph generator and (b) a naive tree-graph constraint implementation, (c) our method naturally imposes the constraint during the graph generation models' training. Our method can be directly applied to plant science and agricultural applications, such as (d) time-series reconstruction of botanical roots. Furthermore, the PlantPose2D with large-scale training demonstrates strong generalization capabilities across diverse datasets, including significantly different out-of-domain plant species, as shown in (e).

to integrate state-of-the-art learning-based graph generation methods, which achieve high-quality image-based graph estimation, and traditional graph algorithms, which compute strictly constrained tree graphs, such as shown in Figs. 3.1c–d. Specifically, we propose to *project* an unconstrained graph into a tree graph by a non-differentiable MST algorithm during each training loop. Our selective feature suppression (SFS) layer then converts the inferred unconstrained graph to the MST-based tree graph in a differentiable manner, thereby naturally incorporating the constraints into the graph generation.

By integrating our feature suppression layer with a state-of-the-art transformer-

based graph generator, we develop **PlantPose2D**[1], which infers tree structures from images capturing plants. Beyond the majority of plant-oriented vision methods that use domain-specific (*i.e.*, species-specific) datasets for the training, we achieve a generalized model that estimates the plant skeletons for varieties of plants, as highlighted in Figure 3.1e. To this goal, we curate or collect 10 datasets spanning real-world, synthetic, and web-sourced plant images, ensuring the model is exposed to various structural patterns and contexts.

We evaluate the effectiveness of PlantPose2D on different classes of plant images, including out-of-domain datasets that are significantly different from those used for training. The results show that our constraint-aware graph generator accurately estimates the target tree structures compared to baselines.

In summary, we develop PlantPose2D, the first end-to-end image-to-graph model that enforces tree topology during training. This method tightly integrates learning-based graph generation with traditional graph algorithms through the proposed SFS layer, which differentiably mimics the behavior of minimum spanning tree projection. As a result, PlantPose2D achieves robust skeleton estimation across diverse plant datasets, demonstrating high tree rates and strong cross-domain generalization. Together with the PlantPose dataset introduced in Chapter 2, PlantPose2D establishes a solid foundation for learning connectivity-aware plant skeletons directly from images.

3.2 Related Work

We propose constraining the graph structures given by image-based graph generators, whose primary goal is plant skeleton estimation. We, therefore, introduce the related work of plant skeleton estimation, graph generation from images, and constrained optimization for neural networks.

[1]This terminology is, of course, an analogy of OpenPose [6] for plants.

3.2.1 Plant Skeleton Estimation

Plant skeleton estimation is actively studied since it becomes a fundamental technique for downstream tasks related to plant phenotyping and cultivation [7].

3D plant skeleton estimation Several methods are proposed to derive plant skeletons from 3D observations [7]. These methods often use point clouds acquired by LiDAR [17, 45] or multi-view stereo (MVS) [10, 18]. Regardless of the 3D acquisition method, these works generally use a two-stage pipeline: Skeletonization [28] followed by graph optimization using MST or Dijkstra's algorithm [29, 41, 46–48], where the graph algorithms are required to convert a set of skeleton positions into a graph.

2D plant skeleton estimation Compared to 3D methods, skeleton estimation from a single 2D image poses significant technical challenges due to the lack of depth information and severe occlusions despite the simplicity of data acquisition. Like 3D methods, existing 2D methods use a two-stage process involving skeletonization and graph optimization. To extract the skeleton regions on 2D images, plant region segmentation is often used for plants with relatively thin leaves [15]. Similarly, a neural network that converts an input image into a map representing 2D skeleton positions is used to mitigate the occlusions [16]. To reason about the direction of intersecting branches, a recent work [2] proposes to use vector fields representing branch direction instead of mask images, similar to the Part Affinity Fields (PAFs) used in OpenPose [6].

Unlike existing two-stage methods, we propose an end-to-end method that directly infers a tree graph representing plant skeletons in a single image. Our experiments show that our end-to-end method achieves better accuracy than a recent two-stage method for 2D images.

3.2.2 Graph Generation from Images

Graph generation from images, sometimes called image-to-graph generation, is studied for extracting thin structures, such as road networks or retinal vessels, or rela-

tions (*e.g.*, scene graphs) from images [4, 5, 19–25, 42–44, 49]. Recent learning-based methods often use object detectors, which detect graph nodes (*e.g.*, intersections in road networks) from images, and then aggregate the combinations of node features to predict the edges defined between two nodes as binary (*i.e.*, existence of edges) or categorical (*e.g.*, classification of edge relations) values. Some studies use external knowledge [50–53] to improve the results.

Graph generators have usually taken autoregressive methods (*e.g.*, [5, 54–57]) that output a graph by starting at an initial node and estimating neighboring graph nodes. Recent GNNs and transformers enable non-autoregressive graph generators [24, 49, 58–61] simultaneously estimating the entire graph. Autoregressive methods are prone to errors during the estimation process, and the state-of-the-art non-autoregressive method, RelationFormer [4], performs better than autoregressive methods, especially for medium to large graphs.

A few recent studies consider graph generation with tree-graph constraints in a different context. For the molecule structure estimation [62–64], these methods assume autoregressive graph generation, making it hard to work with complex and relatively large graphs like botanical plants.

3.2.3 Constrained Optimization for Neural Networks

Constrained optimization is crucial for machine learning. In particular, introducing constraints in neural networks has become a recent trend [32].

Designing differentiable layers The most direct way to introduce additional constraints to neural networks is to make the constraints differentiable. In the continuous domain, it is known that a convex optimization can be implemented as a differentiable layer [65]. However, the design of differentiable layers for combinatorial optimization poses a significant challenge due to the difficulty of differentiation. Wilder *et al.* [66] propose a differentiable layer for linear programming (LP) problems using continuous relaxation. This method is extended to mixed integer linear pro-

gramming (MILP) [67] by splitting the problem into multiple LPs. MST, which we want to use as constraints, is known to be transformed into the class of MILP [33, 34]. However, using differentiable layers for these complex combinatorial problems requires exponential computation time [32] to obtain the exact solution and is practically unrealistic.

Reparameterization for constrained optimization If the constraint function is difficult to differentiate, a simple alternative is to *project* unconstrained inferences or model parameters into constrained space, which can be considered a use of reparameterization [68] for constrained optimization.

In gradient descent optimization, methods projecting unconstrained optimization parameters (*e.g.*, model parameters in neural networks) to the closest ones satisfying the given constraint are called projected gradient descent (PGD). PGD is often used for traditional optimization problems, directly optimizing the input variables [69, 70]. While PGD can be used for neural network optimization, such as for generating adversarial examples [71], designing projection functions for neural networks is challenging. It requires mapping a large number of model parameters into a space satisfying complex constraints, where the constraints are often more naturally defined on the model output.

Instead of designing a projection function for the model parameters, the model's output can be projected onto the subspace that satisfies the constraints during the training loop. Since reparameterization for model output can be easily integrated with existing neural network models, they are often used for domain-specific applications such as coded aperture optimization with hardware constraint [72] and internal organ segmentation with given parametric shape models [73, 74]. We take this approach in our SFS layer, easily plugging it into off-the-shelf end-to-end graph generation methods without preparing the differentiable implementation of the constraints (*i.e.*, MST algorithm).

3.3 Selective Feature Suppression Layer

We here describe the constrained graph generation method. Figure 3.2 summarizes the proposed SFS layer, which casts the original unconstrained edge probabilities to the constrained domain.

3.3.1 Problem Statement

We here consider a simple setting of neural-network-based graph generation, where the model outputs the prediction of the edge probabilities (*i.e.*, the edge exists or not) defined for a pair of nodes, while this can be extended to a multi-class classification setting straightforwardly.

Our goal is to design a tree-constrained graph generator $\mathscr{F}$ that converts a given image I to a tree graph G as

$$G = (V, E) = \mathscr{F}(I) \quad \text{s.t.} \quad E \in E_{tree}, \tag{3.1}$$

where the graph G consists of a set of nodes (or objects) V and edges (or relations) E. Here, E_{tree} denotes all possible edge patterns forming a tree graph given the set of nodes V.

We consider a (non-differentiable) projection function $\mathscr{P}$ that maps an unconstrained graph predicted by graph generators to the constrained graph G. Let the edge probabilities defined between each node pairs as $\{\hat{\mathbf{y}}_{(i,j)}\}_{(i,j)\in V\times V}$.

$$\hat{\mathbf{y}}_{(i,j)} = [\hat{y}^{+}_{(i,j)}, \hat{y}^{-}_{(i,j)}]^{\top} \quad \text{s.t.} \quad \|\hat{\mathbf{y}}_{(i,j)}\|_1 = 1, \tag{3.2}$$

in which $\hat{y}^{+}_{(i,j)}$ and $\hat{y}^{-}_{(i,j)}$ respectively denote the edge existence and non-existence probabilities. The projection function $\mathscr{P}$ then read as

$$(V, E) = \mathscr{P}_{E\in E_{tree}}(V, \{\hat{\mathbf{y}}_{(i,j)}\}), \tag{3.3}$$

which are given by traditional graph algorithms with combinatorial optimization. We assume the projection function $\mathscr{P}$ converts the existence probability (or category prediction) of graph edges while leaving the graph nodes V unchanged. A typical example of $\mathscr{P}$ can be designed using the MST algorithm we use in our PlantPose2D implementation, which projects an arbitrary graph into a tree structure by modifying the existence of graph edges based on the costs defined between each pair of nodes.

We aim to develop a differentiable function $\mathscr{R}$ that mimics the non-differentiable projection $\mathscr{P}$. Plugging with the *unconstrained* graph generator $\hat{\mathscr{F}}$, Equation (3.1) is rewritten as

$$G = (V, E) = \mathscr{R}_{E \in E_{tree}}(V, \{\hat{\mathbf{y}}_{(i,j)}\}), \quad (V, \{\hat{\mathbf{y}}_{(i,j)}\}) = \hat{\mathscr{F}}(I), \tag{3.4}$$

where the whole process is differentiable.

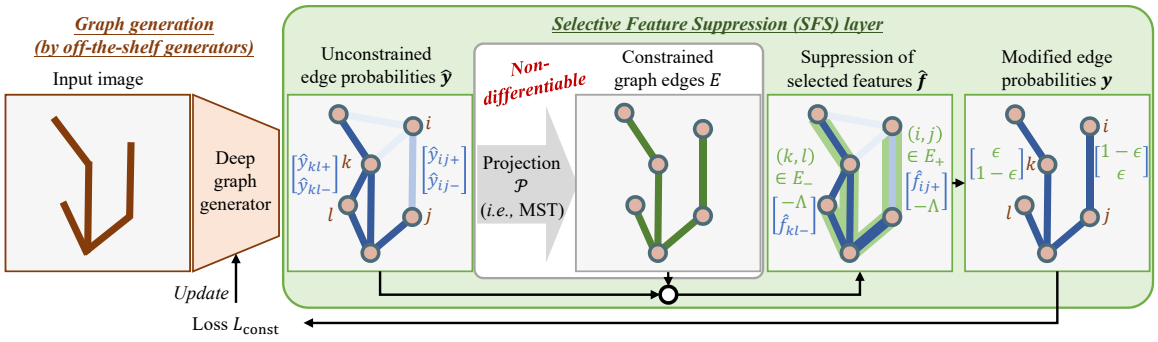


Fig. 3.2 Overview of reparameterization layer that can be easily plugged into off-the-shelf graph generators. Given unconstrained edge predictions by graph generators, our method *projects* it to the closest constrained graph (*i.e.*, tree) using a non-differentiable MST algorithm. Comparing constrained and unconstrained edges, unwanted edge features are selectively suppressed so that the graph becomes a tree.

3.3.2 SFS Layer

Overview We first briefly summarize the intuition of the SFS layer. Naive neural graph generators predict unconstrained edges between node pairs, which may include cycles or disconnected parts. The MST projection converts these predictions into a

valid tree by adding or removing edges. Our key idea is to mimic this projection inside the network in a differentiable way.

We here suppose the multi-head binary classification of edge existence between two given nodes, *i.e.*, the features of the last layer outputs both edge and non-edge features, followed by a softmax layer deciding whether the node pair has an edge. Our SFS layer *selectively* suppresses the features in the last layer, *i.e.*, for graph edges that should be added, we suppress the non-edge feature so that the softmax activation function outputs the edge; for edges that should be removed, we suppress the edge feature instead. All other edges remain unchanged. This selective suppression ensures that the resulting edge existence matches the tree structure given by the MST, while still allowing gradients to flow back to the backbone network.

Formulations Here, we describe an implementation of the SFS layer for constrained graph generation. As described in Equation (3.2), the unconstrained graph generator $\hat{\mathscr{F}}$ computes the probability of *unconstrained* edge existence between the i-th and j-th nodes, $\hat{\mathbf{y}}_{(i,j)} = [\hat{y}^+_{(i,j)}, \hat{y}^-_{(i,j)}]^\top$. In neural networks, $\hat{\mathbf{y}}_{(i,j)}$ is usually computed through the softmax activation σ applied to the output feature vector of the final layer $\hat{\mathbf{f}}_{(i,j)} = [\hat{f}^+_{(i,j)}, \hat{f}^-_{(i,j)}]^\top \in \mathbb{R}^2$ as

$$\hat{\mathbf{y}}_{(i,j)} = \sigma(\hat{\mathbf{f}}_{(i,j)}). \tag{3.5}$$

The set of unconstrained graph edges $\hat{E}$ is then obtained by comparing the edge existence probabilities as

$$\hat{E} = \{(i,j) \mid \hat{y}^+_{(i,j)} > \hat{y}^-_{(i,j)}\}, \tag{3.6}$$

in which $\hat{E}$ records node pairs where the edge exists.

Suppose the projection function $\mathscr{P}$ converts the set of unconstrained edge probabilities $\{\hat{\mathbf{y}}_{(i,j)}\}$ to a set of constrained edges E. Let the difference of two sets be $E^+ = E - \hat{E}$ and $E^- = \hat{E} - E$, denoting the sets of edges newly added and removed by the projection. To mimic discrete (and non-differentiable) inferences by $\mathscr{P}$ in differentiable end-to-end learning, we modify the edge features corresponding to $E^+ \cup E^-$

in the differentiable forward process.

Specifically, what we want to get is the edge probabilities that approximate the constrained edges E, denoted as

$$\mathbf{y}_{(i,j)} = \begin{cases} [1-\epsilon, \quad \epsilon \quad]^\top & ((i,j) \in E^+) \\ [\quad \epsilon \quad, 1-\epsilon]^\top & ((i,j) \in E^-) \\ \left[\hat{y}^+_{(i,j)}, \hat{y}^-_{(i,j)}\right]^\top & (\text{otherwise}). \end{cases} \tag{3.7}$$

When ϵ is small enough, the constrained output $\mathbf{y}_{(i,j)}$ perfectly mimics the output by the projection function $\mathscr{P}$. However, the direct modification of the edge probabilities naturally disconnects the computation graph. Therefore, we modify the unconstrained feature vector $\hat{\mathbf{f}}_{(i,j)}$ so that the corresponding edge probabilities $\mathbf{y}_{(i,j)}$ follows Equation (3.7). Specifically, since $\mathbf{y}_{(i,j)}$ is computed through the softmax function σ, it is achieved via the following minimal modification that *selectively* suppresses the feature values by replacing them with a constant[2] as

$$\begin{aligned} f^-_{(i,j)} &:= -\Lambda \qquad ((i,j) \in E^+) \\ f^+_{(i,j)} &:= -\Lambda \qquad ((i,j) \in E^-), \end{aligned} \tag{3.8}$$

where Λ is assumed to be large enough to make $\exp(-\Lambda) \sim 0$. Given modified features $\mathbf{f}_{(i,j)} = [f^+_{(i,j)}, f^-_{(i,j)}]^\top$, the softmax activation σ normalizes and converts them to edge probability $\mathbf{y}_{(i,j)}$.

In summary, from Eqs. (3.5) and (3.8), the constrained edge prediction between i-th and j-th nodes, $\mathbf{y}_{ij} = [y_{ij+}, y_{ij-}]^\top$, is obtained as

$$\mathbf{y}_{(i,j)} = \begin{cases} \sigma([\hat{f}^+_{(i,j)}, \ -\Lambda \ ]^\top) & ((i,j) \in E^+) \\ \sigma([\ -\Lambda \ , \hat{f}^-_{(i,j)}]^\top) & ((i,j) \in E^-) \\ \sigma([\hat{f}^+_{(i,j)}, \hat{f}^-_{(i,j)}]^\top) & (\text{otherwise}). \end{cases} \tag{3.9}$$

After the reparameterization, the set of edges computed from $\{\mathbf{y}_{(i,j)}\}$ in the same way

[2]See the following Section 3.4 for the derivation.

as Equation (3.6) is guaranteed to be equal to E inferred by the discrete projection function $\mathscr{P}$ when Λ is large enough.

3.4 Analysis of the SFS Layer

This section complements Section 3.3 by providing detailed motivation, gradient behavior, derivations, and case-by-case proofs of the SFS layer. Readers focused on implementation details may skip to Section 3.5.

3.4.1 Motivation

Our method infers a tree graph via MST as formulated in Equation (3.1)– Equation (3.4). Our objective is to optimize the graph generation network so that the **final output (*i.e.*, tree graph via MST) becomes similar to the ground-truth tree graph**. Since MST modifies the edge availability in *unconstrained* inferences, the unconstrained methods evaluating the unconstrained graph edges are indirect. Instead, our method **directly evaluates the quality of the final output tree** by mimicking MST. While experiments highlight our method's benefit, the following theoretical analysis also supports this intuition.

3.4.2 Gradient Behavior

The common auto differentiation libraries automatically compute the gradient of the SFS layer. Although it can disconnect the computation path at a feature, since we keep at least one of the original features (either $\hat{f}^{+}_{(i,j)}$ or $\hat{f}^{-}_{(i,j)}$), the backpropagation path to the backbone graph generation network is not disconnected[3]. Here, we briefly analyze the behavior of the SFS layer. The following subsection provides a detailed analysis, including mathematical proofs.

When using the cross-entropy loss $\mathscr{L}_{\mathrm{CE}}$ to evaluate the availability of the graph edges, the derivative to be backpropagated to the backbone graph generator is ap-

[3]This is akin to the dropout layer often used in neural networks.

proximated as[4]

$$\frac{\partial \mathscr{L}_{\mathrm{CE}}}{\partial \hat{\mathbf{f}}} \sim \begin{cases} [\ 1-t^+\ ,\quad 0\quad\ ]^\top & ((i,j)\in E^+) \\ [\quad 0\quad ,\ 1-t^-\ ]^\top & ((i,j)\in E^-) \\ [y^+-t^+, y^--t^-]^\top & (\text{otherwise}), \end{cases} \tag{3.10}$$

where $\mathbf{t} = [t^+, t^-]^\top \in \{0,1\}^2$ denotes the ground truth edge existence and non-existence for the node pair (i,j). Our method modifies the computation graph of the network when the MST algorithm disagrees with the output of graph generation model (*i.e.*, $(i,j) \in E^+ \cup E^-$), but in different ways for derivatives of each feature value $\frac{\partial \mathscr{L}_{\mathrm{CE}}}{\partial \hat{f}^+}$ or $\frac{\partial \mathscr{L}_{\mathrm{CE}}}{\partial \hat{f}^-}$.

Without loss of generality, we consider the case when the MST algorithm *adds* an edge, *i.e.*, $(i,j) \in E^+$. When the MST *correctly* modify the edge availability (*i.e.*, $t^+ = 1$), the gradient vector becomes small, $\frac{\partial \mathscr{L}_{\mathrm{CE}}}{\partial \hat{\mathbf{f}}} \sim \mathbf{0}$, which is the behavior we expect. On the other hand, if the MST *incorrectly* adds the edge (*i.e.*, $t^+ = 0$), the gradient becomes $[1,0]^\top$, which strongly penalizes the positive edge probability, where the norm of the gradient vector is always larger than unconstrained ones. Therefore, the behavior of our simple reparameterization strategy is reasonable in practice.

3.4.3 Formal Derivation

As discussed in the Section 3.3, we consider the edge probabilities $\hat{\mathbf{y}}_{(i,j)} = [\hat{y}^+_{(i,j)}, \hat{y}^-_{(i,j)}]^\top$ is usually computed through the softmax activation σ applied to the output feature vector of the final layer $\hat{\mathbf{f}}_{(i,j)} = [\hat{f}^+_{(i,j)}, \hat{f}^-_{(i,j)}]^\top$ as

$$\hat{\mathbf{y}}_{(i,j)} = \sigma(\hat{\mathbf{f}}_{(i,j)}) = \left[\frac{\exp(\hat{f}^+_{(i,j)})}{\exp(\hat{f}^+_{(i,j)})+\exp(\hat{f}^-_{(i,j)})}, \frac{\exp(\hat{f}^-_{(i,j)})}{\exp(\hat{f}^+_{(i,j)})+\exp(\hat{f}^-_{(i,j)})} \right]^\top. \tag{3.11}$$

The set of unconstrained graph edges $\hat{E}$ is then obtained by comparing the edge

[4]We omit the subscript (i,j) for simplicity.

existence probabilities as

$$\hat{E} = \{(i,j) \mid \hat{y}^{+}_{(i,j)} > \hat{y}^{-}_{(i,j)}\}, \tag{3.12}$$

in which $\hat{E}$ records node pairs where the edge exists.

Suppose the projection function $\mathscr{P}$ converts the set of unconstrained edge probabilities $\{\hat{\mathbf{y}}_{(i,j)}\}$ to a set of constrained edges E. Let the difference of two sets be $E^{+} = E - \hat{E}$ and $E^{-} = \hat{E} - E$, denoting the sets of edges newly added and removed by the projection. To mimic the discrete (and non-differentiable) inferences by $\mathscr{P}$ in the differentiable end-to-end learning, we modify the edge features corresponding to $E^{+} \cup E^{-}$ in the differentiable forward process. Here, we want to get the edge probabilities that approximate the constrained edges E, which can be denoted as

$$\mathbf{y}_{(i,j)} = \begin{cases} [\ 1\ ,\ 0\ ]^{\top} & ((i,j) \in E^{+}) \\ [\ 0\ ,\ 1\ ]^{\top} & ((i,j) \in E^{-}) \\ \left[\hat{y}^{+}_{(i,j)}, \hat{y}^{-}_{(i,j)}\right]^{\top} & (\text{otherwise}). \end{cases} \tag{3.13}$$

$$\sim \begin{cases} [1-\epsilon,\ \ \epsilon\ ]^{\top} & ((i,j) \in E^{+}) \\ [\ \epsilon\ ,1-\epsilon]^{\top} & ((i,j) \in E^{-}) \\ \left[\hat{y}^{+}_{(i,j)}, \hat{y}^{-}_{(i,j)}\right]^{\top} & (\text{otherwise}). \end{cases} \tag{3.14}$$

When ϵ is small enough, the constrained output $\mathbf{y}_{(i,j)}$ perfectly mimics the output by the projection function $\mathscr{P}$. Our goal is to modify the feature vector $\hat{\mathbf{f}}_{(i,j)}$ so that it makes the probabilities as Equation (3.14) through the softmax activation.

In the SFS layer, we replace the features as

$$\begin{aligned} f^{-}_{(i,j)} &:= -\Lambda \qquad ((i,j) \in E^{+}) \\ f^{+}_{(i,j)} &:= -\Lambda \qquad ((i,j) \in E^{-}), \end{aligned} \tag{3.15}$$

where Λ is assumed to be large enough. Given modified features $\mathbf{f}_{(i,j)} = [f^{+}_{(i,j)}, f^{-}_{(i,j)}]^{\top}$,

the softmax activation σ normalizes and converts them to edge probability $\mathbf{y}_{(i,j)}$ as

$$\mathbf{y}_{(i,j)} = \begin{cases} \sigma([\hat{f}^{+}_{(i,j)},\ -\Lambda\,]^\top) & ((i,j) \in E^{+}) \\ \sigma([\ -\Lambda\,, \hat{f}^{-}_{(i,j)}]^\top) & ((i,j) \in E^{-}) \\ \sigma([\hat{f}^{+}_{(i,j)}, \hat{f}^{-}_{(i,j)}]^\top) & (\text{otherwise}). \end{cases} \tag{3.16}$$

Without loss of generality, we discuss the case in $(i,j) \in E^{+}$. Substituting Equation (3.16) into Equation (3.11) yields

$$\begin{aligned} \mathbf{y}_{(i,j)} &= \left[\frac{\exp(\hat{f}^{+}_{(i,j)})}{\exp(\hat{f}^{+}_{(i,j)})+\exp(-\Lambda)}, \frac{\exp(-\Lambda)}{\exp(\hat{f}^{+}_{(i,j)})+\exp(-\Lambda)}\right]^\top \\ &= \left[\frac{\exp(\hat{f}^{+}_{(i,j)})}{\exp(\hat{f}^{+}_{(i,j)})+\epsilon'}, \frac{\epsilon'}{\exp(\hat{f}^{+}_{(i,j)})+\epsilon'}\right]^\top \qquad ((i,j) \in E^{+}), \end{aligned}$$

where $\epsilon' = \exp(-\Lambda) \sim 0$ when Λ is large enough, leading to $\mathbf{y}_{(i,j)} = [1-\epsilon, \epsilon]^\top$ as in Equation (3.14) by denoting $\epsilon = \frac{\epsilon'}{\exp(\hat{f}^{+}_{(i,j)})+\epsilon'} \sim 0$.

3.4.4 Case-by-case Proofs

We describe a detailed analysis of our reparameterization layer. As described in (3.9), the unconstrained edge feature between i and j-th nodes $\hat{\mathbf{f}}_{(i,j)} = [\hat{f}^{+}_{(i,j)}, \hat{f}^{-}_{(i,j)}]^\top$ is converted to constrained prediction of the edge availability $\mathbf{y}_{(i,j)} = [y^{+}_{(i,j)}, y^{-}_{(i,j)}]^\top$ by selectively suppressing unwanted feature values.

When using the cross-entropy loss $\mathscr{L}_{\mathrm{CE}}$ to evaluate the availability of the graph

Table 3.1 Case-by-case analyses of our reparameterization layer. For the columns of unconstrained features $\hat{\mathbf{f}}$, constrained prediction $\mathbf{y}$, and the ground truth edge availability $\mathbf{t}$, the table shows the index of the larger element. For example, the column $\hat{\mathbf{f}}$ will be + when the edge feature for the positive edge availability is larger, *i.e.*, $\hat{f^+} > \hat{f^-}$. The column (i, j) displays E^+ or E^- if the MST algorithm modifies the edge availability (in which the rows are also highlighted). For the remaining columns, $\uparrow$ and $\downarrow$ denote each value becoming (relatively) large or small, respectively.

Case	Feats & probs $\hat{\mathbf{f}}$	(i,j)	$\mathbf{y}$	GT $\mathbf{t}$	Loss $\mathscr{L}_{\text{CE}}$	Approx. derivatives $\frac{\partial \mathscr{L}_{\text{CE}}}{\partial \hat{f}^+}$	$\frac{\partial \mathscr{L}_{\text{CE}}}{\partial \hat{f}^-}$	$\left\lvert\frac{\partial \mathscr{L}_{\text{CE}}}{\partial \hat{f}^+}\right\rvert$	$\left\lvert\frac{\partial \mathscr{L}_{\text{CE}}}{\partial \hat{f}^-}\right\rvert$	Descriptions
1	+		+	$[1,0]^\top$	$\downarrow$	$y^+ - 1$	y^-	$\downarrow$	$\downarrow$	Unmodified
2	+		+	$[0,1]^\top$	$\uparrow$	y^+	$y^- - 1$	$\uparrow$	$\uparrow$	Unmodified
3	+	E^-	−	$[1,0]^\top$	$\uparrow$	0	1	$\downarrow$	$\uparrow$	MST *incorrectly* modified
4	+	E^-	−	$[0,1]^\top$	$\downarrow$	0	0	$\downarrow$	$\downarrow$	MST *correctly* modified
5	−		−	$[1,0]^\top$	$\uparrow$	$y^+ - 1$	y^-	$\uparrow$	$\uparrow$	Unmodified
6	−		−	$[0,1]^\top$	$\downarrow$	y^+	$y^- - 1$	$\downarrow$	$\downarrow$	Unmodified
7	−	E^+	+	$[1,0]^\top$	$\downarrow$	0	0	$\downarrow$	$\downarrow$	MST *correctly* modified
8	−	E^+	+	$[0,1]^\top$	$\uparrow$	1	0	$\uparrow$	$\downarrow$	MST *incorrectly* modified

edges, the derivative to be backpropagated to the backbone graph generator is [5]

$$\frac{\partial \mathscr{L}_{\text{CE}}}{\partial \hat{\mathbf{f}}} = \begin{cases} [(1-\epsilon) - t^+, \quad 0 \quad]^\top & ((i,j) \in E^+) \\ [\quad 0 \quad , (1-\epsilon) - t^-]^\top & ((i,j) \in E^-) \\ [\ y^+ - t^+ \quad , \quad y^- - t^- \quad]^\top & (\text{otherwise}), \end{cases} \tag{3.17}$$

$$\sim \begin{cases} [\ 1 - t^+ , \quad 0 \quad]^\top & ((i,j) \in E^+) \\ [\quad 0 \quad , \ 1 - t^- \ ]^\top & ((i,j) \in E^-) \\ [y^+ - t^+, y^- - t^-]^\top & (\text{otherwise}), \end{cases} \tag{3.18}$$

where $\mathbf{t} = [t^+, t^-]^\top$ denotes the ground truth edge existence and non-existence for the node pair (i, j). Our method modifies the computation graph of the network when the MST algorithm does not agree with the output of the graph generation model (*i.e.*, $(i,j) \in E^+ \cup E^-$), but in different ways for derivatives of each feature value $\frac{\partial \mathscr{L}_{\text{CE}}}{\partial \hat{f^+}}$ or $\frac{\partial \mathscr{L}_{\text{CE}}}{\partial \hat{f^-}}$.

Table 3.1 summarizes the case-by-case behavior, in which we can categorize the behaviors of the SFS layer into eight cases. Hereafter, we use $E^* \triangleq E^+ \cup E^-$.

[5] We omit the subscript (i, j) for simplicity.

Case $(i,j) \notin E^*$ *(Cases 1, 2, 5, 6)* When the graph algorithm (*i.e.*, MST) does not modify the edge availability (*i.e.*, cases 1, 2, 5, and 6 in the table), the behavior is the same as the usual cross-entropy loss for unconstrained edges.

Case $(i,j) \in E^*$ & $\mathbf{y} \sim \mathbf{t}$ *(Cases 4, 7)* In these cases, the MST algorithm *correctly* suppresses the unwanted features, where the constrained prediction $\mathbf{y}$ becomes the approximation of the ground-truth edge availability $\mathbf{t}$. The loss value becomes small, and the derivative is $\frac{\partial \mathscr{L}_{\mathrm{CE}}}{\partial \hat{\mathbf{f}}} \sim \mathbf{0}$. This is natural since our constrained graph generator produces correct predictions.

Case $(i,j) \in E^*$ & $\mathbf{y} \nsim \mathbf{t}$ *(Cases 3, 8)* In these cases, MST *incorrectly* modifies the edge availability, *i.e.*, the node pair (i,j) belongs to E^+ or E^-, but the constrained prediction $\mathbf{y}$ does not fit the ground truth $\mathbf{t}$. Here, we mathematically discuss the behavior in these cases. Without loss of generality, we focus on Case 3, where MST *incorrectly* removes an edge and compares the methods with and without tree-graph constraints using the SFS layer.

Case 3 (MST* incorrectly *removes an edge) The following discussions can be straightforwardly extended to Case 8, where MST *incorrectly* adds an edge.

Conditions:

- Unconstrained prediction (edge exists): $\hat{f}^+ > \hat{f}^-$,
- MST *removes* the edge: $(i,j) \in E^-$,
- GT edge availability (edge exists): $[t^+, t^-] = [1, 0]$.

Unconstrained method (without SFS layer) The gradient at $\hat{f}^+$ by the unconstrained method is

$$\frac{\partial \mathscr{L}_{\mathrm{unconst}}}{\partial \hat{f}^+} = \frac{\exp(\hat{f}^+)}{\exp(\hat{f}^+) + \exp(\hat{f}^-)} - 1. \tag{3.19}$$

Since $\hat{f}^+ > \hat{f}^-$, it takes the value in the range of $(-0.5, 0)$. Similarly,

$$\frac{\partial \mathscr{L}_{\text{unconst}}}{\partial \hat{f}^-} = \frac{\exp(f^-)}{\exp(f^-) + \exp(f^+)} - 0, \tag{3.20}$$

thus the gradient at $\hat{f}^-$ is inside $(0, 0.5)$. Thus, the gradient vector $\frac{\partial \mathscr{L}_{\text{unconst}}}{\partial \hat{\mathbf{f}}}$ is always shorter than $[-0.5, 0.5]^\top$ (corresponding to the special case $\hat{f}^+ = \hat{f}^-$).

Constrained method (Ours) As described in Eqs. (3.17) and (3.18), our method yields the gradient as

$$\frac{\partial \mathscr{L}_{\text{const}}}{\partial \hat{f}^+} = 0, \qquad \frac{\partial \mathscr{L}_{\text{const}}}{\partial \hat{f}^-} = 1 - \epsilon \sim 1,$$

i.e., $\frac{\partial \mathscr{L}_{\text{const}}}{\partial \hat{\mathbf{f}}} \sim [0, 1]^\top$.

Comparisons While both methods control the features to increase the edge availability, the relation of gradient vectors $\|\frac{\partial \mathscr{L}_{\text{const}}}{\partial \hat{\mathbf{f}}}\| > \|\frac{\partial \mathscr{L}_{\text{unconst}}}{\partial \hat{\mathbf{f}}}\|$ always holds, which means our method strongly penalizes the incorrect estimates by MST by directly comparing the final estimation (*i.e.*, tree graph) with the ground-truth edge availability, which highlights our key motivation—a **direct** control of the tree-constrained graph generation.

3.5 PlantPose2D Method

We develop PlantPose2D, an implementation of the SFS layer in a state-of-the-art graph generator. This section first recaps the graph generator we use [4] and then details how we introduce a tree structure constraint.

3.5.1 RelationFormer: A Brief Recap

RelationFormer [4] is the state-of-the-art non-autoregressive graph generation method. This method uses an end-to-end architecture that combines an object (node) detector

and relation (edge) predictor, which shows superior performance for unconstrained graph generation. The object detection part is based on deformable DETR [75], which is trained to extract graph nodes (*e.g.*, objects) and global features from a given image. Specifically, given the extracted image features, the transformer decoder outputs a fixed number of object queries ([obj]-tokens) representing each of the nodes and a relation query ([rtn]-token) describing the global features, including node relations.

The relation prediction head outputs the relationship (*i.e.*, edge existence or category) from the detected pairs of objects (*i.e.*, [obj]-tokens) and the global relation (*i.e.*, [rtn]-tokens). This module is implemented as a multi-layer perceptron (MLP) headed by layer normalization [76]. RelationFormer is trained using the sum of loss functions related to object detection and edge (relation) estimation, where edge (relation) loss $\mathscr{L}_{\text{edge}}$[6] evaluates the edge existence or category between node pairs using cross-entropy loss.

3.5.2 Tree-Constrained Graph Generation

To introduce the tree structure constraint, we use Kruskal's MST algorithm [77] implemented in NetworkX[7]. To extract a tree from an unconstrained graph predicted by RelationFormer, we use the edge non-existence probabilities $\{\hat{y}^{-}_{(i,j)}\}$ as the edge cost for the MST algorithm to span the tree on edges with higher existence probabilities.

We implement the SFS layer on top of the relation prediction head in the RelationFormer. Specifically, the output features from the MLP after layer normalization are regarded as unconstrained features $\{\hat{\mathbf{f}}\}$. In our experiments, we use $\Lambda = 10$ during training, where $\exp(-\Lambda) = 4.5 \times 10^{-5}$. We show an ablation study changing Λ in the following subsection.

Loss function Our SFS layer affects the evaluation of the edge loss L_{edge} in the graph generator, while the computation of other loss functions, such as for node detection, remains the same as in the original implementation. Our implementation

[6]Denoted as $\mathscr{L}_{\text{rln}}$ in the original paper [4], we use $\mathscr{L}_{\text{edge}}$ for generality.
[7]`https://networkx.org/`, last accessed on September 1, 2025.

uses both loss functions for original (unconstrained) and constrained edges. Denoting the ground-truth edges as $E_{\mathrm{GT}} = \{(i,j) \mid t^{+}_{(i,j)} > t^{-}_{(i,j)}\}$, where $\mathbf{t}_{(i,j)} = [t^{+}_{(i,j)}, t^{-}_{(i,j)}]^{\top} \in \{0,1\}^2$, the loss function for edge availability $\mathscr{L}_{\mathrm{edge}}$ is modified as follows

$$\mathscr{L}_{\mathrm{edge}} = \underbrace{\sum_{(i,j)} \mathscr{L}_{\mathrm{CE}}(\hat{\mathbf{y}}_{(i,j)}, \mathbf{t}_{(i,j)})}_{\mathscr{L}_{\mathrm{unconst}}} + \underbrace{\sum_{(i,j)} \mathscr{L}_{\mathrm{CE}}(\mathbf{y}_{(i,j)}, \mathbf{t}_{(i,j)})}_{\mathscr{L}_{\mathrm{const}}}, \tag{3.21}$$

where $\mathscr{L}_{\mathrm{CE}}$ denotes the cross-entropy loss and λ_{const} controls the weight of the loss for constrained edges $\mathscr{L}_{\mathrm{const}}$.

3.6 Experiments

Datasets used in PlantPose2D We refer readers to Chapter 2 for full dataset descriptions, annotation protocols, and preprocessing. For PlantPose2D, we use the following subsets from the PlantPose dataset:

Domain-specific setting: L-system, Root, Grapevine. **Generalized setting:** Training on: MIPDB, Grapevine, LVD2021 leaf images, LVD2021 vein masks, self-captured, web-sourced; Out-of-domain tests on: Thickened L-system, LVD with background, Mtree, Root.

All graphs follow the unified representation Chapter 2, and image/graph resolutions (*e.g.*, 256×256, 13-pixel sampling) are consistent with Chapter 2 unless otherwise noted.

We conduct experiments to validate the effectiveness of the proposed method in generalized scenarios using synthetic and real-world plant image datasets.

3.6.1 Baselines

Since the constrained graph generation task is new in this paper, there are few established baseline methods. We compare our method with the state-of-the-art methods for 2D plant structure estimation and unconstrained graph generation. Also, as an abla-

tion study, we compare a simpler alternative to our method. The following subsection provides additional comparisons with other baseline methods, including autoregressive graph generation.

Two-stage [2] We implement a 2D plant skeleton estimation method based on a two-stage method involving skeletonization and graph optimization with reference to [2]. Specifically, vector fields of branch directions are generated by a neural network, followed by graph optimization to generate branch structure, in which we find our implementation outperforms the naive re-implementation of the existing method [2]. Specific implementations and analyses are described in the following subsection.

Unconstrained [4] We compare the state-of-the-art (unconstrained) graph generation method, RelationFormer [4]. This method is identical to our method without applying the tree structure constraint.

Test-time constraint As a straightforward implementation of constrained graph generation, we apply MST only in the inference phase, where the graph generator is trained using the same procedure as the unconstrained method.

3.6.2 Details of the Two-Stage Method

Our experiment implements a two-stage baseline involving skeletonization and graph optimization. This baseline implementation is based on ViNet [2], a state-of-the-art plant skeleton estimation method. Since the implementation of [2] is not publicly available, we re-implement the method with reference to the descriptions in the paper. Through the re-implementation, we find room for improvement in the two-stage baseline method. Table 3.2 compares the performance of our two-stage implementation with a naive re-implementation of [2]. Moreover, we also compare the mean squared error (MSE) of the first-stage output of both methods. Our implementation achieves a better performance; thus, we use the improved version for our experiment. In the following, we describe the implementation details.

Table 3.2 Quantitative comparisons between our re-implementation of [2] and our two-stage baseline implementation.

Method	SMD ↓	TOPO score ↑			MSE ↓	
		Prec.	Rec.	F1	Node confidence	Edge direction
Re-implementation of [2]	3.84×10^{-3}	0.459	0.365	0.406	6.95×10^{-3}	1.01×10^{-2}
Our implementation of two-stage method	$\mathbf{4.24 \times 10^{-4}}$	**0.677**	**0.589**	**0.630**	$\mathbf{1.19 \times 10^{-3}}$	$\mathbf{2.67 \times 10^{-3}}$

First stage: Skeletonization Similar to ViNet [2], the first stage of our implementation outputs the prediction of node and edge positions as single-channel confidence maps and two-channel vector fields (hereafter referred to as node confidence maps and edge direction maps, respectively). This step is similar to a widespread human pose estimation method *i.e.*, OpenPose [6], which jointly estimates the confidence of person keypoints and the Part Affinity Fields (*i.e.*, two-channel vector fields).

While ViNet [2] uses a sequence of residual blocks followed by the Stacked Hourglass Network [78] for this stage, we use a pre-trained ResNet50 [79] for image feature extraction. This is for a fair comparison to our PlantPose2D implementation, which also uses ResNet50 as the backbone[8]. We implement an architecture like the Feature Pyramid Network (FPN) [80], illustrated in Figure 3.3, to decode the node & edge maps from the image features. Figure 3.4 visually compares the estimated node & edge maps, showing a better accuracy by our two-stage implementation.

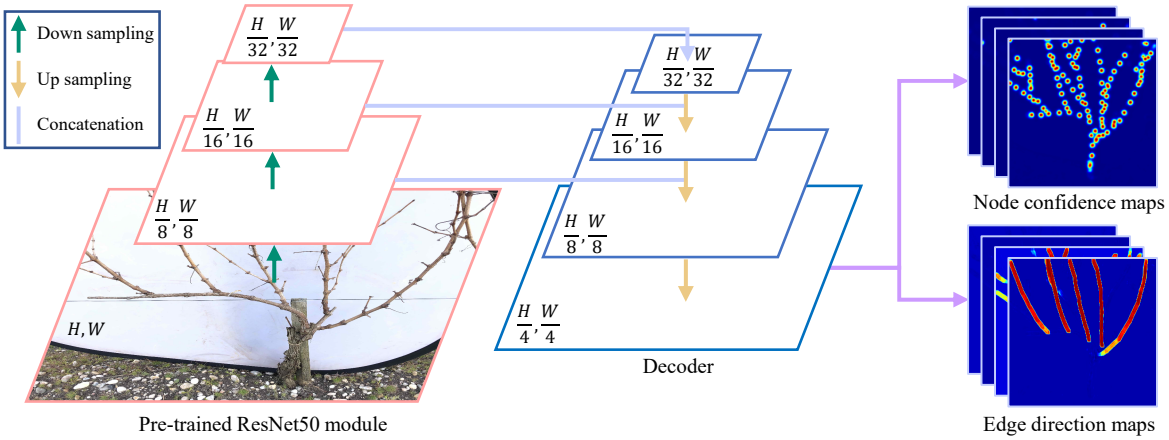


Fig. 3.3 Network architecture for the first (skeletonization) stage of our two-stage baseline method.

[8]ResNet50 is actually used as the node detection module in RelationFormer (that is based on Deformable DETR [75]), which is the basis of our PlantPose2D, and we inherited its implementation.

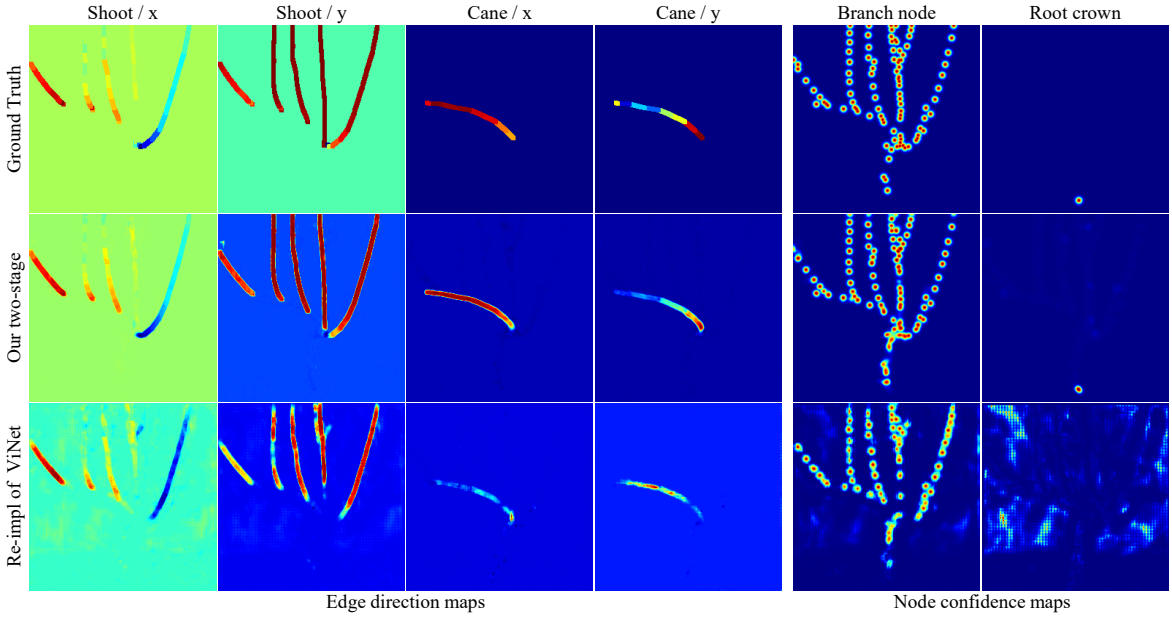


Fig. 3.4 Visual comparisons between a re-implementation of [2] and our two-stage baseline implementation. Our implementation yields better node confidence and edge direction maps, which are the outputs of the first stage of these methods.

The original ViNet estimates multiple classes of nodes (four classes) and edges (five classes) as different maps for the grapevine dataset. Compared to just using binary classes (*i.e.*, a branch exists or not), our two-stage implementation also yields better estimation accuracies using the multiple classes (SMD in 4.2×10^{-4} with multi-class and 1.4×10^{-2} with binary classes). Therefore, we use the multi-class setup for our two-stage implementation of the grapevine dataset. For the other dataset, we use binary classification since we do not have specific class information.

Second stage: Graph algorithm Given the node confidence and edge direction maps, ViNet [2] first extracts the node positions, followed by the computation of the *resistivity* between each node pair, defined using the edge directions and the Euclidean distance between nodes. The final estimates of the graph structure are generated using the Dijkstra algorithm, where the tree structure is obtained by computing the shortest paths from all nodes to the detected root crown. The resistivity is used as the edge cost for the Dijkstra algorithm.

For the second stage, we follow the method in [2] except for the graph algorithm used; namely, we compute MST instead of the shortest paths given by the Dijkstra algorithm, since using MST reduces the SMD metric to 4.2×10^{-4}, compared to 5.9×10^{-4} using the Dijkstra algorithm for the grapevine dataset. Figure 3.5 visually compares the graph algorithms. It also compares the number of classes, *i.e.*, the availability of binary edges and the multi-classes provided in the grapevine dataset as discussed above, showing that using the combination of multi-classes and the MST algorithm achieves better accuracies for the two-stage method.

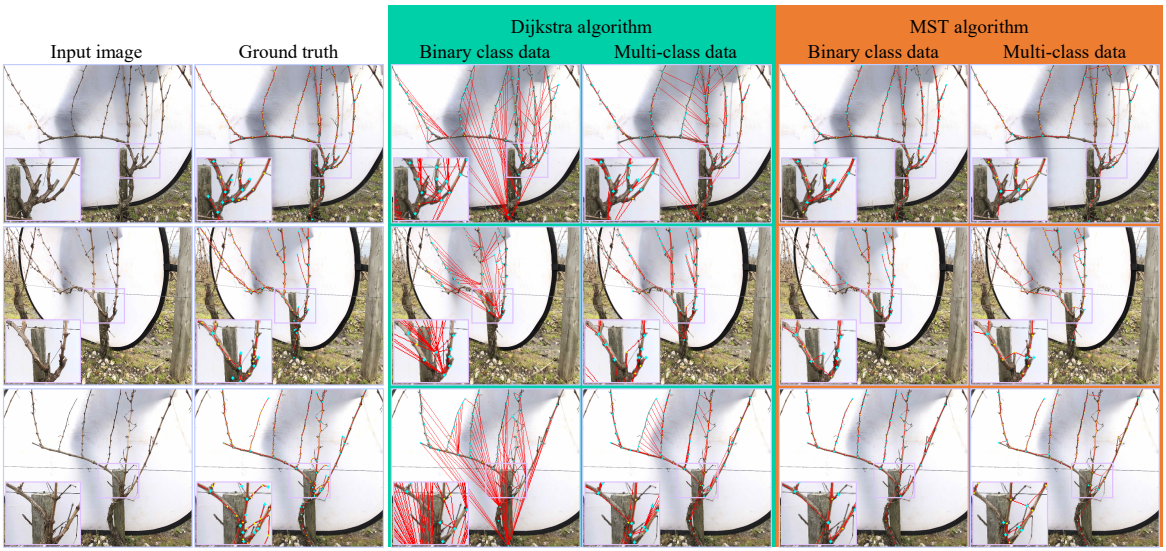


Fig. 3.5 Comparisons of graph algorithms (Dijkstra *vs.* MST) and classification labels (binary *vs.* five classes) for the two-stage method. We use the combination of MST and five class labels for the baseline in our experiment.

3.6.3 Evaluation Metrics

We use different metrics to capture spatial similarity alongside the topological similarity of the predicted graphs.

Street mover's distance (SMD) [5] SMD is a metric to assess the accuracy of the positions of graph edges, which is computed as the Wasserstein distance between the predicted and the ground truth edges. In our implementation, the distance is

computed between densely sampled points on the edges, which is the same procedure as in the original paper proposing the SMD [5].

TOPO score [81] We compute the TOPO scores to evaluate the topological mismatch of the output graph. This metric consists of the precision, recall, and F1 scores of the graph nodes, which are evaluated considering the edge topology. We use the implementation used in the Sat2Graph paper [42], while we only evaluate the nodes with the degree $\neq 2$ that affect the tree structure, *i.e.*, we only evaluate joint and leaf nodes in the graphs.

Tree rate To evaluate how well the output graph satisfies the constraint, we calculate the probability that the output graph forms a tree structure. While it is obvious that the tree rate becomes 100 % for constrained methods, including ours, we are interested in how well the output of the unconstrained graph generation model can reflect the constraint by training on datasets that contain only tree graphs.

3.6.4 Implementation Details

For RelationFormer in our method and the baseline comparison, we use the official implementation[9] on PyTorch. For other hyperparameters, we follow the original RelationFormer implementation used for road network extraction. We used early stopping for all datasets and methods by selecting the model with the best validation performance and terminating training after 30 epochs without improvement. The training of our method takes approximately 141 hours for the synthetic dataset, 10 hours for the root dataset, and 98 hours for the grapevine dataset, all conducted on eight NVIDIA RTX A100 GPUs.

[9] `https://github.com/suprosanna/relationformer`, last accessed on September 1, 2025.

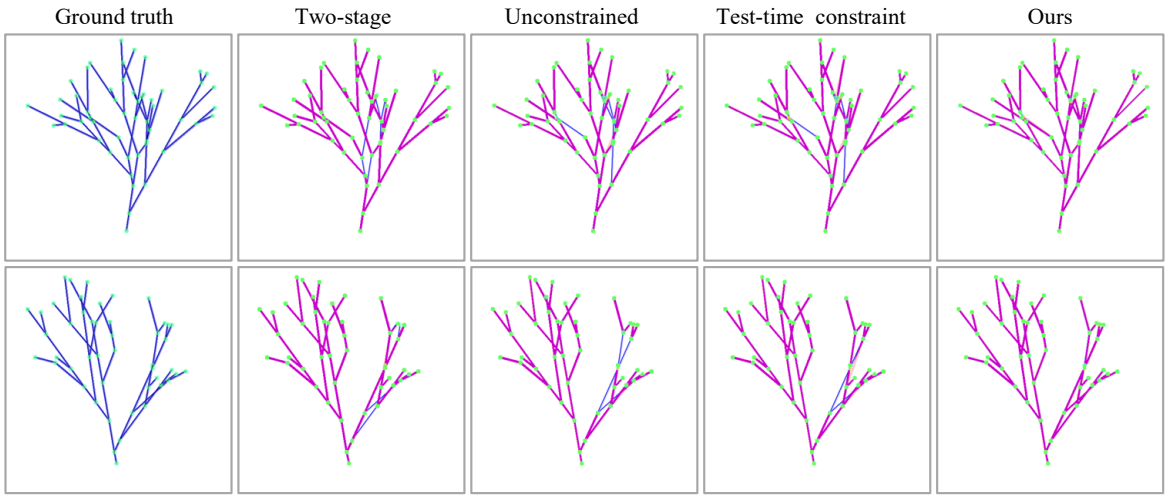
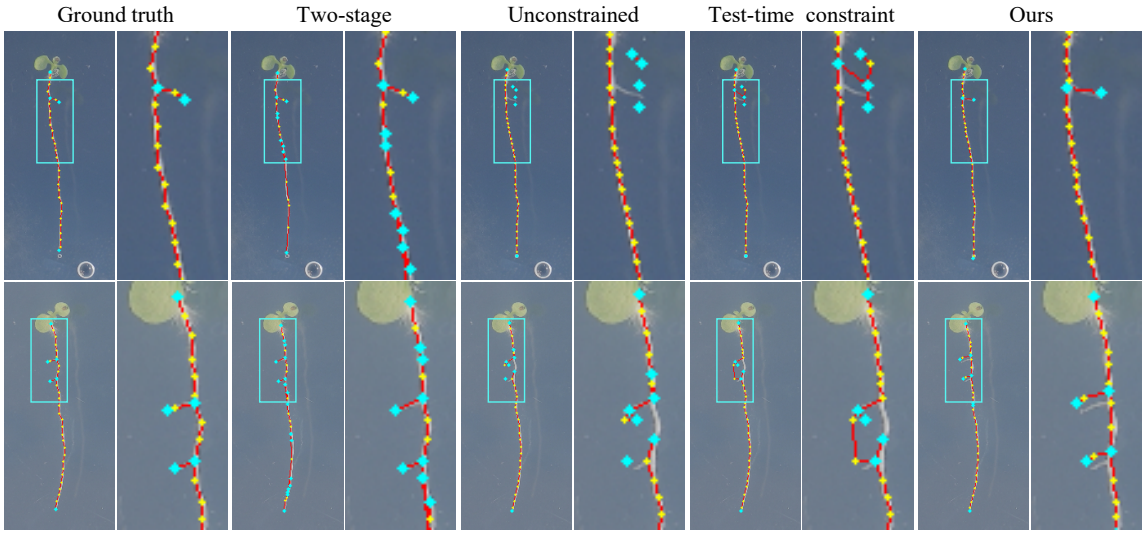


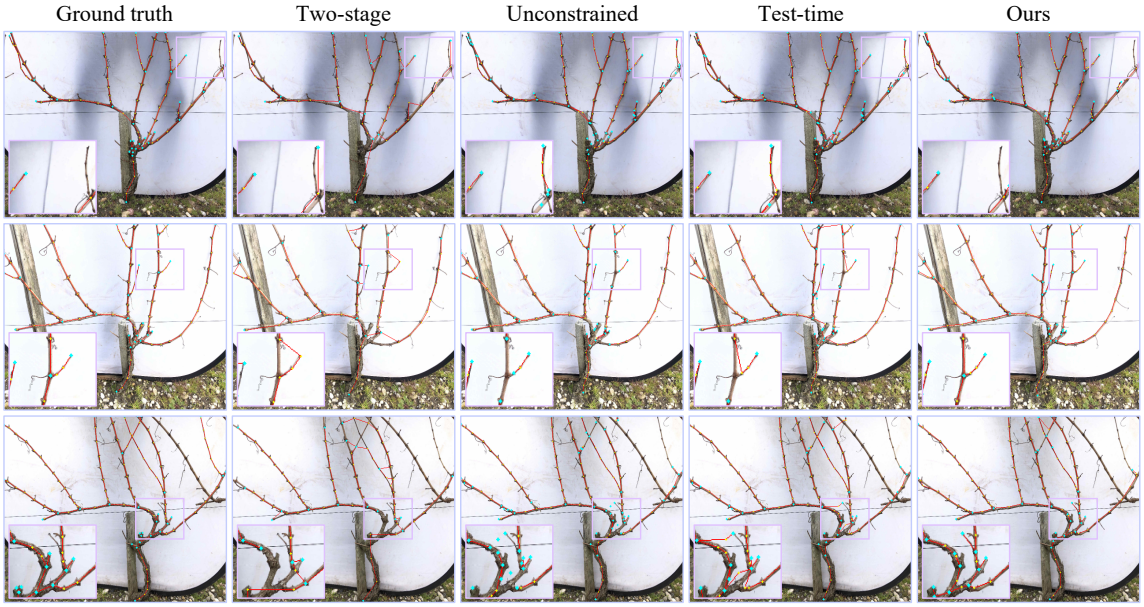
Fig. 3.6 Visual results for the domain-specific training with the synthetic tree pattern dataset. For the synthetic tree pattern dataset, we translucently overlay the estimated and ground truth edges with red and blue lines, respectively. While all methods accurately detect nodes, only our method accurately predicts the availability of edges from given images compared to the baseline methods.

Table 3.3 Quantitative results on domain-specific training. Our method significantly improves both the shape and topology of the predicted graph while enforcing the given constraints. The best scores are highlighted in **bold**.

Dataset	Method	SMD ↓	TOPO score ↑			Tree rate [%]
			Prec.	Rec.	F1	
Synthetic	Two-stage [2]	1.91×10^{-3}	0.940	0.886	0.912	**100.0**
	Unconstrained [4]	1.43×10^{-5}	0.978	0.929	0.953	36.2
	Test-time constraint	6.26×10^{-6}	0.977	0.953	0.965	**100.0**
	Ours	$\mathbf{4.78 \times 10^{-6}}$	**0.986**	**0.968**	**0.977**	**100.0**
Root	Two-stage [2]	4.83×10^{-4}	0.767	0.732	0.749	**100.0**
	Unconstrained [4]	1.19×10^{-4}	0.831	0.633	0.719	35.9
	Test-time constraint	1.52×10^{-4}	0.829	0.771	0.799	**100.0**
	Ours	$\mathbf{8.82 \times 10^{-5}}$	**0.861**	**0.807**	**0.833**	**100.0**
Grapevine	Two-stage [2]	4.24×10^{-4}	0.677	0.589	0.630	**100.0**
	Unconstrained [4]	1.45×10^{-4}	**0.963**	0.559	0.708	0.0
	Test-time constraint	1.47×10^{-4}	0.896	0.840	0.867	**100.0**
	Ours	$\mathbf{1.03 \times 10^{-4}}$	0.899	**0.843**	**0.870**	**100.0**



(a) Visual results for the root dataset.



(b) Visual results for the grapevine dataset.

Fig. 3.7 Visual results for the domain-specific training with the real image datasets: Red lines, yellow dots, and cyan dots indicate the predicted graph edges, nodes, and keypoints (*i.e.*, joints and leaf nodes). Our method accurately estimates the target plant structures compared with baseline methods, demonstrating the applicability of our method for practical uses in plant science and agriculture.

3.6.5 Results

Results on Domain-Specific Training

Results on synthetic dataset Figure 3.6 shows visual results for the synthetic dataset, where the red and blue lines indicate the predicted and ground truth edges, respectively. Since they are shown translucently, if the estimated edge overlaps the true edge, it is displayed in purple. Similarly, cyan and yellow dots indicate the nodes, which merge into green if correctly estimated. From the results, all methods correctly estimate the node positions. The existing unconstrained method outputs isolated edges and cycles. Although the two-stage and test-time constraint methods enforce the tree structure constraint, they often produce incorrect edges. Compared to the baselines, our method accurately generates the graph edges.

The above trend can be quantitatively confirmed in Table 3.3. The unconstrained method produces tree structures with only about 30 % probability, even though all the training graphs form tree structures. Although introducing the test-time constraint and two-stage methods improves the shape and topology, there are still many incorrect estimates. Compared to those baseline methods, our method significantly improves both edge positions and graph topology.

Results on real datasets Figure 3.7 shows results of skeleton estimation for two real-world datasets. For these figures, red lines, yellow dots, and cyan dots indicate the edges, nodes, and keypoints (*i.e.*, joints and leaf nodes), respectively. In agreement with the synthetic results, our method predicts visually better structures, while the unconstrained model hardly produces tree structures. The method with a test-time constraint clearly produces false edges, as shown in the results for the grapevine images. The two-stage method is often sensitive to the node detection error, leading to unnecessary (*cf.* Figure 3.7a) or missing (*cf.* Figure 3.7b) keypoints. In these practical settings, our end-to-end pipeline especially benefits from the simultaneous optimization of edge and node detection, resulting in faithful predictions at both nodes and

edges for real-world datasets.

The quantitative results in Table 3.3 confirm the advantage of our method for real-world scenes. Our method shows particularly compelling results on grapevine datasets with relatively complex branching structures, outperforming the second-best method (test-time constraint) by approximately 30 % improvement on the edge accuracy evaluated by SMD.

Generalization ability of domain-specific training Although the model is trained in a domain-specific manner, our model has a certain ability of generalization. Although the model is trained with the Grapevine dataset with few background textures, it successfully works for grapevine images with background textures (Figure 3.8(a)) and for other tree species (Figure 3.8(b–d)). These results highlight the generalizability of our method.

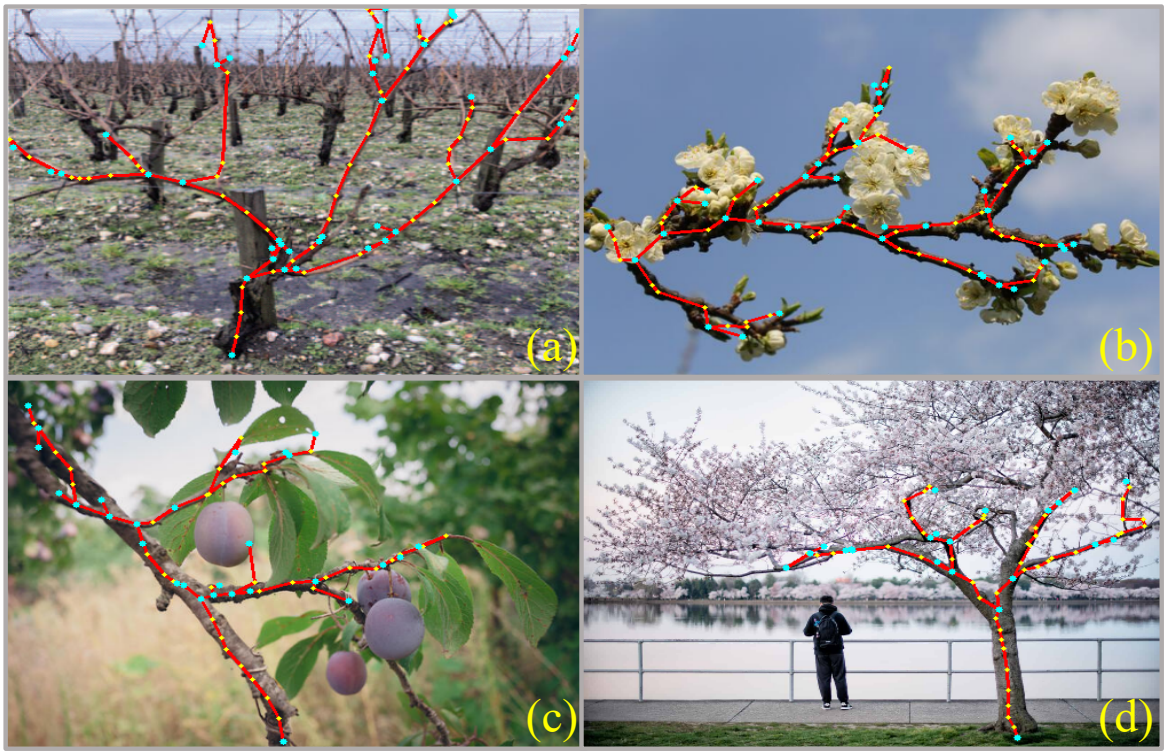


Fig. 3.8 Visual results highlighting generalization ability of domain-specific training using the grapevine dataset for (a) a grapevine tree under a natural background and (b–d) other tree species.

Table 3.4 Results on generalized training. Our method significantly improves both the shape and topology of the predicted graph while enforcing the given constraints, especially for out-of-domain scenarios.

Inner-domain datasets	**Method**	**SMD** ↓	**TOPO score** ↑			**Tree rate** [%]
			Prec.	**Rec.**	**F1**	
Grapevine	Two-stage	1.52×10^{-2}	0.898	0.817	0.855	100.0
	Unconstrained	3.58×10^{-4}	0.956	0.398	0.562	0.0
	Test-time constraint	1.29×10^{-4}	0.876	0.792	0.832	100.0
	Ours	1.57×10^{-4}	0.885	0.798	0.839	100.0
LVD2021 leaf image	Two-stage	6.36×10^{-3}	0.924	0.772	0.842	100.0
	Unconstrained	5.62×10^{-4}	0.963	0.269	0.421	0.0
	Test-time constraint	5.44×10^{-4}	0.881	0.731	0.799	100.0
	Ours	5.41×10^{-4}	0.890	0.755	0.817	100.0
LVD2021 vein mask	Two-stage	1.81×10^{-2}	0.943	0.802	0.867	100.0
	Unconstrained	4.20×10^{-4}	0.979	0.309	0.470	0.0
	Test-time constraint	3.86×10^{-4}	0.931	0.803	0.862	100.0
	Ours	3.75×10^{-4}	0.948	0.836	0.888	100.0
MIPDB	Two-stage	7.45×10^{-1}	0.011	0.012	0.011	39.2
	Unconstrained	1.74×10^{-3}	0.843	0.357	0.502	0.0
	Test-time constraint	1.55×10^{-3}	0.703	0.623	0.660	100.0
	Ours	1.61×10^{-3}	0.723	0.653	0.686	100.0
Self-captured	Two-stage	8.55×10^{-2}	0.486	0.358	0.413	95.2
	Unconstrained	2.12×10^{-3}	0.912	0.354	0.510	1.2
	Test-time constraint	1.68×10^{-3}	0.747	0.648	0.694	100.0
	Ours	1.92×10^{-3}	0.747	0.642	0.691	100.0
Web-sourced	Two-stage	3.36×10^{-1}	0.331	0.188	0.240	72.0
	Unconstrained	9.00×10^{-3}	0.821	0.304	0.443	1.8
	Test-time constraint	7.92×10^{-3}	0.656	0.564	0.606	100.0
	Ours	8.26×10^{-3}	0.634	0.559	0.594	100.0
Out-of-domain datasets	**Method**	**SMD** ↓	**TOPO score** ↑			**Tree rate** [%]
			Prec.	**Rec.**	**F1**	
Thickened L-system	Two-stage	2.02×10^{-2}	0.716	0.578	0.640	100.0
	Unconstrained	6.61×10^{-5}	0.955	0.389	0.553	0.0
	Test-time constraint	5.61×10^{-5}	0.843	0.754	0.796	100.0
	Ours	5.59×10^{-5}	0.892	0.795	0.841	100.0
LVD w/ background	Two-stage	2.99×10^{-2}	0.892	0.596	0.715	98.8
	Unconstrained	2.40×10^{-2}	0.834	0.213	0.340	0.8
	Test-time constraint	2.34×10^{-2}	0.678	0.450	0.541	100.0
	Ours	2.22×10^{-2}	0.725	0.502	0.594	99.8
Tree synthetic	Two-stage	4.97×10^{-2}	0.796	0.216	0.340	97.4
	Unconstrained	1.16×10^{-2}	0.929	0.085	0.156	0.1
	Test-time constraint	8.57×10^{-3}	0.767	0.347	0.478	99.8
	Ours	7.34×10^{-3}	0.772	0.352	0.483	100.0
Root	Two-stage	9.50×10^{-1}	0.032	0.005	0.008	5.0
	Unconstrained	6.17×10^{-2}	0.694	0.482	0.569	8.2
	Test-time constraint	6.06×10^{-2}	0.585	0.599	0.592	94.1
	Ours	2.70×10^{-2}	0.563	0.548	0.555	98.2

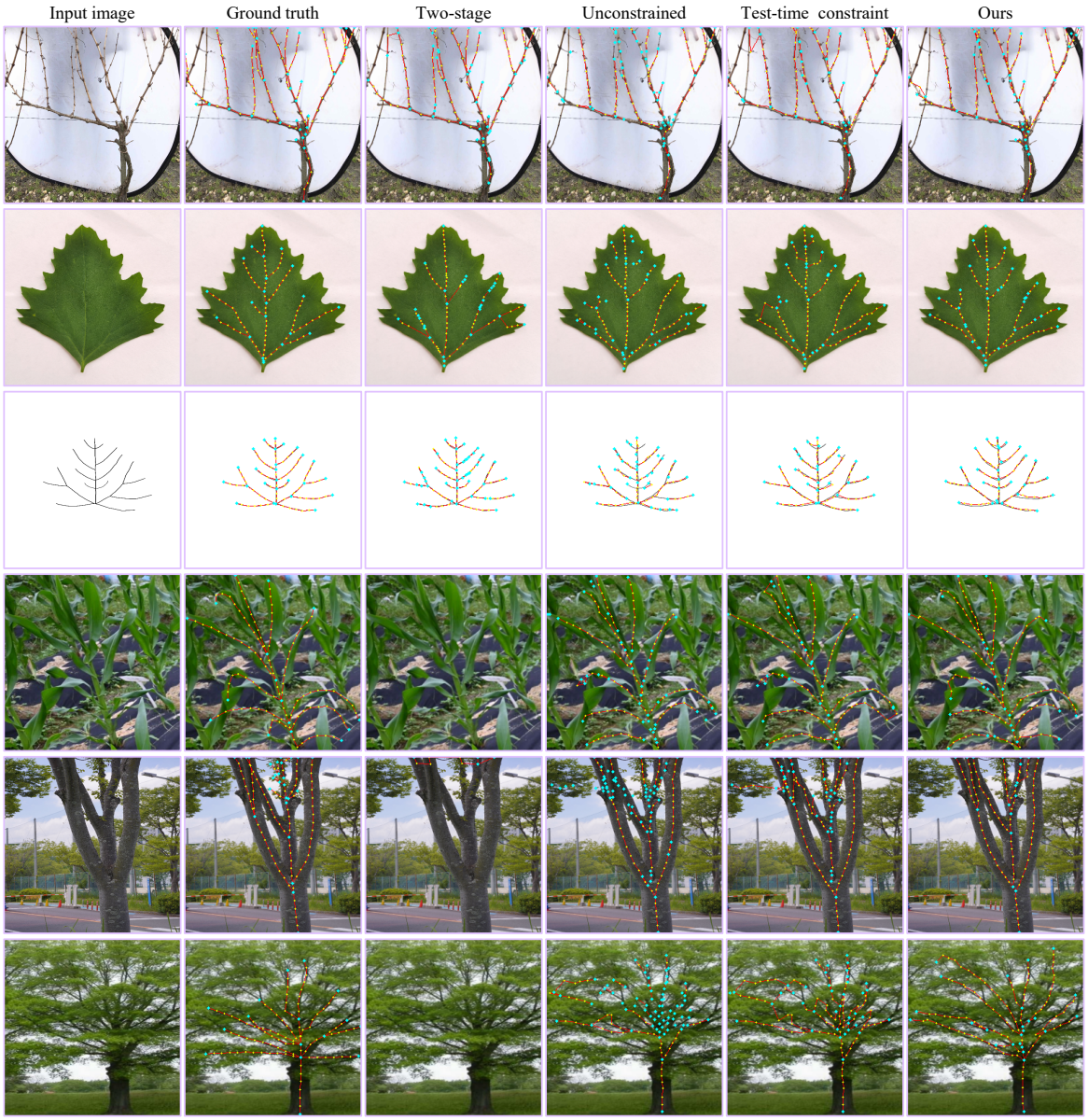


Fig. 3.9 Visual results on the inner-domain datasets. From top to bottom: grapevine, LVD2021 leaf image, LVD2021 vein mask, MIPDB, self-captured, and web-sourced datasets. The two-stage method sometimes produces very few nodes (*e.g.*, on the Self-captured dataset) or even fails to generate any nodes (*e.g.*, on MIPDB and web-sourced dataset). On the grapevine and LVD2021 datasets, it frequently outputs multiple redundant keypoints around actual nodes (visible as dense clusters of blue markers), thereby inflating TOPO scores but also increasing SMD. In contrast, our method achieves a more balanced performance, maintaining stable TOPO scores and low SMD values.

Results on Generalized Datasets

The evaluation of generalized datasets examines the model's performance across diverse plant structures. We group these results into three aspects: (1) *Inner-domain results*, evaluating performance on datasets resembling the training distribution; (2) *Out-of-domain results*, assessing adaptability to novel conditions; and (3) *Generalization ability*, highlighting performance on unseen and diverse data distributions.

Inner-domain results Table 3.4 compares results across training-related datasets, including grapevine, LVD2021 leaf image, LVD2021 vein mask, MIPDB, self-captured, and web-sourced datasets. We evaluate each method using the SMD, TOPO, and tree rate metrics.

The **two-stage** method often generates a small number of redundant nodes near actual branching points during the node detection stage. These redundant nodes are typically close to the actual keypoints. Since TOPO evaluates correctness by searching within a neighborhood around ground truth keypoints, these redundant nodes increase the likelihood of matching. As a result, both precision and recall remain high. However, these redundant nodes introduce short edges that misalign with the actual graph structure, significantly increasing the SMD value.

In contrast, the **unconstrained** method generates numerous independent nodes in areas where keypoints might exist. While this increases the chances of matching predicted nodes to ground truth keypoints, many of these nodes fall outside TOPO's search range. Consequently, TOPO precision appears high due to the excess of predicted nodes, but recall remains low as not all ground truth keypoints are matched.

The **test-time constraint method** provides a tree structure at inference, improving TOPO scores and spatial coherence compared to unconstrained predictions. However, because it applies constraints *only at inference time*, it does not fully leverage structural information to refine the model's ability. Still, the test-time constraint method is a strong baseline that ensures tree structure while maintaining reasonable topological fidelity.

Our method (ours) combines structural priors directly into the learning process. Our method consistently outperforms or closely matches the performance of the test-time constraint method in a wide range of datasets.

In summary, ours and the test-time constraint method are the top contenders on the training datasets. The test-time constraint method provides a simple baseline solution that ensures tree structure by design. At the same time, we further refine the predictions by integrating tree constraints into the inference process more naturally.

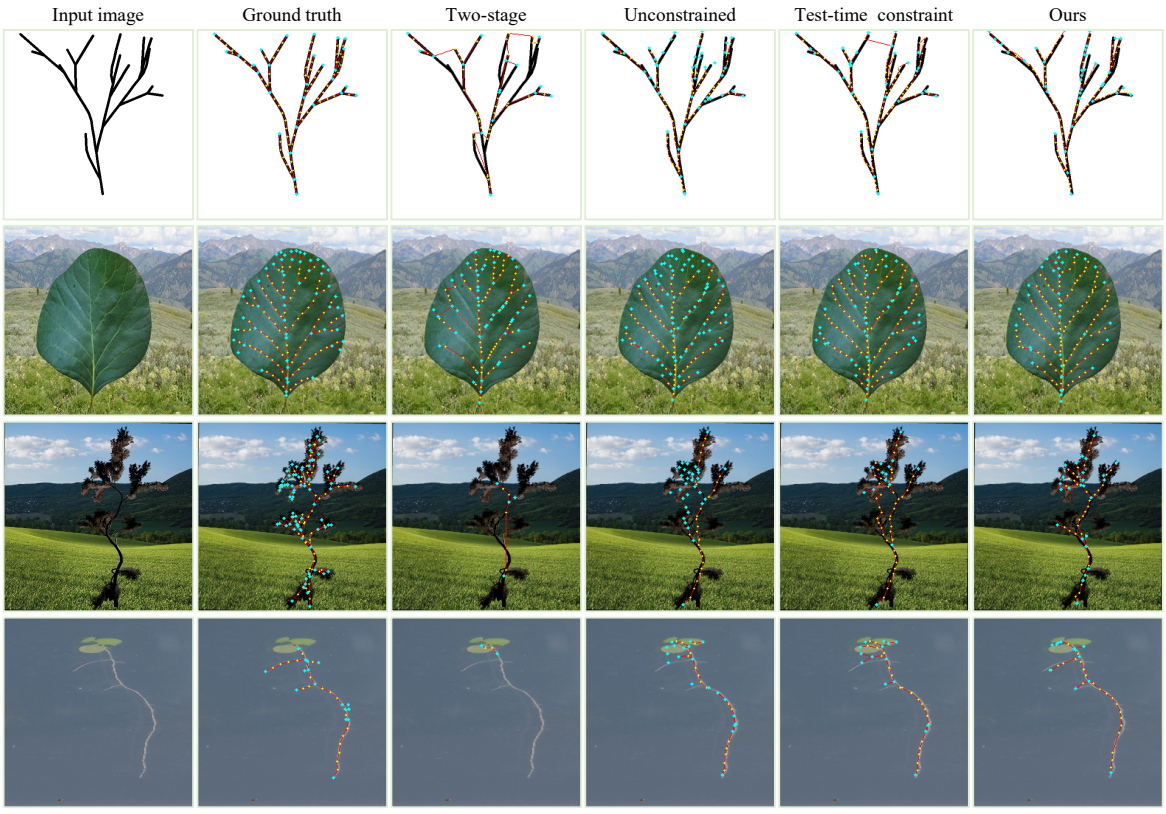


Fig. 3.10 Visual results on the out-of-domain test datasets: thickened L-system, LVD with background, tree synthetic, and root datasets (top to bottom). The unconstrained method generates excessive isolated points and disconnected components, inflating TOPO precision but reducing recall due to poor keypoint coverage. In contrast, our method preserves structural consistency, balancing TOPO precision and recall, and produces coherent tree structures. The root dataset highlights the difficulty of detecting short and thin lateral roots, which remains challenging for all methods.

Out-of-domain results The out-of-domain test datasets, including thickened L-system, LVD with background, tree synthetic, and root datasets, introduce additional complexity and distributional shifts, testing the model's generalization beyond training conditions. Visual examples are shown in Figure 3.10, and detailed metrics are provided in Table 3.4.

On the **thickened L-system** dataset, both the test-time constraint method and ours achieve near-perfect tree rates. Our method achieves a higher TOPO F1 score and lower SMD, showing that internally integrated structural priors can better handle such domain shifts, producing more topologically coherent and spatially accurate outputs.

For the **LVD with background** dataset, which merges natural leaves and environmental variations, as discussed, the two-stage method achieves a relatively high TOPO F1 but suffers from poor SMD due to redundant nodes near branching points. Both the test-time constraint and our method significantly improve SMD, with our approach achieving the lowest value. This proves our method provides the most geometrically accurate tree structures while balancing spatial precision and topological correctness.

Our method outperforms all baselines on the **tree synthetic** dataset, achieving the highest F1, lowest SMD, and a 100 % tree rate. The lowest SMD further highlights the robustness of our combined structural constraints and MST-based pipeline in generating predictions that closely match the ground truth.

Finally, on the **root** dataset, ours and the test-time constraint method perform well, achieving high tree rates. Notably, our method achieves the lowest SMD, signifying better spatial alignment with the ground truth while maintaining comparable F1 scores and consistent topological accuracy.

In terms of inference efficiency, we benchmarked both the test-time constraint method and ours on the LVD with the background dataset (500 test images). Our method required 7.79 seconds per image, while the test-time constraint baseline required 7.57 seconds per image, showing almost identical runtime. (Measured on an RTX 4090, batch size 1, input 256×256.) This indicates that the benefit of our method

mainly comes from improved topological and spatial accuracy, rather than speed.

Generalization ability To evaluate the generalization ability of our method, we qualitatively test our method on the unlabeled web-sourced test dataset. We search and download diverse datasets using an extensive list of keywords, which include various plant species, unique ferns, crops, and plants with distinct appearances. These datasets feature significant variability, ranging from rare species like *Dionaea muscipula* to major crops such as soybean and sugarcane, as well as abstract hand-drawn tree structures.

Some visual results are shown in Figure 3.11. Our method consistently performs well across these datasets, achieving accurate graph reconstruction even in unusual morphological structures or abstract representations. For instance, the model performed well on flowers, rare species, potted plants, and hand-drawn trees.

While the scope is beyond the plant skeleton estimation, we further assess the generalization ability of our model for images other than plants. We tested the pre-trained generalized PlantPose model on retinal vessel images (RETA [82] dataset). The visual results in Figure 3.12 show that the majority of vessels were reconstructed as valid tree graphs, while a few thin vessels were only partially predicted. These results show the potential usability of our method and the pre-trained model for broader domains that require tree graphs, possibly with or without fine-tuning.

Discussion and further insights These out-of-domain evaluations solidify the conclusion that merely imposing constraints at test time (the test-time constraint method) improves unconstrained predictions but may not fully leverage the structural hints available during training. In contrast, our method internalizes these constraints, improving spatial and topological consistency across diverse and previously unseen datasets.

The observed patterns in TOPO and SMD metrics underscore the need for multiple evaluation metrics. The **two-stage** method generates a small number of redundant nodes near actual branching points during the node detection stage. These redundant



Fig. 3.11 Visual results on the unlabeled web-sourced dataset, including flowers, rare plants, potted plants, and hand-drawn trees. These results highlight the diversity of test cases and demonstrate our method's generalization ability across different data domains and styles.

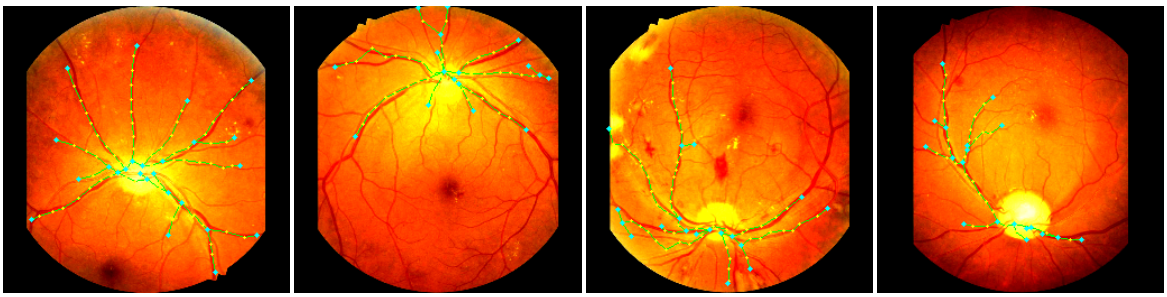


Fig. 3.12 Qualitative results on retinal vessel images. PlantPose, trained only on plant skeletons, recovers most major vessel trees after simple preprocessing (contrast adjustment and rotation).

nodes are typically close to ground truth keypoints, and TOPO's neighborhood-based evaluation matches them easily. Consequently, both TOPO precision and recall remain high. However, these redundant nodes introduce additional short edges that deviate from the actual graph structure, leading to significantly higher SMD values.

On the other hand, the **unconstrained** method produces numerous independent nodes in regions where keypoints might exist. While this abundance increases the likelihood of matching some predicted nodes to ground truth keypoints, many of these nodes fall outside TOPO's search range. As a result, TOPO precision appears artificially high due to the large number of predicted nodes, but recall remains low because not all ground truth keypoints are matched. This, combined with fragmented graphs and a near-zero tree rate, highlights the fundamental structural inconsistencies of the unconstrained method.

In contrast, the test-time constraint method and ours result in consistent tree structures, leading to balanced TOPO scores and stable SMD values. Ours often better balances these two strong contenders, demonstrating that incorporating tree constraints into the model's learning and inference process leads to more robust and generalizable predictions.

3.6.6 Performance Analysis

In this section, we present a detailed analysis of the performance of our proposed method. The effectiveness of our method is evaluated through comprehensive experiments in different scenarios. Specifically, we compare our method with the Auto-regressive (AR) model, and we also analyze the performance when our method is applied solely during the training process.

Comparison with auto-regressive method While we implement the tree-graph constraint on the state-of-the-art non-autoregressive graph generator, RelationFormer [4], other choices of constrained graph generation are viable. Existing works aiming for tree-constrained graph generation, such as in molecule structure estimation [62–64], use auto-regressive graph generation. AR methods are a simpler choice for imposing the constraint since it is relatively straightforward to implement the tree-graph constraint in their graph development process. However, since the auto-regressive methods generate graph nodes and edges progressively, they are prone to breakdowns due to changes in the output order or errors during the generation. This tendency is particularly pronounced for relatively large graphs, including our setup.

To assess the potential of auto-regressive methods, we test the state-of-the-art transformer-based auto-regressive graph generator, Generative Graph Transformer (GGT) [5]. Table 3.5 compares our method and several variations of GGT on the synthetic tree pattern dataset. The results show that the accuracy of GGT falls short compared to our method, although the vanilla GGT (the top row) mostly outputs tree graphs (92 %) without explicitly imposing the tree-graph constraint. We identified that errors by GGT occurring at a particular step in the auto-regressive generation process continuously cause errors in the sequence of the following generations. The GGT was initially designed for small datasets, specifically for graphs with $|V| \leq 10$. For our setup, where $|V| \geq 100$, generating these long sequences in a specific order presents a significant challenge.

Table 3.5 Comparisons of different graph generation models (*i.e.*, RelationFormer [4] and GGT [5]) on the synthetic dataset.

Method	SMD ↓	TOPO score ↑			Tree rate [%]
		Prec.	Rec.	F1	
GGT [5]	2.71×10^{-3}	0.635	0.537	0.582	92.06
GGT w/ test-time constraint	4.13×10^{-3}	0.620	0.545	0.580	98.10
GGT w/ SFS layer	2.80×10^{-3}	0.652	0.584	0.616	99.63
RelationFormer (Unconstraint) [4]	1.43×10^{-5}	0.978	0.929	0.953	36.2
RelationFormer w/ SFS layer (Ours)	4.78×10^{-6}	0.986	0.968	0.977	100.0

Table 3.6 Quantitative results with additional baseline, *train-time constraint*.

Dataset	Method	SFS	SMD ↓	TOPO score ↑			Tree rate [%]
				Prec.	Rec.	F1	
Synthetic	Unconstrained		1.43×10^{-5}	0.978	0.929	0.953	36.2
	Test-time constraint		6.26×10^{-6}	0.977	0.953	0.965	**100.0**
	Train-time constraint	✓	8.44×10^{-6}	**0.987**	0.954	0.970	56.5
	Ours	✓	$\mathbf{4.78 \times 10^{-6}}$	0.986	**0.968**	**0.977**	**100.0**
Root	Unconstrained		1.19×10^{-4}	0.831	0.633	0.719	35.9
	Test-time constraint		1.52×10^{-4}	0.829	0.771	0.799	**100.0**
	Train-time constraint	✓	$\mathbf{7.81 \times 10^{-5}}$	0.853	0.619	0.718	37.2
	Ours	✓	8.82×10^{-5}	**0.861**	**0.807**	**0.833**	**100.0**
Grapevine	Unconstrained		1.45×10^{-4}	0.963	0.559	0.708	0.0
	Test-time constraint		1.47×10^{-4}	0.896	0.840	0.867	**100.0**
	Train-time constraint	✓	1.30×10^{-4}	**0.965**	0.566	0.713	0.0
	Ours	✓	$\mathbf{1.03 \times 10^{-4}}$	0.899	**0.843**	**0.870**	**100.0**

Effectiveness of tree-constraint during training To assess whether our SFS layer (positively) affects the training process itself or not, we evaluate our method *without* using the SFS layer and the MST algorithm during the inference phase, *i.e.*, introducing constraint only during the training phase (called *train-time constraint* hereafter). Table 3.6 summarizes the performances. Inducting the tree constraint during the training phase mostly outperforms the methods without constraints, meaning that the improvement by our method is based on network improvement by the loss propagated via the SFS layer. We also checked the change in the accuracy metric during training, and found that our method consistently achieved better accuracy from the beginning of the training.

Figure 3.13 contrasts *train-time constraint* with our full pipeline. Although train-

ing with SFS injects a tree prior, directly thresholding edge existence probabilities at 0.5 during inference often yields fragmented instances. In the shown example, directly using shortest paths would connect A→C, C→D, and D→E, yielding unnatural connections that ignore branch geometry. By contrast, our method leverages the learned edge probabilities together with an MST at inference to form A→B, C→D, and C→E, recovering a coherent tree consistent with the ground truth. This highlights that (i) imposing the tree constraint during training improves the edge scores themselves, and (ii) applying MST at inference naturally reconnects fragmented predictions into a coherent tree, rather than remaining as isolated parts.



(a) Ground truth (b) Train-time constraint (c) Ours

Fig. 3.13 Effect of tree constraint during training and inference. From left to right: (a) ground-truth skeleton, (b) prediction with train-time constraint only (SFS applied during training, inference still uses a 0.5 edge probability threshold), (c) our method with train-time constraint + MST (Ours). The train-time constraint method suffers from instance fragmentation caused by naive thresholding, while our method recovers natural connectivity consistent with the ground truth.

Analysis of Evaluation Metrics In the experimental results, we report SMD and TOPO scores commonly used in image-to-graph generation studies. While our method achieves promising results, it is not intuitive what each metric evaluates, which has not been investigated so far. To briefly investigate the SMD and TOPO scores, we show an illustrative example in Figure 3.14. The figure compares the cases where TOPO is similar, but SMD reveals large differences between our method and the test-time constraint baseline. In this case, the TOPO score overlooks the apparent mistake in the structure. On the other hand, SMD decreases approximately 40% by our method,

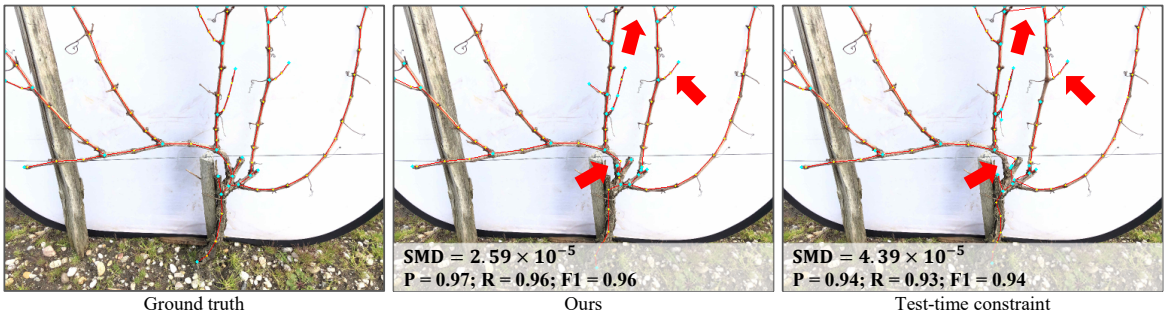


Fig. 3.14 Comparison of our method and test-time MST constraint under generalized training. Both predictions obtain similar TOPO scores ($F1 = 0.94$ vs. 0.96), indicating comparable reachability. However, as highlighted by red arrows, the baseline introduces several spurious or misplaced edges, which are not penalized by TOPO but cause a substantial increase in SMD (almost 2× higher). This example demonstrates that SMD is a more discriminative measure of structural fidelity, capturing global geometric deviations that TOPO may overlook.

which reflects the difference in the shape (implicitly including structure and position) of the reconstructed graph.

Overall, SMD is considered the key indicator of skeleton fidelity in our setup. While TOPO provides complementary evidence, it may not always evaluate structural errors adequately. We hope that providing better evaluation metrics would be an important open problem for image-to-graph generation studies.

Details of Graph Size We use the maximum cap of 384 nodes for the experiments. To further provide the justification and investigate the scalability of our method, we describe the details of how the graph sizes have been determined, and discuss the potential of extending our method for larger scales.

About input resolution: we adopt a fixed input resolution of 256×256 across all 10 datasets to ensure reproducibility and training stability, while emphasizing *topological fidelity* (connectivity, junctions, and endpoints) rather than fine texture details. This choice is consistent with prior image-to-graph works: RelationFormer [4] uses 64×64 for the Toulouse road dataset and 128×128 for the 20 US Cities dataset, RoadTracer [83] slices city-scale 4096×4096 maps into 256×256 tiles, and RNGDet [84], RNGDet++ [85], as well as SAM-Road++ [86] adopt 128–256 pixels patches as their

default input.

About node cap: we cap the maximum number of nodes per graph at 384, which is not a fundamental limitation but a standardized setting under our sampling interval of 13 pixels at 256 × 256. This interval was derived from the observed average inter-junction spacing in grapevine images (≈ 12.8 px ≈ 1/20 of the width). It corresponds to a sampling interval $s \approx 13$ pixels derived from the observed mean inter-junction spacing in grapevine (≈ 12.8 px ≈ 1/20 of the width). Under uniform density, the capacity per tile scales as

$$N_{max}(H, W, s) \approx \left\lfloor \frac{H}{s} \right\rfloor \cdot \left\lfloor \frac{W}{s} \right\rfloor, \tag{3.22}$$

resulting in s=13 at H=W=256, giving $\approx 3.88 \times 10^2$ (rounded to 384 for scheduling).

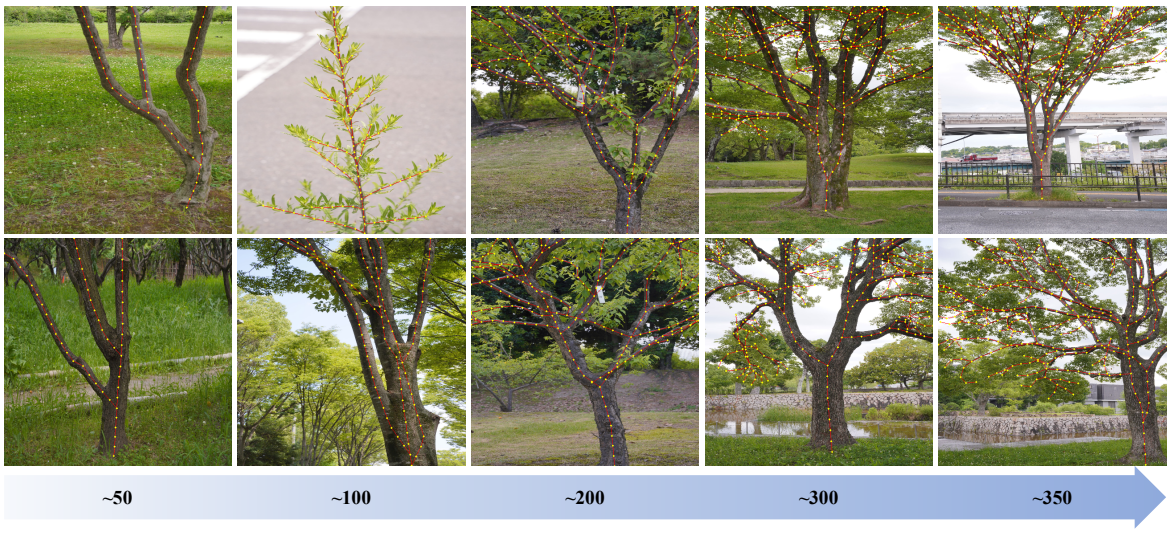


Fig. 3.15 Illustration of scalability with respect to node count. From left to right: examples with ∼50, ∼100, ∼200, ∼300, and ∼350 nodes. Simpler graphs (grass-like or single-trunk plants) have sparse nodes, while more complex trees include dense branching structures.

Scalability Figure 3.15 shows how graphs of different sizes appear in our datasets. From left to right, the examples become progressively more complex: Starting with simple grass-like plants or single trunks (∼50 nodes), then adding visible full plant

structures (∼100–200 nodes), and finally reaching fully developed trees with dense branches (∼300–350 nodes). This trend reflects our dataset statistics: The average node count in the *Self-captured dataset* is 156, while in the *Web-sourced dataset* is 96. When node count grows, the cost of pairwise edge classification increases quadratically ($O(N^2)$), leading to higher memory and runtime.

3.7 Conclusion

We present the first attempt at tree-constrained graph generation from a single image, especially useful for plant skeleton estimation. Our method combines modern learning-based graph generators and traditional graph algorithms, where we project the unconstrained predictions onto constrained graphs during the training loop. Our SFS layer behaves to selectively suppress the gradient to be backpropagated, which exploits the constraint cues from the non-differentiable graph algorithm (*i.e.*, MST). Our method can be easily integrated with off-the-shelf graph generators, as we develop PlantPose2D based on RelationFormer.

Through extensive experiments on six training datasets and four out-of-domain test datasets, we demonstrate that PlantPose2D achieves state-of-the-art results on a wide range of plant images. The strong performance of our model across generalized datasets highlights the effectiveness and versatility of the SFS layer in promoting robust generalization to diverse and previously unseen data distributions. The compelling results on both quantitative metrics and qualitative evaluations underscore its practical utility for plant science and agriculture.

Limitations We use graph algorithms during each training iteration, taking a longer training time than unconstrained methods, where fast GPU-based MST implementations (*e.g.*, [87]) can improve computational performance. The success of our method depends on the accuracy of the underlying graph generation model, as we see a few undetected nodes in the visual results.

Chapter 4

PlantPose3D

Accurately estimating tree skeletons from multi-view images is challenging. While most existing works use skeletonization from 3D point clouds, thin branches with low-texture contrast often involve multi-view stereo (MVS) to produce noisy and fragmented point clouds, which break branch connectivity. Leveraging the recent development in accurate mask extraction from images, we introduce a mask-guided graph optimization framework that estimates a 3D skeleton directly from multi-view segmentation masks, bypassing the reliance on point cloud quality. In our method, a skeleton is modeled as a graph whose nodes store positions and radii while its adjacency matrix encodes branch connectivity. We use 3D Gaussian splatting (3DGS) to render silhouettes of the graph and directly optimize the nodes and the adjacency matrix to fit given multi-view silhouettes in a differentiable manner. Furthermore, we use a minimum spanning tree (MST) algorithm during the optimization loop to regularize the graph to a tree structure. Experiments on synthetic and real-world plants show consistent improvements in completeness and structural accuracy over existing point-cloud-based and heuristic baseline methods.

4.1 Introduction

Traditional methods of plant skeletonization typically rely on high-fidelity 3D point cloud input captured by expensive sensing systems, such as light detection and ranging (LiDAR) and structured light scanners. While LiDAR offers millimeter-level precision, its high cost and operational complexity limit broader applicability [26, 27]. In contrast, photogrammetry-based multi-view image methods offer a more accessible choice [18]. However, the branches' characteristics of low-texture contrast and thin nature often cause multi-view stereo (MVS) to yield noisy and fragmented point clouds, in which branches are often missing or disconnected.

Existing methods of tree skeleton estimation from 3D point clouds typically follow a two-stage routine: first, extracting an initial skeleton from the point clouds and then refining connectivity using heuristic or learning-based rules. Current approaches fall into three families. Among them, particle flow-based models require heavy parameter tuning [88–90]. Geometry-based methods, including thinning [91–93], clustering [31, 94, 95], and minimum spanning tree (MST) refinement [29, 96, 97], are effective under clean and complete data. Recent deep-learning methods like SmartTree and TreePartNet [30, 31] demand large labelled sets and struggle with real-world noise.

To overcome the limitations of point-cloud-based skeletonization, which often yields fragmented or missing branches, we leverage multi-view segmentation masks because they preserve global silhouettes and provide reliable structural constraints. Building on this motivation, we propose a mask-guided graph optimization framework that estimates a 3D tree graph structure directly from such masks. We represent the skeleton as a graph, where nodes have their position and radii of branches, and the edges are represented as an adjacency matrix.

Given the rough initial graph extracted from MVS-based point clouds, we optimize it via 3D Gaussian splatting (3DGS) [35], projecting the graph skeleton onto the image plane. We minimize a multi-view silhouette loss so that the graph fits the observed

silhouettes. The inherent blur of 3D Gaussians achieves robustness to appearance differences among views [35], resulting in camera pose errors and mask errors in our case.

Our method stands on the recent notable development of segmentation algorithms, where yielding accurate 2D masks is essential for guiding the mask-based refinement of the 3D skeleton. Currently, no segmentation model is trained explicitly for tree branches, especially under natural scenes' diverse lighting and occlusion conditions. We, therefore, combine a recent language-grounded segmentation model, Lang-SAM (Language segment-anything, `https://github.com/luca-medeiros/lang-segment-anything` (accessed on 10 May 2025)), and an accurate segmentation model, BiRefNet-HQ [98], for acquiring accurate segmentation of tree branches.

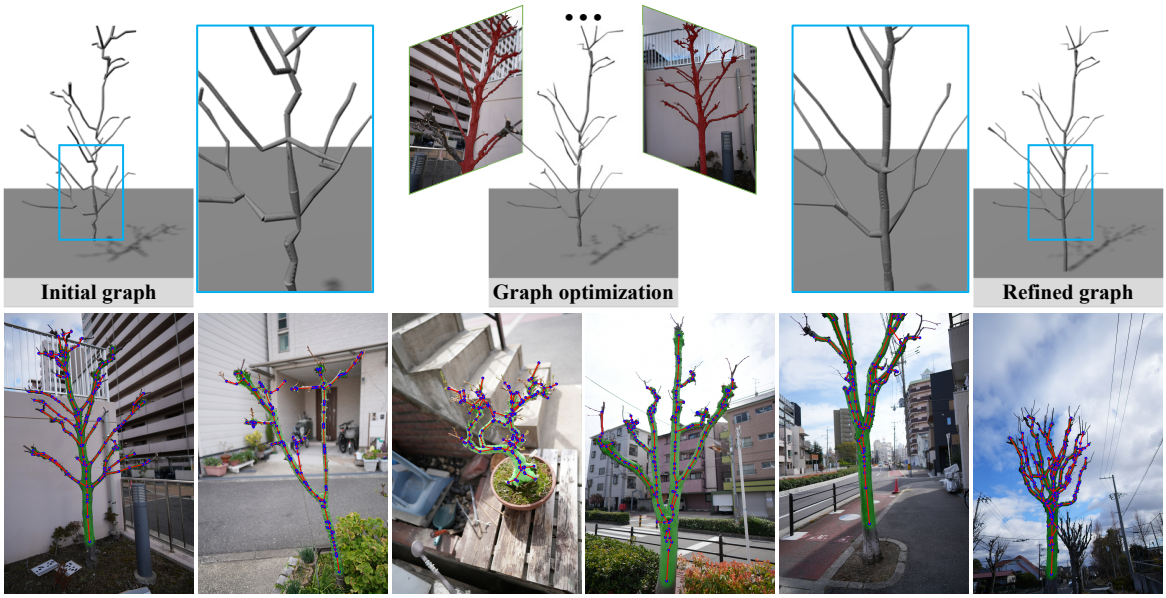


Fig. 4.1 We reconstruct 3D branch structures using refined multi-view masks, visualized as red regions in the adjacent panels of the graph optimization part. Transparent green overlays indicate the predicted tree shape and branch radii. Blue points and red lines represent graph nodes and branch connections, respectively.

In summary, our key contribution is introducing a novel mask-guided optimization framework that estimates 3D tree-structured skeletons. By leveraging differentiable 3DGS, our method transforms the graph into soft silhouettes and aligns them with the

multi-view masks. This design enables the model to mitigate pose and segmentation noise, yielding topologically valid and geometrically accurate skeletons without requiring point-cloud-specific tuning, as shown in our experiments comparing our method with the state-of-the-art tree skeletonization methods.

4.2 Related Work

4.2.1 Three-Dimensional Tree Skeleton Extraction

Estimating the skeletal structure of plants and trees is a long-lived task across computer vision (CV) and plant sciences [7]. For tree skeleton extraction, most approaches rely on 3D point clouds observed by LiDAR [17, 94] or MVS [10, 18]. Regardless of the capturing methods, most methods follow a two-stage pipeline [28]: (1) initial skeletonization to obtain preliminary branch structures and (2) graph refinements, such as using MST or shortest-path algorithms (*e.g.*, Dijkstra's algorithm), to optimize connectivity and topology [29, 41, 46, 47, 99].

Particle-flow modeling A traditional approach to generating plausible skeletons is to simulate particle trajectories guided by physical rules or probabilistic distributions [88–90]. Although effective in estimating missing branches, these methods are susceptible to parameter tuning and require trees to exhibit strong structural self-similarity [88, 90].

Geometry-based methods Most methods directly use the geometry of point clouds to refine the tree structure, such as voxel thinning [91, 92], clustering-based skeleton extraction [31, 94, 95, 100], and spanning tree methods [29, 96, 97]. Thinning methods iteratively erode the data volume but are prone to errors under noisy or incomplete conditions [91, 93, 101]. Clustering methods segment point clouds into smaller components before linking them into a coherent skeleton, requiring precise neighborhood radius selection, but typically struggle with occluded or sparsely sampled data [94, 95].

Spanning tree-based methods [29, 96, 97] guarantee connectivity through global optimization but cannot recover missing branches absent from the initial point cloud. Their pruning thresholds are often species-specific.

Learning-based methods Recent advancements in deep learning have introduced neural approaches for skeletonization. For example, Smart-Tree [30] employs sparse voxel convolutional neural networks to predict medial axes, improving accuracy around complex branch structures. Similarly, TreePartNet [31] decomposes point clouds into generalized cylinders using semantic segmentation and neural clustering modules. Although these methods achieve stronger generalization across tree species, they rely on large-scale supervised training. Moreover, their performance degrades due to a domain gap between synthetic training and real-world inference [13, 14].

To address the limitations of point-cloud-based approaches, we introduce the first attempt at a mask-guided graph optimization framework for tree skeleton extraction, which refines connectivity through differentiable rendering to fit the multi-view segmentation masks.

4.2.2 Image-Guided 3D Structure Estimation via Differentiable Rendering

Although in different contexts, recent sketch-based 3D estimation methods yield 3D line-like structures from 2D supervision using compact primitives such as Bézier curves [102], superquadric strokes [103], or learned sketch embeddings [104]. Among them, Diff3DS [102] is the most relevant to our task, which represents objects as 3D Bézier curves and projects them onto multi-view images using a differentiable renderer based on DiffVG [105]. The silhouettes are denoised using a pre-trained diffusion model. While these methods are helpful for sketch-based estimation, these primitives lack explicit connectivity and topological structure, making it challenging to estimate complete tree skeletons as a graph structure.

Compared with sketch-based methods, our key is to use a graph-based represen-

tation, where branch connectivity and junctions are encoded explicitly. We directly optimize the whole graph structure via differentiable rendering of 3D Gaussian primitives, supporting soft silhouette rendering with spatially varying blur.

4.3 PlantPose3D Method

Our framework estimates accurate 3D plant skeletons from multi-view images, as shown in Figure 4.2. We first extract segmentation masks from multi-view images using the Lang-SAM and BiRefNet-HQ [98], which complementarily extract the semantic region information and fine details. We then initialize a rough tree graph structure by clustering the MVS-based 3DGS point cloud and the MST constraint. We convert the graph structure into Gaussian primitives for differentiable graph rendering and optimize the graph components with silhouette losses while considering tree structure validity.

4.3.1 Preprocessing and Initialization

This section comprises two main components: (1) multi-model mask extraction and (2) graph initialization from filtered point clouds.

Mask Extraction

While our goal is not to propose a new segmentation method, obtaining a high-quality segmentation mask of trees is essential for supervising the graph optimization process described later. Accurate segmentation of plant masks in natural scenes has been challenging due to uncontrolled illumination and background clutter; however, we found that the recent development of image segmentation methods using foundation models enables reasonable segmentation of natural trees.

We adopt a two-stage strategy that combines the strengths of two foundation models: BiRefNet-HQ, a high-resolution foreground segmentation model, and Lang-SAM, a language-grounded model guided by text prompts. The fusion addresses

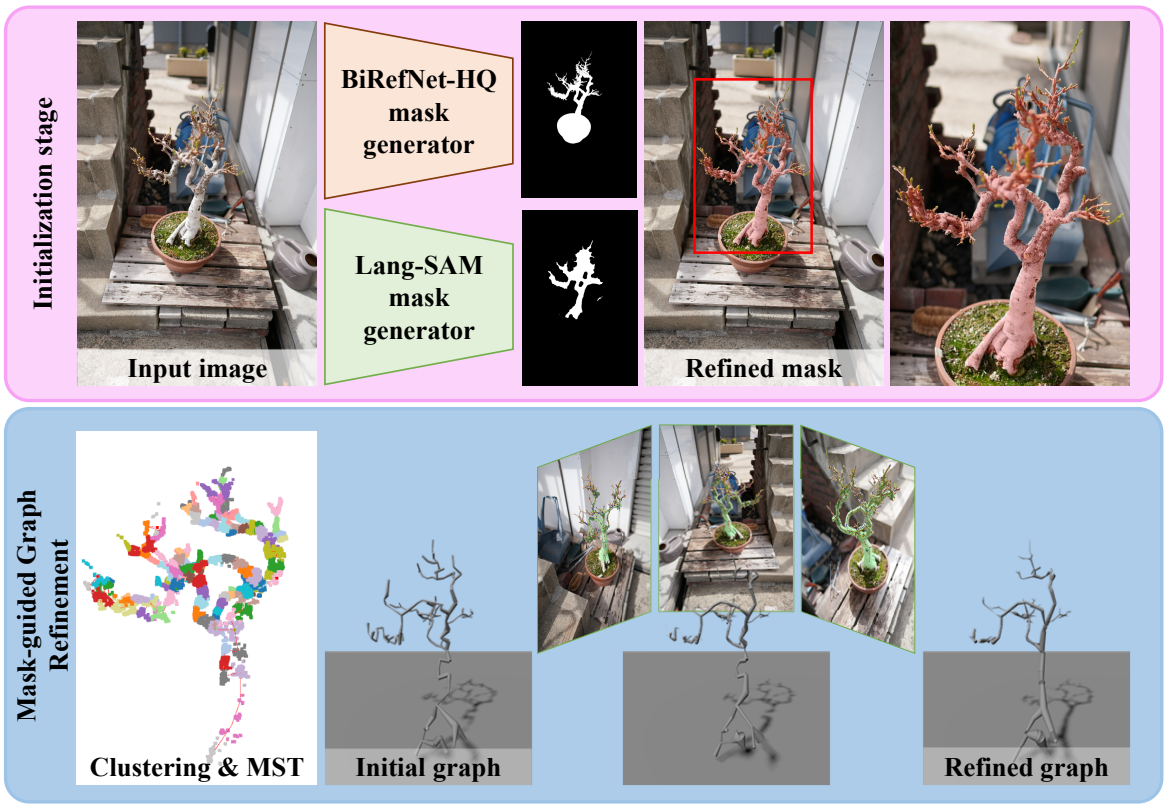


Fig. 4.2 Pipeline overview. The provided input images, two complementary mask generators, BiRefNet-HQ for high-resolution foreground masks, and Lang-SAM for language-guided object masks, are fused into refined masks. We use the MST and clustering to obtain an initial graph. Finally, the initial graph is refined through multi-view masks, producing a 3D plant skeleton that is guaranteed to be a valid tree.

the limitations of each model. BiRefNet-HQ generates spatially accurate masks but struggles to distinguish tree-only foreground from background clutter, while Lang-SAM lacks sufficient spatial resolution to capture fine structural details.

Specifically, given an RGB input image $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$, we first generate a foreground alpha mask $M_{\mathrm{B}} \in \{0,1\}^{H \times W}$ with BiRefNet-HQ, which may contain objects other than the target tree but are judged as the foreground, as shown in Figure 4.3a. To suppress the leakage, we apply Lang-SAM with the text prompt "tree" to obtain the rough area of the target tree $M_{\mathrm{L}} \in \{0,1\}^{H \times W}$. Then, this is followed by the morphological closing $\mathrm{Close}(\cdot, s)$ with a kernel size of $s = 7$ to reduce fragmentation artifacts, as

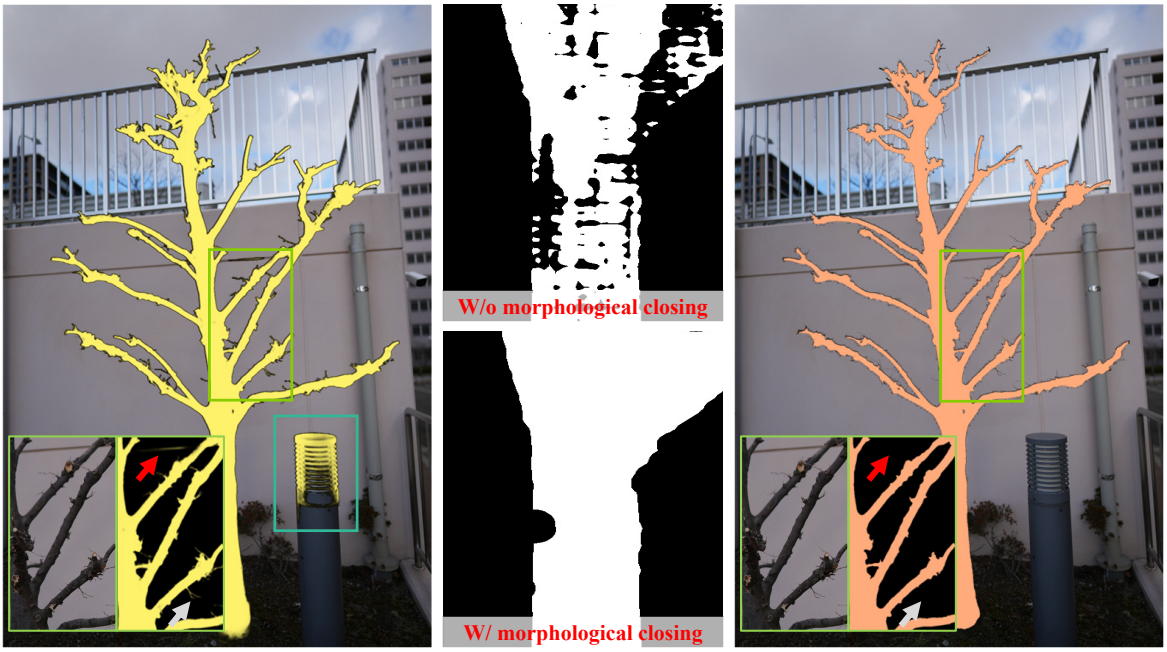
shown in Figure 4.3b. Formally,

$$\tilde{M}_{\mathrm{L}} = \mathrm{Close}(M_{\mathrm{L}},\, s). \tag{4.1}$$

The final mask is computed by intersecting both masks and retaining only the largest connected component (see Figure 4.3(c)) as follows:

$$M_{\mathrm{final}} = \mathrm{LargestCC}\Big(M_{\mathrm{B}} \;\wedge\; \tilde{M}_{\mathrm{L}}\Big), \tag{4.2}$$

where $\wedge$ denotes element-wise logical AND, and LargestCC only retains the largest connected component.



(a) BriefNet-HQ's mask (b) Morphological closing (c) Refined mask

Fig. 4.3 Mask fusion pipeline. (**a**) The output of BirefNet-HQ retains fine structures but includes background noise. (**b**) Lang-SAM prediction before (top) and after (bottom) morphological closing, where closing fills small gaps. (**c**) The refined mask is obtained by combining both predictions, applying morphological filtering and connected component analysis to preserve only the main plant region. As a result, small background artifacts (red arrows) and ultra-thin branches (gray arrows) are effectively suppressed.

Graph Initialization

In urban or roadside scenes, the camera motion is often restricted to part of the whole hemisphere, resulting in noisy and fragmented point clouds by MVS. We thus use 3DGS initialized by the MVS point cloud to further densify the points. Since both clouds often include unreliable outliers, we apply a multi-view mask consistency filtering, *i.e.*, projecting each point into all views and retaining it only if its projection lies at least $N_{\min}$ foreground masks.

Let $\mathscr{P}_{\mathrm{MVS}} = \{p_i \in \mathbb{R}^3\}$ and $\mathscr{P}_{\mathrm{GS}} = \{p_i \in \mathbb{R}^3\}$ denote the filtered point clouds from MVS and 3DGS, respectively. We then yield the dense point cloud as follows:

$$\mathscr{P}_{\mathrm{init}} = \mathscr{P}_{\mathrm{MVS}} \cup \mathscr{P}_{\mathrm{GS}}. \tag{4.3}$$

We clean $\mathscr{P}_{\mathrm{init}}$ using radius-based and statistical outlier removal, then apply K-means clustering. Let a tuple $v_i = (\boldsymbol{c}_i, r_i), (1 \leq i \leq K)$ contain a cluster centroid $\boldsymbol{c}_i \in \mathbb{R}^3$ and cluster's median local radius $r_i \in \mathbb{R}_+$. We connect the nodes using an MST algorithm, where the edge cost is defined as the Euclidean distance between $\boldsymbol{c}_i$, yielding the initial graph:

$$G_{\mathrm{init}} = \Big(\mathscr{V},\, \mathscr{E}_{\mathrm{init}}\Big) = \Big(\{\,(\boldsymbol{c}_i, r_i)\,\}_{i=1}^{K},\; \mathscr{E}_{\mathrm{init}}\Big), \tag{4.4}$$

where each node v_i encodes the position and radius, and each edge $(i, j) \in \mathscr{E}_{\mathrm{init}}$ reflects a branch connection determined by the MST.

To enable the gradient-based optimization of the adjacency structure, we represent the graph connectivity using a differentiable adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{K \times K}$. Specifically, we initialize adjacency probabilities with values close to 0 or 1 based on initial edges as follows:

$$A_{ij} = \begin{cases} 1 - \epsilon, & (i, j) \in \mathscr{E}_{\mathrm{init}} \\ \epsilon, & (i, j) \notin \mathscr{E}_{\mathrm{init}} \end{cases}, \tag{4.5}$$

where $\epsilon = 10^{-6}$ prevents gradient saturation. To facilitate optimization, we reparam-

eterize the adjacency probabilities using the inverse sigmoid transformation:

$$\boldsymbol{A}_{\text{init}} = \text{logit}(\boldsymbol{A}) = \log\left(\frac{\boldsymbol{A}}{1-\boldsymbol{A}}\right). \tag{4.6}$$

4.3.2 Mask-Guided Graph Refinement

Three-Dimensional Gaussian Splatting

Three-dimensional Gaussian splatting, introduced by Kerbl et al. [35], represents a scene as a set of anisotropic 3D Gaussians, facilitating efficient differentiable rendering. Each Gaussian primitive G is parameterized by a 3D center $\boldsymbol{\mu} \in \mathbb{R}^3$ and a covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{3\times3}$:

$$G(\boldsymbol{p}) = e^{-\frac{1}{2}(\boldsymbol{p}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{p}-\boldsymbol{\mu})}, \tag{4.7}$$

where $\boldsymbol{p} \in \mathbb{R}^3$ denotes any 3D spatial location. The covariance matrix $\boldsymbol{\Sigma}$ is decomposed into a scaling matrix $\boldsymbol{S} = \text{diag}(s_1, s_2, s_3)$ and a rotation matrix $\boldsymbol{R} \in \text{SO}(3)$ as follows:

$$\boldsymbol{\Sigma} = \boldsymbol{R}\boldsymbol{S}\boldsymbol{S}^\top\boldsymbol{R}^\top. \tag{4.8}$$

In the 3DGS implementation, scale and rotation are represented by a scaling vector $\boldsymbol{s} = [s_1, s_2, s_3]^\top \in \mathbb{R}^3_{\geq 0}$ and the quaternion form $\boldsymbol{q}$, respectively. Both parameters are optimized independently in the pipeline along with the optimization of Gaussian appearances $\{o, \boldsymbol{c}\}$ containing opacity o, and color $\boldsymbol{c}$ is represented as its RGB direct component (DC) and view-dependent spherical harmonics coefficients. The projected opacity α of a Gaussian at a pixel location is as follows:

$$\alpha = 1 - e^{-o\cdot G(\boldsymbol{p})}. \tag{4.9}$$

In our pipeline, we specifically exploit opacity values to encode the probability of structural edges, which is detailed in the following section.

Mask-Guided Graph Refinement

Our key contribution is formulating the structural refinement of plant skeleton graphs as a **self-supervised** optimization problem. Here, "self-supervised" means that no ground-truth 3D skeletons are required. Starting from the initial graph, we render Gaussian samples along its edges using differentiable 3DGS and minimize a silhouette loss that aligns the projected skeleton with the observed masks.

Formally, given an edge connecting node v_i and v_j with respective radii r_i and r_j, we uniformly sample points along the edge to create Gaussian primitives:

$$\boldsymbol{p}_n^{(ij)} = \boldsymbol{c}_i + \frac{n}{N}(\boldsymbol{c}_j - \boldsymbol{c}_i), \quad r_n^{(ij)} = r_i + \frac{n}{N}(r_j - r_i), \quad n = 0, \ldots, N, \tag{4.10}$$

where N denotes the number of sampled Gaussians per edge, ensuring dense coverage (we use N=64 in all experiments).

To align the graph structure with silhouette masks, we initialize each Gaussian with isotropic scale parameters $(s_x, s_y, s_z) = (r_n^{(ij)}, r_n^{(ij)}, r_n^{(ij)})$ and set the rotation and color parameters to fixed constants, which are not updated during optimization.

We explicitly parameterize edge existence probabilities using the differentiable adjacency matrix $\boldsymbol{A}$ defined earlier. The opacity of Gaussian primitives sampled along an edge (i, j) is thus directly tied to the adjacency probability A_{ij} as follows:

$$o_k^{(ij)} = A_{ij} = \gamma(A_{init}^{(ij)}), \tag{4.11}$$

where $\gamma(\cdot)$ denotes the sigmoid activation.

During rendering, pixel-level opacity α is computed using the accumulation of projected Gaussian opacities, yielding a differentiable mask:

$$\alpha(\boldsymbol{x}) = 1 - \prod_{g \in \mathscr{G}\text{proj}(\boldsymbol{x})} (1 - o_g \cdot G_g(\boldsymbol{x})), \tag{4.12}$$

where $\mathscr{G}_{\text{proj}}(\boldsymbol{x})$ is the set of Gaussians projected onto pixel $\boldsymbol{x}$.

4.3.3 Structure-Aware Graph Optimization

Our framework refines both the topology and geometry of the plant skeleton graph through two complementary objectives:

$$\mathscr{L}_{\text{total}} = \mathscr{L}_{\text{silhouette}} + \mathscr{L}_{\text{graph}}. \tag{4.13}$$

Silhouette Supervision

As described above, we render the graph structure using 3DGS and obtain a soft silhouette mask $\alpha(\boldsymbol{x})$. We supervise this mask using the binary foreground segmentation mask $M(\boldsymbol{x})$ extracted from multi-view images.

To guide the adjacency matrix $\boldsymbol{A}$ toward tree-like connectivity, we introduce a tree graph prior via a differentiable selective feature suppression (SFS) layer proposed in TreeFormer [8]. This layer converts an arbitrary graph to an MST-based tree graph in a differentiable manner, and thus, we convert the adjacency matrix $\boldsymbol{A}$ to the one representing an MST, $\boldsymbol{A}_{\text{mst}}$. We then obtain the MST graph's soft silhouette mask $\alpha_{\text{mst}}(\boldsymbol{x})$ in the same manner as computing the soft silhouette mask $\alpha(\boldsymbol{x})$.

The final silhouette loss to minimize is as follows:

$$\mathscr{L}_{\text{silhouette}} = \lambda_{\text{GS}} \cdot \mathscr{L}_{\text{GS}} + \lambda_{\text{MST}} \cdot \mathscr{L}_{\text{MST}}, \tag{4.14}$$

where

$$\mathscr{L}_{\text{GS}} = \frac{1}{|\Omega|} \sum_{\boldsymbol{x} \in \Omega} |\alpha(\boldsymbol{x}) - M(\boldsymbol{x})|, \tag{4.15}$$

$$\mathscr{L}_{\text{MST}} = \frac{1}{|\Omega|} \sum_{\boldsymbol{x} \in \Omega} |\alpha_{\text{mst}}(\boldsymbol{x}) - M(\boldsymbol{x})|, \tag{4.16}$$

and Ω denotes the set of foreground-relevant pixels across all views, *i.e.*, pixels within the union of all refined masks.

This setup encourages $\boldsymbol{A}$ to remain fully differentiable while being softly aligned with a plausible tree topology through silhouette supervision.

Graph Geometry Regularization

To encourage physically plausible and topologically meaningful structures, we incorporate five regularization terms inspired by trees' characteristics. Unless otherwise specified, we follow the same notation as in previous sections: $\boldsymbol{c}_i \in \mathbb{R}^3$ and $r_i \in \mathbb{R}_+$ denote the position and radius of node i, $(i, j) \in \mathscr{E}$ denotes an undirected edge between nodes i and j, and θ_{ijk} is the angle formed by two incident edges at node j. And the Hyper-Parameters follow Section 4.4.1.

Repulsion Loss $\mathscr{L}_{rep}$ Repulsion loss penalizes overly close node pairs using a Gaussian kernel:

$$\mathscr{L}_{\text{rep}} = \sum_{i<j} e^{-\frac{\|\boldsymbol{c}_i - \boldsymbol{c}_j\|^2}{\sigma_{\text{rep}}^2}}.$$

Edge Length Loss $\mathscr{L}_{edge\text{-}short}$ Edge length loss penalizes short edges to suppress trivial links:

$$\mathscr{L}_{\text{edge-short}} = \sum_{(i,j)\in\mathscr{E}} e^{-\frac{\|\boldsymbol{c}_i - \boldsymbol{c}_j\|^2}{\sigma_{\text{edge}}^2}}.$$

Angle Fold Loss $\mathscr{L}_{angle}$ Angle fold loss penalizes sharp back-and-forth turns (i.e., zig-zag edges) by enforcing a minimum angle between consecutive edges. For each triple (i, j, k) sharing a common pivot node j, we compute the angle θ_{ijk} formed between edges $\mathscr{E}^{(ij)}$ and $\mathscr{E}^{(jk)}$, and apply the following:

$$\mathscr{L}_{\text{angle}} = \sum_{(i,j,k)} \text{softplus}\left(\cos\theta_{ijk} - \cos\theta_{\min}\right).$$

Midpoint Direction Loss $\mathscr{L}_{mid}$ Midpoint direction loss penalizes directionally aligned edges that are spatially too close. For each edge pair (i, j, k) that shares a common node j, we consider the pair of edges $\mathscr{E}^{(ij)}$ and $\mathscr{E}^{(jk)}$, and apply the loss if they are nearly collinear:

$$\mathscr{L}_{\text{mid}} = \sum_{(i,j,k)} \text{softplus}\left(\|\boldsymbol{m}^{(ij)} - \boldsymbol{m}^{(jk)}\|_2 - d_{\max}\right),$$

where $\boldsymbol{m}^{(ij)} = \frac{1}{2}(\boldsymbol{c}_i + \boldsymbol{c}_j)$ and $\boldsymbol{m}^{(jk)} = \frac{1}{2}(\boldsymbol{c}_j + \boldsymbol{c}_k)$ denote the midpoints of the two edges. We consider the pair only if the angle between the two edges satisfies the following:

$$\cos(\theta_{ijk}) = \frac{(\boldsymbol{c}_j - \boldsymbol{c}_i)^\top (\boldsymbol{c}_k - \boldsymbol{c}_j)}{\|\boldsymbol{c}_j - \boldsymbol{c}_i\| \cdot \|\boldsymbol{c}_k - \boldsymbol{c}_j\|} > \tau_{\text{dir}}.$$

Radius Lower Bound Loss $\mathscr{L}_{\textit{radius}}$ Radius lower bound loss prevents branch radii from collapsing:

$$\mathscr{L}_{\text{radius}} = \sum_{i \in \mathscr{E}} \text{softplus}(r_{\min} - r_i).$$

We combine these terms into the full geometry-aware loss:

$$\begin{aligned} \mathscr{L}_{\text{graph}} = \lambda_{\text{rep}} \mathscr{L}_{\text{rep}} + \lambda_{\text{edge-short}} \mathscr{L}_{\text{edge-short}} + \lambda_{\text{angle}} \mathscr{L}_{\text{angle}} \\ + \ \lambda_{\text{mid}} \mathscr{L}_{\text{mid}} + \lambda_{\text{radius}} \mathscr{L}_{\text{radius}}. \end{aligned} \tag{4.17}$$

Together, these modules form a fully differentiable pipeline that transforms raw 2D masks into a geometry- and topology-consistent 3D tree structure without the need for dataset-specific tuning or post-processing heuristics. The hyper-parameter settings are summarized in Section 4.4.1.

4.3.4 Overall Algorithm

To clearly illustrate our mask-guided tree skeleton estimation pipeline, we summarize our overall method in Algorithm 1.

4.4 Experiments

4.4.1 Implementation Details

We implement our method with PyTorch and 3DGS's CUDA kernels for rasterization. All experiments were conducted on a Windows workstation equipped with an AMD 3970X CPU and an NVIDIA RTX 4090 GPU. Our method is implemented as a modular PyTorch framework, utilizing the gsplat rasterizer [106] for rendering.

Algorithm 1: Mask-Guided Tree Skeleton Estimation.

Input: Multi-view RGB images $\{\mathbf{I}_v\}_{v=1}^{V}$ and cameras $\{\Pi_v\}$
Output: Refined graph $G = (\mathscr{V}, \mathscr{E})$
/* 1. Mask Extraction (Section 4.3.1) */;
foreach $v \in \{1, \ldots, V\}$ **do**
  $M_{\mathrm{B}} \leftarrow \texttt{BiRefNetHQ}(\mathbf{I}_v)$;
  $M_{\mathrm{L}} \leftarrow \texttt{LangSAM}(\mathbf{I}_v,$ "tree." $)$;
  $\tilde{M}_{\mathrm{L}} \leftarrow \mathrm{Close}(M_{\mathrm{L}}, s)$;
  $M_v \leftarrow \mathrm{LargestCC}(M_{\mathrm{B}} \wedge \tilde{M}_{\mathrm{L}})$;
/* 2. Point-Cloud Densification (Section 4.3.1) */;
$\mathscr{P}_{\mathrm{MVS}} \leftarrow \texttt{MVS}(\{\mathbf{I}_v, \Pi_v\})$;
$\mathscr{P}_{\mathrm{GS}} \leftarrow \texttt{3DGS}(\mathscr{P}_{\mathrm{MVS}})$;
$\mathscr{P} \leftarrow \mathscr{P}_{\mathrm{MVS}} \cup \mathscr{P}_{\mathrm{GS}}$;
$\mathscr{P} \leftarrow \texttt{MaskFilter}(\mathscr{P}, \{M_v, \Pi_v\}, N_{\min})$;
$\mathscr{P} \leftarrow \texttt{OutlierRemoval}(\mathscr{P})$;
/* 3. Graph Initialization */;
$\{\boldsymbol{c}_i, r_i\}_1^K \leftarrow K\text{-}\texttt{means}(\mathscr{P}, K)$;
$\mathscr{E}_{\mathrm{init}} \leftarrow \texttt{MST}(\{\boldsymbol{c}_i\})$;
/* 4. Optimization Loop */;
for $t = 1$ **to** T **do**
  $\mathscr{G} \leftarrow \texttt{SampleGaussians}(\{\boldsymbol{c}_i, r_i\}, \boldsymbol{A}, N)$;
  $\alpha, \alpha_{\mathrm{mst}} \leftarrow \texttt{Render3DGS}(\mathscr{G})$;
  Compute $\mathscr{L}_{\mathrm{silhouette}}, \mathscr{L}_{\mathrm{graph}}$;
  $\mathscr{L}_{\mathrm{total}} = \mathscr{L}_{\mathrm{silhouette}} + \mathscr{L}_{\mathrm{graph}}$;
  Update $\{\boldsymbol{c}_i, r_i, \boldsymbol{A}\}$ with Adam;
return $G = \big(\{\boldsymbol{c}_i, r_i\}, \boldsymbol{A}\big)$;

Hyper-Parameters We summarize all hyper-parameters used in the silhouette supervision and geometry-aware optimization. Unless otherwise specified, the following values are fixed throughout all experiments.

For silhouette loss, we set $\lambda_{\mathrm{GS}} = 0.1$ and $\lambda_{\mathrm{MST}} = 1.0$. For geometry regularization, we fix all weights $\lambda_* = 0.05$ and use thresholds $\sigma_{\mathrm{rep}} = 2\,\mathrm{cm}$, $\sigma_{\mathrm{edge}} = 2\,\mathrm{cm}$, $\theta_{\min} = 20^\circ$, $\tau_{\mathrm{dir}} = 0.90$, $d_{\max} = 2\,\mathrm{cm}$, and $r_{\min} = 1\,\mathrm{cm}$.

The number of clusters K is chosen empirically depending on dataset complexity: $K = 50$ for the simple Mtree dataset, $K = 100$ for the L-system-based dataset, and $K = 150$ for the real-world dataset. This choice follows the scalability analysis

in Section 3.6.6 of PlantPose2D, where average node counts of $100-200$ were reported. While K affects node density in the initial clustering, the MST projection ensures the final skeleton remains a stable tree topology across these values. Note that the cost of pairwise edge classification scales quadratically with the node count ($O(N^2)$), so larger K increases memory and runtime, motivating the chosen range as a balance between fidelity and efficiency.

4.4.2 Datasets

We evaluate our method using two datasets that reflect both controlled and real-world conditions: one synthetic (hereafter referred to as the *synthetic dataset*) and one real-world (the *real-world dataset*), as illustrated in Figure 4.4.



(a) Synthetic dataset (b) Real-world dataset

Fig. 4.4 Datasets used in our experiments. (**a**) Synthetic dataset. Blue boxes highlight L-system-based trees characterized by relatively straight and uniform branches, while pink boxes highlight Mtree-based trees exhibiting more pronounced curvature and significant diameter variation. (**b**) Real-world dataset. Images of urban and roadside trees were captured from the sidewalk side to minimize background clutter and occlusion.

We construct two subsets for synthetic data to reflect different modeling challenges. We build the first subset using a 3D extension of the L-system-based tree generation method from TreeFormer [8]. While the original method generates 2D tree structures for image annotation, we adapt it to produce procedurally generated 3D trees. This enables full control over structural parameters and access to ground-truth topology for

quantitative evaluation. We generate 50 random trees with varying branching patterns and morphological complexity. These L-system trees are designed at a moderate scale, approximately 2–5 m in height, matching typical real-world sampling sizes.

To further diversify the dataset and introduce more complex structures, we additionally generate a second synthetic subset using a modular tree framework, Mtree (`https://github.com/MaximeHerpin/modular_tree` (accessed on 10 May 2025)). Similarly, we generate 50 random Mtree trees reaching up to 20 meters in height. These trees exhibit significantly more curved and organically shaped branches, with pronounced diameter variations from trunk to terminal branches. These characteristics present a more challenging testbed for our method, particularly in accurately predicting branch radii.

Our real-world dataset consists of urban and roadside vegetation captured using a Sony α7R V camera (Sony Group Corporation, Tokyo, Japan) at a nominal resolution of 9352×6254 pixels. The scenes include approximately 6 trees, comprising common street-side landscaping trees and ornamental species. Image capture is performed from the sidewalk side to avoid occlusions and suppress background clutter, resulting in partial views and permanent self-occlusions. These conditions introduce substantial visual challenges, including occluded branches, uneven lighting, and non-uniform background textures, thereby simulating practical constraints in real-world estimation tasks. We also note that some tree species exhibit anatomical irregularities, such as galls or nodal swellings, that introduce additional complexity during estimation.

4.4.3 Evaluation Metrics

We use complementary metrics to evaluate both spatial accuracy and topological fidelity of the predicted plant skeletons.

Chamfer Distance To assess geometric similarity, we compute the bidirectional Chamfer distance between the predicted and ground-truth skeletons. For each skeleton, we sample points uniformly along every edge using fixed-length segments. Specif-

ically, we use a spacing of 1 mm for L-system trees and 1 cm for Mtree trees, reflecting the scale differences between the two synthetic subsets. Let $\mathscr{P}_{\text{pred}}$ and $\mathscr{P}_{\text{gt}}$ denote the two resulting point clouds. The Chamfer distance is computed as follows:

$$\text{CD}(\mathscr{P}_{\text{pred}}, \mathscr{P}_{\text{gt}}) = \frac{1}{|\mathscr{P}_{\text{pred}}|} \sum_{p \in \mathscr{P}_{\text{pred}}} \min_{q \in \mathscr{P}_{\text{gt}}} \|p - q\|_2^2 + \frac{1}{|\mathscr{P}_{\text{gt}}|} \sum_{q \in \mathscr{P}_{\text{gt}}} \min_{p \in \mathscr{P}_{\text{pred}}} \|q - p\|_2^2. \quad (4.18)$$

This captures both under-prediction and over-extension in the predicted graph structure.

Node and Edge Count To evaluate topological correctness, we compare the number of key nodes and edges between the predicted and ground-truth graphs. We simplify each graph by removing intermediate points on straight-line segments and retaining only nodes with *degree* $\neq 2$ (*i.e.*, endpoints and branching points). The resulting simplified graph is denoted $\mathscr{G}_{\text{simp}}$, and we report the total number of nodes and edges in this graph as structural indicators. A well-predicted graph should match the ground truth in both key node count and structural connectivity.

Tree Rate To evaluate how well the output graph satisfies the constraint, we calculate the probability that the output graph forms a tree structure. We consider a graph as a tree if it is connected and contains no cycles, *i.e.*, $|\mathscr{E}| = |\mathscr{V}| - 1$.

4.4.4 Baselines

To ensure a comprehensive evaluation, we compare our method with two representative baselines: geometry-based and learning-based methods.

adTree adTree [29] is a classical geometry-based pipeline for accurate and automatic 3D tree estimation. AdTree extracts an initial skeleton by building an MST based on Dijkstra's shortest paths. It iteratively prunes redundant or noisy components, and cylinders are fitted through an optimization process to approximate branch geometry. As a purely geometric method, adTree does not rely on any learned parameters, making

it robust to unseen species but sensitive to the completeness and density of the input point cloud. In practice, the default duplicate removal threshold (0.001) retains too many noisy points, leading to severely over-connected graphs (13,012 nodes vs. 28 in the ground truth). A higher threshold, like 0.1, oversimplifies the structure. Following careful tuning, we adopt 0.05, which generates a reasonable 26-node skeleton close to the ground truth, as shown in Figure 4.5.

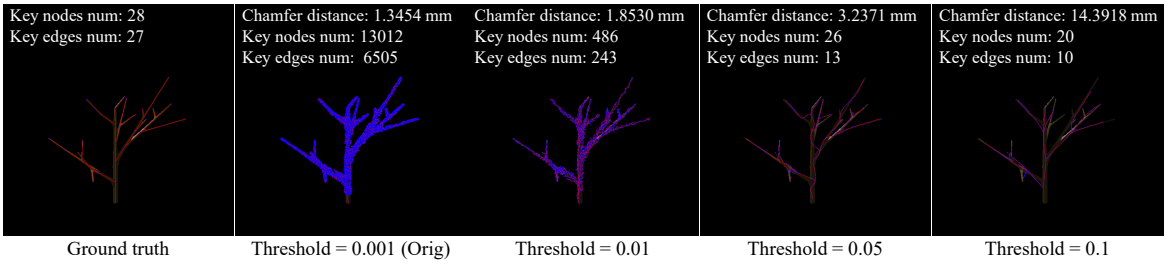


Fig. 4.5 Effect of duplication removal threshold in AdTree. Visual comparison under varying thresholds illustrates that the default value (0.001) leads to excessive node retention and an over-connected graph, artificially lowering the Chamfer score. A high threshold (0.1) oversimplifies the structure, while a moderate threshold (0.05) yields a result close to the ground truth with better structural fidelity.

Smart-Tree Smart-Tree [30] represents a recent learning-based method that approximates branch skeletons from point clouds using a supervised approach. It employs a sparse voxel convolutional neural network to predict each point's radius and medial axis direction, followed by a greedy post-processing algorithm to extract the final skeleton. Unlike geometry-based methods that require species-specific parameter tuning, Smart-Tree learns structural priors from data and generalizes across diverse tree morphologies. However, it relies on large-scale supervised training and may degrade under domain shift.

To ensure a fair comparison, we normalize the input conditions across all methods. Specifically, we transform our predicted point cloud into the same upright orientation as the baselines and rescale it to match the overall bounding box dimensions of each baseline' s output. This process reduces discrepancies on a global scale and orientation, ensuring that structural evaluation focuses on skeleton quality rather than

Table 4.1 Quantitative evaluation on synthetic datasets. The Chamfer distance measures geometric similarity. $\Delta|E|$ and $\Delta|N|$ represent the absolute differences in the number of predicted edges and nodes relative to the ground-truth counts (average graph sizes: 29.32 key nodes and 28.32 key edges for the L-system-based dataset; 15.36 key nodes and 14.40 key edges for the Mtree dataset).

Method	L-system-based dataset				Mtree dataset			
	Chamfer distance (mm)↓	$\Delta\|E\|$↓	$\Delta\|N\|$↓	**Tree rate** % ↓	**Chamfer distance (cm)**↓	$\Delta\|E\|$↓	$\Delta\|N\|$↓	**Tree rate** % ↓
AdTree	3.06±0.68	14.18	1.04	0.0	53.91±79.19	**1.52**	10.40	0.0
Smart-Tree	8.86±16.66	2.22	22.28	0.0	31.15±76.69	3.66	20.76	0.0
Ours	**0.19±0.20**	**0.22**	**0.22**	**100.0**	**15.66±8.12**	2.28	**2.24**	**100.0**

external input variation.

4.4.5 Results on the Synthetic Dataset

We evaluate all methods on two synthetic datasets with distinct structural properties: the L-system-based and Mtree datasets. Table 4.1 presents quantitative comparisons.

L-system-based Dataset Figure 4.6 presents qualitative comparisons on the L-system-based dataset. From top to bottom, we show the ground-truth skeletons and results from our method, AdTree, and Smart-Tree. Visually, our method exhibits the closest alignment with the green masks in branch radii and node placement.

AdTree tends to generate overly curved structures, even when the underlying tree shape is relatively straight, which introduces geometric bias. Although Smart-Tree can produce dense point distributions, it often fails to capture thin branches, leading to incomplete skeletons and high Chamfer distance variance. Quantitatively, our method achieves the lowest average and standard deviation in Chamfer distance (0.19 ± 0.20 mm) and a minor error in node count. AdTree also performs relatively well in terms of connectivity, while SmartTree suffers from inconsistent predictions. Furthermore, ours is the only method that yields a valid tree in every test case (Tree Rate 100%), while both baselines produce graphs with cycles or disconnected parts (Tree Rate 0%).

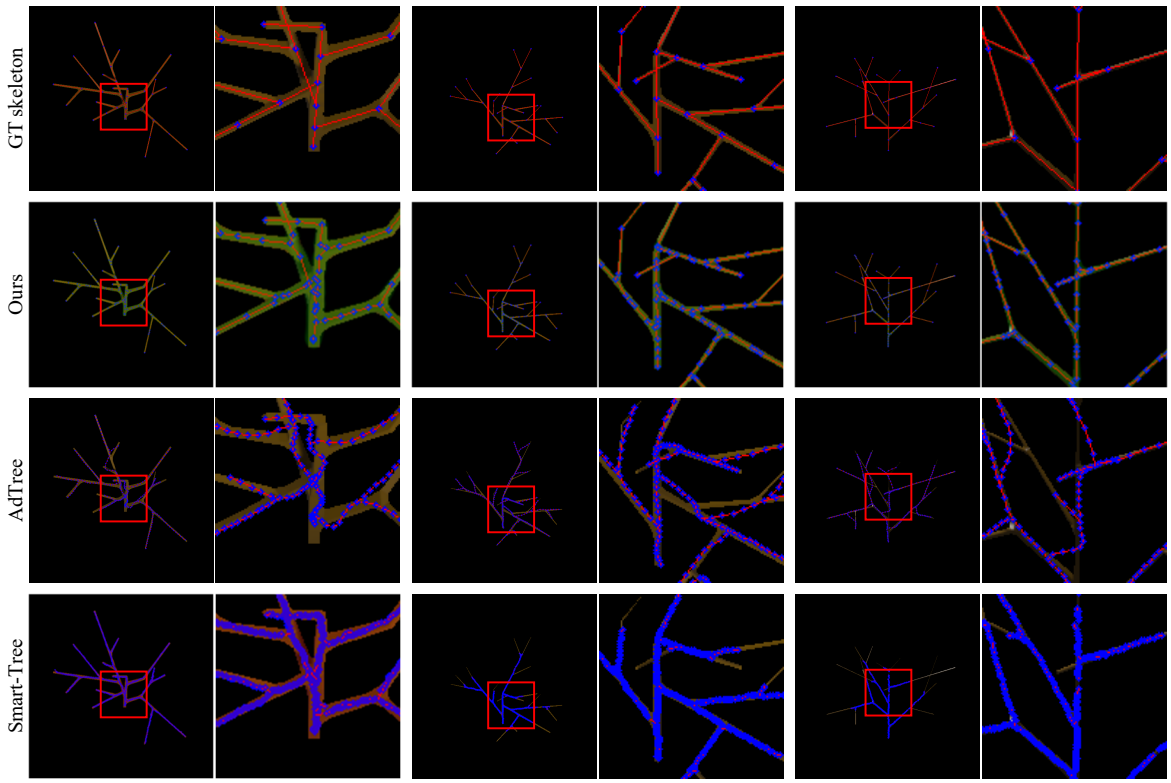


Fig. 4.6 **Qualitative results on the L-system-based dataset**. From top to bottom: ground-truth skeletons, our method, AdTree, and Smart-Tree. We overlay green MST-based mask contours to visualize the alignment between predicted radii and ground-truth structure. Our method consistently achieves the most accurate and well-aligned predictions. AdTree tends to overfit straight structures with curved paths, while Smart-Tree suffers from over-segmentation and fails to reconstruct thin branches, despite its high point density.

Mtree Dataset Figure 4.7 shows the more challenging Mtree dataset results. AdTree continues producing geometrically biased, over-curved structures due to its reliance on global point trends that can smooth nearby branches. Smart-Tree performs better overall but often fails to recover thin terminal branches.

Our method remains robust across these conditions, producing structurally complete skeletons and accurately modeling branch radii variation. We again achieve the lowest Chamfer distance (15.66 ± 8.12 cm) and perfect tree rate (100%), validating the strength of our mask-guided refinement and structure-aware graph optimization.

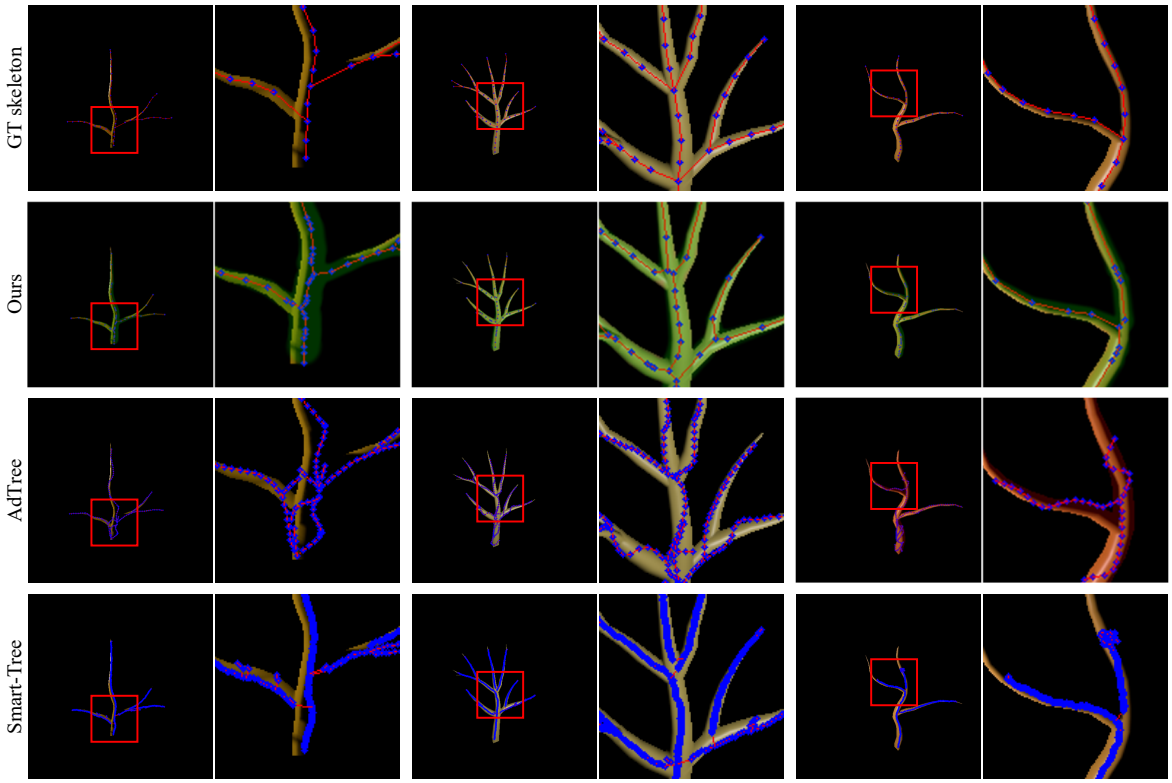


Fig. 4.7 Qualitative results on the Mtree dataset. From top to bottom: ground-truth skeletons, our method, AdTree, and Smart-Tree. As with the L-system results, green MST-based masks highlight branch regions. While all methods show similar trends, the increased curvature and diameter variation in Mtree trees amplify the weaknesses of AdTree and Smart-Tree. Our method remains the most faithful to the ground truth in both geometry and structure.

4.4.6 Results on the Real-World Dataset

We evaluate all methods on six real-world trees, including five common urban landscaping species and one potted plant. As shown in Figure 4.8 and Figure 4.9, the visual complexity of natural scenes, such as occlusion, irregular branching, and thin structures, amplifies the limitations of baseline methods.

AdTree continues to produce overly curved branches, especially in trunk-adjacent regions, as it globally fits trends in the point cloud and often merges nearby but distinct branches into a single curved trajectory. Smart-Tree, which is based on clustering, frequently produces disconnected components rather than coherent tree-like

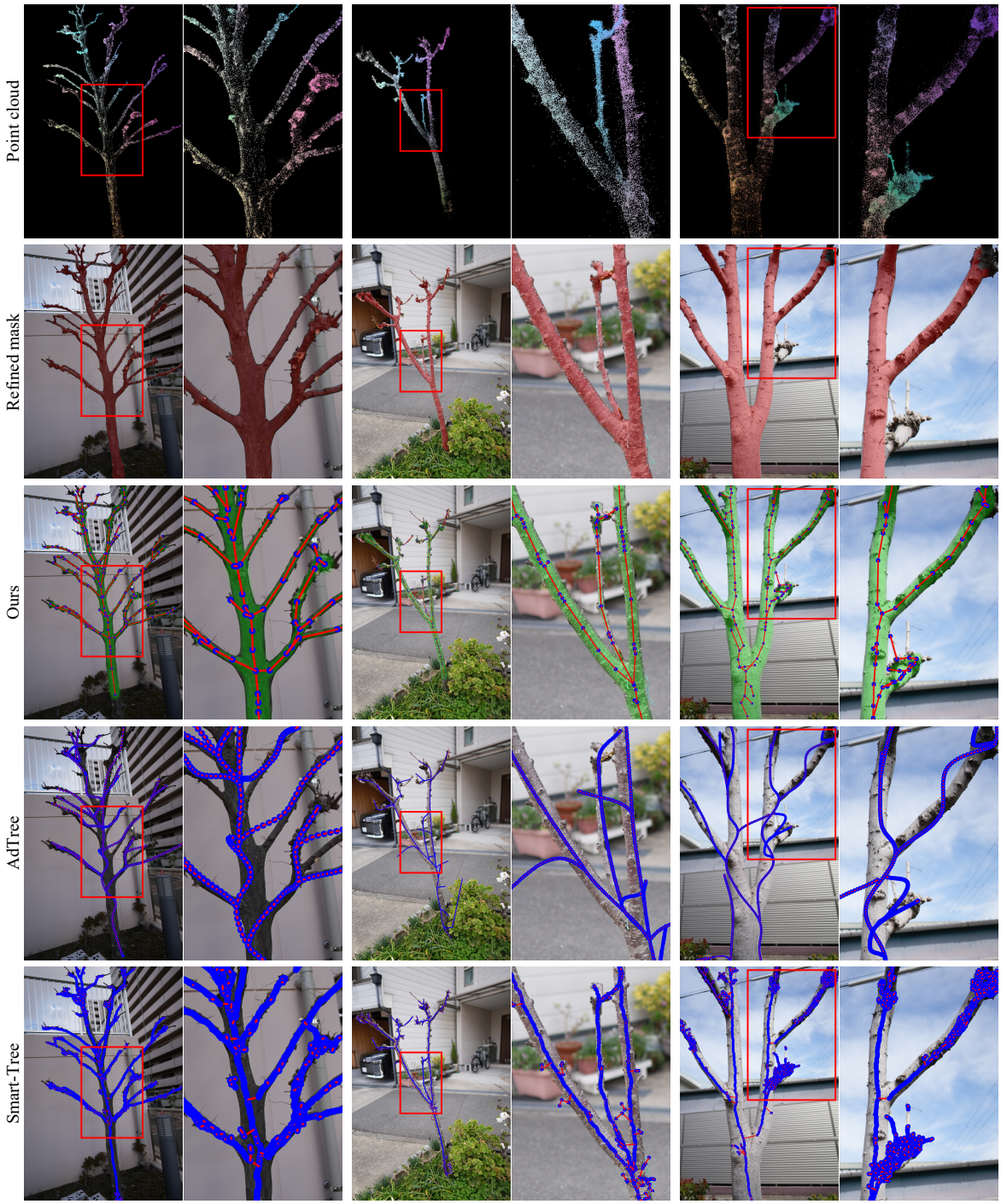


Fig. 4.8 Qualitative results on the real-world dataset. From top to bottom: point cloud, refined mask, our method, AdTree, and Smart-Tree. We overlay green MST-based mask contours to visualize the alignment between predicted radii and ground-truth structure. To enhance clarity, the visualization thickness of nodes (blue) and edges (red) is adjusted based on tree species.

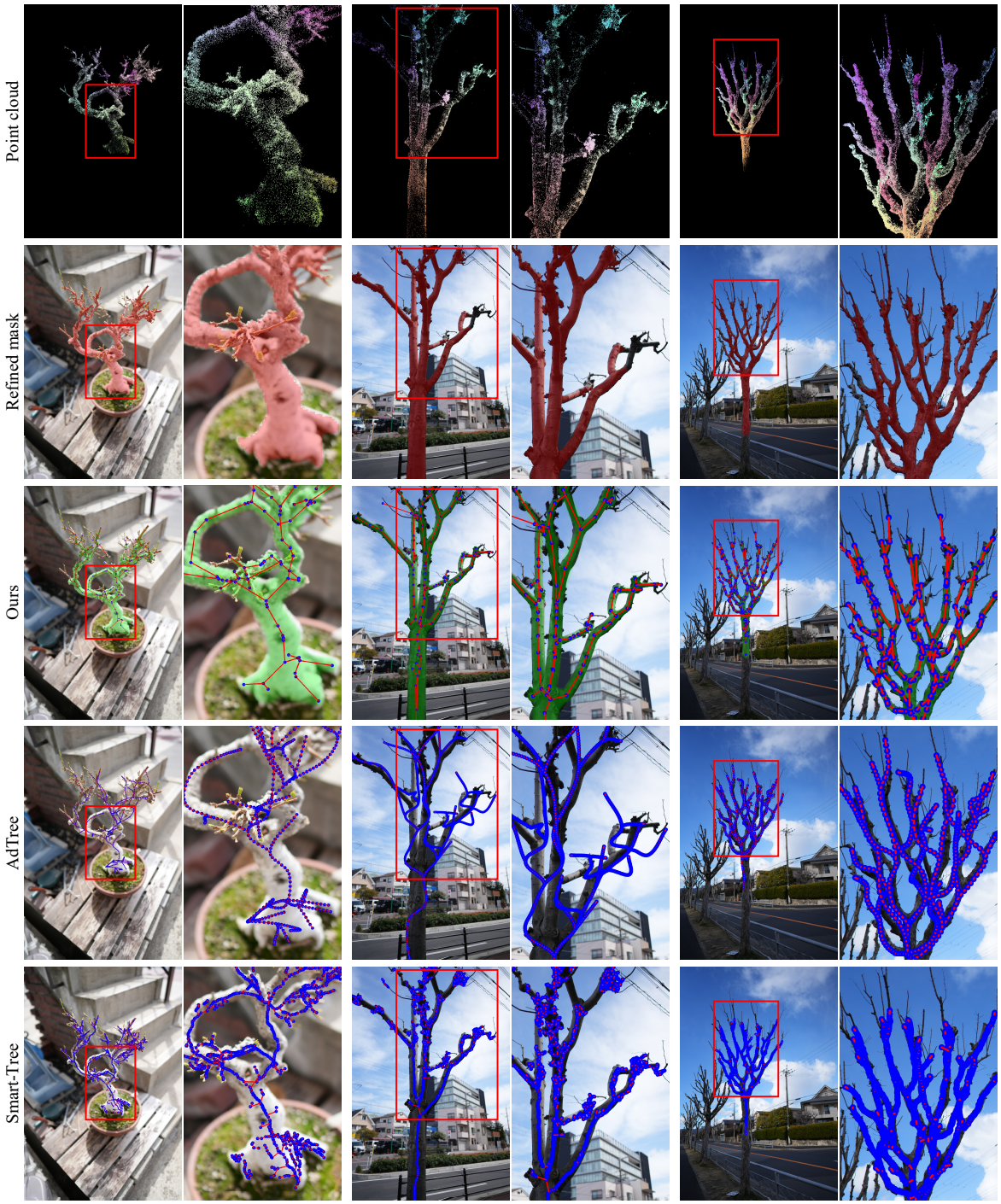


Fig. 4.9 Qualitative results on the real-world dataset. From top to bottom: point cloud, refined mask, our method, AdTree, and Smart-Tree. We overlay green MST-based mask contours to visualize the alignment between predicted radii and ground-truth structure. To enhance clarity, the visualization thickness of nodes (blue) and edges (red) is adjusted based on tree species.

structures. Moreover, continuous geometry is often over-segmented due to local variations in point density along a single branch. It introduces spurious thin sub-branches, especially in denser regions, resulting in structural noise even along otherwise straight branches.

In addition, we observe that certain anatomical features, such as nodal swellings, galls, or terminal ends of branches, often receive disproportionately dense matches during MVS reconstruction. These localized point cloud accumulations are not representative of true geometric importance. However, they pose significant challenges for baseline methods: AdTree tends to overfit these regions, generating undesired curves near trunk junctions, while Smart-Tree often segments them into densely stacked components, introducing clutter and misinterpreted sub-structures.

In contrast, our method maintains structural connectivity and faithfully estimates branching geometry. Notably, even when refined 2D masks (shown in red) are incomplete in certain views, our model concludes plausible geometry by integrating the multi-view masks' distribution through the soft blur of 3DGS. All estimated graphs are valid trees (tree rate 100%), and the alignment between predicted radii and the overlaid green MST masks confirms high fidelity in topology and shape.

4.4.7 Ablation Study

We conduct controlled ablations to isolate the effect of each major loss term. As shown in Figure 4.10, removing any of the two key objectives leads to degraded geometry and topology.

Without Geometry Regularization ($\mathscr{L}_{graph}$) Disabling the graph geometry regularization leads to unnatural tree-like geometry. In particular, without constraints on edge length, angle smoothness, or directional spacing, the estimated graph loses key features in real plants, resulting in unnatural zig-zagging edges, sharp angular bends, and uneven node spacing.

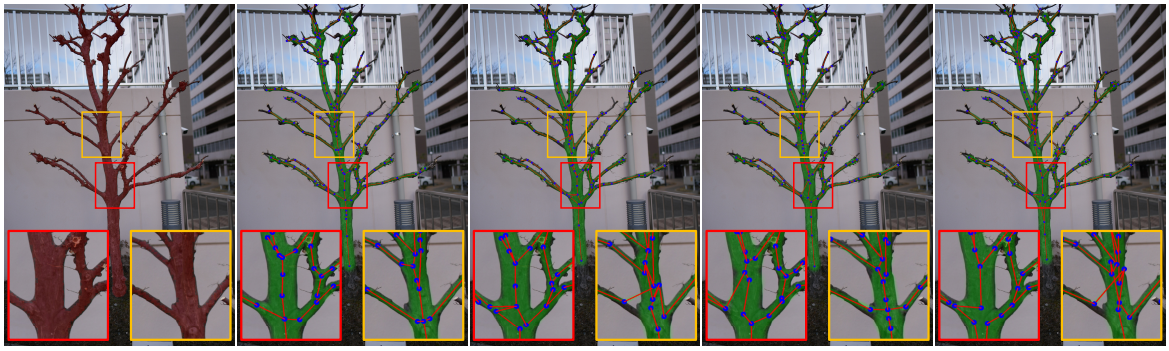


Fig. 4.10 Ablation study on loss components. From left to right: refined mask, full model (ours), without $\mathscr{L}_{\text{graph}}$, without the SFS layer, and without both. All results are visualized on the same real-world scene with the predicted skeleton overlaid in green. Removing geometry regularization causes jagged edges and sharp angles, while removing the SFS layer disrupts edge connectivity realism. Removing both leads to severe geometry and topology degradation, confirming the importance of each loss term.

Without an SFS Layer The SFS layer softly reweights the predicted adjacency matrix to align with the MST structure. Unlike the geometric artifacts caused by removing $\mathscr{L}_{\text{graph}}$, which disrupts local angles and spacing, removing the SFS layer leads to incorrect edge connectivity. This topological inconsistency undermines the plausibility of branch bifurcations and the overall tree structure.

Without an SFS Layer and Geometry Regularization Removing both the SFS layer and $\mathscr{L}_{\text{graph}}$ combines the weaknesses of topological and geometric inconsistency. The predicted graph exhibits misplaced connections, zig-zagging edges, sharp angles, and uneven spacing. Without any structural prior, the graph fails to capture both the physical realism and the edge-connect logic of natural trees.

Quantitative results on synthetic datasets. To complement the qualitative observations above, we further conducted controlled ablations on synthetic datasets where ground-truth skeletons are available. As shown in Table 4.2, removing either $\mathscr{L}_{\text{graph}}$ or the SFS layer degrades performance in terms of Chamfer distance as well as node/edge count errors. The full model achieves the lowest Chamfer distance

and the smallest deviations in graph size across both L-system and Mtree datasets, confirming that geometry regularization and the SFS layer contribute complementary improvements in accuracy and structural fidelity.

Table 4.2 Quantitative ablation study on synthetic datasets. Chamfer distance measures geometric similarity, while $\Delta|E|$ and $\Delta|N|$ denote absolute differences in predicted edge and node counts relative to ground-truth. Average ground-truth graph sizes: 29.32 nodes / 28.32 edges for the L-system dataset, 15.36 nodes / 14.40 edges for the Mtree dataset.

Method	**L-system-based dataset**			**Mtree dataset**		
	Chamfer distance (mm)↓	$\Delta\|E\|$↓	$\Delta\|N\|$↓	**Chamfer distance (cm)**↓	$\Delta\|E\|$↓	$\Delta\|N\|$↓
W/o Geometry loss	1.11±1.69	0.32	0.32	17.83±11.99	2.50	2.47
W/o SFS layer	0.99±1.68	0.28	0.28	24.60±13.41	3.60	3.57
W/o SFS + Geometry	0.19±0.35	0.39	0.39	17.12±5.90	3.20	3.17
Full model (Ours)	**0.19±0.20**	**0.22**	**0.22**	**15.66±8.12**	**2.28**	**2.24**

4.5 Conclusion

We introduced a differentiable graph optimization framework for estimating 3D tree skeletons from multi-view segmentation masks. Our method jointly refines node locations, branch radii, and edge probabilities by coupling an MST-initialized graph with 3D Gaussian rendering. Thanks to the parameterization of the adjacency matrix and the mask-guided optimization, every prediction yields a valid tree structure. On both synthetic and real-world datasets, we achieve a 100% tree rate while significantly reducing Chamfer distance, node count error, and edge error compared to AdTree and Smart-Tree. The soft blur in 3DGS makes our approach robust to pose noise and partial mask inaccuracies.

Limitations and Future Work Our datasets contain bare or lightly occluded branches. Heavy foliage may degrade mask quality and, by extension, silhouette supervision. We plan to collect leaf-on scenes and investigate joint leaf/branch segmentation to maintain accuracy under severe occlusion. Compared with purely geometric

pipelines like AdTree, our mask-guided refinement adds roughly five minutes of optimization per tree on a single RTX 4090. Although acceptable for offline processing, this overhead prevents real-time deployment. Future work will explore incremental optimization and a lightweight neural renderer to shorten convergence time.

Chapter 5

Discussion and Conclusion

This chapter presents a brief summary of the thesis and outlines concrete directions for future research that build on the dataset, models, and analyses presented earlier.

5.1 Summary

This thesis aims at making plant skeleton estimation both structurally faithful and broadly generalizable, with the ultimate goal of supporting downstream tasks in phenotyping, growth modeling, and agricultural robotics. To achieve this goal, we address three fundamental problems that are commonly shared by existing methods. They are summarized as follows.

- Existing datasets provide only branch-level supervision and suffer from strong domain shifts, limiting cross-species and cross-style generalization (Chapter 2).
- Current 2D image-to-graph models decode graphs without structural constraints, often yielding cycles or disconnected components that break the natural tree structure (Chapter 3).
- Point-cloud–based 3D estimation pipelines are fragile under sparse or uneven density, causing fragmentation and unreliable skeletonization (Chapter 4).

A summary of the introduced datasets, algorithms, and techniques in each chapter is presented in the following.

Chapter 2: PlantPose dataset We establish the PlantPose dataset, the first unified benchmark for plant skeleton estimation, consolidating ten heterogeneous subsets into a single canonical representation. To overcome the supervision gap of existing datasets, the PlantPose dataset converts branch-level annotations into complete tree graphs using automated pipelines. The dataset is designed according to four guiding principles: connectivity awareness, diversity across biology, diversity across organs, and diversity across styles and scenes. It includes real-world datasets such as MIPDB [1], Grapevine [2], and LVD2021 [3], as well as extensions through self-captured, web-sourced, and root data. In addition, synthetic d atasets generated by L-systems and the Mtree framework broaden structural and visual diversity. To enable fair assessment of generalization, we standardize all data into a unified graph representation and propose a generalized train/test split with six training datasets and four out-of-domain test sets, explicitly probing organ, style, background, and shape shifts.

Chapter 3: PlantPose2D We introduce PlantPose2D, the first end-to-end framework for tree-constrained graph generation from a single plant image. Unlike existing image-to-graph models that decode unconstrained structures and enforce the tree topology only through post-processing, our method integrates structural priors directly into training. The key component is the selective feature suppression layer, which projects intermediate edge scores onto an MST and suppresses features that would otherwise induce cycles or disconnections. This design ensures the predicted graphs remain visually natural and topologically valid throughout optimization.

Through extensive experiments across six training datasets and four out-of-domain test datasets, PlantPose2D achieves consistently higher tree rates and lower structural errors than unconstrained, two-stage, and test-time constraint baselines. In particular, the model shows strong generalization to challenging conditions, including background clutter, hand-drawn line styles, and novel species. The results confirm that incorpo-

rating tree constraints into both learning and inference yields skeletons that are not only geometrically accurate but also robust across diverse domains.

Chapter 4: PlantPose3D We propose PlantPose3D, a mask-guided graph optimization framework that estimates 3D plant skeletons directly from multi-view segmentation masks. Unlike conventional pipelines that rely on dense and uniform point clouds, our method bypasses the fragility of point-cloud–based skeletonization by optimizing a graph representation with differentiable 3D Gaussian splatting. Starting from an MST-initialized structure, the framework refines node positions, branch radii, and connectivity to minimize silhouette loss across views, ensuring geometric fidelity and strict tree validity.

Extensive evaluations on synthetic and real-world data demonstrate that PlantPose3D consistently achieves a 100% tree rate while significantly reducing Chamfer distance, node count error, and edge error compared to baselines such as AdTree and Smart-Tree. The soft blur of 3DGS further mitigates pose noise and incomplete masks, enabling robust estimation under challenging imaging conditions. These results confirm that embedding silhouette-based supervision into a structure-aware graph optimization pipeline provides a principled and effective alternative to traditional point-cloud skeletonization.

5.2 Future Work

Although the proposed dataset, 2D model, and 3D framework address the three central challenges of plant skeleton estimation, several directions remain open for further exploration:

- **Expanding organ and species coverage.** PlantPose consolidates shoots, leaf veins, and roots, but does not yet capture mature organs with dense foliage or seasonal variation. Future extensions should include leaf-on scenes and multi-organ integration, enabling models to reason jointly about stems, leaves, and

other organs under realistic growth conditions.

- **Acceleration of 2D inference.** PlantPose2D currently runs at only 0.3–0.5 fps, which is insufficient for real-time applications such as automated pruning or robotic perception. A promising direction is developing lighter and faster baselines that retain tree-graph fidelity while significantly improving inference speed, facilitating large-scale deployment in natural conditions.

- **Beyond single-instance graphs.** Our current 2D pipeline assumes a single plant per image. However, plants are densely distributed in natural or agricultural environments and often overlap. Future work could extend the framework toward multi-instance skeleton parsing, where strategies such as minimum spanning *forests* may be explored to enforce connectivity within each plant while separating multiple individuals simultaneously.

- **Handling heavy foliage and occlusion in 3D.** Current 3D experiments are primarily conducted on bare or lightly occluded plants. In practice, leaves often obscure stems, making mask-based supervision unreliable. An important extension is to develop models that predict probabilistic distributions of hidden skeletons, allowing trunk or branch structures to be inferred even under dense leaf occlusion.

Outlook. Taken together, these future directions point toward a more complete and practical framework for plant structure analysis. By extending coverage to complex organs, improving inference speed, and enabling multi-instance and occlusion-robust skeletonization, we move closer to scalable tools that can operate in real agricultural environments. Ultimately, these advances will support next-generation phenotyping platforms, enable more autonomous agricultural robots, and deepen biological understanding of plant growth in realistic field conditions.

References

[1] P. Wang, J. Chang, W. Deng, B. Liu, H. Lai, Z. Hou, L. Dong, Q. Chen, Y. Zhou, Z. Zhang, H. Liu, and J. Ruan, "MIPDB: A maize image-phenotype database with multi-angle and multi-time characteristics," *bioRxiv*, 2024.

[2] T. Gentilhomme, M. Villamizar, J. Corre, and J.-M. Odobez, "Towards smart pruning: ViNet, a deep-learning approach for grapevine structure estimation," *Computers and Electronics in Agriculture*, vol. 207, p. 107736, 2023.

[3] L. Li, W. Hu, J. Lu, and C. Zhang, "Leaf vein segmentation with self-supervision," *Computers and Electronics in Agriculture*, 2022.

[4] S. Shit, R. Koner, B. Wittmann, J. Paetzold, I. Ezhov, H. Li, J. Pan, S. Sharifzadeh, G. Kaissis, V. Tresp *et al.*, "Relationformer: A unified framework for image-to-graph generation," in *Proceedings of the European Conference on Computer Vision*, 2022, pp. 422–439.

[5] D. Belli and T. Kipf, "Image-conditioned graph generation for road network extraction," in *Proceedings of NeurIPS Workshop on Graph Representation Learning*, 2019.

[6] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7291–7299.

[7] F. Okura, "3D modeling and reconstruction of plants and trees: A cross-cutting review across computer graphics, vision, and plant phenotyping," *Breeding Science*, vol. 72, no. 1, pp. 31–47, 2022.

[8] X. Liu, H. Santo, Y. Toda, and F. Okura, "Treeformer: Single-view plant skeleton estimation via tree-constrained graph generation," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, February 2025, pp. 8154–8164.

[9] L. Cabrera-Bosquet, C. Fournier, N. Brichet, C. Welcker, B. Suard, and F. Tardieu, "High-throughput estimation of incident light, light interception and radiation-use efficiency of thousands of plants in a phenotyping platform," *New Phytologist*, vol. 212, no. 1, pp. 269–281, 2016.

[10] S. Wu, W. Wen, B. Xiao, X. Guo, J. Du, C. Wang, and Y. Wang, "An accurate skeleton extraction approach from 3d point clouds of maize plants," *Frontiers in Plant Science*, vol. 10, p. 248, 2019.

[11] R. Arikapudi and S. Vougioukas, "Robotic tree-fruit harvesting with arrays of cartesian arms: A study of fruit pick cycle times," *Computers and Electronics in Agriculture*, vol. 211, p. 108023, 08 2023.

[12] A. Zahid, M. S. Mahmud, L. He, P. Heinemann, D. Choi, and J. Schupp, "Technological advancements towards developing a robotic pruner for apple trees: A review," *Computers and Electronics in Agriculture*, vol. 189, p. 106383, 10 2021.

[13] J. L. Cárdenas, C. J. Ogayar, F. R. Feito, and J. M. Jurado, "Modeling of the 3d tree skeleton using real-world data: A survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 12, pp. 4920–4935, 2023.

[14] R. Hartley, S. Jayathunga, J. Morgenroth, and G. Pearse, "Tree branch characterisation from point clouds: a comprehensive review," *Current Forestry Reports*, vol. 10, pp. 360–385, 07 2024.

[15] V. Giuffrida, M. Minervini, and S. Tsaftaris, "Learning to count leaves in rosette plants," in *Proceedings of Workshop on Computer Vision Problems in Plant Phenotyping*, 01 2015, pp. 1.1–1.13.

[16] T. Isokane, F. Okura, A. Ide, Y. Matsushita, and Y. Yagi, "Probabilistic plant modeling via multi-view image-to-image translation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2906–2915.

[17] M. Ai, Y. Yao, Q. Hu, Y. Wang, and W. Wang, "An automatic tree skeleton extraction approach based on multi-view slicing using terrestrial LiDAR scans data," *Remote Sensing*, vol. 12, no. 22, 2020.

[18] S. Wu, W. Wen, Y. Wang, J. Fan, C. Wang, W. Gou, and X. Guo, "MVS-Pheno: A portable and low-cost phenotyping platform for maize shoots using multiview stereo 3D reconstruction," *Plant Phenomics*, vol. 2020, 2020.

[19] D. Acuna, H. Ling, A. Kar, and S. Fidler, "Efficient interactive annotation of segmentation datasets with polygon-RNN++," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 859–868.

[20] W. Li, Y. Lu, K. Zheng, H. Liao, C. Lin, J. Luo, C.-T. Cheng, J. Xiao, L. Lu, C.-F. Kuo *et al.*, "Structured landmark detection via topology-adapting deep graph learning," in *Proceedings of the European Conference on Computer Vision*, 2020, pp. 266–283.

[21] W. Li, W. Zhao, H. Zhong, C. He, and D. Lin, "Joint semantic-geometric learning for polygonal building segmentation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021, pp. 1958–1965.

[22] H. Ling, J. Gao, A. Kar, W. Chen, and S. Fidler, "Fast interactive object annotation with curve-GCN," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5252–5261.

[23] I. Khokhlov, L. Krasnov, M. V. Fedorov, and S. Sosnin, "Image2SMILES: Transformer-based molecular optical recognition engine," *Chemistry Methods*, vol. 2, no. 1, p. e202100069, 2022.

[24] R. Li, S. Zhang, and X. He, "SGTR: End-to-end scene graph generation with transformer," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 19 486–19 496.

[25] J. Liang, N. Homayounfar, W.-C. Ma, Y. Xiong, R. Hu, and R. Urtasun, "PolyTransform: Deep polygon transformer for instance segmentation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9128–9137.

[26] X. Liang, V. Kankare, J. Hyyppä, Y. Wang, A. Kukko, H. Haggrén, X. Yu, H. Kaartinen, A. Jaakkola, F. Guan, M. Holopainen, and M. Vastaranta, "Terrestrial laser scanning in forest inventories," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 115, pp. 63–77, 2016.

[27] K. Olofsson, J. Holmgren, and H. Olsson, "Tree stem and height measurements using terrestrial laser scanning and the ransac algorithm," *Remote Sensing*, vol. 6, no. 5, pp. 4323–4344, 2014.

[28] A. Bucksch, "A practical introduction to skeletons for the plant sciences," *Applications in Plant Sciences*, vol. 2, no. 8, p. 1400005, 2014.

[29] S. Du, R. Lindenbergh, H. Ledoux, J. Stoter, and L. Nan, "AdTree: Accurate, detailed, and automatic modelling of laser-scanned trees," *Remote Sensing*, vol. 11, no. 18, 2019.

[30] H. Dobbs, O. Batchelor, R. Green, and J. Atlas, "Smart-tree: Neural medial axis approximation of point clouds for 3d tree skeletonization," in *Proceedings on Pattern Recognition and Image Analysis*, 2023, pp. 351–362.

[31] Y. Liu, J. Guo, B. Benes, O. Deussen, X. Zhang, and H. Huang, "TreePartNet: neural decomposition of point clouds for 3D tree reconstruction," *ACM Transactions on Graphics*, vol. 40, no. 6, p. 232, 2021.

[32] J. Kotary, F. Fioretto, P. Van Hentenryck, and B. Wilder, "End-to-end constrained optimization learning: A survey," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2021.

[33] Y.-S. Myung, C.-H. Lee, and D.-W. Tcha, "On the generalized minimum spanning tree problem," *Networks*, vol. 26, no. 4, pp. 231–241, 1995.

[34] P. C. Pop, "The generalized minimum spanning tree problem: An overview of formulations, solution procedures and latest advances," *European Journal of Operational Research*, vol. 283, no. 1, pp. 1–15, 2020.

[35] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, "3D Gaussian splatting for real-time radiance field rendering," *ACM Transactions on Graphics*, vol. 42, no. 4, pp. 139:1–139:14, 2023.

[36] N. Asada, X. Liu, K. Xu, R. Miyakawa, Y. Yang, H. Santo, Y. Toda, and F. Okura, "Leafgen: Structure-aware leaf image generation for annotation-free leaf instance segmentation," *Plant Phenomics*, 2025.

[37] X. Liu, K. Xu, R. Shinoda, H. Santo, and F. Okura, "Masks-to-skeleton: Multi-view mask-based tree skeleton extraction with 3d gaussian splatting," *Sensors*, vol. 25, no. 14, 2025.

[38] J. Guo, H. Jiang, B. Benes, O. Deussen, X. Zhang, D. Lischinski, and H. Huang, "Inverse procedural modeling of branching structures by inferring L-systems," *ACM Transactions on Graphics*, vol. 39, no. 5, pp. 1–13, 2020.

[39] A. Lindenmayer, "Mathematical models for cellular interactions in development I. Filaments with one-sided inputs," *Journal of Theoretical Biology*, vol. 18, no. 3, pp. 280–299, 1968.

[40] M. Gaillard, C. Miao, J. c. Schnable, and B. Benes, "Sorghum segmentation by skeleton extraction," in *Proceedings of the European Conference on Computer Vision Workshops*, 2020, pp. 296–311.

[41] T. Miao, C. Zhu, T. Xu, T. Yang, N. Li, Y. Zhou, and H. Deng, "Automatic stem-leaf segmentation of maize shoots using three-dimensional point cloud," *Computers and Electronics in Agriculture*, vol. 187, p. 106310, 2021.

[42] S. He, F. Bastani, S. Jagwani, M. Alizadeh, H. Balakrishnan, S. Chawla, M. M. Elshrif, S. Madden, and M. A. Sadeghi, "Sat2Graph: Road graph extraction through graph-tensor encoding," in *Proceedings of the European Conference on Computer Vision*, 2020, pp. 51–67.

[43] Z. Xu, Y. Liu, Y. Sun, M. Liu, and L. Wang, "RNGDet++: Road network graph detection by transformer with instance segmentation and multi-scale features enhancement," *IEEE Robotics and Automation Letters*, pp. 1–8, 2023.

[44] Z. Xu, Y. Sun, and M. Liu, "iCurb: Imitation learning-based detection of road curbs using aerial images for autonomous driving," *IEEE Robotics and Automation Letters*, vol. 6, pp. 1097–1104, 2021.

[45] H. Xu, N. Gossett, and B. Chen, "Knowledge and heuristic-based modeling of laser-scanned trees," *ACM Transactions on Graphics*, vol. 26, no. 4, p. 19, 2007.

[46] H. Huang, S. Wu, D. Cohen-Or, M. Gong, H. Zhang, G. Li, and B. Chen, "L1-medial skeleton of point cloud," *ACM Transactions on Graphics*, vol. 32, no. 4, p. 65, 2013.

[47] I. Ziamtsov and S. Navlakha, "Machine learning approaches to improve three basic plant phenotyping tasks using three-dimensional point clouds," *Plant Physiology*, vol. 181, no. 4, pp. 1425–1440, 2019.

[48] A. Chaudhury and C. Godin, "Skeletonization of plant point cloud data using stochastic optimization framework," *Frontiers in Plant Science*, vol. 11, 2020.

[49] Y. Cong, M. Yang, and B. Rosenhahn, "RelTR: Relation transformer for scene graph generation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 9, pp. 11 169–11 183, 2023.

[50] S. Abdelkarim, A. Agarwal, P. Achlioptas, J. Chen, J. Huang, B. Li, K. W. Church, and M. Elhoseiny, "Exploring long tail visual relationship recognition with large vocabulary," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2020, pp. 15 901–15 910.

[51] M.-J. Chiou, H. Ding, H. Yan, C. Wang, R. Zimmermann, and J. Feng, "Recovering the unbiased scene graphs from the biased ones," in *Proceedings of ACM International Conference on Multimedia*, 2021, pp. 1581–1590.

[52] S. Sharifzadeh, S. M. Baharlou, M. Schmitt, H. Schütze, and V. Tresp, "Improving scene graph classification by exploiting knowledge from texts," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022, pp. 2189–2197.

[53] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2014, pp. 1532–1543.

[54] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. Battaglia, "Learning deep generative models of graphs," *arXiv preprint arXiv:1803.03324*, 2018.

[55] Q. Liu, M. Allamanis, M. Brockschmidt, and A. L. Gaunt, "Constrained graph variational autoencoders for molecule design," in *Advances in Neural Information Processing Systems*, 2018.

[56] Y. Luo, K. Yan, and S. Ji, "Graphdf: A discrete flow model for molecular graph generation," in *Proceedings of the International Conference on Machine Learning*, 2021, pp. 7192–7203.

[57] X. Zhou, B. Li, B. Benes, S. Fei, and S. Pirk, "Deeptree: Modeling trees with situated latents," *IEEE Transactions on Visualization and Computer Graphics*, 2023.

[58] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proceedings of the International Conference on Learning Representations*, Y. Bengio and Y. LeCun, Eds., 2014.

[59] T. N. Kipf and M. Welling, "Variational graph auto-encoders," in *Proceedings on NeurIPS Workshop on Bayesian Deep Learning*, 2016.

[60] J. Yang, Y. Z. Ang, Z. Guo, K. Zhou, W. Zhang, and Z. Liu, "Panoptic scene graph generation," in *Proceedings of the European Conference on Computer Vision*, 2022, pp. 178—-196.

[61] C. Zang and F. Wang, "Moflow: An invertible flow model for generating molecular graphs," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 617–626.

[62] S. Ahn, B. Chen, T. Wang, and L. Song, "Spanning tree-based graph generation for molecules," in *Proceedings of the International Conference on Learning Representations*, 2022.

[63] W. Jin, R. Barzilay, and T. Jaakkola, "Junction tree variational autoencoder for molecular graph generation," in *Proceedings of the International Conference on Machine Learning*, 2018, pp. 2323–2332.

[64] ——, "Hierarchical generation of molecular graphs using structural motifs," in *Proceedings of the International Conference on Machine Learning*, 2020.

[65] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[66] B. Wilder, B. Dilkina, and M. Tambe, "Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, pp. 1658–1665.

[67] A. Ferber, B. Wilder, B. Dilkina, and M. Tambe, "Mipaal: Mixed integer program as a layer," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2020, pp. 1504–1511.

[68] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *Proceedings of the International Conference on Learning Representations*, 2014.

[69] C. Xie, M. Tan, B. Gong, J. Wang, A. L. Yuille, and Q. V. Le, "Adversarial examples improve image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 819–828.

[70] H. Gupta, K. H. Jin, H. Q. Nguyen, M. T. McCann, and M. Unser, "CNN-based projected gradient descent for consistent CT image reconstruction," *IEEE Transactions on Medical Imaging*, vol. 37, no. 6, pp. 1440–1453, 2018.

[71] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proceedings of the International Conference on Learning Representations*, 2018.

[72] M. Yoshida, A. Torii, M. Okutomi, K. Endo, Y. Sugiyama, R.-i. Taniguchi, and H. Nagahara, "Joint optimization for compressive video sensing and reconstruction under hardware constraints," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 634–649.

[73] S. Bohlender, I. Öksüz, and A. Mukhopadhyay, "A survey on shape-constraint deep learning for medical image segmentation," *IEEE Reviews in Biomedical Engineering*, vol. 16, pp. 225–240, 2021.

[74] Y. Li, C. P. Ho, M. Toulemonde, N. Chahal, R. Senior, and M.-X. Tang, "Fully automatic myocardial segmentation of contrast echocardiography sequence using random forests guided by shape model," *IEEE Transactions on Medical Imaging*, vol. 37, no. 5, pp. 1081–1091, 2017.

[75] X. Zhu, W. Su, L. Lu, B. Li, X. Wang, and J. Dai, "Deformable DETR: Deformable transformers for end-to-end object detection," in *Proceedings of the International Conference on Learning Representations*, 2021.

[76] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[77] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, 1956.

[78] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *Proceedings of the European Conference on Computer Vision*, 2016, pp. 11–16.

[79] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[80] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2117–2125.

[81] S. He, F. Bastani, S. Abbar, M. Alizadeh, H. Balakrishnan, S. Chawla, and S. Madden, "RoadRunner: Improving the precision of road network inference

from gps trajectories," in *Proceedings of ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2018, pp. 3–12.

[82] X. Lyu, L. Cheng, and S. Zhang, "The reta benchmark for retinal vascular tree analysis," *Scientific Data*, vol. 9, no. 1, p. 397, 2022.

[83] F. Bastani, S. He, S. Abbar, M. Alizadeh, H. Balakrishnan, S. Chawla, S. Madden, and D. DeWitt, "Roadtracer: Automatic extraction of road networks from aerial images," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4720–4728.

[84] Z. Xu, Y. Liu, L. Gan, Y. Sun, X. Wu, M. Liu, and L. Wang, "Rngdet: Road network graph detection by transformer in aerial images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, p. 1–12, 2022.

[85] Z. Xu, Y. Liu, Y. Sun, M. Liu, and L. Wang, "Rngdet++: Road network graph detection by transformer with instance segmentation and multi-scale features enhancement," *arXiv preprint arXiv:2209.10150*, 2023.

[86] P. Yin, K. Li, X. Cao, J. Yao, L. Liu, X. Bai, F. Zhou, and D. Meng, "Towards satellite image road graph extraction: A global-scale dataset and a novel method," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025, pp. 1527–1537.

[87] V. Vineet, P. Harish, S. Patidar, and P. Narayanan, "Fast minimum spanning tree for large graphs on the GPU," in *Proceedings of Conference on High Performance Graphics*, 2009, pp. 167–171.

[88] B. Neubert, T. Franken, and O. Deussen, "Approximate image-based tree-modeling using particle flows," *ACM Transactions on Graphics*, vol. 26, no. 3, p. 88, 2007.

[89] X. Zhang, H. Li, M. Dai, W. Ma, and L. Quan, "Data-driven synthetic modeling of trees," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 9, pp. 1214–1226, 2014.

[90] T. Isokane, F. Okura, A. Ide, Y. Matsushita, and Y. Yagi, "Probabilistic plant modeling via multi-view image-to-image translation," in *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2906–2915.

[91] B. Gorte and N. Pfeifer, "Structuring laser-scanned trees using 3d mathematical morphology," *International Archives of Photogrammetry and Remote Sensing*, vol. 35, pp. 929–933, 2004.

[92] A. Bucksch and R. C. Lindenbergh, "CAMPINO: A skeletonization method for point cloud processing," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 63, pp. 115–127, 2008.

[93] Z. Su, Y. Zhao, C. Zhao, X. Guo, and Z. Li, "Skeleton extraction for tree models," *Mathematical and Computer Modelling*, vol. 54, no. 3, pp. 1115–1120, 2011.

[94] H. Xu, N. Gossett, and B. Chen, "Knowledge and heuristic-based modeling of laser-scanned trees," *ACM Transactions on Graphics*, vol. 26, no. 4, p. 19, 2007.

[95] D.-M. Yan, J. Wintz, B. Mourrain, W. Wang, F. Boudon, and C. Godin, "Efficient and robust reconstruction of botanical branching structure from laser scanned points," in *Proceedings on IEEE International Conference on Computer-Aided Design and Computer Graphics*, 2009, pp. 572–575.

[96] Y. Livny, F. Yan, M. Olson, B. Chen, H. Zhang, and J. El-Sana, "Automatic reconstruction of tree skeletal structures from point clouds," *ACM Transactions on Graphics*, vol. 29, no. 6, p. 151, 2010.

[97] W. Zhen, L. Zhang, T. Fang, P. T. Mathiopoulos, H. Qu, C. Dong, and W. Yuebin, "A structure-aware global optimization method for reconstructing 3-d tree models from terrestrial laser scanning data," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 52, pp. 5653–5669, 2014.

[98] P. Zheng, D. Gao, D.-P. Fan, L. Liu, J. Laaksonen, W. Ouyang, and N. Sebe, "Bilateral reference for high-resolution dichotomous image segmentation," *CAAI Artificial Intelligence Research*, vol. 3, p. 9150038, 2024.

[99] A. Chaudhury and C. Godin, "Skeletonization of plant point cloud data using stochastic optimization framework," *Frontiers in Plant Science*, vol. 11, 2020.

[100] A. Verroust and F. Lazarus, "Extracting skeletal curves from 3D scattered data," in *Proceedings International Conference on Shape Modeling and Applications*, 1999, pp. 194–201.

[101] J. Bartolozzi and M. Kuruc, "A hybrid approach to procedural tree skeletonization," in *ACM SIGGRAPH 2017 Talks*, 2017.

[102] Y. Zhang, L. Wang, C. Zou, T. Wu, and R. Ma, "Diff3DS: Generating view-consistent 3d sketch via differentiable curve rendering," in *Proceedings of the International Conference on Learning Representations*, 2025.

[103] C. Choi, J. Lee, J. Park, and Y. M. Kim, "3doodle: Compact abstraction of objects with 3d strokes," *ACM Transactions on Graphics*, 07 2024.

[104] X. Xing, C. Wang, H. Zhou, J. Zhang, Q. Yu, and D. Xu, "Diffsketcher: Text guided vector sketch synthesis through latent diffusion models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 15 869–15 889, 2023.

[105] T.-M. Li, M. Lukáč, G. Michaël, and J. Ragan-Kelley, "Differentiable vector graphics rasterization for editing and learning," *ACM Transactions on Graphics*, vol. 39, no. 6, pp. 193:1–193:15, 2020.

[106] V. Ye, R. Li, J. Kerr, M. Turkulainen, B. Yi, Z. Pan, O. Seiskari, J. Ye, J. Hu, M. Tancik, and A. Kanazawa, "gsplat: An open-source library for Gaussian splatting," *arXiv preprint arXiv:2409.06765*, 2024.