in silico Plants, 2026, pp. 1–42
DOI added during production
Submitted for review
Paper

Vision-Driven Neural Approximation of Procedural L-Systems for Structural Plant Phenotyping

William Bushell,[1,*] Nazifa Khan[1] and Ian McQuillan[1]

[1]Department of Computer Science, University of Saskatchewan, Canada

[*]Corresponding author. william.bushell@usask.ca

Abstract

Procedural plant models, such as parametric L-systems, can generate biologically realistic structures, but fitting their parameters to match a real target plant in phenotyping settings remains difficult. An existing approach evaluates structural dissimilarity between simulated and real plants using a cost based on matching components geometrically, then searches for better candidate parameters using repeated simulate-and-evaluate methods, but discrete rule selection and discrete matching prevent gradient computation, forcing derivative-free search that requires many simulator evaluations. Here, we instead learn a differentiable neural surrogate to the dissimilarity objective and use it to guide gradient-based parameter search, verifying optimized candidate parameters by rerunning the original simulator-and-matching pipeline. We introduce a phytomorphic surrogate that computes this dissimilarity objective through a process conceptually similar to the original pipeline, using probabilistic landmark generation, differentiable Sinkhorn-based soft assignment, and learned temporal cost aggregation. We compare this surrogate against a baseline multi-layer perceptron that computes cost directly from parameters, and find that despite the baseline achieving higher held-out predictive accuracy, the phytomorphic surrogate achieves lower (better matching) simulator-verified costs on most plants. Supplementary correlation analysis and ablation experiments further suggest that predictive accuracy and optimization utility are only weakly related. These results propose that surrogate design and evaluation for optimizing non-differentiable procedural simulators should prioritize structural fidelity to the true objective, and not predictive accuracy alone. All source code is publicly available at `https://github.com/bushwill/Phytomorphic-Neural-Network-2026`.

1. Introduction

Procedural modelling provides a powerful mathematical framework for generating complex topological and geometric structures from a description. Lindenmayer systems (L-systems) are formal parallel string rewriting systems developed specifically to model the growth of structures by applying rewriting rules in parallel to all letters of a string, and then iteratively rewriting strings. Graphically interpreting these strings with a simulator such as vlab (Algorithmic Botany, 2026) can produce biologically faithful architectures that realistically simulate the developmental behaviours of real-world plants (Lindenmayer, 1968; Prusinkiewicz and Lindenmayer, 1990). Parametric L-systems extend this framework by including numeric parameters in the strings and rules themselves, allowing the simulator to consider quantities such as branch length in both growth behaviour and graphical output. This capability makes them highly valuable across computer graphics (Lee et al., 2023), biological simulation (Room et al., 1996), and agricultural research. It has also been found that using synthetic images from an L-system to augment real image datasets for the purpose of training artificial neural networks can improve automated phenotype prediction; see (Ubbens et al., 2018) for *Arabidopsis thaliana* and (Darvishpour et al., 2026) in canola. Furthermore, it has been found that realism within the synthetic images has a substantial effect on prediction (Khan et al., 2025), and therefore model calibration is important.

While the forward process of generating a structure from a rule set is fairly well understood, inverse problems of automatically learning L-systems from data remain difficult. One form of the inverse problem assumes the underlying L-system is known and seeks to recover the numeric parameters that cause the simulator to approximately match a target plant geometry (Stava et al., 2014; Khan et al., 2026); a broader and more difficult problem is to infer the L-system itself from observations, as studied in inverse procedural

Received: 31 August 2026
© The Author(s) 2026. Copyright and licence statements to be updated by the publisher during production.

modelling and neural L-system inference work (McQuillan et al., 2018; Guo et al., 2020; Lee et al., 2023). In this paper, we focus on the former: given plant images and a fixed L-system, we seek continuous parameter values whose generated plant structure best matches the target.

A common approach to this known-model inverse problem is to treat the simulator as a black box (Farooq et al., 2009; Khan et al., 2026). Given a candidate parameter vector, the L-system is forward executed to generate a synthetic plant, and the resulting sequence of structures is compared against a target plant using a geometric similarity measure. In plant phenotyping settings, both real and simulated plants can be represented by sets of morphological landmarks, such as branch points and end points, extracted from images or topology annotations (Ubbens et al., 2018; Khan et al., 2025, 2026). Because landmarks are unordered, comparing two observations requires establishing correspondences. One approach is to form a pairwise cost matrix and solve the minimum cost one-to-one assignment problem; the Hungarian algorithm provides an exact solution to this problem (Kuhn, 1955). A genetic algorithm is then used with this cost function to explore the parameter space in search of the lowest cost simulation to the real plant. Stava et al. (2014) addressed inverse procedural modelling using stochastic sampling for procedural tree models. These methods treat the simulator as a non-differentiable black box, but require many repeated simulator evaluations during search, since each candidate parameter vector requires a full simulation and matching computation. This is primarily because the objective is not differentiable with respect to the L-system parameters: discrete string-rewriting events and conditional rule applications can cause sudden changes in the generated structure (Stava et al., 2014), and the Hungarian assignment itself is discrete. Gradient-based optimization therefore cannot be applied directly, and existing methods such as genetic algorithms rely on derivative-free search (Farooq et al., 2009; Khan et al., 2026).

To bypass these computational limitations, recent work has increasingly integrated deep learning with procedural generation. As highlighted in comprehensive surveys (Liu et al., 2021), ANNs have been used to approximate structure-generation and grammar-based models: convolutional and transformer architectures have been used to infer L-system branching rules from images (Guo et al., 2020), or generate L-system strings directly (Lee et al., 2023). These approaches provide fast feed-forward predictions, but solve an inference task directly rather than providing a smooth scalar loss minimizable with respect to a fixed simulator's continuous parameters. A different line of research has instead constructed differentiable surrogates for procedural models (Gaillard et al., 2022; Hu et al., 2022; Hossain et al., 2024): smooth approximations of part of the original pipeline, such as simulator output or mismatch cost, through which gradients can propagate with respect to input parameters unavailable from the original objective. Replacing these non-differentiable components with a differentiable surrogate therefore makes continuous gradient-based parameter optimization possible.

However, reliably using a neural surrogate for direct parameter optimization is not the same as predicting simulator cost accurately. Standard feedforward networks are powerful function approximators, but they do not explicitly encode the topological, geometric, or relational structure of the generated plant (Bronstein et al., 2021; Battaglia et al., 2018), so a purely data-driven model may achieve strong predictive accuracy while still producing a misleading optimization landscape. This can lead to surrogate exploitation, where the optimizer finds parameter regions with low predicted cost but high true simulator cost (Jin, 2011; Janner et al., 2019), echoing concerns elsewhere in machine learning where unconstrained data-driven models can violate physical or structural constraints unless appropriate priors are included (Raissi et al., 2019; Willard et al., 2022). Because plant morphology is naturally represented as an unordered and dynamically changing set of landmarks, relevant architectures must also account for point-set structure. Prior work on point-set generation, permutation-invariant set processing, and differentiable assignment, provides tools for this setting, including fixed-size point prediction with existence probabilities (Fan et al., 2017), PointNet and Set Transformer architectures for unordered sets (Qi et al., 2017; Lee et al., 2019), and Sinkhorn-based relaxations of discrete assignment problems (Cuturi, 2013; Mena et al., 2018). These ideas motivate a surrogate architecture that represents plant structure explicitly and compares predicted and target landmarks through differentiable matching.

To address this misalignment between predictive accuracy and optimization behaviour, we introduce a fully differentiable phytomorphic surrogate framework, using phytomorphic to mean that the surrogate is shaped by plant-specific structure. Rather than directly regressing from L-system parameters to the single mismatch cost, it computes its prediction through intermediate representations of plant landmarks and their geometric correspondence to the target. Our architecture decomposes procedural approximation into three specialized modules. First, a Structure Generation Network maps continuous L-system parameters to 2D spatial coordinates and existence probabilities for distinct plant components. Second, a continuous Sinkhorn Assignment Network replaces exact landmark assignment with entropy-regularized optimal transport (Cuturi, 2013; Sarlin et al., 2020), computing a soft assignment matrix that weights a Euclidean physical cost matrix to provide a differentiable relaxation of discrete matching (Mena et al., 2018). Finally, a Cost Aggregation Network synthesizes the daily optimal transport distances across the temporal growth sequence into a unified scalar penalty. By constraining the surrogate to compute cost through explicit geometric structure and differentiable matching, the resulting loss landscape is intended to be more faithful to the true objective in regions relevant for optimization.

We then apply this comparative framework to images of maize (Tardieu et al., 2017; Pound et al., 2017; Ubbens et al., 2018; Khan et al., 2025, 2026; Cieslak et al., 2022), examining whether held-out predictive accuracy is sufficient to characterize optimization utility. To isolate the effect of

structural constraints, we evaluate a multilayer perceptron (MLP) cost regressor and the phytomorphic surrogate under the same protocol, reporting test-set R^2 and simulator-verified Hungarian cost after surrogate-guided optimization. This design allows us to evaluate whether architectural priors improve the downstream inverse-optimization landscape, and to identify failure modes where low predicted cost or low verified cost may not correspond to a meaningful learned objective.

2. Preliminaries

An alphabet U is a finite set of symbols, U^* is the set of all words including the empty word, and U^+ is the set of all non-empty words. A context-free L-system (0L-system) is defined as a tuple $G = (U, \omega, P)$, where U is a finite alphabet, $\omega \in U^+$ is the axiom (the initial state), and $P \subseteq U \times U^*$ is a finite set of productions with each typically written as $a \to \chi$ which maps a predecessor symbol $a \in U$ to a successor word $\chi \in U^*$. If every symbol in U is the predecessor for exactly one production, the system is deterministic and is termed a D0L-system.

Let $\mu \in U^*$ denote a word over the alphabet U. The L-system derivation relation $\Rightarrow$ is defined by $a_1 \cdots a_n \Rightarrow x_1 \cdots x_n$, where $a_i \in U$ and $a_i \to x_i \in P$ for $1 \leq i \leq n$. If

$$\omega = \mu_1 \Rightarrow \mu_2 \Rightarrow \cdots \Rightarrow \mu_n,$$

then the sequence $(\mu_1, \mu_2, \ldots, \mu_n)$ is called a developmental sequence of length n.

To model continuous plant traits (such as age, length, or delay), L-systems are commonly extended to parametric L-systems (Prusinkiewicz and Lindenmayer, 1990). In this setting, symbols become parameterized modules, written for example as $A(p_1, \ldots, p_k)$ with $p_i \in \mathbb{R}$ for all i. Productions may include conditional logic typically written in the form:

$$\mathit{predecessor} : \mathit{cond}(\mathbf{p}) \to \mathit{successor}(\mathit{expr}(\mathbf{p})).$$

This ensures a rule is applied only when its predicate evaluates to true. For example, the production

$$A(x) : x < 5 \to A(x+1)\, B(x)$$

increments a state variable and writes a new module B only while the threshold condition ($x < 5$) holds.

The discrete words produced by an L-system are converted into explicit continuous geometry using a turtle graphics interpretation (Prusinkiewicz, 1986; Prusinkiewicz and Lindenmayer, 1990). In many L-system models, symbols used only for graphical interpretation, such as rotations or bracket commands, are treated as constants and are assumed to rewrite by identity productions unless explicit productions are provided. In the turtle paradigm, a virtual "turtle" traverses 3D space while maintaining a current position and orientation vector. As the turtle reads the L-system word from left to right, each module is interpreted as an imperative drawing or rotation command. The most common commands include **Forward draw:** $F(d)$ moves the turtle forward by a distance d in its current direction while drawing a segment; **Rotations:** Symbols such as $+$ and $-$ apply a rotation by a fixed angle (with additional symbols, e.g., $\wedge$, &, /, and \, to apply pitch and roll rotations in 3D); **Branching:** The bracket symbols [and] implement branching via a Last-In-First-Out stack. When [is encountered, the turtle pushes its current state (position and orientation) to the stack and begins drawing a new branch. When] is encountered, the turtle pops a state from the stack and returns to it. These strings can therefore be interpreted by a simulator such as vlab (Algorithmic Botany, 2026), which generates a developmental simulation.

3. Data and Methods

Figure 1 summarizes the complete surrogate-guided parameter-search workflow.

Figure 2 expands the surrogate-guided parameter search denoted in 1 as a yellow hexagon.

We use the 13-parameter maize L-system described in (Cieslak et al., 2022). The parameter definitions are given in Table 1, while the axiom and production rules used in the present experiments are provided in Supplementary Section B. Hence, the simulator is controlled by an input vector $\mathbf{x} \in \mathbb{R}^d$ (with $d = 13$). Table 1 summarizes the 13 parameters, their types, and intended effects.

3.1. Problem Definition: Inverse Procedural Phenotyping

We start with a specific parametric L-system. For a given parameter vector $\mathbf{x}$, vlab produces a synthetic maize plant using the parametric L-system. Given $\mathbf{x}$, and a sequence of real plant topology images of a target real plant R, we measure the dissimilarity between the simulated plant from $\mathbf{x}$ and R using a landmark-matching cost described below in Section 3.4. We will first describe it at a high-level. Both real and synthetic plants are represented using two types of landmarks: branch points (BPs) and end points (EPs). For matching, we group a BP and an EP together as one matching pair. Real plant landmarks are extracted from topology-image skeletons, and synthetic plant landmarks are extracted from recorded internode and leaf positions produced by the vlab simulator (Section 3.3). For each day, we construct one instance of the bipartite matching problem between real and synthetic BP–EP pairs using a joint pairwise cost, and solve it with the Hungarian algorithm. Then, day costs are summed over the growth sequence to obtain a single scalar objective $C^R(\mathbf{x})$ for the L-system output generated by $\mathbf{x}$ compared to the plant R. Hence, $C^R(\mathbf{x})$ is a cost representing the geometric difference of a time-series of synthetic plant images to a time-series of real plant images.

The inverse problem addressed in this work is defined as finding:

$$\mathbf{x}^* \;=\; \arg\min_{\mathbf{x}} \; C^R(\mathbf{x}).$$

Directly minimizing C^R is seemingly computationally expensive, with essentially an infinite number of possible $\mathbf{x}$ vectors, and it is incompatible with gradient-based optimization due to discontinuities introduced by procedural

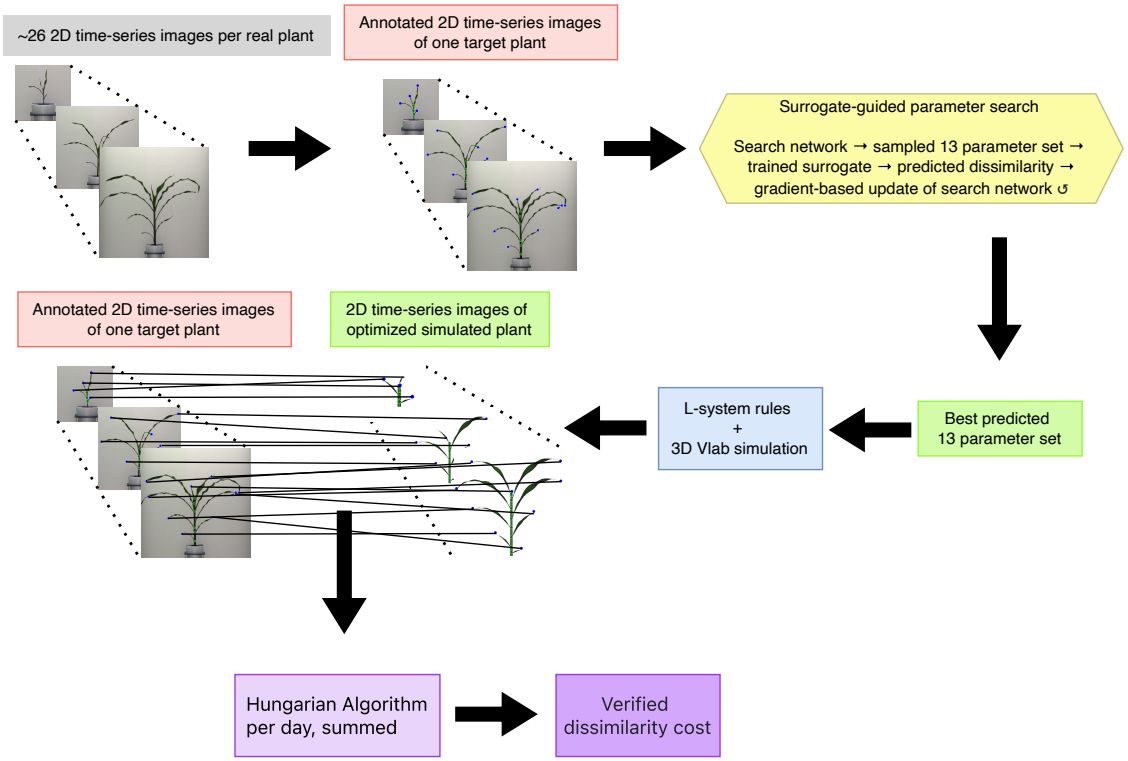


Figure 1 High-level overview of the surrogate-guided parameter-search workflow. The input is a real plant's observations over time, as a time series of two-dimensional corresponding branch point (BP) and end point (EP) coordinate sets. The surrogate-guided parameter search step requires a fully trained surrogate model, that can be leveraged for rapid parameter search to ultimately output a best candidate parameter set. The vlab simulation generates a time-series of three-dimensional plant geometry of this candidate parameter set; this geometry is then projected into a time-series of two-dimensional landmark sequences, identical to the real plant's time-series. The synthetic plant and real plant time-series of 2D coordinates are compared daily using the Hungarian algorithm, and summed together to give a final dissimilarity cost, encompassing the entire growth of the real and synthetic plants, beginning at sprout and ending approximately 26 days after.

rule selection. Here, we instead learn differentiable surrogate models that approximate $C^R(\mathbf{x})$ while providing stable gradients, and use them to search for an $\mathbf{x}^*$ such that $C^R(\mathbf{x}^*)$ is effectively small, producing a synthetic plant that closely matches the real target.

We know the overall mapping from $\mathbf{x}$ to the generated topology and geometry is non-smooth, as discussed in Section 1. Rather than differentiating through the full procedural rewriting and turtle interpretation, we optimize a differentiable proxy objective $\hat{C}^R(\mathbf{x})$ that approximates $C^R(\mathbf{x})$ while providing stable gradients for optimization. For learning and evaluation, simulated plants are converted into a common 2D topology representation from which we extract sets of morphological landmarks (BPs and EPs) for each day in the growth sequence. Hence, these are known automatically from the L-system.

3.2. UNL-Component Plant Phenotyping Dataset (UNL-CPPD)

All real reference targets in this work come from the UNL-Component Plant Phenotyping Dataset (UNL-CPPD) (Das Choudhury et al., 2018), a publicly available time-series maize phenotyping dataset. Figure 3 shows an example from Day 27 of `Plant_063-32`, with pipeline-extracted landmarks overlaid. Each target plant is provided as a sequence of daily 2D topology images. We treat these daily topology images as the observational input that defines the inverse phenotyping target for each plant. This dataset contains 334 images, with 23-26 observations per plant. The active pipeline did not use the dataset's GroundTruth JSON fields, instead extracting landmarks from the provided topology images.

3.3. 2D Topology Representation and Landmark Sets

We represent each real or simulated plant on day $t \in \{1, \ldots, T\}$ by BP and EP coordinate sets, B_t and E_t:

$$B_t = \{\, b_{t,i} \in \mathbb{R}^2 \,\}_{i=1}^{|B_t|}, \qquad E_t = \{\, e_{t,j} \in \mathbb{R}^2 \,\}_{j=1}^{|E_t|}.$$

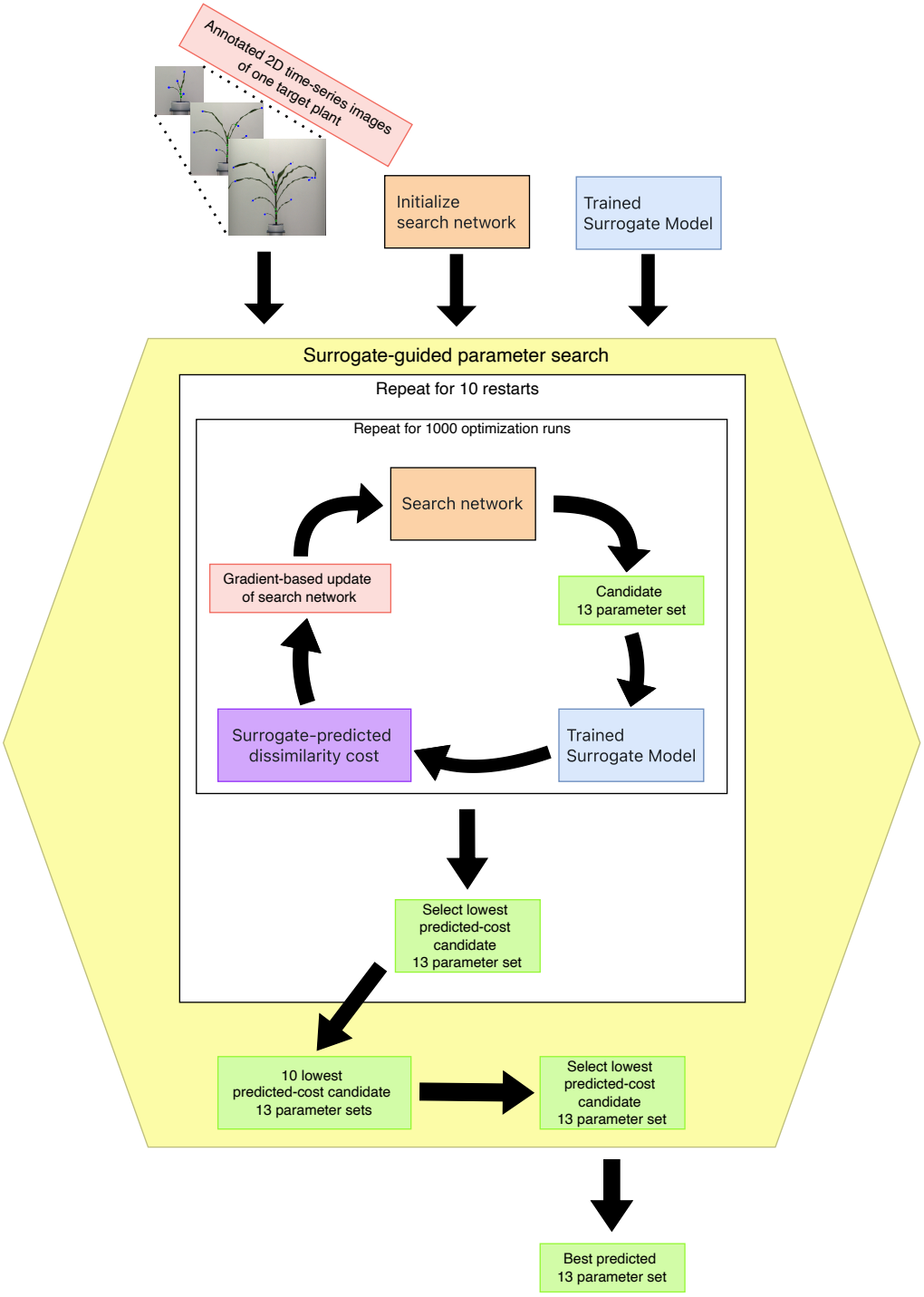


Figure 2 Optimization overview for the surrogate-guided parameter search. A frozen trained surrogate evaluates candidate 13-dimensional parameter sets generated by a randomly initialized search network relative to the real plant target. Within each restart, the predicted dissimilarity is used to compute gradients and update the search network over 1,000 optimization steps per restart. The candidate with the lowest predicted cost is retained for that restart. Multiple random restarts are then performed, and the overall candidate with the lowest verified dissimilarity is selected.

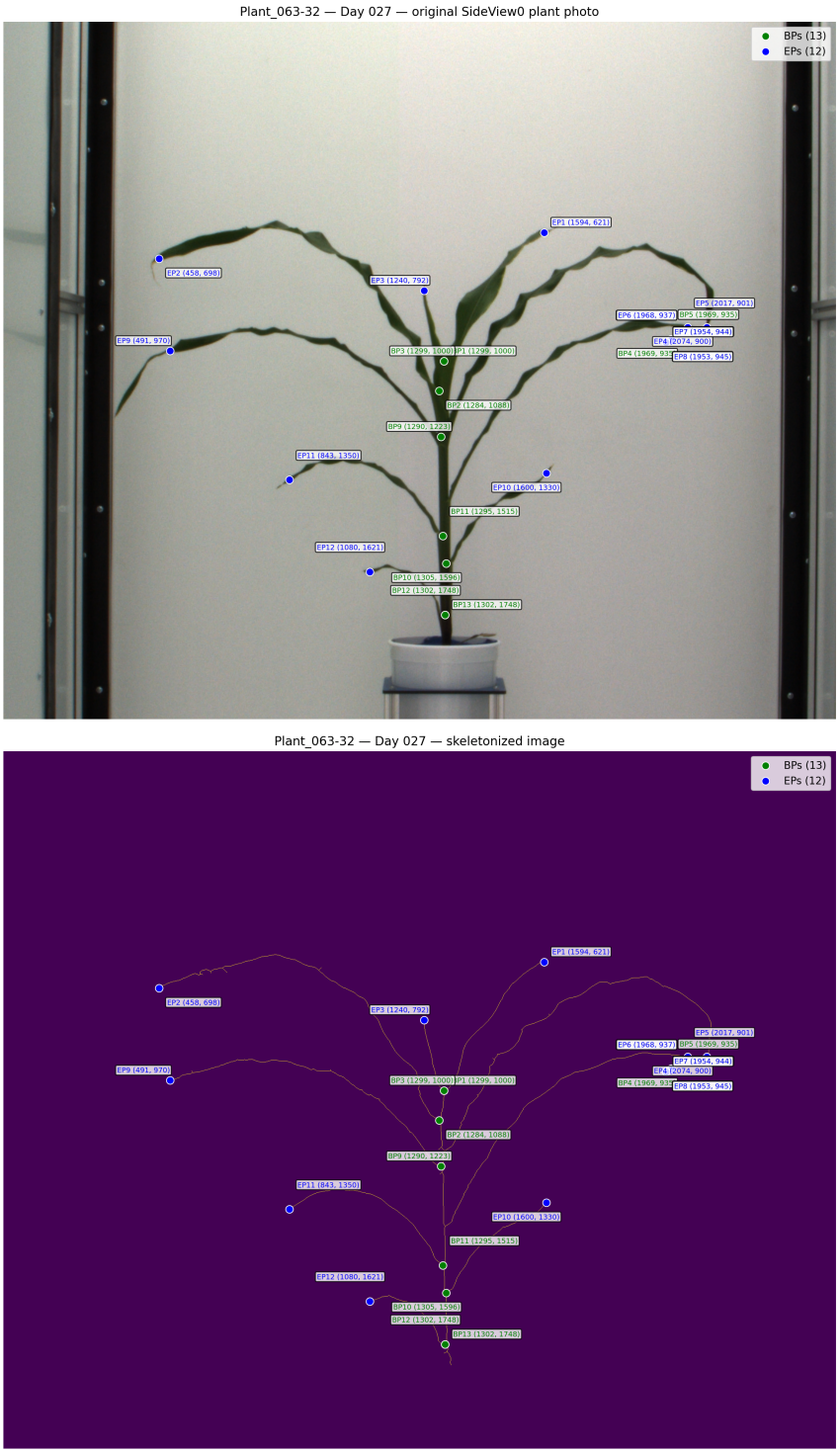


Figure 3 Two-dimensional topology representation and landmark sets for the real target plant `Plant_063-32` on day 27 (26th observation). The upper image shows the original plant image with the pipeline-extracted BP and EP coordinates overlaid. The lower image shows the corresponding binary skeleton representation with the same pipeline-extracted BP and EP landmarks overlaid. The skeleton representation emphasizes the plant's branching structure while abstracting away colour and some appearance-related detail, but not all. Coordinate labels indicate the two-dimensional landmark locations used by the landmark-matching objective.

Table 1 Maize L-system parameter space used in this work.

Index	Parameter name	Type	Description
0	MAX_PHYTOMERS	float threshold	Numeric threshold for phytomer production; because the L-system tests $n <$ val(`MAX_PHYTOMERS`) with integer n, a positive non-integer value effectively permits its ceiling number of phytomers.
1	PLASTOCHRON	float	Interval between successive phytomer initiations; controls growth rate over time.
2	PLANT_ROLL_ANGLE	float	Global roll rotation of the plant around the main axis; affects overall orientation in projection.
3	PLANT_DOWN_ANGLE	float	Global downward tilt (droop) of the plant; affects posture.
4	BRANCH_ANGLE	float	Emergence/branching angle of organs from the main stem; influences leaf/stem divergence angles.
5	LEAF_LEN	float	Leaf length scale; controls blade extent.
6	EXP_LEAF_WID	float	Exponent controlling width variation along the leaf blade; shapes tapering.
7	LEAF_WID	float	Leaf width scale; controls blade thickness/area.
8	LEAF_BEND_SCALE	float	Leaf bending magnitude; affects curvature / droop of leaves.
9	LEAF_TWIST_SCALE	float	Leaf twisting magnitude; affects torsion along the blade.
10	NODE_LEN	float	Internode (node-to-node) length scale; controls vertical spacing of phytomers.
11	INT_WID	float	Internode width (radius/diameter) scale; controls stem thickness.
12	EXP_INT_RAD	float	Exponent controlling how internode radius changes along the stem; shapes tapering of the stalk.

For real plants, topology-image skeletons are processed with Skan to identify paths and their endpoints. For each type-1 path, the parser considers its two endpoints together and uses their occurrence among type-2 path endpoints to assign BP and EP labels. This creates a heuristic, path-based BP–EP pair, but does not guarantee that the landmarks are biologically correct. For simulated plants, BP coordinates are taken from recorded internode (I) module positions and EP coordinates from recorded leaf (L) module positions.

For ground-truth matching, the 3D positions recorded by vlab are orthographically projected onto a fixed 2454×2056 pixel canvas and expressed in the same [row, column] order as the real landmarks, without target-specific registration.

A plant (real or simulated) is therefore described by a per-day sequence of these landmark sets, $\{(B_t, E_t)\}_{t=1}^{T}$. Our real image dataset has at most 26 observations per plant, and so we use $T \leq 26$. Some real plants have fewer available days due to missing early or late observations, and all temporal costs are computed over the available overlap between the real and synthetic sequences.

3.4. Ground-truth Landmark Matching Cost (Hungarian Assignment)

This subsection describes the C^R cost function. For two coordinates $p = (p_x, p_y)$ and $q = (q_x, q_y)$ in $\mathbb{R}^2$,

$$d(p,q) \;=\; \|p-q\|_2 \;=\; \sqrt{(p_x-q_x)^2+(p_y-q_y)^2},$$

which is the Euclidean distance between the two points.

The Hungarian algorithm takes a weighted, undirected bipartite graph as input, represented by a matrix of pairwise weights between the two vertex sets of the graph, and computes an optimal minimum-cost one-to-one assignment between the two vertex sets of the bipartite graph (in polynomial time). This produces a permutation in which each vertex is matched to exactly one vertex. The resulting assignment cost measures the dissimilarity between the simulated plant and the target real plant.

To compare a real plant R with a synthetic plant $S(\mathbf{x})$ with parameter set $\mathbf{x}$, a different graph is created for each day t.

For each day t, the implementation constructs the matching problem over BP–EP pairs, rather than matching the two landmark types independently. For real plants, each pair is formed from the two endpoints of one Skan type-1 path. Endpoint roles are assigned heuristically using whether each endpoint also appears among type-2 path endpoints; these pairs have no phytomer identity. For synthetic plants, the BP and EP lists are paired by index, with each index corresponding to a phytomer. Concretely, let

$$R_t = \{(r^B_{t,i}, r^E_{t,i})\}_{i=1}^{|R_t|}, \qquad S_t(\mathbf{x}) = \{(s^B_{t,j}(\mathbf{x}), s^E_{t,j}(\mathbf{x}))\}_{j=1}^{|S_t(\mathbf{x})|},$$

Here, i and j index real and synthetic BP–EP pairs, respectively. The parser omits the final extracted real EP from the matching list but retains its BP. The real-side matching count is determined by the retained EP count, and costs are scored only for assignments with valid EP indices. This trailing BP is not matched or penalized separately. The day-t pairwise cost matrix is

$$(D_t(\mathbf{x}))_{ij} = \left\| r^B_{t,i} - s^B_{t,j}(\mathbf{x}) \right\|_2 + \left\| r^E_{t,i} - s^E_{t,j}(\mathbf{x}) \right\|_2 .$$

Each entry compares real BP–EP pair i with synthetic BP–EP pair j using the sum of their BP and EP distances. Hungarian matching is run once per day on this single joint matrix to obtain day cost c_t.

When tuple-set sizes differ, the matrix is expanded to a square size

$$n_t = \max\big(|R_t|,\ |S_t(\mathbf{x})|\big).$$

Unmatched assignments are penalized after matching: each unmatched tuple contributes the maximum valid matched tuple cost from that day (computed over matched (i, j) pairs using the same joint BP+EP distance above); if no valid pairs exist, a fixed penalty of 1000.0 is used.

The total ground-truth phenotyping objective for a real plant R is

$$C^R(\mathbf{x}) \ = \ \sum_{t=1}^{T} c_t.$$

Thus, $C^R(\mathbf{x})$ is the dissimilarity between the synthetic plant generated from $\mathbf{x}$ and the target real plant R. This objective is non-differentiable with respect to $\mathbf{x}$ because of both the discrete rule-selection events in the L-system simulator and discontinuous changes in the optimal assignment.

We refer to $C^R(\mathbf{x})$ as the vlab cost of $S(\mathbf{x})$ to R throughout the remainder of this paper, reflecting its computation from simulated plant geometry.

3.5. Two-Stage Framework Overview

Our approach separates learning the objective from optimizing it. We do the following for each real plant R. In the offline dataset generation step, we randomly sample L-system parameter vectors $\mathbf{x}$ from a fixed distribution. For each vector, we run the vlab simulator and compute its ground-truth dissimilarity $C^R(\mathbf{x})$. This produces a supervised dataset of input-output pairs $(\mathbf{x}, C^R(\mathbf{x}))$ (Section 3.7).

Stage 1: Surrogate training

For each target real plant R, we train a single surrogate model $\hat{C}^R(\mathbf{x})$ to predict the ground-truth cost $C^R(\mathbf{x})$. In the dataset generation procedure, vlab simulator evaluations produce parameter-cost pairs $(\mathbf{x}, C^R(\mathbf{x}))$ as training objectives. The dataset generation procedure is illustrated in Figure 4. The datasets are then used for supervised regression with the MLP and phytomorphic surrogate. Both model architectures and their respective training objectives are described in Sections 3.9.1 and 3.9.2.

Stage 2: Surrogate-guided optimization

After training, we treat $\hat{C}^R(\mathbf{x})$ as a differentiable proxy objective and use a randomly initialized search network to generate candidate parameter vectors $\mathbf{x}$. Within each restart, gradients of $\hat{C}^R(\mathbf{x})$ update the search network for a number of steps, after which the candidate with the lowest predicted cost is chosen for verification. Multiple random restarts are performed, and the final candidate is selected by verified vlab cost. The optimization procedure is illustrated in Figure 2.

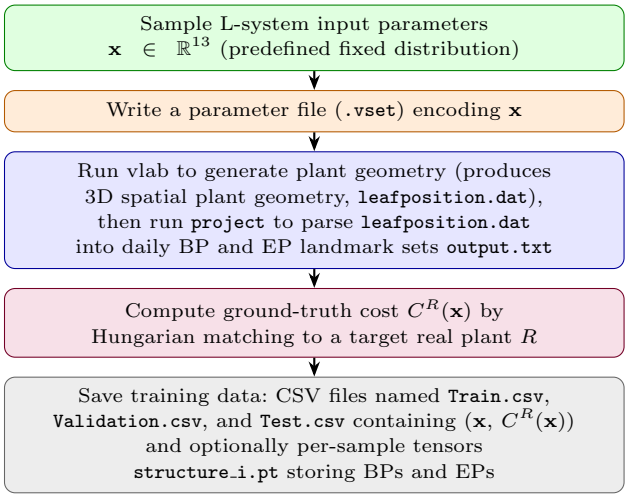


Figure 4 Offline synthetic-data generation and ground-truth labeling. For each target plant R, L-system parameter vectors are sampled from a fixed distribution, written to simulator input files, evaluated with vlab, and converted into daily BP and EP landmark sets. The resulting landmark sequences are compared with the target using the ground-truth Hungarian matching procedure to obtain $C^R(\mathbf{x})$. The resulting parameter-cost pairs are stored for subsequent surrogate training; model training and optimization are not shown in this figure.

3.6. Containerized Experimental Environment

All experiments ran in a Dockerized environment via Docker Compose, covering dataset generation, vlab simulation, surrogate training, optimization, and verification in a single container. Experiments ran on CPU in a virtual machine, because Docker required administrator privileges and GPU resources could not be partitioned for individual use. The Dockerfile and experiment scripts are included in the public project repository.

The repository setup commands, experiment-stage workflow, and source-file map are provided in Appendix J.

3.7. Synthetic Dataset Generation

Synthetic training samples are generated by sampling raw L-system parameter vectors $\mathbf{x} \in \mathbb{R}^{13}$ from a fixed distribution and executing vlab for each sample. The dataset stores parameters in raw units; input standardization is applied later by the surrogate models using training-split statistics (Section 3.8). Datasets are generated per target plant; for each plant we create separate `Train.csv`, `Validation.csv`, and `Test.csv` splits using the same sampling distribution but a plant-specific ground-truth cost label $C^R(\mathbf{x})$. Figures 5 and 6 show samples from one synthetic dataset used for training.

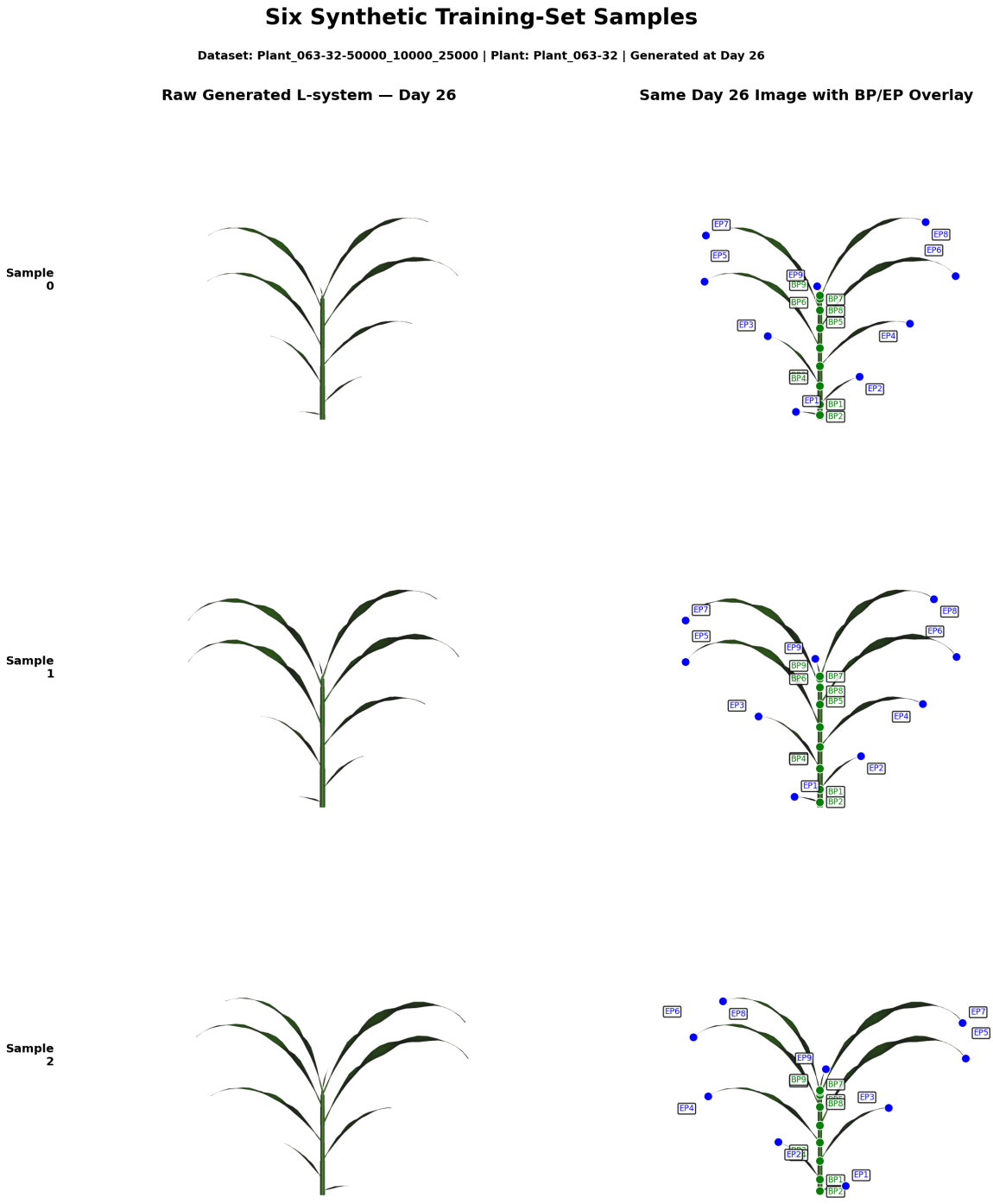


Figure 5 Synthetic dataset examples used for surrogate training.

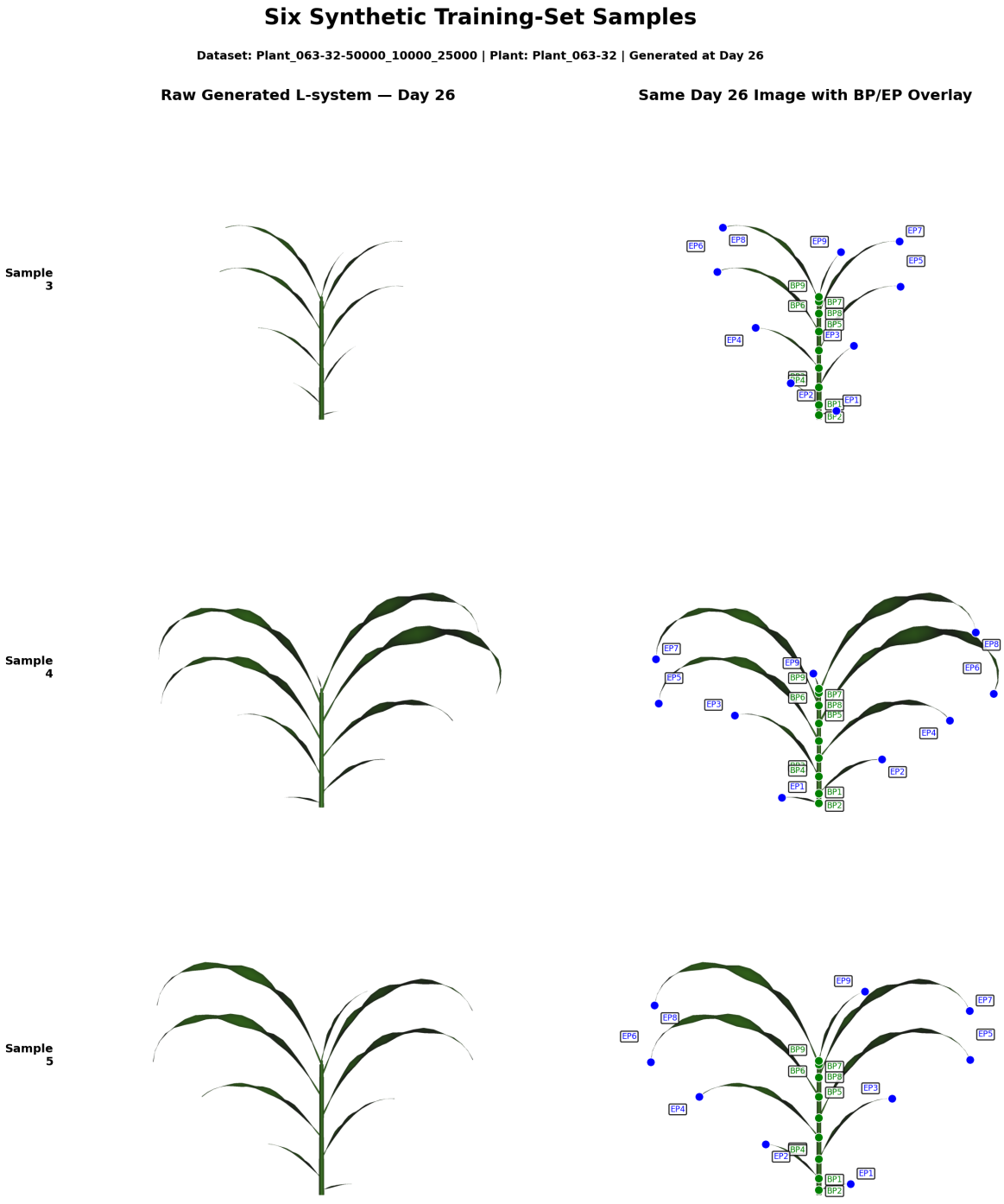


Figure 6 Additional synthetic maize examples used for surrogate training.

Gaussian Parameter Sampling

Parameters are sampled independently; except for the global roll angle, each parameter follows a univariate Gaussian distribution. For parameters x_k with generator mean μ_k^{gen} and standard deviation σ_k^{gen},

$$x_k \sim \mathcal{N}\Big(\mu_k^{\text{gen}},\, (\sigma_k^{\text{gen}})^2\Big)\,.$$

All parameters follow this rule except the global roll angle, which is sampled from a symmetric two-mode "chirality" mixture: unlike the other parameters, roll is a viewing-orientation variable with two plausible mirror-image modes in 2D projection, so with probability 0.5 the roll mean is $+90^\circ$ and with probability 0.5 it is -90°, each with standard deviation 10°:

$$x_{\text{roll}} \sim \tfrac{1}{2}\mathcal{N}(+90,\, 10^2) \;+\; \tfrac{1}{2}\mathcal{N}(-90,\, 10^2).$$

The generator distribution parameters $(\mu_k^{\text{gen}}, \sigma_k^{\text{gen}})$ used in our experiments are reported in Table 2.

Table 2 Gaussian sampling distributions used to generate synthetic L-system parameters in raw units.

Index	Parameter	μ^{gen}	σ^{gen}
0	MAX_PHYTOMERS	10.0	1.0
1	PLASTOCHRON	3.0	0.1
2	PLANT_ROLL_ANGLE	$\pm$90.0	10.0
3	PLANT_DOWN_ANGLE	0.0	4.0
4	BRANCH_ANGLE	135.0	5.0
5	LEAF_LEN	5.0	1.0
6	EXP_LEAF_WID	0.5	0.01
7	LEAF_WID	1.0	0.1
8	LEAF_BEND_SCALE	90.0	3.0
9	LEAF_TWIST_SCALE	180.0	3.0
10	NODE_LEN	0.7	0.05
11	INT_WID	0.9	0.01
12	EXP_INT_RAD	0.5	0.01

Dataset Size and Splits

For each target plant, we generated $N = 85,000$ synthetic samples: $N_{\text{train}} = 50,000$ for training, $N_{\text{val}} = 10,000$ for validation, and $N_{\text{test}} = 25,000$ for testing, with normalization statistics computed from the training split only. Smaller regimes (1,000/200/500 and 20,000/4,000/10,000 Train/Validation/Test) were separately generated to conduct preliminary tuning and scaling studies, using `Plant_023-1`.

For each sample we store the raw parameter vector $\mathbf{x}$ and its scalar label $C^R(\mathbf{x})$ in CSV files. For hierarchical surrogate training, we additionally store the simulated landmark tensors (daily BP and EP sets with masks) as per-sample serialized tensors (e.g., `structure_i.pt`).

3.8. Supervision, Normalization, and Verification

Each simulated sample is labelled with the ground-truth Hungarian landmark-matching cost $C^R(\mathbf{x})$ (Section 3.4), which serves as the supervision signal for all surrogate models.

To stabilize training and optimization, all models normalize inputs using statistics computed from the training split only. Let $\mu_{x,k}$ and $\sigma_{x,k}$ denote the mean and population standard deviation of parameter x_k over the training set $(k = 0, \ldots, 12)$. We standardize each dimension as

$$\widetilde{x}_k \;=\; \frac{x_k - \mu_{x,k}}{\sigma_{x,k} + \varepsilon},$$

where ε is a small stability constant; $\widetilde{x}_k$ is unbounded; large $|\widetilde{x}_k|$ indicates parameters are far from the training distribution.

Similarly, let μ_C^R and σ_C^R denote the mean and population standard deviation of $C^R(\mathbf{x})$ over the training set for target plant R. The normalized cost target is

$$\widetilde{C}^{\,R}(\mathbf{x}) \;=\; \frac{C^R(\mathbf{x}) - \mu_C^R}{\sigma_C^R}.$$

$\mathcal{L}_{\text{cost}}$ is computed as MSE between normalized predictions and targets, with predictions converted back to raw units for reporting. All statistics $(\mu_{x,k}, \sigma_{x,k}, \mu_C^R, \sigma_C^R)$ are computed once from the training split and stored with the trained model, then reused unchanged for validation, testing, and surrogate-guided optimization.

3.9. Surrogate Model Families

We evaluate two surrogate families, differing in whether cost is predicted directly or through explicit structural matching. All surrogate models were implemented in PyTorch, standardizing the 13-dimensional input internally using stored training-set statistics. The MLP predicts cost directly, whereas the phytomorphic surrogate denormalizes the CAN output, applies Softplus in raw cost space, and re-normalizes the resulting prediction for the regression loss.

3.9.1. Baseline Surrogate: Multi-Layer Perceptron (MLP)

For each real plant R, the MLP maps $\mathbf{x} \in \mathbb{R}^{13}$ directly to a scalar prediction $\hat{C}^R(\mathbf{x}) \in \mathbb{R}$, with no explicit landmark representation or matching step.

Inputs and costs are normalized using training-split statistics (Section 3.8), with $\widetilde{\cdot}$ denoting z-score normalization. The MLP is trained to minimize

$$\mathcal{L}_{\text{MLP}} = \mathcal{L}_{\text{cost}} = \frac{1}{B}\sum_{b=1}^{B}\left(\widetilde{\hat{C}}^{\,R}(\mathbf{x}_b) - \widetilde{C}^{\,R}(\mathbf{x}_b)\right)^2. \qquad (1)$$

The MLP architecture and output mapping are detailed in Supplementary Section C.1.

3.9.2. Phytomorphic Surrogate: Sinkhorn-Based Structural Proxy

High-level overview

Rather than regressing directly to a scalar, the phytomorphic surrogate (Figure 7) estimates vlab cost through a full pipeline mirroring the L-system simulation and Hungarian

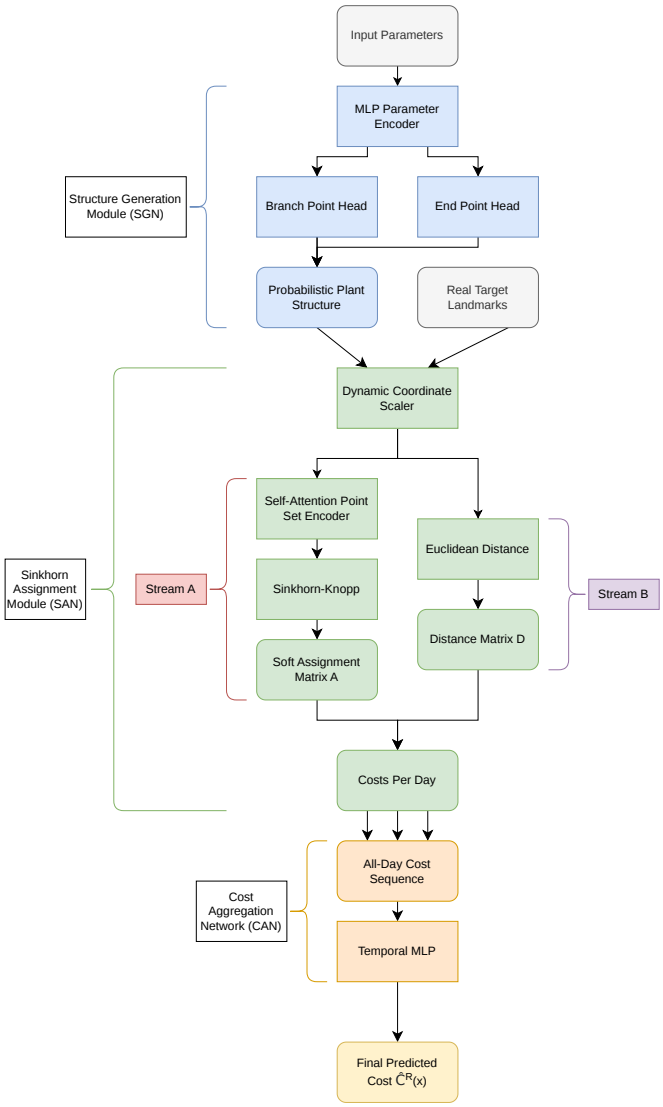


Figure 7 Overview of the phytomorphic surrogate pipeline used in this study.

evaluation, mapping parameters to landmarks, landmarks to soft matches, and soft matches to a final scalar cost:

$$\mathbf{x} \rightarrow \text{predicted landmark sets} \rightarrow \text{differentiable set matching} \rightarrow \text{temporal aggregation} \rightarrow \hat{C}^R(\mathbf{x}).$$

It predicts one BP and one EP set with per-point existence probabilities, broadcasts them across available days, compares them to real landmarks via Sinkhorn-based differentiable assignment rather than Hungarian discrete assignment, and aggregates the resulting day-wise mismatch costs through a learned module. In this setting, Sinkhorn provides a continuous relaxation of bipartite matching by assigning soft correspondence probabilities instead of the Hungarian algorithm's hard one-to-one matches (Cuturi, 2013; Mena et al., 2018). Because matches are learned via differentiable optimal transport, the output is ordering-invariant and grounded in geometric mismatch rather than an arbitrary function of $\mathbf{x}$. The phytomorphic surrogate uses the SGN-SAN-CAN architecture described below.

Full surrogate-training settings are provided in Supplementary Section C.6.

Training Objective

The total loss combines cost regression with an auxiliary structure-guidance term applied to the Structure Generation Network when ground-truth landmarks are available:

$$\mathcal{L}_{\text{phyto}} = \mathcal{L}_{\text{cost}} + 0.01\,\mathcal{L}_{\text{guide}},$$

with $\mathcal{L}_{\text{cost}}$ as in Equation (1). The role of $\mathcal{L}_{\text{guide}}$ is to stabilize SGN training by providing direct geometric supervision before downstream Sinkhorn matching and temporal aggregation. It aligns generated 2D point sets with true simulated landmarks using two complementary terms: a coordinate-alignment term (ℓ_{coord}) and a cardinality term (ℓ_{count}), for both BPs and EPs. Detailed definitions of these terms are provided in Supplementary Section C.2.

Structure Generation Network (SGN)

The Structure Generation Network (SGN) replaces the non-differentiable vlab string-rewriting and turtle-interpretation steps with a continuous neural approximation, taking the normalized parameter vector $\tilde{\mathbf{x}}$ and predicting a probabilistic 2D landmark representation.

An MLP encoder produces a latent representation, which is passed to two separate heads predicting BPs and EPs independently.

Since the number of landmarks varies with $\mathbf{x}$, each head outputs a fixed number of point slots, each with a 2D coordinate and an existence logit converted to a probability via sigmoid.

Slots with high probability act as active landmarks, while near-zero slots are softly suppressed in later matching and loss terms, giving a continuous alternative to explicitly choosing landmark counts.

The SGN architecture and output parameterization are detailed in Supplementary Section C.3.

The resulting BP and EP sets are passed to the Sinkhorn Assignment Network, which computes differentiable geometric matching costs against the observed plant structure.

Sinkhorn Assignment Network (SAN)

For each day t, the SAN evaluates geometric mismatch between the SGN's probabilistic point sets and real plant point sets, broadcasting the probabilistic BP/EP sets across days and applying day-specific coordinate scaling before differentiable Sinkhorn-based matching.

Real point sets are padded to a fixed maximum count with a validity mask $m_j^{\text{real}} \in \{0, 1\}$. The dynamic coordinate-scaling implementation is detailed in Supplementary Section C.4.

Stream A: Learned Soft Assignment The scaled predicted and real point sets are each passed through a shared, permutation-equivariant Point Set Encoder; synthetic features bundle coordinates with their existence

probabilities, $(x_i, y_i, p_i^{\text{syn}})$, and real features concatenate coordinates with the validity mask, $(x_j, y_j, m_j^{\text{real}})$. Self-attention within each set contextualizes each point's embedding relative to the plant's structure (Sarlin et al., 2020). BP and EP streams each use a shared point-set encoder.

The point-set encoder implementation is detailed in Supplementary Section C.4.

The embeddings are compared via a temperature-scaled similarity matrix

$$S_{ij} = \frac{e_i^{\text{syn}} \cdot e_j^{\text{real}}}{T}, \qquad T = \max\left(\exp(\tau), 10^{-4}\right),$$

with learnable log-temperature τ: lower T sharpens toward near-discrete matching, higher T spreads mass more evenly. Invalid real points are masked with a large negative value prior to normalization to prevent matching by embedding similarity.

We project S into an approximately doubly stochastic matrix A via the Sinkhorn-Knopp algorithm (Cuturi, 2013), computed in log-space. The normalization steps and iteration count are detailed in Supplementary Section C.4. After K iterations, $A_{ij} = \exp\left(Z_{ij}^{(K)}\right)$, where the resulting matrix A is the soft assignment matrix, with entry A_{ij} giving the matching probability mass between synthetic point i and real point j.

Stream B: Physical Geometric Error We separately compute the pairwise Euclidean distance between scaled synthetic and real coordinates.

$$D_{ij} = \left\| \mathbf{r}_i^{\text{syn}} - \mathbf{r}_j^{\text{real}} \right\|_2,$$

anchoring the final cost to true physical distance rather than a purely learned similarity space.

Two-Stream Combination and Final Expected Cost Stream A and B combine into a differentiable expected cost:

$$\text{Cost}_{\text{sink}} = \sum_{i=1}^{N} \sum_{j=1}^{N} A_{ij}\, D_{ij} \left(p_i^{\text{syn}} \cdot m_j^{\text{real}}\right).$$

The $\left(p_i^{\text{syn}} \cdot m_j^{\text{real}}\right)$ term restricts physical penalties to comparisons against valid real landmarks, and enables gradients to flow to the SGN through three paths: D_{ij} (moving predicted coordinates toward real ones), A_{ij} (refining learned embeddings), and p_i^{syn} (suppressing existence probability where no valid match exists).

The SAN runs this pipeline independently for BPs and EPs, summing the results into the total assignment cost for day t:

$$c_t = \text{Cost}_{\text{sink}}^{\text{BP}} + \text{Cost}_{\text{sink}}^{\text{EP}},$$

repeated for $t = 1, \ldots, T$ to produce the day-wise cost sequence passed to the Cost Aggregation Network.

For clarity, SAN is a differentiable surrogate approximation: it computes BP and EP soft-assignment costs independently and then sums them. In contrast, the ground-truth vlab objective in Section 3.4 applies Hungarian matching once per day on an joint BP–EP pair cost matrix.

Cost Aggregation Network (CAN)

The CAN combines the per-day assignment costs produced by the SAN into a single scalar objective. Uniformly summing the day-wise costs c_t, as is done for the ground-truth objective $C^R(\mathbf{x})$, causes large geometric errors of late-stage growth to dominate the objective and mask structural errors from early development. To address this, the CAN instead learns an aggregation function h_ϕ that maps the full cost sequence to a scalar,

$$\hat{c} = h_\phi(c_1, \ldots, c_T),$$

allowing it to weight developmental days non-uniformly rather than applying a fixed sum. The CAN architecture and implementation details are provided in Supplementary Section C.5. The CAN output $\hat{c}$ is in normalized cost space. The model converts it to raw cost units and applies Softplus:

$$\hat{C}^R(\mathbf{x}) = \text{softplus}\left(\sigma_C^R \hat{c} + \mu_C^R\right),$$

where μ_C^R and σ_C^R are the training-set cost mean and standard deviation for target plant R. During training, this nonnegative raw-cost prediction is standardized as

$$\widetilde{\hat{C}}^R(\mathbf{x}) = \frac{\hat{C}^R(\mathbf{x}) - \mu_C^R}{\sigma_C^R}$$

before computing the cost-regression loss against the similarly normalized target. The raw prediction $\hat{C}^R(\mathbf{x})$ is used as the surrogate cost during optimization.

3.9.3. Phytomorphic Ablation Architectures

We evaluated four ablated variants of the full SGN-SAN-CAN architecture, each retaining the same training data, optimization procedure, loss definition, and training configuration as the full model unless stated otherwise.

The no-encoder variant replaces the BP/EP point-set encoders with independent linear projections Linear$(3, 64)$, retaining the coordinate scaler, Sinkhorn assignment, and CAN.

The no-scaler variant removes the Dynamic Coordinate Scaler, using raw coordinates directly for feature construction and pairwise distance; the encoders, Sinkhorn assignment, and CAN remain active.

The no-aggregator variant retains the encoders, scaler, and Sinkhorn assignment, but replaces the CAN with the arithmetic mean of the padded/truncated 26-dimensional daily-cost sequence.

The hollow variant combines all three: linear projections instead of encoders, raw coordinates instead of scaling, and mean aggregation instead of the CAN, with the underlying Sinkhorn cost computation retained.

3.10. Surrogate-Guided Optimization Details

Once trained, the surrogate serves as a differentiable proxy objective, and optimization seeks $\mathbf{x}^*$ minimizing $\hat{C}^R(\mathbf{x})$.

Directly optimizing $\mathbf{x}$ via gradient descent is risky: although the simulator accepts arbitrary parameter values, surrogates are trained on a fixed distribution and are most

reliable near the training region. Extrapolation beyond it enters out-of-distribution territory where the surrogate's loss landscape is undefined and exploitable. Gradient clipping limits the magnitude of the gradients, which can help prevent large updates, but it does not keep $\mathbf{x}$ within parameter bounds.

Instead, we frame the search as a generative task using an implicit neural prior (Ulyanov et al., 2018). Rather than optimizing $\mathbf{x}$ directly, a lightweight generator network (`OptimizerNet`) maps a fixed latent scalar $z \in \mathbb{R}^1$ to a 13-dimensional output via a small MLP with a final Sigmoid activation, bounding the raw output to $(0, 1)^{13}$. This is deterministically scaled to a dynamic bounding box derived from the surrogate's training statistics. Letting $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ be the training parameter mean and standard deviation, we define

$$\mathbf{x}_{\min} = \boldsymbol{\mu} - c\boldsymbol{\sigma}, \qquad \mathbf{x}_{\max} = \boldsymbol{\mu} + c\boldsymbol{\sigma},$$

with $c = 2.0$, and for each parameter index k:

$$x_k \;=\; \mathrm{OptimizerNet}(z)_k \,(x_{\max,k} - x_{\min,k}) \;+\; x_{\min,k}.$$

Searching the smooth weight-manifold of `OptimizerNet` rather than raw parameter space naturally regularizes the optimizer and keeps candidates within the 2σ training bounds.

Gradient Flow and Search Strategy

The trained surrogate's weights are frozen during optimization; the objective is simply

$$\mathcal{L}_{\mathrm{opt}} \;=\; \hat{C}^R(\mathbf{x}) \;=\; \hat{C}^R(\mathrm{OptimizerNet}(z)).$$

For the phytomorphic surrogate, gradients flow through the differentiable structural modules into `OptimizerNet`; for the MLP, gradients flow directly through the feedforward regressor. The optimizer and search configuration are provided in Supplementary Section C.7.

Since surrogates of highly non-linear procedural spaces contain flat regions and local minima, we use a multi-restart strategy to reduce the risk of stalling on flat plateaus.

Ground-Truth Verification

After optimization, the candidate $\mathbf{x}^*$ with the lowest surrogate-predicted cost per restart is passed back into the vlab simulator, and its exact Hungarian matching cost $C^R(\mathbf{x}^*)$ is computed. The restart with the lowest verified cost is the final reported solution and serves as the definitive success metric for the inverse phenotyping task.

3.11. Preliminary Model-Selection Protocol

Before the final all-plant experiments, we conducted a four-phase preliminary protocol on the fixed reference target `Plant_023-1` to select dataset scale, training budget, model hyperparameters, and the phytomorphic architecture used in the final experiments. Verified vlab cost after surrogate-guided optimization was the primary selection metric, while held-out test R^2 was retained as a diagnostic measure. Full configurations, hyperparameter grids, and results for this protocol are provided in Appendix D.

3.12. Final All-Plant Experiment Configuration

For the final evaluation, we repeated the inverse-optimization experiment across all 13 UNL-CPPD target plants listed in Appendix A, including `Plant_023-1`, which was also used as the fixed target for preliminary model selection. For each target plant, a synthetic dataset was generated using the parameter-sampling distribution described in Section 3.7, with plant-specific ground-truth cost labels.

The final comparison used both surrogate families with the hyperparameters selected in Phase 3; the exact learning rates, batch sizes, and epoch budget are provided in Supplementary Section C.8. Two independent training replicates were run for each target plant and model family. For the phytomorphic surrogate, stored simulator landmark tensors were used during training so the structural guidance term $\mathcal{L}_{\mathrm{guide}}$ remained active.

After training, each replicate was optimized using the configuration in Section 3.10. Final results are reported as mean ± standard deviation across the two replicates for each target plant and model. Two replicates provide only a coarse estimate of run-to-run variance; variance was non-trivial for some plants, so individual per-plant comparisons carry more uncertainty than the aggregate patterns reported across all 13 plants.

4. Results

We report two complementary measures throughout the results. Predictive accuracy is reported as R^2 between surrogate-predicted vlab costs and ground-truth costs on held-out data. Unless otherwise stated, R^2 is computed on the test split. Optimization utility is measured by verified vlab cost. After surrogate-guided optimization, the selected candidate parameters $\mathbf{x}^*$ are rerun through vlab and evaluated with the true Hungarian landmark-matching objective $C^R(\mathbf{x}^*)$. Lower verified vlab cost indicates better agreement with the real target plant and is the primary metric for model selection and comparison. Unless otherwise stated, aggregate values are reported as mean ± standard deviation across independently trained replicates. We report stage-specific runtime measurements in Appendix H, Table 20; these measurements are not combined into an end-to-end wall-clock estimate.

4.1. Preliminary Model-Selection Summary

The four-phase preliminary protocol selected the 20k-sample dataset configuration for early screening, the full 50k-sample training set with 50 epochs for final tuning, and final hyperparameters of learning rate 10^{-3}, batch size 8 for the MLP and learning rate 5×10^{-4}, batch size 8 for the phytomorphic surrogate. Ablation testing confirmed the full SGN-SAN-CAN phytomorphic architecture over four reduced variants, as the hollow and no-aggregator variants exhibited degenerate flat-objective behaviour despite low verified cost. Complete configurations, hyperparameter grids, and phase-by-phase results are provided in Appendix D.

Table 3 Final all-plant results across the 13 UNL-CPPD target plants. Values report mean ± standard deviation over two independently trained replicates per model and plant. Lower verified vlab cost is better.

Target plant	MLP Test R^2	MLP vlab cost	Phytomorphic Test R^2	Phytomorphic vlab cost
`Plant_001-9`	0.9963 ± 0.0002	$38,752 \pm 13,515$	0.9821 ± 0.0021	$39,770 \pm 14,945$
`Plant_006-25`	0.9951 ± 0.0006	$54,544 \pm 369$	0.9851 ± 0.0110	$39,598 \pm 83$
`Plant_008-19`	0.9959 ± 0.0002	$53,224 \pm 97$	0.9912 ± 0.0012	$47,001 \pm 12,124$
`Plant_016-20`	0.9968 ± 0.0004	$49,165 \pm 548$	0.9933 ± 0.0008	$47,524 \pm 1,742$
`Plant_023-1`	0.9967 ± 0.0003	$42,720 \pm 828$	0.9912 ± 0.0010	$32,850 \pm 720$
`Plant_045-1`	0.9982 ± 0.0002	$43,407 \pm 98$	0.9960 ± 0.0001	$35,128 \pm 4,686$
`Plant_047-25`	0.9961 ± 0.0000	$55,652 \pm 126$	0.9892 ± 0.0066	$54,112 \pm 3,138$
`Plant_063-32`	0.9929 ± 0.0004	$61,968 \pm 12,700$	0.9816 ± 0.0082	$53,070 \pm 3,172$
`Plant_070-11`	0.9967 ± 0.0009	$53,205 \pm 8$	0.9546 ± 0.0444	$50,299 \pm 4,174$
`Plant_071-8`	0.9964 ± 0.0004	$55,786 \pm 17$	0.9949 ± 0.0001	$53,958 \pm 2,593$
`Plant_076-24`	0.9979 ± 0.0001	$39,939 \pm 358$	0.9946 ± 0.0014	$39,820 \pm 80$
`Plant_104-24`	0.9976 ± 0.0000	$35,554 \pm 315$	-1.9790 ± 4.2083	$31,389 \pm 5,063$
`Plant_191-28`	0.9919 ± 0.0006	$51,325 \pm 3,941$	0.9815 ± 0.0120	$35,843 \pm 336$
Mean across plants	0.9961 ± 0.0016	$48,804 \pm 8,170$	0.7582 ± 0.8225	$43,105 \pm 7,826$

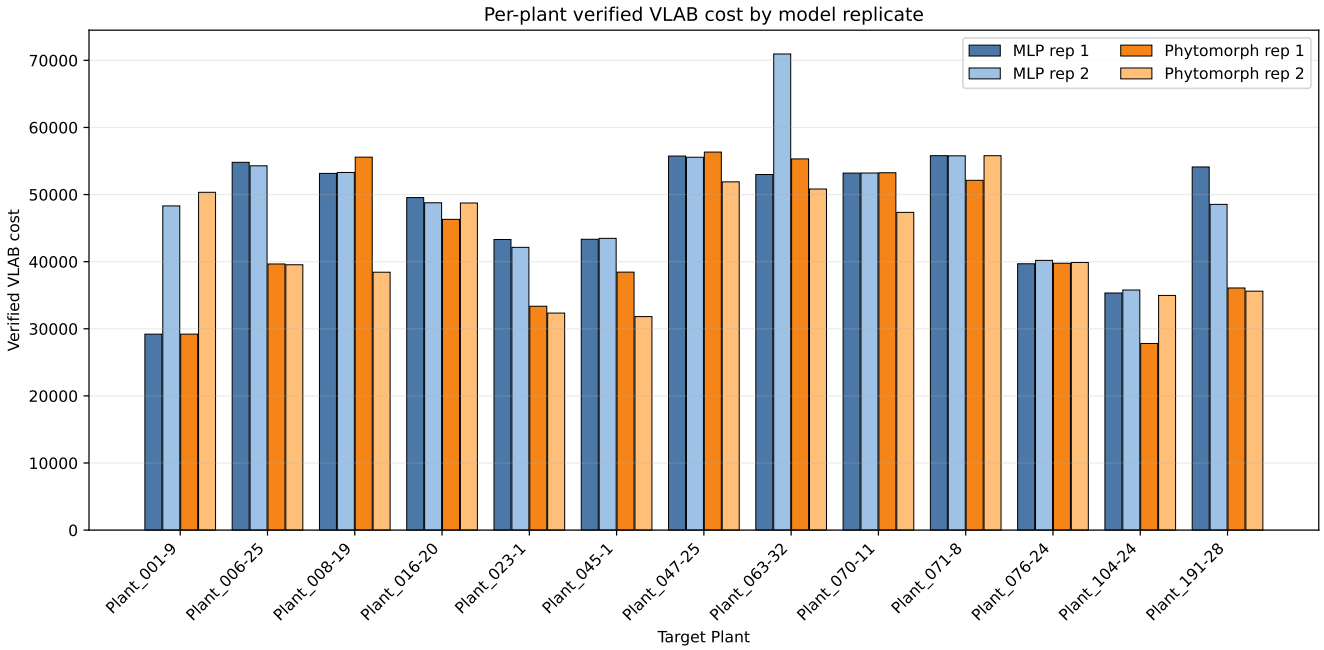


Figure 8 Verified vlab cost for every plant, model, and replicate. Bars are grouped by target plant, with the MLP surrogate and the phytomorphic surrogate shown for each replicate. Lower is better.

4.2. Final All-Plant Optimization Results

Table 3 reports per-plant test R^2 and verified vlab cost for the MLP surrogate and the phytomorphic surrogate across all 13 UNL-CPPD target plants. Figure 8 visualizes these results. The complete all-plant experiment results are provided in Appendix Table 19. It is important to note that the single collapse replicate (`Plant_104-24` in Appendix Table 19) gave a negative R^2 value of -4.95, causing the mean in Table 3 to drop to -1.97. This is explained in Section 5.6.

4.3. Qualitative Plant Comparisons

Figures 9 and 10 show six selected comparisons. For each plant, the displayed model output is the lowest-verified-cost result among its two replicates. The cases include the strongest Phytomorph-favored result (`Plant_006-25`), three near-median cases, the strongest MLP-favored result (`Plant_076-24`), and the Phytomorph R^2-collapse replicate (`Plant_104-24`). Figure 12 in the appendix provides all 13 target plant comparisons.



Figure 9 Target plants and optimized model outputs. From top to bottom: `Plant_001-9` (typical case), `Plant_006-25` (strongest Phytomorph-favored result), and `Plant_063-32` (typical case).

4.4. Predictive Accuracy Versus Optimization Utility

The per-plant results in Table 3 separate the surrogate's predictive accuracy (test R^2) from its optimization utility (verified vlab cost after surrogate-guided search). The MLP surrogate obtained the higher test R^2 on all 13 plants, whereas the phytomorphic surrogate obtained the lower verified vlab cost on 12 of 13. Consequently, only one plant (`Plant_001-9`) selected the same model under both metrics: MLP had both higher test R^2 and lower verified cost there, and its cost advantage was marginal (1,017 units, 2.6%). The two metrics therefore selected different models on the other 12 plants.

Rank correlations between test R^2 and verified cost were correspondingly weak (Appendix Table 21). Across the 26 per-plant, per-model means, the association was negligible and non-significant (Spearman $\rho = 0.12$, $p = 0.55$), as it was within the phytomorphic model alone (Spearman $\rho = 0.13$, $p = 0.68$; $n = 13$). For the MLP's 13 per-plant means, test R^2 was negatively associated with verified cost (Spearman $\rho = -0.632$, $p = 0.021$; Pearson $r = -0.610$, $p = 0.027$),

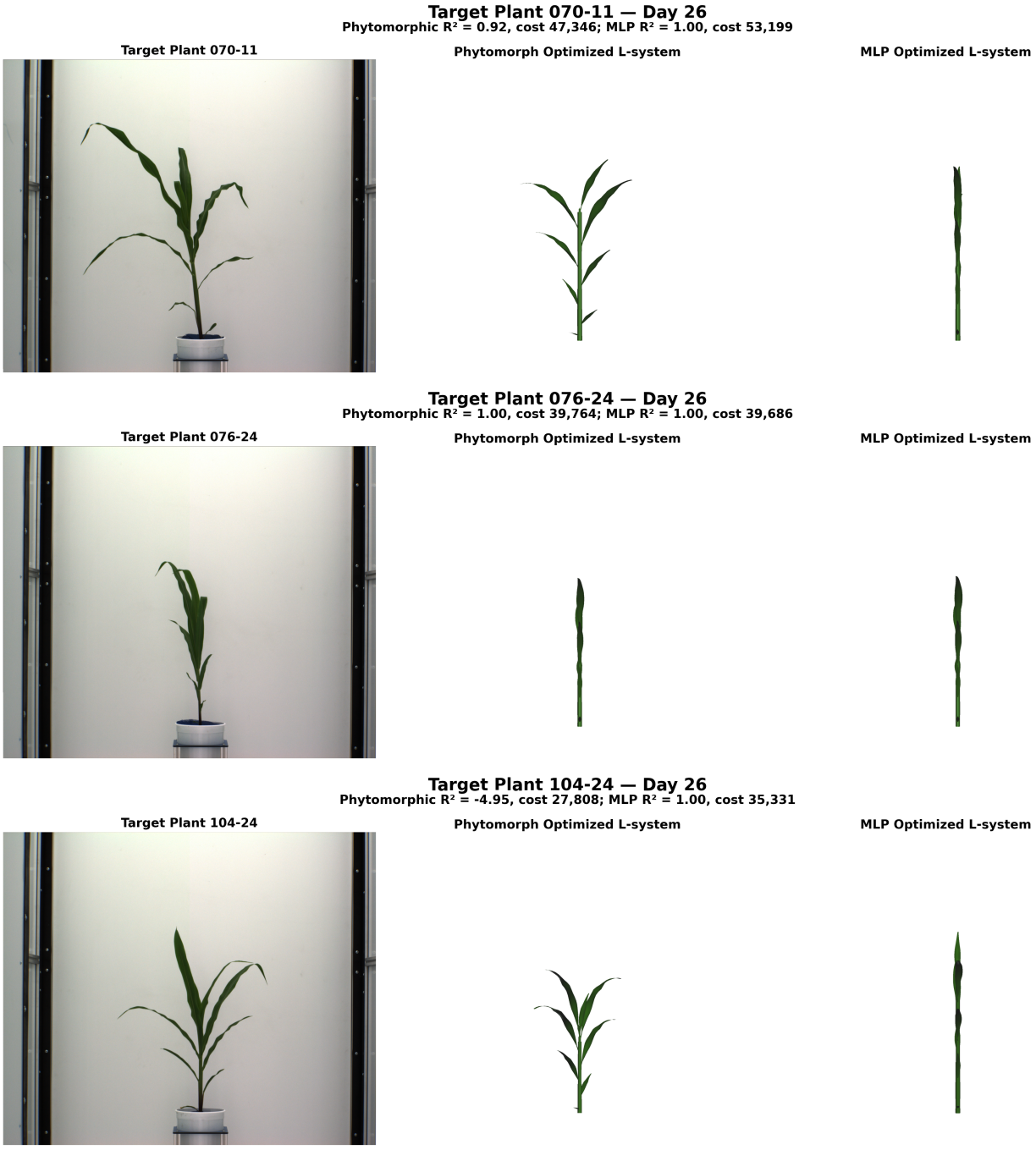


Figure 10 Additional target plants and optimized model outputs. From top to bottom: `Plant_070-11` (typical case), `Plant_076-24` (strongest MLP-favored result), and `Plant_104-24` (Phytomorph R^2-collapse replicate).

providing suggestive evidence of a within-model association in these plants but not establishing that predictive accuracy generally tracks optimization utility across model families. A pooled Pearson correlation over all 52 raw replicates was reported nominally significant ($r = 0.28$, Pearson $p = 0.048$) but was produced almost entirely by the single collapsed `Plant_104-24` replicate. Removing it reduced the Pearson correlation to near zero ($r = 0.02$, Pearson $p = 0.87$; $n = 51$), while the rank correlation was weak both with and without that replicate (Spearman $\rho = 0.02$ and -0.04). That replicate is examined in Section 5.6.

4.5. Parameter Boundary Analysis

Surrogate-guided optimization constrains each of the 13 parameters to a 2σ interval around the training-set mean (Section 3.10), derived from the sampling distributions in Table 2. Figure 11 reports the mean absolute z-score of each optimized parameter across the two independently trained

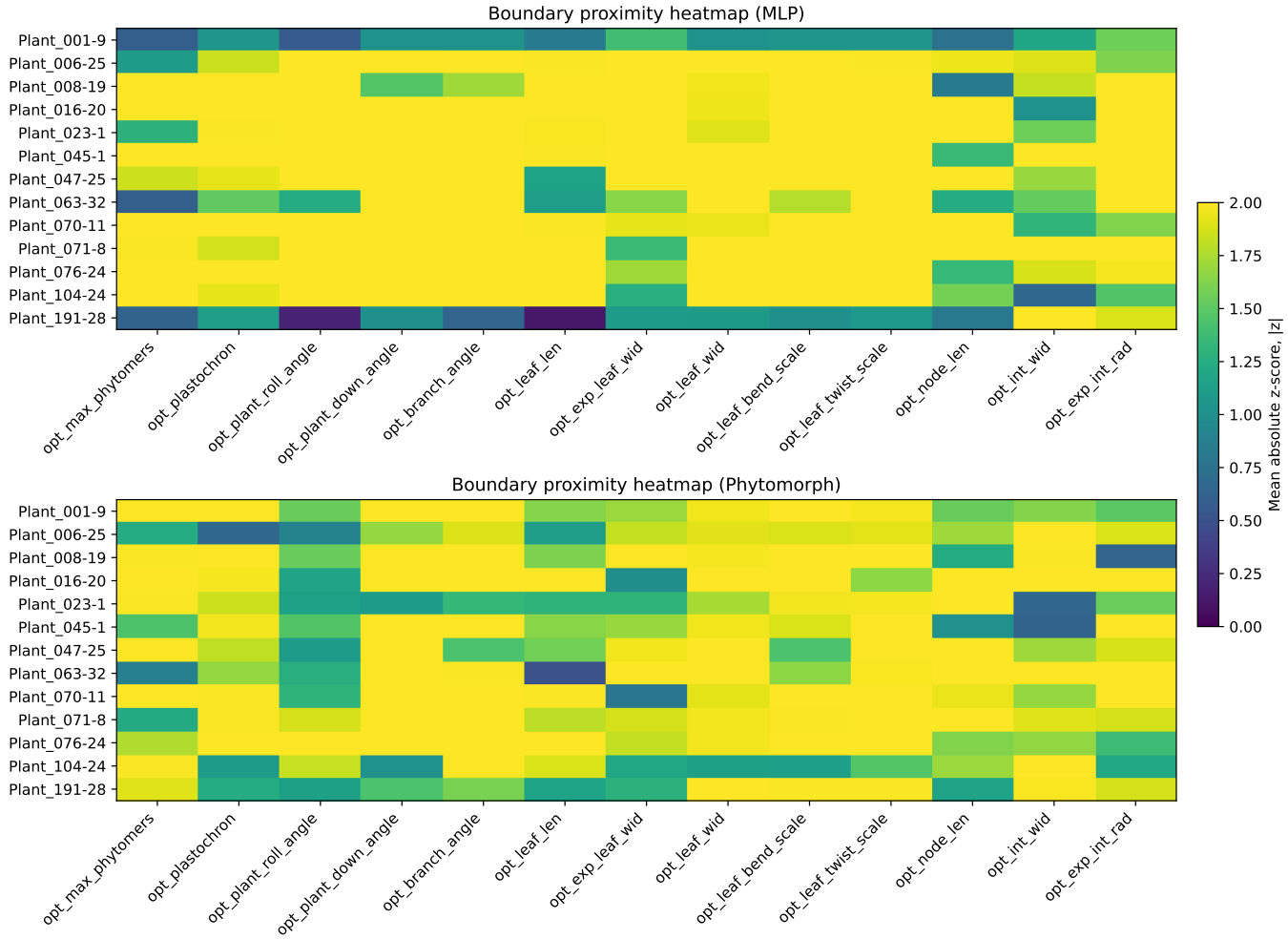


Figure 11 Mean absolute z-scores $|\widetilde{x}_k|$ of the optimized parameter sets found by each model, computed from the surrogate's stored training-set input statistics. Larger values indicate optimized parameters that lie further from the training-distribution mean and closer to the 2σ optimization bounds.

replicates. We interpret these boundary patterns in Section 5.5.

5. Discussion

5.1. Predictive Accuracy Versus Optimization Utility

The MLP achieved higher test R^2 on all 13 plants, yet the phytomorphic surrogate achieved lower verified costs on 12 of 13. The MLP provided an optimized L-system on `Plant_076-24` with a verified cost roughly 80 units lower than the phytomorph's, but per-plant means across replicates still favoured the phytomorph for `Plant_076-24`.

The comparisons in Figures 9 and 10 provide visual examples of some of the optimized L-systems across targets and surrogate types. Visually, the phytomorph optimized L-system matches features of `Plant_006-25`, having the same number of prominent phytomers, and has similar leaf structures and branching angles, while failing to match the decurved nature of the leaves. The typical cases, `Plant_001-9`, `Plant_063-32`, and `Plant_070-11` show the range in behaviours the surrogates can express during optimization. The phytomorph generally visually performs well in this experiment, matching visual/spatial characteristics of the leaves and often closely matching phytomers. The MLP often produces an optimized L-system that appears flat in the 2D projection, which is associated with `PLANT_ROLL_ANGLE` values near 180 (see Section 5.5), but in a few cases it visually matches the target plant. Target plant `Plant_063-32` is an interesting case: both surrogates produce optimized L-systems that look visually similar to the target. The Phytomorph appears to match the phytomers' spatial coordinates more closely, consistent with its lower verified cost and the visual comparison. The MLP, however, appears to better match the leaves' shape and width. Its output also has five phytomers on the right side, matching the target, whereas the Phytomorph output has only four.

Correlation between test R^2 and verified cost was weak and largely driven by a single collapsed replicate (Section 4.4), reinforcing that held-out regression fit is not a reliable proxy for optimization utility. This divergence is consistent with the surrogates' distinct pathways: the MLP maps parameters to cost through a single feedforward path, so reducing its error requires tracking the target, whereas the phytomorphic

surrogate's longer pipeline can admit near-constant, input-independent outputs that still lower loss by predicting close to the target mean (Sections 5.6, 5.4). Surrogates that preserve elements of the downstream objective pathway appear more likely to yield useful optimization landscapes, and should be compared using end-to-end, simulator-verified protocols rather than predictive accuracy alone.

5.2. Role of Structural Inductive Bias in Surrogate-Guided Optimization

The empirical results support the view that structural inductive bias can improve surrogate usefulness for optimization even when it does not maximize predictive R^2. The phytomorphic surrogate constrains scalar cost prediction through intermediate landmark generation, differentiable assignment, and temporal aggregation, and under this constrained pathway achieved lower verified vlab cost than the MLP on most target plants despite lower test R^2.

The ablation study (Section 5.4) provides additional evidence: variants that removed key structural components showed failure modes not captured by scalar fit alone, indicating that structural modules shape the optimization landscape exposed to parameter search, not just representation capacity. Structural inductive bias therefore appears to function as a practical regularizer on the search landscape rather than as a direct mechanism for maximizing held-out regression metrics.

5.3. Contextual Comparison to a Prior Optimization Method

For external context, we compare our best verified vlab costs to those reported by (Khan et al., 2026), which uses exact Hungarian matching with genetic algorithm optimization. Our paper and (Khan et al., 2026) uses similar cost methodology, therefore the cost values are primarily comparable. This comparison contextualizes performance and is not the primary goal of this study.

Table 4 shows that the genetic algorithm (GA) baseline achieved lower best verified vlab cost on 11 of 13 plants, while PhytoNN achieved lower cost on 2 of 13 plants (`Plant_006-25` and `Plant_008-19`). This outcome is expected given the optimization setup: GA directly optimizes the true simulator objective and is allowed a very large online simulation budget, whereas our method optimizes a learned differentiable proxy and verifies selected candidates afterward. (Khan et al., 2026) uses population size 100 for 1,000 generations per GA run (100,000 forward simulations), repeated five times per plant (500,000 total), whereas our method uses a fixed offline budget of 85,000 samples per plant (50,000 train, 10,000 validation, 25,000 test), performing search primarily in surrogate space with simulator calls reserved for verification.

As reported in (Khan et al., 2026), each GA run required approximately 2 days of compute per target plant, and with 5 runs per plant, obtaining these results required approximately 10 days of compute per plant, or roughly 130 days in total across all 13 plants under a sequential accounting.

Therefore, the importance of these results is not that the surrogate always beats GA on the final minimum, but that a differentiable surrogate with explicit structural inductive bias can produce competitive verified solutions via a fundamentally different, gradient-based pathway.

5.4. Architectural Insights from the Ablation Study

Low verified cost alone did not indicate a functioning surrogate. Both the hollow and no-aggregator variants achieved low verified cost but showed flat-objective behaviour on audit: the hollow variant never learned (test $R^2 \approx 0$) and produced a constant predicted cost, and the no-aggregator variant behaved similarly. We therefore do not treat these low costs as evidence of a useful architecture; the no-aggregator result only shows that removing the temporal aggregator made the objective fragile here, not that the aggregator is unnecessary in general.

The no-encoder and no-scaler ablations were more informative. Removing the point-set encoder preserved strong test R^2 but worsened verified cost, showing that contextual set encoding matters more for optimization than for scalar fitting. The no-scaler variant was numerically unstable (one replicate collapsed in R^2), though its verified cost stayed near the full model. We selected the full phytomorphic surrogate for final experiments, as it retained high accuracy, avoided the flat-objective failures above, and achieved strong verified cost.

5.5. Boundary Behaviour and Optimization Stability

The boundary analysis shows that optimized parameter sets from the two surrogate families exhibit different search behaviour under the same 2σ-bounded optimization domain: the MLP surrogate more frequently produced candidates near bound limits, while the phytomorphic surrogate more often returned interior solutions.

This difference should not be read as one model being more exploitative than the other: boundary contact can reflect either surrogate instability near the training-supported region's edges, or legitimate pressure toward lower-cost regions that partially lie outside the imposed bounds. The phytomorphic inductive bias may regularize the objective surface and reduce extreme extrapolative moves, but could equally suppress signals that would otherwise drive the optimizer toward boundary-adjacent candidates, while the MLP may simply expose steeper edge-approaching gradients on some targets. Boundary behaviour is therefore best understood as a joint property of architecture and the imposed constraints.

The MLP surrogate frequently finds solutions where `PLANT_ROLL_ANGLE` is near 180. This outputs a synthetic plant that looks flat from the perspective for 2D annotation capture. This can be seen more clearly in Appendix figure 12, in 9/13 MLP optimized L-systems.

5.6. R^2 Collapse Analysis

In a few instances, the phytomorphic model's R^2 value collapsed to a value near or below zero during training, most notably `Plant_104-24` Replicate 1, which collapsed by epoch

Table 4 Contextual cross-study comparison of best verified vlab costs per plant. Lower is better.

Plant ID	PhytoNN best (this paper)	GA best (Khan et al.)	Difference (PhytoNN - GA)	Difference (%)
Plant_001-9	29,202	28,241	961	3.40%
Plant_006-25	39,539	40,228	-689	-1.71%
Plant_008-19	38,428	43,012	-4,584	-10.66%
Plant_016-20	46,305	37,614	8,691	23.10%
Plant_023-1	32,340	30,110	2,230	7.41%
Plant_045-1	31,815	27,989	3,826	13.67%
Plant_047-25	51,893	40,916	10,977	26.83%
Plant_063-32	50,828	27,088	23,740	87.64%
Plant_070-11	47,346	30,437	16,909	55.55%
Plant_071-8	52,124	42,282	9,842	23.28%
Plant_076-24	39,764	20,274	19,490	96.14%
Plant_104-24	27,808	26,071	1,737	6.66%
Plant_191-28	35,605	33,522	2,083	6.21%

20 to a final value of -4.95. This behaviour was specific to the phytomorphic surrogate: across all 13 target plants, the MLP surrogate never produced a collapsed R^2, remaining above 0.99 in every run.

The collapse was intermittent rather than systematic. It appeared in only four phytomorphic replicates (Plant_006-25, Plant_047-25, Plant_071-8, and Plant_104-24). In three of these, the run recovered within the training budget; only Plant_104-24 Replicate 1 remained collapsed at the final epoch. This reinforces that predictive accuracy and optimization utility are independent properties of a surrogate: despite its catastrophic R^2 collapse, this replicate achieved the lowest verified vlab cost of any phytomorphic replicate for Plant_104-24 (27,808), outperforming both MLP replicates and approaching the genetic-algorithm baseline (26,071; Table 4), showing that a surrogate need not fit the true cost well to still yield a useful optimization landscape for this target. See the Plant_104-24 entry in Figure 10. The phytomorph optimized L-system looks visually similar to the real target plant, and has the lowest verified cost among surrogate runs for this target.

5.7. Limitations and Future Directions

This study has several limitations. Preliminary model selection was performed on one fixed target plant (Plant_023-1) before final all-plant evaluation, so selected settings could retain target-specific bias. Optimization was restricted to a fixed 2σ parameter box, which reduces out-of-distribution exploitation but may prevent valid improving directions. The objective is based on 2D topology landmarks and day-wise matching, which does not capture full 3D morphology or appearance-level traits. Intermittent instability in the phytomorphic pipeline indicates sensitivity to architectural and optimization choices. The two-replicate budget per plant and model also limits confidence in individual per-plant comparisons, given the variance observed for some plants. Increasing the number of optimization restarts and gradient steps per restart could plausibly lower the best verified vlab cost further for both surrogates.

Future work should prioritize cross-target model selection during tuning, adaptive bound or trust-region schedules with explicit simulator verification, and richer geometric objectives that extend beyond 2D landmark agreement. It should also strengthen surrogate stability through improved matching/aggregation constraints and broaden baselines to include additional optimization-oriented surrogate families.

6. Conclusion

This paper pursued two goals: designing a differentiable, objective-based phytomorphic neural surrogate capable of approximating L-system-driven plant generation, and demonstrating its practical utility on a real parameter-optimization problem – matching a simulated maize plant to a target real plant by minimizing a non-differentiable Hungarian landmark-matching cost. We introduced a two-stage framework that trains a neural surrogate on a single target plant, then uses it to guide parameter search, with candidates verified against the original simulation. Comparing a direct MLP cost regressor against the phytomorphic surrogate across 13 target plants, the phytomorphic surrogate achieved better optimization performance despite the MLP's consistently higher held-out predictive accuracy. Ablation results further showed that the phytomorphic surrogate's structural components meaningfully shape its cost landscape and downstream optimization utility. These findings indicate that an inductively biased, differentiable surrogate can practically improve optimization on a non-differentiable procedural simulator beyond what a direct regressor provides, and that such surrogates should be evaluated primarily by simulator-verified task performance rather than held-out predictive accuracy alone.

Acknowledgments

UNL-Component Plant Phenotyping Dataset (UNL-CPPD) provided by the University of Nebraska-Lincoln.

Author contributions statement

William Bushell (W.B.) developed the surrogate models, ran experiments, analyzed results, and drafted the paper. Nazifa Khan (N.K.) provided the L-system/phenotyping pipeline, prior genetic-algorithm framework, and dataset support. Ian McQuillan (I.M.) supervised the project, contributed to study design, and revised the paper.

Conflict of interest

The authors declare no competing interests.

Funding

The work of I.M. was supported by the Natural Sciences and Engineering Research Council of Canada Grant 2022-05092, and the work of W.B. was supported by funding from the Natural Sciences and Engineering Research Council CREATE program.

Data availability

All source code for the phytomorphic surrogate neural network, baseline surrogate models, training scripts, and evaluation procedures is available at: `https://github.com/bushwill/Phytomorphic-Neural-Network-2026`. The repository includes: implementation of the hierarchical phytomorphic surrogate neural network architecture; scripts for L-system plant simulation, data preprocessing, and cost calculation; training and evaluation workflows for all models discussed in this paper; and a complete automated pipeline process to produce the experiments detailed in this paper, and the produced results.

Generative AI disclosure

The authors used AI-assisted writing tools to support LaTeX-specific formatting, drafting of alternative phrasings, and article organization. All scientific content, interpretations, and final text were reviewed and approved by the authors.

REFERENCES

Algorithmic Botany. Virtual laboratory. `https://github.com/AlgorithmicBotany/vlab`, 2026. Software repository, accessed 24 June 2026.

P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

M. Cieslak, N. Khan, P. Ferraro, R. Soolanayakanahally, S. J. Robinson, I. Parkin, I. McQuillan, and P. Prusinkiewicz. L-system models for image-based phenomics: case studies of maize and canola. *in silico Plants*, 4(1): diab039, 01 2022. ISSN 2517-5025. doi: 10.1093/insilicoplants/diab039. URL `https://doi.org/10.1093/insilicoplants/diab039`.

M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems*, volume 26, pages 2292–2300, 2013.

A. Darvishpour, M. Cieslak, and A. Runions. The effects of synthetic data and label distribution on canola branch counting, 2026. URL `https://arxiv.org/abs/2607.09630`.

S. Das Choudhury, S. Bashyam, Y. Qiu, A. Samal, and T. Awada. Holistic and component plant phenotyping using temporal image sequence. *Plant Methods*, 14:35, 2018. ISSN 1746-4811 (Print); 1746-4811 (Electronic); 1746-4811 (Linking). doi: 10.1186/s13007-018-0303-x.

H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 605–613, 2017.

H. Farooq, M. N. Zakaria, M. F. Hassan, and S. Sulaiman. An approach to derive parametric L-system using genetic algorithm. In *International Visual Informatics Conference*, pages 455–466. Springer, 2009.

M. Gaillard, V. Krs, G. Gori, R. Mech, and B. Benes. Automatic differentiable procedural modeling. *Computer Graphics Forum*, 41(2), 2022.

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 (TOG)*, 39(5):1–13, 2020.

I. Hossain, I.-C. Shen, and O. van Kaick. Approximating procedural models of 3d shapes with neural networks. *Computer Graphics Forum*, 43(2), 2024.

Y. Hu, P. Guerrero, M. Hašan, H. Rushmeier, and V. Deschaintre. Node graph optimization using differentiable proxies. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–9, 2022.

M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

N. A. Khan, M. Cieslak, M. Eramian, and I. McQuillan. Effectiveness of training with procedurally generated synthetic images of crop plants. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 5490–5500, 2025.

N. A. Khan, W. Bushell, M. Cieslak, and I. McQuillan. Learning L-system models of plants matching real images using genetic algorithm. In *Proceedings of the 23rd International Conference on Unconventional Computing and Natural Computing (UCNC 2026)*, Lecture Notes in

Computer Science, 2026.

H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955.

J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning (ICML)*, pages 3744–3753, 2019.

J. J. Lee, B. Li, and B. Benes. Latent L-systems: Transformer-based tree generator. *ACM Transactions on Graphics (TOG)*, 42(4):1–13, 2023.

A. Lindenmayer. Mathematical models for cellular interactions in development, parts i and ii. *Journal of Theoretical Biology*, 18:280–315, 1968.

J. Liu, S. Snodgrass, A. Khalifa, S. Risi, G. N. Yannakakis, and J. Togelius. Deep learning for procedural content generation. *Neural Computing and Applications*, 33(1): 19–37, 2021.

I. McQuillan, J. Bernard, and P. Prusinkiewicz. Algorithms for inferring context-sensitive L-systems. In *International Conference on Unconventional Computation and Natural Computation*, pages 116–130. Springer, 2018. doi: 10.1007/978-3-319-92435-9_9.

G. Mena, D. Belanger, S. Linderman, and J. Snoek. Learning latent permutations with Gumbel-Sinkhorn networks. In *International Conference on Learning Representations (ICLR)*, 2018.

M. P. Pound, J. A. Atkinson, A. J. Townsend, M. H. Wilson, M. Griffiths, A. S. Jackson, A. Bulat, G. Tzimiropoulos, D. M. Wells, E. H. Murchie, T. P. Pridmore, and A. P. French. Deep machine learning provides state-of-the-art performance in image-based plant phenotyping. *GigaScience*, 6(10):gix083, 2017.

P. Prusinkiewicz. Graphical applications of l-systems. In *Proceedings on Graphics Interface '86/Vision Interface '86*, page 247–253, CAN, 1986. Canadian Information Processing Society.

P. Prusinkiewicz and A. Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag, New York, 1990. doi: 10.1007/978-1-4613-8476-2.

C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017.

M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

P. Room, J. Hanan, and P. Prusinkiewicz. Virtual plants: new perspectives for ecologists, pathologists and agricultural engineers. *Trends in Plant Science*, 1(1): 33–38, 1996.

P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4938–4947, 2020.

O. Stava, S. Pirk, J. Kratt, B. Chen, R. Měch, O. Deussen, and B. Benes. Inverse procedural modeling of trees. *Computer Graphics Forum*, 33(6):118–131, 2014. doi: 10.1111/cgf.12328.

F. Tardieu, L. Cabrera-Bosquet, T. Pridmore, and M. MacMillan. Plant phenomics, from sensors to knowledge. *Current Biology*, 27(15):R770–R783, 2017. doi: 10.1016/j.cub.2017.05.055.

J. Ubbens, M. Cieslak, P. Prusinkiewicz, and I. Stavness. The use of plant models in deep learning: an application to leaf counting in rosette plants. *Plant Methods*, 14(1): 1–10, 2018.

D. Ulyanov, A. Vedaldi, and V. Lempitsky. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9446–9454, 2018.

J. Willard, X. Jia, S. Xu, M. Steinbach, and V. Kumar. Integrating scientific knowledge with machine learning for engineering and environmental systems. *ACM Computing Surveys (CSUR)*, 55(4):1–37, 2022.

Appendices

A. UNL-CPPD Target Plant Day Ranges

Table 5 Day ranges for the UNL-CPPD target plants used in the final all-plant experiments.

Target plant	First day	Last day	Number of days
`Plant_001-9`	2	27	26
`Plant_006-25`	2	27	26
`Plant_008-19`	2	27	26
`Plant_016-20`	2	27	26
`Plant_023-1`	5	27	23
`Plant_045-1`	2	27	26
`Plant_047-25`	2	27	26
`Plant_063-32`	2	27	26
`Plant_070-11`	2	27	26
`Plant_071-8`	2	27	26
`Plant_076-24`	2	27	26
`Plant_104-24`	2	27	26
`Plant_191-28`	2	26	25

B. Maize L-System Axiom and Production Rules

The fixed maize L-system used in this study is the 13-parameter model described in (Cieslak et al., 2022). We provide the axiom and production rules here. The parameters appearing in these rules correspond to the parameter definitions in Table 1.

At a conceptual level, the model consists of apex modules A, internode modules I, and leaf modules L. The apex A periodically decomposes into a new internode I, a lateral leaf L, and a renewed apex A. The resulting modules age over time and are graphically interpreted as three-dimensional plant geometry (Prusinkiewicz, 1986). Their recorded positions are converted into the two-dimensional landmark time-series sequences used for comparison with the target plant. Simplified behaviour of each of these three primary modules is found in Table 6.

Before presenting the simulator-specific implementation, we summarize the developmental logic of the maize L-system in a simplified presentation. The axiom is as follows:

$$\omega = A(a_0, 0) \tag{2}$$

The initial apex age is a_0 = PLASTOCHRON, and the second argument of A denotes the phytomer index.

Table 6 Simplified production rules for the maize L-system. Each rule is applied when its stated condition is satisfied.

Production rule	Condition
$A(a,n) \rightarrow A(a + \Delta a, n)$	Applied at every simulation step, where $\Delta a = \text{DAGE} \cdot \text{DT}$.
$A(a,n) \rightarrow I(a,n)[L(a,n)]A(a, n+1)$	Applied when $a \geq$ PLASTOCHRON and $n <$ MAX_PHYTOMERS. The apex age is then reset by $a \leftarrow a -$ PLASTOCHRON.
$L(a,n) \rightarrow L(a + \Delta a, n)$	Applied at every simulation step.
$I(a,n) \rightarrow I(a+\Delta a, n)$	Applied at every simulation step.

The following listings give the corresponding simulator-specific implementation, including the age-update rules, apex decomposition rule, landmark recording, and geometric interpretation rules.

The system uses a maximum derivation length of `STEPS` = 1000 and a decomposition depth of 100. The axiom is shown below. The modules `Init`, `A`, `L`, and `VisualAid` are simulator modules; `SB` and `EB` delimit a branch.

Listing 1. Axiom of the fixed maize L-system.

```
Axiom:
Init()
SetColor(255)
SB
Down(val(PLANT_DOWN_ANGLE))
RollL(val(PLANT_ROLL_ANGLE))
A(val(PLASTOCHRON),0)
EB
VisualAid
```

During production group 1, the age of the apex, internode, and leaf modules increases at each simulation step.

Listing 2. Age-update productions for the fixed maize L-system.

```
production:
group 1:

A(age,n):
{
    age += DAGE*DT;
    produce A(age,n);
}

L(age,n):
{
    age += DAGE*DT;
    produce L(age,n);
}

I(age,n):
{
    age += DAGE*DT;
    produce I(age,n);
}
```

The apex decomposition rule creates a new internode and leaf when the apex reaches the PLASTOCHRON threshold and the maximum number of phytomers has not been reached. The new leaf is placed using the branching-angle function, and the apex is recreated with an incremented phytomer index.

Listing 3. Apex decomposition rule for phytomer creation.

```
decomposition:
maximum depth: 100;

A(age,n):
{
    if (age >= val(PLASTOCHRON)
        && n < val(MAX_PHYTOMERS))
    {
        age = age - val(PLASTOCHRON);

        produce
            I(age,n)
            GetPos(0,0,0)
            SB
                Down(val(BRANCH_ANGLE)
                     * br_target_angle(n))
                L(age,n)
            GetPos(0,0,0)
            EB
            RollL(180.0)
            A(age,n+1);
    }
}
```

Production group 2 records the positions of the internode and leaf landmarks for each developmental day. The simulator writes these positions as three-dimensional coordinates, which are subsequently projected into the two-dimensional landmark representation used by the matching objective.

Listing 4. Landmark-recording productions.

```
production:
group 2:

Init():
{
    fprintf(leafPositionFile, "Day: %g\n",
            std::floor(sim_time));
    produce Init;
}

L(age,n) > GetPos(x,y,z):
{
    fprintf(leafPositionFile, "L %g %g %g %g\n",
            n, x, y, z);
    produce L(age,n);
}

I(age,n) > GetPos(x,y,z):
{
    fprintf(leafPositionFile, "I %g %g %g %g\n",
            n, x, y, z);
    produce I(age,n);
}
```

The graphical interpretation rules convert internode and leaf modules into three-dimensional geometry. The internode rule draws a cylinder whose length and width depend on the internode-related parameters. The leaf rule generates a leaf object whose properties and shape are controlled by corresponding parameters and functions from a function set.

Listing 5. Interpretation rules for internodes and leaves.

```
interpretation:
maximum depth: 100;

I(age,n):
{
    if (age <= 0.)
        produce;

    float len =
        val(NODE_LEN)
        * internode_target_len(n)
        * internode_length(age);

    float wid =
        powf(len,val(EXP_INT_RAD))
        * val(INT_WID)
        * internode_width(n);

    produce SetWidth(wid) F(len);
}

L(age,n):
{
    if (age <= 0.)
        produce;

    ++num_leaves;

    float len =
        val(LEAF_LEN)
        * leaf_target_len(n)
        * leaf_length(age);

    nproduce
        SetWidth(powf(len,val(EXP_LEAF_WID))
        * val(LEAF_WID)
        * leaf_width(0));

    nproduce
        Down(br_angle(age))
        StartGC();

    float x = 0.f;

    for (x = 0.; x < 1.; x += DX)
    {
        nproduce
            BlendedContour(0,1,powf(x,0.2));

        nproduce
            F(len*DX);

        nproduce
            SetWidth(powf(len,val(EXP_LEAF_WID))
            * val(LEAF_WID)
            * leaf_width(x))
            Down(leaf_bend(x)*len*DX)
            RollL(leaf_twist(x)*len*DX);
    }

    nproduce F(len*(1.-x));
    produce EndGC();
}
```

The simulator's turtle interpretation uses three-dimensional coordinates for geometry generation. For the phenotyping objective, the recorded geometry is converted to daily two-dimensional BP and EP landmark sets for comparison with the real target plant, as described in Section 3.3.

C. Neural Surrogate Architectures and Implementation

C.1. Baseline MLP

The MLP surrogate is a three-layer fully connected network:

$$\text{Linear}(13, 128) \to \text{ReLU} \to \text{Linear}(128, 128) \to \text{ReLU} \to \text{Linear}(128, 1).$$

Its scalar output is de-normalized to vlab cost units using the stored training-set output statistics, with no additional nonlinearity.

C.2. Structure-Guidance Loss

For a single sample with a nonempty target set, let P be the predicted points with existence probabilities p_i, and let Y be the padded target set with existence booleans m_j. Using normalized weights

$$w_i = \frac{p_i}{\sum_{k \in P} p_k}, \qquad v_j = \frac{m_j}{\sum_{k \in Y} m_k},$$

the coordinate loss is the probability-weighted nearest-neighbor distance:

$$\ell_{\text{coord}} = \sum_{i \in P} w_i \min_{j \in Y} \|x_i - y_j\|_2 + \sum_{j \in Y} v_j \min_{i \in P} \|y_j - x_i\|_2.$$

We use nearest-neighbor alignment because landmark sets are unordered and variable in size. Nearest-neighbor distances provide a simple permutation-invariant geometric signal without requiring a hard one-to-one correspondence at this stage. The bidirectional form reduces one-sided collapse (e.g., predicted points covering only part of the target).

The count loss penalizes mismatch between predicted expected and true valid landmark counts:

$$\ell_{\text{count}} = \left(\sum_{i \in P} p_i - \sum_{j \in Y} m_j \right)^2.$$

Its purpose is to calibrate existence probabilities so SGN learns not only where landmarks should be, but also how many should be active.

The total guidance loss is

$$\mathcal{L}_{\text{guide}} = \frac{1}{B} \sum_{b=1}^{B} \left[\ell_{\text{coord}}^{\text{BP}} + \ell_{\text{coord}}^{\text{EP}} + 0.1 \left(\ell_{\text{count}}^{\text{BP}} + \ell_{\text{count}}^{\text{EP}} \right) \right]_b.$$

The weighting choices (0.01 on $\mathcal{L}_{\text{guide}}$, 0.1 on ℓ_{count}) were selected to keep cost regression as the primary objective while using structure guidance as a weak regularizer. For each landmark type, if no valid targets occur anywhere in a minibatch, the corresponding coordinate loss is set to zero. If a sample has no targets while other samples in the minibatch do, its predicted-to-target distances are masked to 10^6 in the coordinate-loss calculation; its target-to-predicted term has no valid target entries. Its count loss still compares the predicted count with a target count of zero. Thus, an empty-target sample in a mixed minibatch can incur a large coordinate penalty. The guidance loss is weighted by 0.01 in the total loss.

C.3. Structure Generation Network (SGN)

A shared encoder processes the normalized parameter vector:

$$\text{Linear}(13, 128) \to \text{ReLU} \to \text{Linear}(128, 128) \to \text{ReLU} \to \text{Linear}(128, 64) \to \text{ReLU},$$

producing a 64-dimensional latent representation passed to two heads (BP and EP), each:

$$\text{Linear}(64, 128) \to \text{ReLU} \to \text{Linear}(128, 150).$$

Each 150-dimensional output is reshaped into $N_{\max} = 50$ slots of (x, y, ℓ), with coordinates scaled by 200 and ℓ converted to a probability via sigmoid:

$$p_i = \text{sigmoid}(\ell_i) = \frac{1}{1 + \exp(-\ell_i)}.$$

$$P_{\text{BP}} = \left\{ \left(\mathbf{r}_i^{\text{BP}}, p_i^{\text{BP}} \right) \right\}_{i=1}^{N_{\max}}, \qquad P_{\text{EP}} = \left\{ \left(\mathbf{r}_i^{\text{EP}}, p_i^{\text{EP}} \right) \right\}_{i=1}^{N_{\max}},$$

where $\mathbf{r}_i \in \mathbb{R}^2$ is a predicted coordinate and $p_i \in (0, 1)$ is the probability the slot is a valid landmark.

C.4. Sinkhorn Assignment Network (SAN)

For each day t, we compute the absolute coordinate mean and standard deviation statistics of the valid real landmarks, $\phi_t \in \mathbb{R}^4$ (BP and EP concatenated), and pass them through a small MLP g_θ producing a bounded scale modifier.

$$\boldsymbol{\Delta}_t = g_\theta(\phi_t) \in \mathbb{R}^2.$$

This modifier adjusts a learnable base scale $\mathbf{b} \in \mathbb{R}^2$ (shared across all days) through a bounded tanh nonlinearity, so that the day-specific scale can deviate from the base scale only within a fixed range:

$$\mathbf{s}_t = \exp(\mathbf{b}) \odot \exp(\alpha \tanh(\boldsymbol{\Delta}_t)),$$

with α a bounding constant. The resulting scale s_t is applied to both real and synthetic coordinates for day t, preserving relative geometry while normalizing magnitude across the sequence. The Dynamic Coordinate Scaler maps ϕ_t through

$$\text{Linear}(4, 32) \to \text{ReLU} \to \text{Linear}(32, 2)$$

to produce $\boldsymbol{\Delta}_t = (\Delta_{\text{BP}}, \Delta_{\text{EP}})$, with base log-scale initialized as $\mathbf{b} = (\log 200.0,\ \log 200.0)$ and bounding constant $\alpha = 0.1$.

BP and EP streams each use a shared point-set encoder:

$$\text{Linear}(3, 64) \to \text{ReLU} \to \text{Linear}(64, 64),$$

followed by four-head self-attention (embedding dimension 64, 16 per head), a residual connection, and LayerNorm. No positional encodings, dropout, or cross-attention were used; padded real landmarks were masked only when constructing the assignment-score matrix, not during self-attention.

The log-temperature τ was initialized to zero. Invalid padded real landmarks received a score mask of -10^9; synthetic rows were not masked.

Initializing $Z_{ij}^{(0)} = S_{ij}$, we alternate row/column normalization using LogSumExp:

$$Z_{ij}^{(k+0.5)} = Z_{ij}^{(k)} - \log \sum_{j'} \exp\left(Z_{ij'}^{(k)}\right), \tag{3}$$

$$Z_{ij}^{(k+1)} = Z_{ij}^{(k+0.5)} - \log \sum_{i'} \exp\left(Z_{i'j}^{(k+0.5)}\right). \tag{4}$$

Sinkhorn normalization used $K = 7$ log-space iterations.

C.5. Cost Aggregation Network (CAN)

Per-day costs were padded or truncated to length 26 and processed by

$$\text{Linear}(26, 64) \to \text{ReLU} \to \text{Linear}(64, 32) \to \text{ReLU} \to \text{Linear}(32, 1),$$

with no dropout, BatchNorm, or LayerNorm.

C.6. Training Configuration

All surrogate models were trained in PyTorch using supervised regression to the ground-truth vlab cost labels, on CPU within the Dockerized environment described in Section 3.6. Each epoch corresponds to one full pass over the training loader, with minibatches shuffled at the start of each epoch.

All training runs used Adam with PyTorch's default momentum parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$), varying only the learning rate, with no weight decay. A `ReduceLROnPlateau` scheduler was used, and gradients were clipped to maximum norm 1.0.

Training losses were computed in normalized cost space. Validation tracked both the normalized MSE used for model selection and a de-normalized MSE in raw vlab cost units; reported test performance used R^2 on the held-out split.

Final training runs trained for the full configured number of epochs rather than stopping early. `best_model.pt` was saved at the lowest normalized validation loss, `final_model.pt` at the end of training; optimization used `final_model.pt`. In the separate preliminary tuning workflow, early stopping used verified vlab cost with patience as specified in Section 3.11.

Each replicate used a deterministic seed (base seed plus replicate index, e.g. rep 1 seed = 43), giving reproducible but distinct initializations and minibatch orders across replicates while keeping the train/validation/test split fixed. Training budgets differed by experiment type (Table 7): preliminary tuning used patience-based stopping where applicable, while ablation testing and final all-plant experiments used fixed epoch budgets with no early stopping.

Table 7 Training budgets used across experiment types.

Experiment type	Max epochs	Patience
Initial HP screening	100	5
Convergence analysis	25/50	5/10
Final HP tuning	50	10
Ablation testing	20	-
All-plant experiments	20	-

C.7. Surrogate-Guided Optimization Configuration

For the final evaluation and Stage 4 ablation, each trained surrogate was frozen and optimized with 10 random restarts of 1,000 gradient steps per restart, using the `OptimizerNet` architecture and bounding procedure described above. Unlike training replicates, surrogate-guided optimization restarts were not assigned explicit deterministic seeds; restart initialization and jitter therefore varied stochastically across optimizer runs.

The `OptimizerNet` weights were optimized with Adam using learning rate 0.1, with a step-decay learning-rate scheduler applied using decay factor 0.8 and step size $\max(50, S/4)$, where S is the total number of optimization steps. Generated parameter values include `MAX_PHYTOMERS`, which are passed to vlab without rounding. In the L-system, this value is used as the threshold in $n <$ val(`MAX_PHYTOMERS`), where n starts at 0 and increases by 1. Thus, a positive non-integer threshold effectively permits $\lceil$val(`MAX_PHYTOMERS`)$\rceil$ phytomers; for example, a threshold of 8.018 permits 9.

Within each restart, the optimizer tracked the candidate with the lowest surrogate-predicted cost, and the best candidate from each restart was verified by rerunning vlab and recomputing the true Hungarian matching cost. The final selected candidate was the restart candidate with the lowest verified vlab cost. For final experiment execution, the optimization runner used `final_model.pt` when present and otherwise fell back to `best_model.pt`.

Small Gaussian jitter (0.01) was injected into generated parameters before evaluating surrogate cost to avoid stalling on flat plateaus.

C.8. Final All-Plant Experiment Configuration

For the final evaluation, we repeated the inverse-optimization experiment across all 13 UNL-CPPD target plants listed in Appendix A, including `Plant_023-1`, which was also used as the fixed target for preliminary model selection. For each target plant, a synthetic dataset was generated using the parameter-sampling distribution described in Section 3.7, with only the ground-truth cost labels changing across targets since each plant defines a different reference morphology.

The final comparison used both surrogate families with the hyperparameters selected in Phase 3: the MLP with learning rate 10^{-3}, batch size 8, and the phytomorphic surrogate with learning rate 5×10^{-4}, batch size 8, both trained for a fixed 20 epochs without early stopping. Two independent training replicates were run for each target plant and model family. For the phytomorphic surrogate, stored simulator landmark

Table 8 Summary of preliminary testing phases.

Phase	Data	Reps	Training budget	Purpose
Initial HP screening	1k/200/500 and 20k/4k/10k splits	3	100 epochs, patience 5	Select initial broad HPs
Convergence analysis	25%, 50%, 100% of 50k train split	2	25/5 and 50/10	Select data scale and budget
Final HP tuning	Full 50k train split	2	50 epochs, patience 10	Select final HPs
Ablation testing	Full 50k train split	2	20 fixed epochs	Evaluate phytomorphic variants

tensors were used during training so the structural guidance term $\mathcal{L}_{\text{guide}}$ remained active.

After training, each replicate was optimized using the configuration in Section 3.10. Final results are reported as mean ± standard deviation across the two replicates for each target plant and model; the main text reports compact per-plant and aggregate summaries, while complete replicate-level results are provided in the appendix. Two replicates provide only a coarse estimate of run-to-run variance. This budget was constrained by the training and optimization cost of the phytomorphic pipeline (Appendix H). Variance was non-trivial for some plants, so individual per-plant comparisons carry more uncertainty than the aggregate patterns reported across all 13 plants.

D. Preliminary Model-Selection: Configuration and Results

Preliminary testing was performed on `Plant_023-1` as the fixed reference target. Each preliminary configuration was evaluated end-to-end: the surrogate was trained on the corresponding synthetic training split, assessed on held-out data, frozen for surrogate-guided parameter optimization, and then verified by rerunning vlab and recomputing the true Hungarian landmark-matching cost.

Optimization budgets differed across preliminary phases. Stages 1-3 used a reduced surrogate-guided optimization budget of 3 restarts and 250 steps per restart per optimization call (750 total steps per call) for fast model selection. Stage 4 ablation used the full optimization budget of 10 restarts and 1,000 steps per restart, with training run in train-only mode and optimization executed afterward as a separate step.

Table 8 summarizes the preliminary testing phases, and Table 9 gives the corresponding hyperparameter or variant grids.

Table 9 Hyperparameter and variant grids used in preliminary testing.

Phase	MLP settings	Phytomorphic settings	Notes
Initial HP screening	LR $\{10^{-4}, 5\times10^{-4}, 10^{-3}\}$; BS $\{1, 16, 32\}$	LR $\{10^{-4}, 5\times10^{-4}, 10^{-3}\}$; BS $\{1, 16, 32\}$	Both 1k and 20k regimes
Convergence analysis	LR 10^{-3}; BS 1	LR 5×10^{-4}; BS 16	Training fractions: 25%, 50%, 100%
Final HP tuning	LR $\{5\times10^{-4}, 10^{-3}\}$; BS $\{1, 4, 8\}$	LR $\{5\times10^{-4}, 10^{-3}\}$; BS $\{8, 16, 32\}$	Full 50k train split
Ablation testing	-	Full, hollow, no aggregator, no encoder, no scaler	LR 5×10^{-4}; BS 8

D.1. Phase 1: Initial Hyperparameter Tuning

Initial hyperparameter tuning on 1k and 20k training datasets identified learning rates and batch sizes for subsequent experiments. Table 10 summarizes the best-performing hyperparameter configurations for each model and dataset size. These findings motivated the selection of the 20k dataset configurations of MLP (lr=1×10^{-3}, bs=1) and Phytomorphic (lr=5×10^{-4}, bs=16) for subsequent convergence and final tuning phases. Stage 1 used a reduced optimization budget (3 restarts, 250 steps per restart), so verified vlab costs are intended for within-stage ranking only.

Table 10 Phase 1: Best Hyperparameter Configurations (1k and 20k datasets).

Model	Dataset	LR	Test R^2	Verified vlab cost
MLP	1k	1e-3, BS=16	0.9463 ± 0.0038	$73,545 \pm 325$
MLP	20k	1e-3, BS=1	0.9851 ± 0.0019	$57,327 \pm 5,331$
Phytomorphic	1k	1e-3, BS=1	0.5112 ± 0.4510	$65,122 \pm 6,870$
Phytomorphic	20k	5e-4, BS=16	0.9512 ± 0.0217	$59,346 \pm 406$

1k Training Dataset

Table 11: Phase 1 complete hyperparameter sweep on the 1k training dataset. Each row reports one replicate.

Model	Epochs Trained	Rep	BS	LR	Test R^2	Verified vlab cost
MLP	8	1	1	0.0001	0.945528	73,846.379932
MLP	6	2	1	0.0001	0.930787	70,873.133792
MLP	10	3	1	0.0001	0.948158	73,798.280120
MLP	8	1	16	0.0001	0.860054	73,300.420573
MLP	8	2	16	0.0001	0.866279	70,854.175375
MLP	6	3	16	0.0001	0.813634	71,732.258530
MLP	10	1	32	0.0001	0.828681	73,301.069448
MLP	8	2	32	0.0001	0.815028	70,854.175375
MLP	11	3	32	0.0001	0.833035	73,298.606934
MLP	11	1	1	0.0005	0.948280	83,604.589726
MLP	6	2	1	0.0005	0.943788	73,574.158571
MLP	9	3	1	0.0005	0.950909	73,835.145683
MLP	8	1	16	0.0005	0.945123	73,648.348400
MLP	11	2	16	0.0005	0.945223	73,295.833136
MLP	6	3	16	0.0005	0.934961	73,979.998397
MLP	7	1	32	0.0005	0.930211	73,973.336009
MLP	10	2	32	0.0005	0.944133	73,358.732687
MLP	8	3	32	0.0005	0.936504	73,732.239595
MLP	6	1	1	0.001	0.949653	73,833.898264
MLP	9	2	1	0.001	0.949830	74,025.246238
MLP	9	3	1	0.001	0.945955	73,859.607714
MLP	8	1	16	0.001	0.946722	73,363.869357
MLP	6	2	16	0.001	0.943356	73,347.411473
MLP	9	3	16	0.001	0.950968	73,924.282004
MLP	6	1	32	0.001	0.940483	73,979.998397
MLP	10	2	32	0.001	0.946748	73,528.329830
MLP	6	3	32	0.001	0.931810	73,979.998397
Phytomorphic	9	1	1	0.0001	0.109152	70,331.276518
Phytomorphic	8	2	1	0.0001	0.570858	65,968.913109
Phytomorphic	19	3	1	0.0001	0.737961	55,835.866381
Phytomorphic	7	1	16	0.0001	-0.009141	72,979.652929
Phytomorphic	10	2	16	0.0001	-0.001291	74,557.904090
Phytomorphic	8	3	16	0.0001	0.002915	60,138.747650
Phytomorphic	8	1	32	0.0001	-0.013411	63,286.499764
Phytomorphic	9	2	32	0.0001	-0.015251	70,737.852256
Phytomorphic	9	3	32	0.0001	-0.000967	56,768.935233
Phytomorphic	10	1	1	0.0005	0.804016	55,976.481544
Phytomorphic	7	2	1	0.0005	0.716128	65,407.319922
Phytomorphic	14	3	1	0.0005	0.499655	73,303.235787
Phytomorphic	12	1	16	0.0005	0.106576	60,629.443956
Phytomorphic	7	2	16	0.0005	0.034113	67,574.950684
Phytomorphic	8	3	16	0.0005	0.237192	77,874.719613
Phytomorphic	10	1	32	0.0005	0.030506	69,112.485047
Phytomorphic	11	2	32	0.0005	0.085413	72,511.307308
Phytomorphic	6	3	32	0.0005	0.010676	59,031.713519
Phytomorphic	8	1	1	0.001	0.668056	73,252.692370
Phytomorphic	9	2	1	0.001	0.003045	56,544.158173
Phytomorphic	8	3	1	0.001	0.862359	65,569.059865
Phytomorphic	14	1	16	0.001	-0.107516	62,670.793582
Phytomorphic	12	2	16	0.001	0.644079	67,031.537624
Phytomorphic	15	3	16	0.001	0.682659	56,579.568896
Phytomorphic	15	1	32	0.001	0.005274	64,857.537943
Phytomorphic	8	2	32	0.001	0.298836	57,604.722181
Phytomorphic	6	3	32	0.001	0.045991	56,585.934407

20k Training Dataset

Table 12: Phase 1 complete hyperparameter sweep on the 20k training dataset. Each row reports one replicate.

Model	Epochs Trained	Rep	BS	LR	Test R^2	Verified vlab cost
MLP	7	1	1	0.0001	0.969239	63,893.391897
MLP	11	2	1	0.0001	0.975570	61,298.603880
MLP	7	3	1	0.0001	0.969968	63,451.465023

Model	Epochs Trained	Rep	BS	LR	Test R^2	Verified vlab cost
MLP	9	1	16	0.0001	0.966035	63,929.076851
MLP	7	2	16	0.0001	0.959460	63,740.631090
MLP	8	3	16	0.0001	0.963852	63,741.118260
MLP	6	1	32	0.0001	0.919759	64,179.442000
MLP	7	2	32	0.0001	0.951512	63,496.826401
MLP	8	3	32	0.0001	0.958564	63,669.434373
MLP	7	1	1	0.0005	0.974392	63,705.411285
MLP	19	2	1	0.0005	0.989093	54,265.028013
MLP	9	3	1	0.0005	0.982687	54,225.112117
MLP	6	1	16	0.0005	0.961505	63,488.851906
MLP	15	2	16	0.0005	0.984902	61,814.246136
MLP	9	3	16	0.0005	0.976663	63,593.797150
MLP	8	1	32	0.0005	0.968909	63,546.586842
MLP	15	2	32	0.0005	0.981736	59,814.415307
MLP	9	3	32	0.0005	0.973271	63,593.797150
MLP	14	1	1	0.001	0.986584	54,277.901513
MLP	9	2	1	0.001	0.982889	63,483.027198
MLP	12	3	1	0.001	0.985956	54,220.710235
MLP	7	1	16	0.001	0.972954	63,433.117643
MLP	12	2	16	0.001	0.986257	62,545.949818
MLP	9	3	16	0.001	0.978191	63,622.041886
MLP	7	1	32	0.001	0.969019	64,192.953834
MLP	8	2	32	0.001	0.974093	63,547.250015
MLP	10	3	32	0.001	0.979864	63,671.499593
Phytomorphic	10	1	1	0.0001	0.963213	59,268.174112
Phytomorphic	8	2	1	0.0001	0.849280	63,885.446205
Phytomorphic	13	3	1	0.0001	0.656611	61,941.230823
Phytomorphic	14	1	16	0.0001	0.877817	59,345.810039
Phytomorphic	13	2	16	0.0001	0.761180	62,782.765772
Phytomorphic	10	3	16	0.0001	0.823801	58,617.164991
Phytomorphic	10	1	32	0.0001	-0.783068	59,699.384226
Phytomorphic	19	2	32	0.0001	0.952474	60,305.179296
Phytomorphic	14	3	32	0.0001	0.911901	61,564.663632
Phytomorphic	9	1	1	0.0005	0.953218	61,593.874317
Phytomorphic	12	2	1	0.0005	0.732364	59,800.848881
Phytomorphic	6	3	1	0.0005	0.933807	59,537.538399
Phytomorphic	21	1	16	0.0005	0.973960	58,880.207026
Phytomorphic	10	2	16	0.0005	0.922590	59,619.387575
Phytomorphic	11	3	16	0.0005	0.957146	59,538.887868
Phytomorphic	13	1	32	0.0005	0.846239	63,416.751719
Phytomorphic	7	2	32	0.0005	0.495309	63,962.323973
Phytomorphic	7	3	32	0.0005	0.760159	64,015.651581
Phytomorphic	12	1	1	0.001	0.933966	63,489.878856
Phytomorphic	12	2	1	0.001	0.279668	61,828.767877
Phytomorphic	7	3	1	0.001	0.929051	59,586.330836
Phytomorphic	11	1	16	0.001	0.970107	59,327.961720
Phytomorphic	9	2	16	0.001	0.582294	63,828.245263
Phytomorphic	11	3	16	0.001	0.958784	59,696.294963
Phytomorphic	22	1	32	0.001	0.977682	59,429.695700
Phytomorphic	10	2	32	0.001	0.954725	63,461.940652
Phytomorphic	8	3	32	0.001	0.487875	59,640.356680

Table 13 Phase 2 convergence results for the 50-epoch, patience-10 condition. Data fractions are relative to the 50k-sample training set. Values report mean ± standard deviation over two replicates.

Model	Training Fraction	Samples	Epochs Trained	Test R^2	Verified vlab cost
MLP	25%	12.5k	25.5 ± 0.7	0.9920 ± 0.0011	$71,815 \pm 187$
MLP	50%	25k	22.5 ± 6.4	0.9938 ± 0.0012	$71,360 \pm 189$
MLP	100%	50k	18.5 ± 3.5	0.9948 ± 0.0006	$71,912 \pm 75$
Phytomorphic	25%	12.5k	18.0 ± 7.1	0.9640 ± 0.0036	$73,429 \pm 736$
Phytomorphic	50%	25k	13.0 ± 2.8	0.9697 ± 0.0050	$73,062 \pm 1,254$
Phytomorphic	100%	50k	16.0 ± 2.8	0.9888 ± 0.0000	$65,776 \pm 6,263$

Table 14 Complete Phase 2 convergence sweep across model, data fraction, epoch budget, and replicate. Fractions are relative to the 50k-sample training set.

Model	Fraction	Max Epochs	Patience	Rep.	Epochs	Test R^2	Best vlab cost
MLP	25%	25	5	1	12	0.988202	71,986
MLP	25%	25	5	2	13	0.987067	71,976
MLP	25%	50	10	1	25	0.992768	71,947
MLP	25%	50	10	2	26	0.991218	71,682
MLP	50%	25	5	1	8	0.987031	71,973
MLP	50%	25	5	2	13	0.991479	71,226
MLP	50%	50	10	1	27	0.994580	71,493
MLP	50%	50	10	2	18	0.992934	71,226
MLP	100%	25	5	1	16	0.994539	71,859
MLP	100%	25	5	2	11	0.994310	71,966
MLP	100%	50	10	1	21	0.995251	71,859
MLP	100%	50	10	2	16	0.994387	71,966
Phytomorphic	25%	25	5	1	8	0.957958	73,950
Phytomorphic	25%	25	5	2	8	0.945989	74,491
Phytomorphic	25%	50	10	1	13	0.961458	73,950
Phytomorphic	25%	50	10	2	23	0.966600	72,908
Phytomorphic	50%	25	5	1	10	0.966204	73,949
Phytomorphic	50%	25	5	2	6	0.971901	72,175
Phytomorphic	50%	50	10	1	15	0.966204	73,949
Phytomorphic	50%	50	10	2	11	0.973270	72,175
Phytomorphic	100%	25	5	1	13	0.988324	70,205
Phytomorphic	100%	25	5	2	9	0.984026	61,348
Phytomorphic	100%	50	10	1	18	0.988792	70,205
Phytomorphic	100%	50	10	2	14	0.988769	61,348

D.2. Phase 2: Convergence Analysis

Using the hyperparameters selected in Phase 1, we evaluated surrogate convergence across three training set fractions: 25%, 50%, and 100% of the 50k-sample training set, corresponding to 12.5k, 25k, and 50k training samples. Each model was trained under two epoch-budget settings: 25 epochs with patience 5 and 50 epochs with patience 10. Table 13 summarizes the longer-budget condition. Based on these results, Phase 3 used the full training set with 50 epochs and patience 10.

D.3. Phase 3: Final Hyperparameter Tuning

Using the full 50k-sample training set selected in Phase 2, we performed a final hyperparameter sweep with a 50-epoch maximum and patience of 10, and identified learning rates and batch sizes for the main experiment. Table 15 reports the best configuration for each model ranked by mean verified vlab cost. Stage 3 also used the reduced optimization budget (3 restarts, 250 steps per restart), so verified vlab costs are used for final preliminary selection within this stage.

Table 15 Phase 3: Best final hyperparameter configurations ranked by mean verified vlab cost. Values report mean ± standard deviation over two replicates.

Model	LR	BS	Epochs Trained	Test R^2	Verified vlab cost
MLP	1e-3	8	23.0 ± 2.8	0.9969 ± 0.0003	$69,445 \pm 2,031$
Phytomorphic	5e-4	8	28.5 ± 14.8	0.9910 ± 0.0046	$56,875 \pm 1,559$

Table 16 Complete Phase 3 final hyperparameter sweep on the full 50k-sample training set. Each row reports one replicate.

Model	LR	BS	Rep.	Epochs Trained	Test R^2	Verified vlab cost
MLP	5e-4	1	1	25	0.997060	69,558.848020
MLP	5e-4	1	2	16	0.996538	71,955.249786
MLP	5e-4	4	1	25	0.997450	71,078.698626
MLP	5e-4	4	2	21	0.997275	70,742.393446
MLP	5e-4	8	1	21	0.997543	70,895.151948
MLP	5e-4	8	2	22	0.997287	70,625.176535
MLP	1e-3	1	1	21	0.995251	71,859.224137
MLP	1e-3	1	2	16	0.994387	71,965.574951
MLP	1e-3	4	1	25	0.997107	69,994.745122
MLP	1e-3	4	2	16	0.995607	71,719.772220
MLP	1e-3	8	1	21	0.996750	70,881.203666
MLP	1e-3	8	2	25	0.997146	68,008.444198
Phytomorphic	5e-4	8	1	39	0.994225	55,772.757113
Phytomorphic	5e-4	8	2	18	0.987750	57,978.055187
Phytomorphic	5e-4	16	1	18	0.988792	70,204.697441
Phytomorphic	5e-4	16	2	14	0.988769	61,348.087679
Phytomorphic	5e-4	32	1	13	0.973073	65,680.328739
Phytomorphic	5e-4	32	2	13	0.981243	57,523.029655
Phytomorphic	1e-3	8	1	25	0.993260	62,945.854683
Phytomorphic	1e-3	8	2	18	0.991515	62,921.038768
Phytomorphic	1e-3	16	1	37	0.989662	60,697.213188
Phytomorphic	1e-3	16	2	11	0.986938	70,139.829492
Phytomorphic	1e-3	32	1	12	0.988637	63,137.399426
Phytomorphic	1e-3	32	2	15	0.988817	69,631.882319

D.4. Phase 4: Phytomorphic Ablation Tests

Using the selected phytomorphic configuration from Phase 3, we evaluated four ablated variants against the full phytomorphic model: a hollow variant, a no-aggregator variant, a no-encoder variant, and a no-scaler variant. Table 17 summarizes the ablation results. Although the hollow and no-aggregator variants produced low verified vlab costs in this single-target ablation, optimization-log audits showed degenerate flat-objective behaviour in these variants, with surrogate-predicted costs collapsing to nearly constant values across optimization restarts in affected runs (Section 5.4). We therefore treated these low costs as unstable ablation outcomes rather than reliable evidence of surrogate performance, and used the full phytomorphic surrogate in the final multi-plant experiments. Unlike stages 1-3, stage 4 used the full optimization budget (10 restarts, 1,000 steps per restart), so its verified vlab costs are directly comparable in optimization depth to the final all-plant protocol. We note that the Phase 4 ablation results and the `Plant_023-1` row of Table 3 are independent stochastic replicates of the same full phytomorphic configuration, not repeated reports of a single run. Their difference ($42{,}188 \pm 242$ vs. $32{,}850 \pm 720$) reflects run-to-run variance.

Table 17 Phase 4 ablation results for the phytomorphic surrogate. Values report mean ± standard deviation over two replicates.

Variant	Test R^2	Verified vlab cost
Full phytomorphic	0.9919 ± 0.0010	$42,188 \pm 242$
Hollow	$-1.0\times10^{-6} \pm 0.0$	$39,993 \pm 2,742$
No aggregator	0.4912 ± 0.1072	$40,599 \pm 1,761$
No encoder	0.9912 ± 0.0003	$48,550 \pm 197$
No scaler	-31.3769 ± 45.7566	$40,705 \pm 1,775$

Table 18 Phase 4 complete ablation results for the phytomorphic surrogate. Each row reports one replicate.

Variant	Rep	Test R^2	Predicted vlab cost	Verified vlab cost
Hollow	1	−0.000001	43,576.61	41,939.07
Hollow	2	−0.000001	43,576.61	38,046.54
Full phytomorphic	1	0.992652	34,160.18	42,359.31
Full phytomorphic	2	0.991221	32,166.94	42,016.52
No aggregator	1	0.415324	43,576.61	41,844.89
No aggregator	2	0.566989	43,576.61	39,354.75
No encoder	1	0.991468	49,353.00	48,688.80
No encoder	2	0.991010	77,308.34	48,410.29
No scaler	1	0.977908	36,038.14	41,960.03
No scaler	2	−63.731712	0.00	39,450.73

E. Per-Replicate All-Plant Results

Table 19: Per-replicate results for all 13 target plants. Test R^2 is the held-out coefficient of determination. Predicted Cost is the surrogate's estimated vlab cost at the optimized parameters. Verified Cost is the simulator-evaluated vlab cost at those parameters. Lower verified cost is better. The MLP surrogate is denoted `mlp` and the phytomorphic surrogate `phytomorph`.

Target plant	Model	Rep	Test R^2	Predicted Cost	Verified Cost
`Plant_001-9`	mlp	1	0.9961	28,507	29,195
	mlp	2	0.9965	31,915	48,309
	phytomorph	1	0.9835	28,731	29,202
	phytomorph	2	0.9806	29,253	50,337
`Plant_006-25`	mlp	1	0.9947	41,422	54,805
	mlp	2	0.9955	41,304	54,283
	phytomorph	1	0.9929	39,548	39,656
	phytomorph	2	0.9774	40,289	39,539
`Plant_008-19`	mlp	1	0.9961	30,418	53,155
	mlp	2	0.9958	19,861	53,292
	phytomorph	1	0.9920	37,537	55,573
	phytomorph	2	0.9903	39,706	38,428
`Plant_016-20`	mlp	1	0.9965	38,323	49,552
	mlp	2	0.9971	38,455	48,778
	phytomorph	1	0.9938	51,182	46,305
	phytomorph	2	0.9927	49,593	48,742
`Plant_023-1`	mlp	1	0.9965	29,366	43,305
	mlp	2	0.9969	33,423	42,135
	phytomorph	1	0.9919	33,637	33,359
	phytomorph	2	0.9905	33,622	32,340
`Plant_045-1`	mlp	1	0.9981	31,631	43,338
	mlp	2	0.9983	28,323	43,476

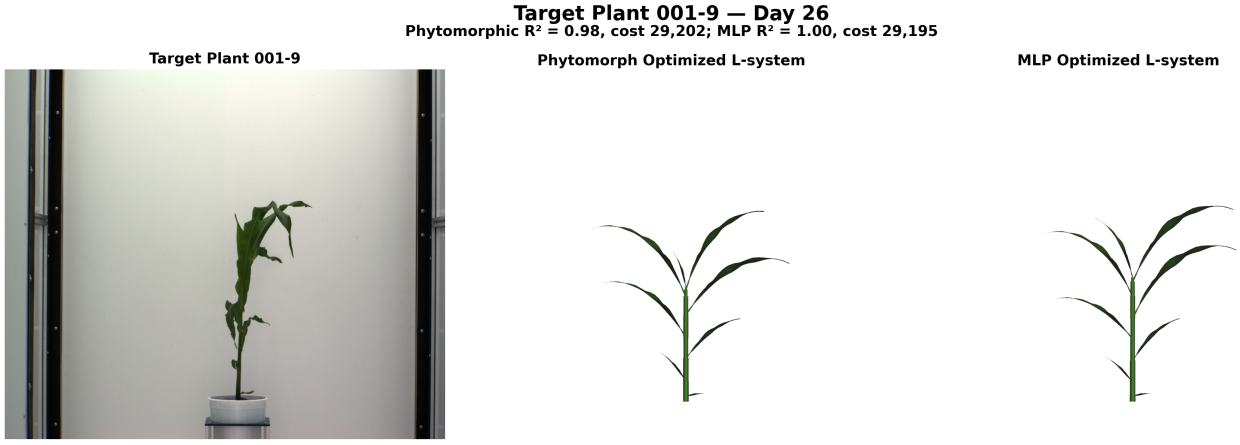
continued on next page

Table 19 – *continued from previous page*

Target plant	Model	Rep	Test R^2	Predicted Cost	Verified Cost
	phytomorph	1	0.9960	31,375	38,442
	phytomorph	2	0.9961	31,456	31,815
Plant_047-25	mlp	1	0.9961	30,668	55,740
	mlp	2	0.9960	33,301	55,563
	phytomorph	1	0.9938	40,876	56,331
	phytomorph	2	0.9845	39,868	51,893
Plant_063-32	mlp	1	0.9926	47,153	52,988
	mlp	2	0.9932	43,437	70,948
	phytomorph	1	0.9874	51,921	55,313
	phytomorph	2	0.9758	51,390	50,828
Plant_070-11	mlp	1	0.9961	40,395	53,199
	mlp	2	0.9973	44,648	53,210
	phytomorph	1	0.9860	47,369	53,251
	phytomorph	2	0.9232	39,808	47,346
Plant_071-8	mlp	1	0.9961	41,466	55,798
	mlp	2	0.9967	38,772	55,774
	phytomorph	1	0.9950	46,080	52,124
	phytomorph	2	0.9948	43,378	55,791
Plant_076-24	mlp	1	0.9978	30,195	39,686
	mlp	2	0.9980	28,988	40,192
	phytomorph	1	0.9956	31,821	39,764
	phytomorph	2	0.9937	32,069	39,877
Plant_104-24	mlp	1	0.9976	16,323	35,331
	mlp	2	0.9977	7,003	35,776
	phytomorph	1	−4.9547	71,806	27,808
	phytomorph	2	0.9967	22,282	34,970
Plant_191-28	mlp	1	0.9923	41,774	54,112
	mlp	2	0.9914	36,852	48,538
	phytomorph	1	0.9731	34,806	36,080
	phytomorph	2	0.9899	35,057	35,605

F. All-Plant Qualitative Comparisons

Figure 12: Day-26 target plants and optimized Phytomorph and MLP outputs for all 13 UNL-CPPD plants.

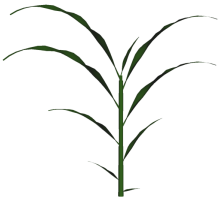


Target Plant 006-25 — Day 26
Phytomorphic R^2 = 0.98, cost 39,539; MLP R^2 = 1.00, cost 54,283

Target Plant 006-25



Phytomorph Optimized L-system

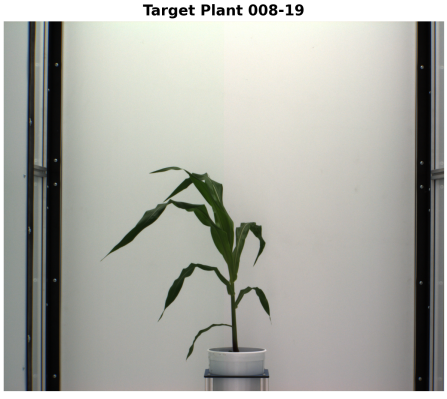


MLP Optimized L-system



Target Plant 008-19 — Day 26
Phytomorphic R^2 = 0.99, cost 38,428; MLP R^2 = 1.00, cost 53,155

Target Plant 008-19



Phytomorph Optimized L-system

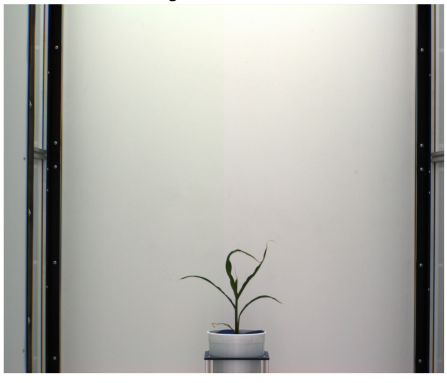


MLP Optimized L-system



Target Plant 016-20 — Day 26
Phytomorphic R^2 = 0.99, cost 46,305; MLP R^2 = 1.00, cost 48,778

Target Plant 016-20



Phytomorph Optimized L-system

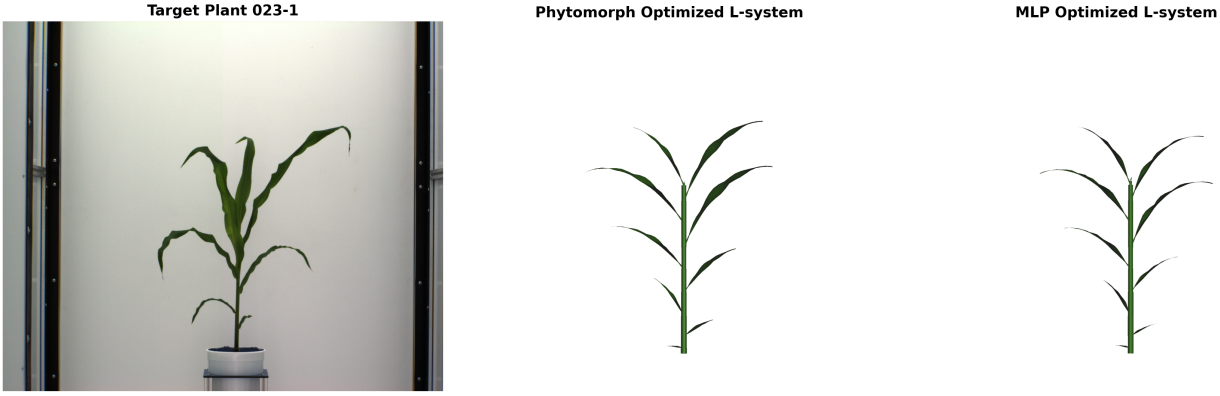


MLP Optimized L-system



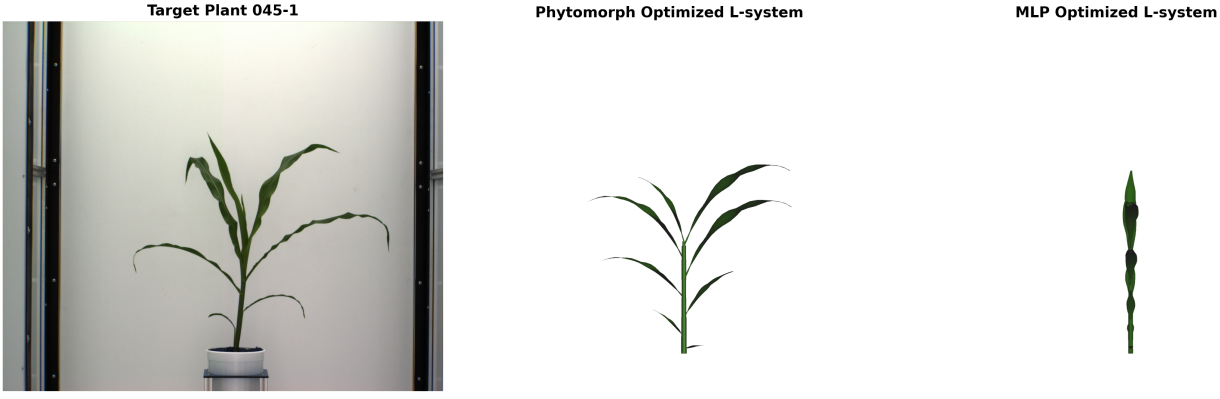
Target Plant 023-1 — Day 26
Phytomorphic R^2 = 0.99, cost 32,340; MLP R^2 = 0.94, cost 32,332

Target Plant 023-1 | Phytomorph Optimized L-system | MLP Optimized L-system



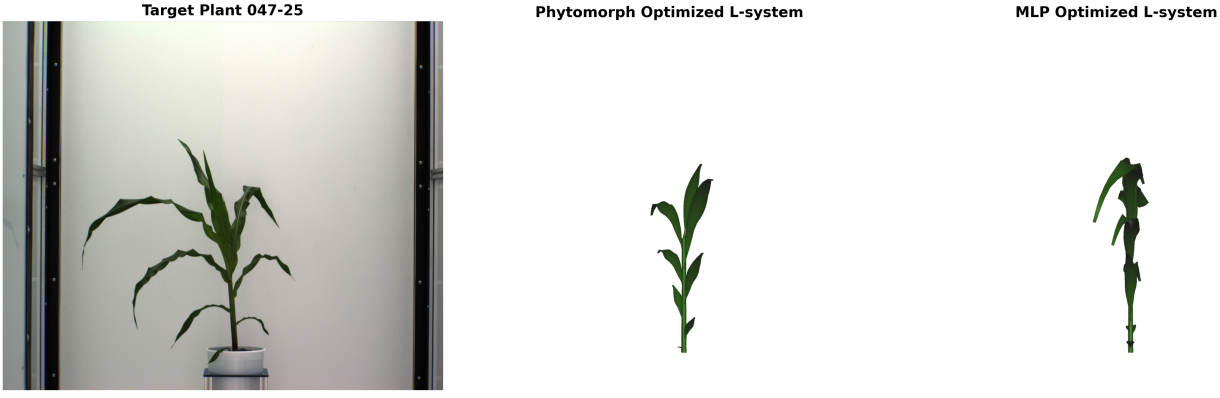
Target Plant 045-1 — Day 26
Phytomorphic R^2 = 1.00, cost 31,815; MLP R^2 = 1.00, cost 43,338

Target Plant 045-1 | Phytomorph Optimized L-system | MLP Optimized L-system



Target Plant 047-25 — Day 26
Phytomorphic R^2 = 0.98, cost 51,893; MLP R^2 = 1.00, cost 55,563

Target Plant 047-25 | Phytomorph Optimized L-system | MLP Optimized L-system

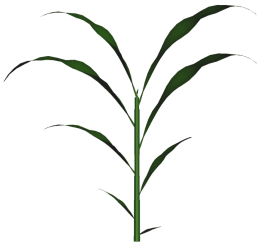


Target Plant 063-32 — Day 26
Phytomorphic R^2 = 0.98, cost 50,828; MLP R^2 = 0.99, cost 52,988

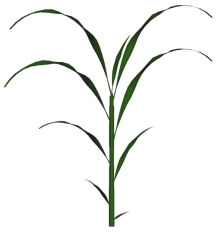
Target Plant 063-32



Phytomorph Optimized L-system

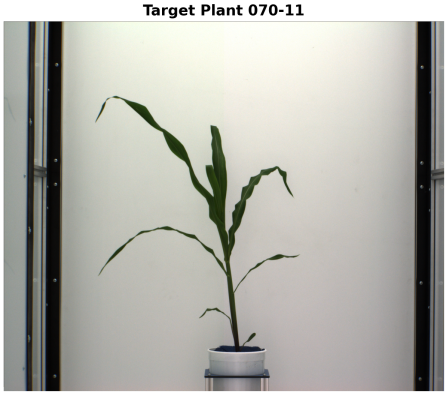


MLP Optimized L-system



Target Plant 070-11 — Day 26
Phytomorphic R^2 = 0.92, cost 47,346; MLP R^2 = 1.00, cost 53,199

Target Plant 070-11



Phytomorph Optimized L-system



MLP Optimized L-system

Target Plant 071-8 — Day 26
Phytomorphic R^2 = 0.99, cost 52,124; MLP R^2 = 1.00, cost 55,774

Target Plant 071-8



Phytomorph Optimized L-system



MLP Optimized L-system



Target Plant 076-24 — Day 26
Phytomorphic R^2 = 1.00, cost 39,764; MLP R^2 = 1.00, cost 39,686

Target Plant 076-24



Phytomorph Optimized L-system



MLP Optimized L-system



Target Plant 104-24 — Day 26
Phytomorphic R^2 = -4.95, cost 27,808; MLP R^2 = 1.00, cost 35,331

Target Plant 104-24



Phytomorph Optimized L-system

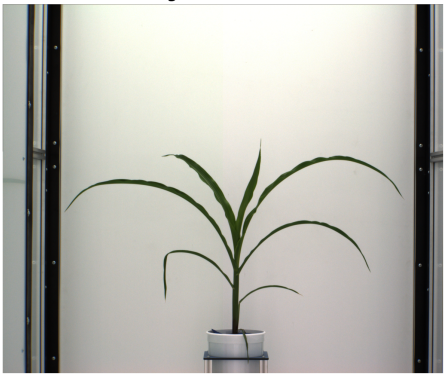


MLP Optimized L-system

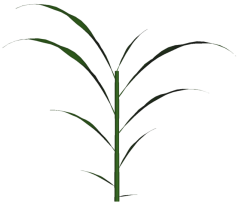


Target Plant 191-28 — Day 26
Phytomorphic R^2 = 0.99, cost 35,605; MLP R^2 = 0.88, cost 38,777

Target Plant 191-28



Phytomorph Optimized L-system



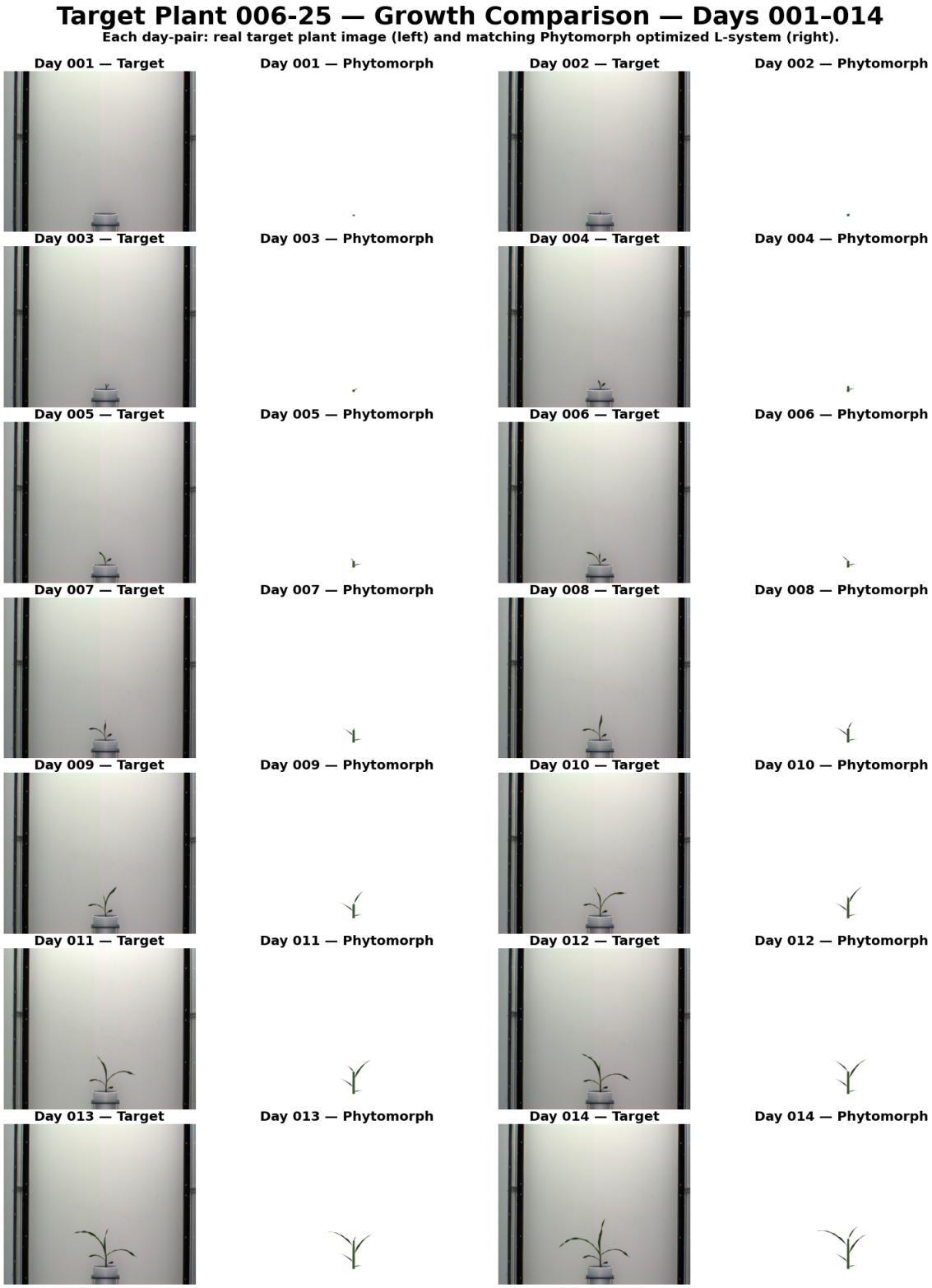
MLP Optimized L-system



G. Plant 006-25 Comparison Over Days

The simulated sequence uses the Phytomorph parameter set with the lowest verified vlab cost across the two Plant 006-25 replicates.

Figure 13: Daily comparison of target Plant 006-25 and its optimized L-system simulation across source-image Days 001–027, with 26 target observations on Days 002–027.





H. Runtime and Compute Summary

All runtimes reported here were extracted from pipeline log files, including `generation_log.txt`, `metrics.json`, `summary_results.csv`, and optimizer output timestamps. All jobs were run in Docker on CPU (no GPU acceleration). Table 20 reports detailed runtime accounting for the 13-plant final evaluation scope, including dataset generation, surrogate training, and optimization.

Table 20 Runtime summary for dataset generation, surrogate training, and surrogate-guided optimization (13-plant final evaluation scope).

Stage	Statistic	Value (seconds)	Value (DD:HH:MM:SS)
Simulation + matching (per sample)	Mean	0.1214	00:00:00:00.1214
Dataset generation (per plant, 85,000 samples)	Mean	10,322.22	00:02:52:02
Dataset generation (13 plants, 1,105,000 samples)	Total	134,188.84	01:13:16:29
MLP training (per run)	Mean	446.08	00:00:07:26
MLP training (per run)	Min / Max	423 / 503	00:00:07:03 / 00:00:08:23
Phytomorph training (per run)	Mean	48,584.84	00:13:29:45
Phytomorph training (per run)	Min / Max	39,725 / 55,847	00:11:02:05 / 00:15:30:47
Surrogate training (all runs, sequential)	Total	1,274,803.81	14:18:06:44
Surrogate training (all runs, wall-clock)	Total	654,914.83	07:13:55:15
MLP optimization (per run)	Mean	21.991	00:00:00:21.991
Phytomorph optimization (per run)	Mean	9,099.239	00:02:31:39.239
All optimization runs	Sum of per-run durations	237,151.988	02:17:52:31.988

Experiments used parallel workers, so summed per-run durations are not equivalent to elapsed wall-clock time. We report per-run and stage-level timing measures separately and do not present component sums as end-to-end wall-clock duration.

Optimization durations exclude vlab/LPFG verification. Per-run durations are summarized separately from their sum across runs; that sum is not an elapsed wall-clock duration.

I. Correlation Statistics

Table 21 Correlation between test R^2 and verified vlab cost under different aggregations. Spearman ρ is the primary statistic; Pearson r is included for reference. The pooled raw-replicate correlation is significant only because of a single collapsed `Plant_104-24` replicate, and falls to near zero when that replicate is removed.

Aggregation	n	Pearson r	Pearson p	Spearman ρ	Spearman p
All means (both models)	26	0.350	0.079	0.123	0.548
MLP means	13	−0.610	0.027	−0.632	0.021
Phytomorphic means	13	0.424	0.149	0.126	0.681
All raw replicates	52	0.276	0.048	0.024	0.867
All raw replicates, minus collapse	51	0.024	0.867	−0.035	0.809

J. Codebase Organization and Reproduction Workflow

The experiment code is located in `vlab/oofs/ext`. Docker and Docker Compose are environment requirements. From the repository directory, run `./start.sh` to create the container, then run `. ./connect.sh` to attach to it. Once attached, run `. ./setup.sh` to install VLAB, and other necessary configurations. Press Enter when prompted to finalize the default configuration.

From `vlab/oofs/ext` (your terminal session should be redirected automatically from `setup.sh`), run `./run_experiment.sh` to execute the numbered stages in order. An optional experiment tag can be supplied as the first argument (e.g., `./run_experiment.sh trial_01`), and if it is omitted, the script generates a tag from the current date and time. Experiment settings are primarily defined inline in the numbered bash shell scripts rather than in a separate configuration.

In this paper, the phytomorphic surrogate is the standard Sinkhorn model implemented in `model_sinkhorn.py`; Stage 05 trains it alongside the MLP baseline.

For Stages 01–03, `tune_models.py` performs optimization and LPFG verification during each tuning configuration. Verified vlab cost is used for early stopping: training stops when the verified cost fails to improve for the configured patience or when the epoch limit is reached. Stage 04 uses `--train-only` for ablation training, with optimization and verification run afterward.

Tuning and convergence outputs are stored under `Hyperparameter Tuning/`, `Convergence Tests/`, and `Final HP Tune/`; ablation results under `Ablation Tests/`; main training outputs under `Training Data/`; and optimization results and experiment summaries under `Optimizer Data/`. Stage logs and state are stored in `Research Pipeline/`.

Table 22 Numbered experiment stages and core implementation files.

Experiment stages

Stage and script	Purpose and execution path
`00_dataset.sh`	Defines the target plants and train/validation/test sizes, then calls `generate_dataset.py` to create any missing datasets. Outputs are stored under `Datasets/`.
`01_hp_tuning.sh`	Runs learning-rate and batch-size searches for the MLP and Sinkhorn models on small and medium datasets using `tune_models.py`.
`02_convergence.sh`	Tests training-set fractions and epoch/patience settings using `tune_models.py`; `convergence_test.py` summarizes the convergence results.
`03_final_hp_tuning.sh`	Runs a narrower hyperparameter search on the large dataset using `tune_models.py`.
`04_ablation.sh`	Trains Sinkhorn variants with selected components removed using `--train-only`, then optimizes and verifies them with `optimizer_script.py`.
`05_experiment.sh`	Trains the baseline MLP and standard Sinkhorn models across the listed plants using `train_models.py --skip-evaluation`, then optimizes and verifies their outputs with `optimizer_script.py`. Stage 5 selects these two model families; although `model_hungarian.py` is present in the codebase, it is not selected by this stage.

Core implementation files

Files	Role in the workflow
`generate_dataset.py`	Samples the 13 L-system parameters, generates synthetic plants, measures structural cost against a real-plant reference, and writes CSV splits and saved structures.
`utils_nn.py`	Connects the Python workflow to plant-image parsing, geometry comparison, parameter-file writing, and LPFG execution. Reference images are stored in `Real Plants/`. It computes the daily Hungarian assignment using `Munkres.compute`.
`lsystem/lsystem.l` and supporting files	`lsystem.l` defines the procedural plant. `functions.fset`, `functions.tset`, `contours.cset`, `materials.mat`, and `view.v` provide its functions, contours, materials, and view settings.
`lsystem/project.cpp`	Converts simulator-produced 3D positions into the point coordinates used by the comparison code; the scripts compile or use the resulting project executable.
`model_base.py`, `model_mlp.py`, `model_sinkhorn.py`	`model_base.py` provides shared dataset handling, structure generation, matching and loss utilities, and temporal aggregation. The other files implement the MLP baseline and Sinkhorn model, respectively.
`train_models.py`, `tune_models.py`, `optimizer_script.py`	Handle model training and validation, hyperparameter/convergence runs, and surrogate-guided optimization with LPFG verification, respectively.
`convergence_test.py`, `hp_tuning_evaluation.py`, `evaluate_models.py`	Provide convergence summaries and downstream tuning or experiment analyses. These reporting scripts are separate from the numbered end-to-end runner. The defaults in `evaluate_models.py` point to a historical experiment and should be aligned with the experiment tag and models being analyzed.
`lsystem/anim.a`, `generate_animated_lsystem.py`	Support optional animated rendering and visualization; they are not central to the numbered training pipeline.