Supplementary Material 2: Detailed Python Workflow for Data Preprocessing, Model Training, and Evaluation

Table S1. Computational Environment and Data Preprocessing Workflow

Step/Component	Specification/Action	Notes (Applies to All Models)
Computational Environment	Python 3.12.6 (Jupyter Notebook), Windows 11, Intel i7-1260P CPU, 16 GB RAM	Package versions in Supplementary Materials
Key Libraries Used	pandas, numpy, scikit-learn (1.4.2), imbalanced-learn, matplotlib, seaborn, tensorflow.keras (2.16.1), xgboost, SHAP (0.45.0)	
Raw Data Import	File: growth_physo_anatomy_cleaned2.xlsx	Data and code provided as Supplementary Files
Data Cleaning	Dropped categorical columns (cultivar_en, water_cond)	Retained only relevant numeric features
Target Variable Encoding	LabelEncoder (scikit-learn): classes = moderate, susceptible, tolerant	Target encoded as integers 0, 1, 2
Feature Selection	All physiological/anatomical features used; for hybrid, top 10 by SHAP	See details in respective model tables
Class Balancing	SMOTE (random_state=42) applied before train/test split	Ensures equal class representation
Train/Test Split	80% training / 20% test, stratified, random_state=42 and 5 fold cross validation	Class proportions preserved in both sets and folds
Feature Scaling	StandardScaler for MLP and SVM (fit on train set only); RF/XGB use raw values	Scaling method chosen by model type
Random Seed Initialization	42 applied to NumPy, scikit-learn, TensorFlow/Keras, SMOTE, and splits	Ensures analyses are exactly reproducible
Software Reproducibility	Jupyter notebooks, and environment files provided	

Table S2. Random Forest (RF) Model Implementation Details

Step/Component	Specification/Value	Notes
Software Environment	Python (Jupyter Notebook), pandas, numpy, scikit-learn (1.4.2), imblearn, matplotlib, seaborn	Version details in supplement
Data Preprocessing	Dropped categorical columns, target label-encoded	Only relevant features kept
Class Balancing	SMOTE, random_state=42	Applied before train-test split
Train/Test Split	80% train / 20% test, stratified, random_state=42	Ensures class proportions are preserved
Feature Scaling	Not required for Random Forest	Trees are insensitive to scaling
Feature Selection	All features except target used	Feature importance later via Gini index
Model Architecture	RandomForestClassifier: grid search over n_estimators: [100, 200, 300] max_depth: [10, 20, 30] min_samples_split: [2, 5, 10] min_samples_leaf: [1, 2, 4] max_features: ['sqrt', 'log2']	Hyperparameters chosen by 5-fold CV
Model Training	Best model trained on full training set	random_state=42 for reproducibility
Model Validation	Evaluated on held-out test set (20%) and 5-fold cross-validation	Unseen data for fair assessment and folds
Evaluation Metrics	Accuracy, macro precision, macro recall, macro F1-score, balanced accuracy, Matthews correlation coefficient (MCC), Cohen's kappa, log loss, macro ROC-AUC (one-vs-rest), hamming loss, confusion matrix, classification report	All reported in results and supplement
Interpretability	Feature importance from RF (Gini index); visualized as barplot	Supports trait-based biological insights
Reproducibility	random_state=42 everywhere; code and data provided	Enables exact re-running of analysis
Visualization	Confusion matrix, ROC curves, feature importance plot	Provided in supplement and notebook

Appendix Table S3. Multi-Layer Perceptron (MLP) Model Implementation Details

Step/Component	Specification/Value	Notes
Software Environment	Python (Jupyter Notebook), pandas, numpy, scikit-learn (1.4.2), tensorflow.keras (2.16.1), imblearn, matplotlib, seaborn	Version details in supplement
Data Preprocessing	Dropped categorical columns, target label-encoded	Consistent with RF workflow
Class Balancing	SMOTE, random_state=42	Applied before train-test split
Train/Test Split	80% train / 20% test, stratified, random_state=42	As in RF
Feature Scaling	StandardScaler (fit on train, apply to test)	Required for neural networks
Feature Selection	All features except target used	SHAP analysis for interpretability
Model Architecture	Keras Sequential: 3 hidden layers (512, 256, 128, LeakyReLU α=0.1) BatchNorm and Dropout (0.3) after each Softmax output for 3 classes	Designed for multiclass
Model Training	Adam optimizer (lr=0.001), batch size 16, max 200 epochs Early stopping (patience=10), reduce LR on plateau	random_state=42 set for reproducibility
Model Validation	Best model chosen by validation loss on hold-out test set and 5-fold cross-validation	Early stopping to prevent overfitting
Evaluation Metrics	(Same as RF): Accuracy, macro precision, macro recall, macro F1-score, balanced accuracy, MCC, Cohen's kappa, log loss, macro ROC-AUC (one-vs-rest), hamming loss, confusion matrix, classification report	All reported in results and supplement
Interpretability	SHAP DeepExplainer for feature importance	Supports understanding of complex model
Reproducibility	random_state=42 everywhere, tf.random.set_seed(42)	Code and data provided
Visualization	Training curves, confusion matrix, ROC curves, SHAP plots	Provided in supplement and notebook

Table S4. Hybrid (Stacking Ensemble) Model Implementation Details

Step/Component	Specification/Value	Notes
Software Environment	Python (Jupyter Notebook), pandas, numpy, scikit-learn (1.4.2), xgboost, tensorflow.keras (2.16.1), imblearn, matplotlib, seaborn	Version details in supplement

Step/Component	Specification/Value	Notes
Data Preprocessing	Dropped categorical columns, target label-encoded	Same as other models
Class Balancing	SMOTE, random_state=42	Applied before train-test split
Train/Test Split	80% train / 20% test, stratified, random_state=42	Consistent across all models
Feature Scaling	StandardScaler for MLP/SVM; not needed for RF/XGB	Applied to each pipeline as required
Feature Selection	Top 10 features by SHAP value	Used for all base models and meta-learner
Model Architecture	**Base Models:** - Random Forest (n_estimators=100, max_depth=20) - XGBoost (n_estimators=200) - SVM (RBF kernel, C=1.0) - MLP (as above) **Meta-Learner:** Logistic Regression (scikit-learn, default params)	StackingClassifier (scikit-learn) used
Model Training	Base models trained on same train data; meta-learner on out-of-fold base model predictions	Ensures no data leakage
Model Validation	Evaluated on held-out test set (20%) and 5-fold cross-validation	Test set not used in model selection
Evaluation Metrics	(Same as RF/MLP): Accuracy, macro precision, macro recall, macro F1-score, balanced accuracy, MCC, Cohen's kappa, log loss, macro ROC-AUC (one-vs-rest), hamming loss, confusion matrix, classification report	All reported in results and supplement
Interpretability	SHAP summary and interaction plots for meta-learner	Highlights top predictive traits
Reproducibility	random_state=42 everywhere; code and data provided	Fully reproducible pipeline
Visualization	Confusion matrix, ROC curves, SHAP plots, feature rankings	Provided in supplement and notebook