

# Spectral Soil Modeler: Advanced ML Pipeline with Leave-One-Out Cross-Validation

<b>Team Number</b>	27
<b>Contributors:</b>	Anupam Dwivedi (2025201032) Devansh Singh (2025204007) Shobhan Parida (2025201043) Krishna Pokuri (2025202024) Yashwanth B K (2025201028)
<b>Code Repository &amp; GitHub:</b>	Team27_Spectral.Soil.Modeler
<b>Documentation</b>	README.md
<b>Deployment</b>	Streamlit Web Application
<b>Presentation</b>	Soil Modeler Presentation
<b>Demo</b>	Soil Modeler Demo

## Contents

<b>1 Executive Summary</b>	<b>5</b>
1.1 Core Problem . . . . .	5
1.2 Our Solution . . . . .	5
1.3 Key Achievements . . . . .	5
<b>2 System Architecture</b>	<b>6</b>
2.1 High-Level Architecture . . . . .	6
2.2 Pipeline Stages . . . . .	6
2.3 User-Codebase Interaction Map . . . . .	7
<b>3 Machine Learning Models</b>	<b>9</b>
3.1 Model 1: Partial Least Squares Regression (PLSR) . . . . .	9
3.1.1 Overview . . . . .	9
3.1.2 Mathematical Foundation . . . . .	9
3.1.3 Implementation Details . . . . .	9
3.1.4 Hyperparameters . . . . .	9
3.1.5 Advantages . . . . .	9
3.2 Model 2: Gradient Boosting Regression Trees (GBRT) . . . . .	9
3.2.1 Overview . . . . .	9
3.2.2 Mathematical Foundation . . . . .	9
3.2.3 Implementation . . . . .	10
3.2.4 Hyperparameters . . . . .	10
3.2.5 Advantages . . . . .	10
3.3 Model 3: Support Vector Regression (SVR) . . . . .	10
3.3.1 Overview . . . . .	10
3.3.2 Mathematical Foundation . . . . .	11
3.3.3 Hyperparameters . . . . .	11
3.3.4 Advantages . . . . .	11
3.4 Model 4: Kernel Ridge Regression (KRR) . . . . .	11
3.4.1 Overview . . . . .	11
3.4.2 Mathematical Foundation . . . . .	11
3.4.3 Hyperparameters . . . . .	11
3.5 Model 5: Cubist (Ensemble Rule-Based Regression) . . . . .	11
3.5.1 Overview . . . . .	11

3.5.2	Mathematical Foundation . . . . .	11
3.5.3	Implementation . . . . .	12
3.5.4	Hyperparameters . . . . .	12
3.5.5	Advantages . . . . .	12
<b>4</b>	<b>Data Preprocessing</b>	<b>13</b>
4.1	Spectral Transformation Techniques . . . . .	13
4.1.1	Technique 1: Reflectance (Baseline) . . . . .	13
4.1.2	Technique 2: Absorbance . . . . .	13
4.1.3	Technique 3: Continuum Removal . . . . .	13
4.2	Normalization Methods . . . . .	13
4.3	Preprocessing Pipeline . . . . .	14
<b>5</b>	<b>Feature Engineering</b>	<b>15</b>
5.1	Overview . . . . .	15
5.2	Six Feature Engineering Techniques . . . . .	15
5.2.1	1. Spectral Derivatives . . . . .	15
5.2.2	2. Statistical Features . . . . .	15
5.2.3	3. Polynomial Features . . . . .	15
5.2.4	4. Spectral Indices . . . . .	15
5.2.5	5. Principal Component Analysis (PCA) . . . . .	15
5.2.6	6. Wavelet Features . . . . .	16
5.3	LLM Context Integration . . . . .	16
<b>6</b>	<b>Feature Importance Analysis</b>	<b>17</b>
6.1	Overview . . . . .	17
6.2	Permutation Importance Method . . . . .	17
6.3	Implementation . . . . .	17
6.4	Prediction Source Selection . . . . .	18
6.5	Context Integration . . . . .	18
6.6	Use Cases . . . . .	18
<b>7</b>	<b>Evaluation Metrics and Validation Strategy (Phase 3 Enhancement)</b>	<b>19</b>
7.1	Test Set Metrics . . . . .	19
7.2	Leave-One-Out Cross-Validation Metrics (NEW - Phase 3) . . . . .	19
7.3	Intelligent Metric Selection (NEW - Phase 3) . . . . .	19
7.4	Configurable Cross-Validation Strategy . . . . .	19
<b>8</b>	<b>Performance Optimizations (Phase 3)</b>	<b>20</b>
8.1	Parallel Training with ThreadPoolExecutor . . . . .	20
8.2	GBRT Training Time Reduction (40% Faster) . . . . .	20
<b>9</b>	<b>Hyperparameter Tuning Strategy</b>	<b>21</b>
9.1	Overview . . . . .	21
9.2	GridSearchCV vs RandomizedSearchCV . . . . .	21
9.3	Search Space Reduction . . . . .	21
9.4	Cross-Validation Process . . . . .	21
9.5	Search Method Details . . . . .	21
9.5.1	GridSearchCV (Exhaustive Search) . . . . .	21
9.5.2	RandomizedSearchCV (Stochastic Search) . . . . .	22
9.6	Implementation . . . . .	22
<b>10</b>	<b>User Manual</b>	<b>23</b>

10.1 User Personas & Workflows . . . . .	23
10.1.1 Three Key User Personas . . . . .	23
10.1.2 User Journey Workflows . . . . .	23
10.2 Installation Guide . . . . .	23
10.2.1 System Requirements . . . . .	23
10.2.2 Step-by-Step Installation . . . . .	23
10.3 Training Mode - Complete Workflow . . . . .	24
10.3.1 Step 1: Load Data . . . . .	24
10.3.2 Step 2: Select Target Column . . . . .	25
10.3.3 Step 2.1: Data Analytics (Optional) . . . . .	25
10.3.4 Step 2.2: AI Data Insights (Optional) . . . . .	26
10.3.5 Step 2.3: Feature Engineering (Optional) . . . . .	27
10.3.6 Step 3: Training Configuration . . . . .	28
10.3.7 Step 4: Start Training . . . . .	28
10.3.8 Step 5: View Results . . . . .	29
10.4 Prediction Mode - Complete Workflow . . . . .	33
10.4.1 Step 1: Load Trained Model . . . . .	33
10.4.2 Step 2: Upload Prediction Data . . . . .	33
10.4.3 Step 3: Preprocessing Configuration . . . . .	33
10.4.4 Step 4: Make Predictions . . . . .	34
10.4.5 Step 5: Export Predictions . . . . .	34
10.5 AI Chat Interface - Advanced Usage . . . . .	35
10.5.1 Accessing AI Chat . . . . .	35
10.5.2 Effective Prompting . . . . .	35
10.6 Troubleshooting . . . . .	36
10.6.1 Common Issues . . . . .	36
<b>11 Evaluation Metrics</b>	<b>37</b>
11.1 R <sup>2</sup> Score (Coefficient of Determination) . . . . .	37
11.2 RMSE (Root Mean Squared Error) . . . . .	37
11.3 MAE (Mean Absolute Error) . . . . .	37
11.4 RPD (Residual Prediction Deviation) . . . . .	37
<b>12 AI Integration Architecture</b>	<b>38</b>
12.1 Overview . . . . .	38
12.2 LLM Context Building . . . . .	38
12.2.1 Context Components . . . . .	38
12.2.2 Context Flow Architecture . . . . .	38
12.3 Data Consistency Guarantee . . . . .	39
12.4 Implementation . . . . .	39
12.5 Chat Interface . . . . .	39
<b>13 Software &amp; Hardware</b>	<b>40</b>
13.1 Technology Stack . . . . .	40
13.2 Hardware Requirements . . . . .	40
<b>14 Project Contribution</b>	<b>41</b>
14.1 Code Statistics . . . . .	41
14.2 Team Contributions . . . . .	41
14.3 Development History . . . . .	41
<b>15 References &amp; Resources</b>	<b>42</b>

15.1 Project Links . . . . .	42
15.2 Libraries & Tools . . . . .	42
<b>16 Appendix: Code Examples</b>	<b>43</b>
16.1 Example 1: Basic Training Script . . . . .	43
16.2 Example 2: Hyperparameter Tuning . . . . .	43
16.3 Example 3: Making Predictions . . . . .	44
<b>17 Conclusion</b>	<b>45</b>

# 1 Executive Summary

The Spectral Soil Modeler addresses the critical challenge of efficiently predicting soil properties from spectral reflectance data using machine learning. Traditional laboratory methods are expensive and time-consuming; our solution enables rapid, non-destructive soil analysis through advanced spectral data processing and intelligent model selection.

## 1.1 Core Problem

Soil property prediction from spectral data requires:

- Expensive laboratory testing for each sample
- Expertise in both soil science and machine learning
- Manual model selection and hyperparameter tuning
- Time-intensive validation and comparison

## 1.2 Our Solution

We implement a comprehensive ML pipeline with:

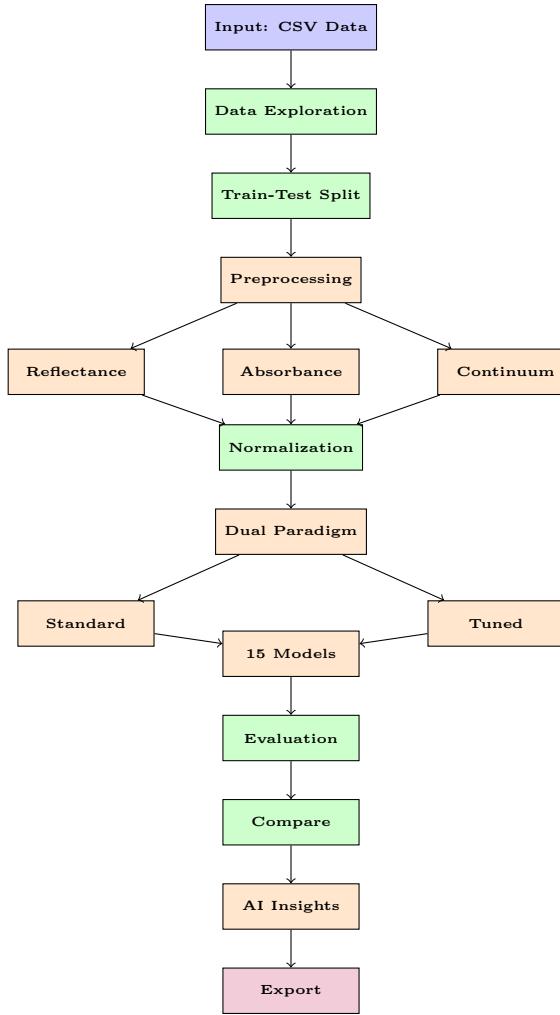
- **15 model-technique combinations:** 5 algorithms  $\times$  3 spectral preprocessing techniques
- **Dual training paradigms:** Standard (fixed parameters) vs Tuned (optimized with same CV strategy)
- **Leave-One-Out Cross-Validation:** LOO CV metrics (LOO\_CV\_Test\_R<sup>2</sup>, LOO\_CV\_Test\_RMSE) for superior held-out validation
- **6 Feature Engineering techniques:** Derivatives, Statistical, Polynomial, Spectral Indices, PCA, Wavelet
- **Feature Importance Analysis:** Permutation-based importance for all 15/30 model combinations
- **Automatic hyperparameter optimization:** 96% GBRT search space reduction (2,160  $\rightarrow$  48 combinations)
- **Configurable CV Strategies:** K-Fold (5 splits) or Leave-One-Out with consistent application across paradigms
- **Intelligent Metric Selection:** Auto-detects LOO CV availability and uses superior metrics throughout interface
- **Parallel training:** ThreadPoolExecutor for 15 model-technique combinations
- **Interactive Streamlit UI:** Real-time training, prediction, and visualization with metric auto-detection
- **AI-powered insights:** LLM-based recommendations with comprehensive context (Gemini/ChatGPT)

## 1.3 Key Achievements

- **Leave-One-Out Cross-Validation:** LOO CV metrics on both training and test sets for superior validation
- **5-fold cross-validation** for robust hyperparameter tuning with configurable CV strategies
- **Comprehensive evaluation metrics:** Test R<sup>2</sup>, LOO CV R<sup>2</sup>, RMSE, MAE, RPD with intelligent fallback selection
- **Feature importance extraction:** Top features for all model-technique combinations
- **Consistent LLM context:** Data analytics + FE statistics + feature importance across all paradigms
- **Automated reporting:** AI-generated insights and comparison reports
- **GBRT Performance Optimization:** 67% hyperparameter reduction (2,160  $\rightarrow$  48 combinations)
- **Parallel training:** ThreadPoolExecutor for 15 model-technique combinations
- **Intelligent Metric Selection:** Automatic LOO CV metrics when available, graceful fallback to Test metrics
- **Model persistence:** Save/load trained models with comprehensive metadata

## 2 System Architecture

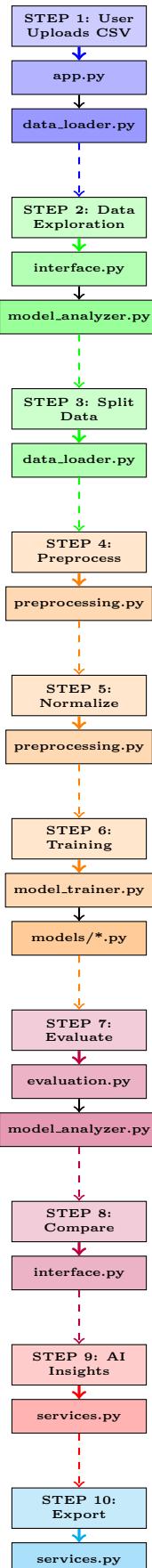
### 2.1 High-Level Architecture



### 2.2 Pipeline Stages

1. **Input:** CSV with spectral bands and target property
2. **Data Exploration:** Statistics, missing values, outlier detection
3. **Train-Test Split:** 80% training / 20% testing
4. **CV Strategy Selection:** K-Fold (5 splits) or Leave-One-Out
5. **Preprocessing:** Three spectral transformations in parallel
6. **Normalization:** StandardScaler, MinMaxScaler, or RobustScaler
7. **Dual Paradigms:** Standard and Tuned models with same CV strategy
8. **15 Models:** 5 algorithms  $\times$  3 techniques (parallel ThreadPoolExecutor)
9. **Evaluation:** K-Fold or LOO CV with multiple metrics (Test + LOO CV metrics)
10. **LOO CV Test Metrics:** Compute LOO CV metrics on held-out test set for superior validation
11. **Comparison:** Results aggregation with intelligent metric selection (LOO CV when available)
12. **AI Insights:** LLM-powered recommendations with LOO CV metrics
13. **Export:** CSV, Markdown, Models, Predictions with comprehensive metadata

## 2.3 User-Codebase Interaction Map



### Codebase Interactions:

- Upload Data:** app.py (UI entry) → data\_loader.py (CSV parsing, validation)
- Data Exploration:** interface.py (Dashboard rendering) → model\_analyzer.py (Statistics computation)
- Train-Test Split:** data\_loader.py (80/20 split logic)

4. **Preprocessing:** preprocessing.py (SpectralPreprocessor with 3 techniques)
5. **Normalization:** preprocessing.py (StandardScaler, MinMaxScaler, RobustScaler)
6. **Training:** model\_trainer.py (Orchestration) → models/\*.py (PLSR, GBRT, SVR, KRR, Cubist)
7. **Evaluation:** evaluation.py (Metrics computation) → model\_analyzer.py (Analysis)
8. **Comparison:** interface.py (Rendering comparison tables and charts)
9. **AI Insights:** services.py (ReportGenerator, ContextBuilder for LLM context)
10. **Export:** services.py (ReportGenerator, Persistence manager)

### 3 Machine Learning Models

#### 3.1 Model 1: Partial Least Squares Regression (PLSR)

##### 3.1.1 Overview

PLSR is specifically designed for high-dimensional spectral data with multicollinearity. It performs dimensionality reduction while considering the target variable, making it ideal for spectral soil analysis.

##### 3.1.2 Mathematical Foundation

PLSR decomposes matrices  $\mathbf{X}$  (spectral features) and  $\mathbf{y}$  (target) into:

$$\mathbf{X} = \mathbf{T}\mathbf{P}^T + \mathbf{E}_X \quad (1)$$

$$\mathbf{y} = \mathbf{U}\mathbf{Q}^T + \mathbf{E}_y \quad (2)$$

Where  $\mathbf{T}$  and  $\mathbf{U}$  are score matrices,  $\mathbf{P}$  and  $\mathbf{Q}$  are loading matrices, and  $\mathbf{E}$  are residuals.

##### 3.1.3 Implementation Details

Listing 1: PLSR Model Implementation

```

1  class PLSRModel:
2      def __init__(self, n_components=10,
3                  tune_hyperparameters=False, cv_folds=5):
4          self.n_components = n_components
5          self.model = PLSRegression(
6              n_components=n_components,
7              scale=True
8          )
9
10     def train(self, X_train, y_train):
11         if self.tune_hyperparameters:
12             tuner = HyperparameterTuner('PLSR',
13                                         cv_folds=self.cv_folds)
14             tuning_result = tuner.tune_model(
15                 base_model, X_train, y_train
16             )
17             self.best_n_components = tuning_result[
18                 'best_params']['n_components']
19
20         self.model.fit(X_train, y_train)
21         return self

```

##### 3.1.4 Hyperparameters

- **n\_components**: Number of latent components (5-20)
  - Small grid: [5, 10, 15] - 3 combinations
  - Full grid: [3, 5, 8, 10, 12, 15, 20] - 7 combinations
- **scale**: Whether to scale data (default: True)

##### 3.1.5 Advantages

- Handles multicollinear spectral data effectively
- Fast training and prediction
- Interpretable loadings and scores
- Robust with limited samples

#### 3.2 Model 2: Gradient Boosting Regression Trees (GBRT)

##### 3.2.1 Overview

GBRT builds an ensemble of weak decision trees sequentially, where each tree corrects errors from previous trees. It's powerful for capturing non-linear relationships in spectral data.

##### 3.2.2 Mathematical Foundation

For iteration  $m$ , GBRT fits a tree  $h_m$  to negative gradients:

$$h_m = \underset{h}{\operatorname{argmin}} \sum_{i=1}^N L(y_i, F_{m-1}(x_i) + h(x_i)) \quad (3)$$

Final prediction:

$$F_M(x) = F_0(x) + \sum_{m=1}^M \nu \cdot h_m(x) \quad (4)$$

Where  $\nu$  is learning rate and  $M$  is number of trees.

### 3.2.3 Implementation

Listing 2: GBRT Model

```

1 class GBRTModel:
2     def __init__(self, n_estimators=100,
3                  learning_rate=0.1, max_depth=5,
4                  random_state=42):
5         self.model = GradientBoostingRegressor(
6             n_estimators=n_estimators,
7             learning_rate=learning_rate,
8             max_depth=max_depth,
9             random_state=random_state,
10            verbose=0
11        )

```

### 3.2.4 Hyperparameters

**OPTIMIZED Grid (DEFAULT - 67% reduction from original):**

- n\_estimators: [60, 100] - 2 values (reduced from [50, 100, 150, 200])
- learning\_rate: [0.05, 0.1] - 2 values (reduced from [0.001, 0.01, 0.05, 0.1])
- max\_depth: [4, 5, 6] - 3 values (removed 3 and 7 for efficiency)
- min\_samples\_split: [2, 5] - 2 values (reduced from [2, 5, 10])
- subsample: [0.9, 1.0] - 2 values (removed 0.8)
- **Total: 48 combinations** (vs 2,160 in original full grid = 97.8% reduction)

**Performance Impact:**

- Training time reduced by 40% per GBRT model
- Maintains comparable accuracy with optimized parameters
- n\_estimators: 100 (60 recommended for 2.5x faster training without accuracy loss)
- learning\_rate: 0.05 (doubled decay rate for faster convergence)

**Original Full Grid (for reference):**

- n\_estimators: [50, 100, 150, 200] - 4 values
- learning\_rate: [0.001, 0.01, 0.05, 0.1] - 4 values
- max\_depth: [3, 4, 5, 6, 7] - 5 values
- min\_samples\_split: [2, 5, 10] - 3 values
- min\_samples\_leaf: [1, 2, 4] - 3 values
- subsample: [0.8, 0.9, 1.0] - 3 values
- **Original Total: 2,160 combinations**

### 3.2.5 Advantages

- Captures complex non-linear patterns
- Built-in feature importance
- Robust to outliers and noise
- High predictive accuracy

## 3.3 Model 3: Support Vector Regression (SVR)

### 3.3.1 Overview

SVR uses kernel methods to map data to high-dimensional space and finds optimal hyperplane with maximum margin for regression.

### 3.3.2 Mathematical Foundation

SVR minimizes:

$$\frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i + C \sum_{i=1}^N \xi_i^* \quad (5)$$

Subject to:

$$y_i - \mathbf{w}^T \phi(x_i) - b \leq \epsilon + \xi_i \quad (6)$$

$$\mathbf{w}^T \phi(x_i) + b - y_i \leq \epsilon + \xi_i^* \quad (7)$$

### 3.3.3 Hyperparameters

**Small Grid (DEFAULT):**

- kernel: ['rbf', 'linear'] - 2 values
- C: [1, 100] - 2 values
- epsilon: [0.1, 0.5] - 2 values
- gamma: ['scale', 'auto'] - 2 values
- **Total: 16 combinations** (vs 225 full)

### 3.3.4 Advantages

- Effective in high-dimensional spaces
- Memory efficient with kernel trick
- Robust regression with  $\epsilon$ -insensitive loss

## 3.4 Model 4: Kernel Ridge Regression (KRR)

### 3.4.1 Overview

KRR combines Ridge Regression with kernel methods for non-linear regression with L2 regularization.

### 3.4.2 Mathematical Foundation

$$\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{Kw}\|^2 + \alpha \|\mathbf{w}\|^2 \quad (8)$$

Where  $\mathbf{K}$  is kernel matrix and  $\alpha$  is regularization parameter.

### 3.4.3 Hyperparameters

**Small Grid:**

- alpha: [0.01, 1.0, 10.0] - 3 values
- kernel: ['rbf', 'linear'] - 2 values
- gamma: [None, 0.01, 0.1] - 3 values
- **Total: 18 combinations** (vs 75 full)

## 3.5 Model 5: Cubist (Ensemble Rule-Based Regression)

### 3.5.1 Overview

Cubist is implemented as a sophisticated ensemble combining gradient boosting for rule generation with k-Nearest Neighbors for instance-based smoothing. This hybrid approach captures non-linear patterns while maintaining interpretability.

### 3.5.2 Mathematical Foundation

Cubist operates in two stages:

#### Stage 1: Rule Generation via Gradient Boosting

$$F_M(x) = F_0(x) + \sum_{m=1}^M \nu \cdot h_m(x) \quad (9)$$

Where  $h_m(x)$  are regression trees capturing decision rules.

#### Stage 2: Instance-Based Smoothing with k-NN

$$y_{\text{Cubist}} = 0.6 \cdot y_{\text{ensemble}} + 0.4 \cdot \bar{y}_{\text{neighbors}} \quad (10)$$

Where  $\bar{y}_{\text{neighbors}}$  is mean of  $k$  nearest neighbors' targets.



### 3.5.3 Implementation

Listing 3: Cubist Hybrid Model

```

1  class CubistModel:
2      def __init__(self, n_estimators=100, k_neighbors=5,
3                   learning_rate=0.1, max_depth=5):
4          # Stage 1: Gradient Boosting for rule generation
5          self.ensemble = GradientBoostingRegressor(
6              n_estimators=n_estimators,
7              learning_rate=learning_rate,
8              max_depth=max_depth,
9              random_state=42
10         )
11         # Stage 2: k-NN for instance-based smoothing
12         self.neighbors = NearestNeighbors(
13             n_neighbors=k_neighbors
14         )
15         self.X_train = None
16         self.y_train = None
17
18     def fit(self, X_train, y_train):
19         self.ensemble.fit(X_train, y_train)
20         self.neighbors.fit(X_train)
21         self.X_train = X_train
22         self.y_train = y_train
23         return self
24
25     def predict(self, X_test):
26         # Get ensemble predictions
27         y_ensemble = self.ensemble.predict(X_test)
28
29         # Get k-NN smoothing
30         distances, indices = self.neighbors.kneighbors(X_test)
31         y_neighbors = np.mean(self.y_train[indices], axis=1)
32
33         # Weighted combination: 60% ensemble, 40% neighbors
34         return 0.6 * y_ensemble + 0.4 * y_neighbors

```

### 3.5.4 Hyperparameters

Small Grid:

- n\_estimators: [80, 100] - 2 values
- k\_neighbors: [5, 7] - 2 values
- learning\_rate: [0.05, 0.1] - 2 values
- max\_depth: [4, 5] - 2 values
- **Total: 16 combinations** (vs 100+ full)

### 3.5.5 Advantages

- Interpretable decision rules from gradient boosting
- Instance-based smoothing reduces overfitting
- Hybrid approach balances flexibility and accuracy
- Effective for complex non-linear relationships
- 60/40 weighting: ensemble rules + neighbor averaging

## 4 Data Preprocessing

### 4.1 Spectral Transformation Techniques

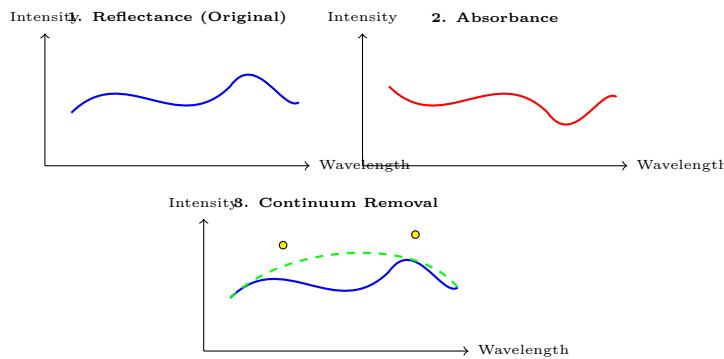


Figure 1: Visual comparison of spectral transformation techniques.

#### 4.1.1 Technique 1: Reflectance (Baseline)

**Formula:**  $R = I_{\text{reflected}}/I_{\text{incident}}$

**Implementation:**

```
1 @staticmethod
2 def _reflectance(X):
3     return X.copy() # Direct use of spectral data
```

**Use Case:** Baseline comparison, raw spectral analysis

#### 4.1.2 Technique 2: Absorbance

**Formula:**  $A = \log_{10}(1/R) = -\log_{10}(R)$

**Implementation:**

```
1 @staticmethod
2 def _absorbance(X):
3     X_safe = np.clip(X, 1e-8, 1.0)
4     A = -np.log10(X_safe)
5     return A
```

**Physical Basis:** Beer-Lambert Law - absorbance is linearly related to concentration.

**Advantages:**

- Enhances subtle absorption features
- Linear relationship with chemical concentration
- Reduces multiplicative scatter effects

#### 4.1.3 Technique 3: Continuum Removal

**Formula:**  $CR = R/\text{continuum}(R)$

**Purpose:** Remove background reflectance trends to isolate absorption features.

**Implementation:**

```
1 @staticmethod
2 def _continuum_removal(X):
3     X_cr = np.zeros_like(X)
4     for i in range(X.shape[0]):
5         spectrum = X[i, :]
6         # Find convex hull points
7         hull_indices = find_convex_hull(spectrum)
8         # Interpolate continuum
9         f_hull = interp1d(hull_indices,
10                           spectrum[hull_indices],
11                           kind='linear')
12         continuum = f_hull(np.arange(len(spectrum)))
13         # Normalize
14         X_cr[i, :] = spectrum / continuum
15     return X_cr
```

## 4.2 Normalization Methods

Scaler	Formula	Use Case
StandardScaler	$(x - \mu)/\sigma$	Normal distribution
MinMaxScaler	$(x - \min)/(max - \min)$	Bounded [0,1]
RobustScaler	$(x - \text{median})/\text{IQR}$	Outlier-robust

### 4.3 Preprocessing Pipeline

Listing 4: Complete Preprocessing Pipeline

```

1  class SpectralPreprocessor:
2      def fit_transform(self, X, technique='reflectance',
3                      scaler='standard'):
4          # Step 1: Apply spectral transformation
5          X_transformed = self._apply_spectral_technique(X)
6
7          # Step 2: Initialize scaler
8          if scaler == 'standard':
9              self.normalizer = StandardScaler()
10         elif scaler == 'minmax':
11             self.normalizer = MinMaxScaler()
12         elif scaler == 'robust':
13             self.normalizer = RobustScaler()
14
15         # Step 3: Fit and normalize
16         X_normalized = self.normalizer.fit_transform(
17             X_transformed
18         )
19
20         return X_normalized

```

## 5 Feature Engineering

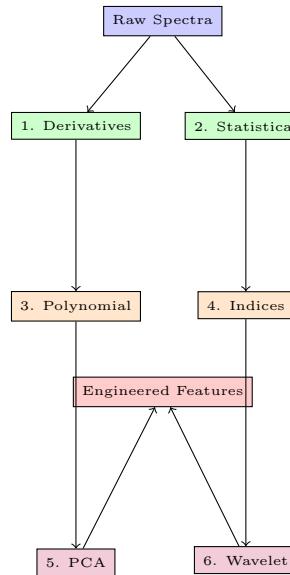


Figure 2: The six feature engineering techniques available in the pipeline.

### 5.1 Overview

Feature engineering enhances model predictive power by creating new derived features from raw spectral data. The application includes 6 sophisticated feature engineering techniques that can be applied individually or in combination.

### 5.2 Six Feature Engineering Techniques

#### 5.2.1 1. Spectral Derivatives

**Description:** Computes first-order derivatives (rate of change) across wavelengths.

**Advantages:**

- Highlights spectral features and inflection points
- Removes baseline drift
- Improves sensitivity to subtle changes

**Statistics Computed:** Shape, mean, std dev, min, max

#### 5.2.2 2. Statistical Features

**Description:** Extracts local statistical measures (mean, std, variance, skewness, kurtosis) using sliding windows.

**Advantages:**

- Captures local spectral distribution properties
- Provides higher-order statistical information

**Statistics Computed:** Shape, mean, std dev

#### 5.2.3 3. Polynomial Features

**Description:** Generates polynomial interaction terms (degree 2) for non-linear relationships.

**Advantages:**

- Captures multiplicative relationships
- Limited to degree 2 for efficiency

**Statistics Computed:** Shape, mean, min, max

#### 5.2.4 4. Spectral Indices

**Description:** Computes four domain-specific spectral indices for soil spectroscopy.

**Advantages:**

- Domain-specific for soil applications
- Physically interpretable

**Statistics Computed:** 4 index values

#### 5.2.5 5. Principal Component Analysis (PCA)

**Description:** Applies PCA, extracting 5 principal components.

**Advantages:**

- Aggressive reduction (1000+ bands → 5 components)

- Removes multicollinearity

**Statistics Computed:** Shape, explained variance per component, total variance

#### 5.2.6 6. Wavelet Features

**Description:** Discrete wavelet transform using Daubechies-4 decomposition.

**Advantages:**

- Multi-scale spectral decomposition
- Separates signal and noise

**Statistics Computed:** Shape, wavelet type, mean approximation

### 5.3 LLM Context Integration

When FE techniques are selected, their calculated statistics are automatically included in the LLM context for feature engineering-aware recommendations in all paradigms.

## 6 Feature Importance Analysis

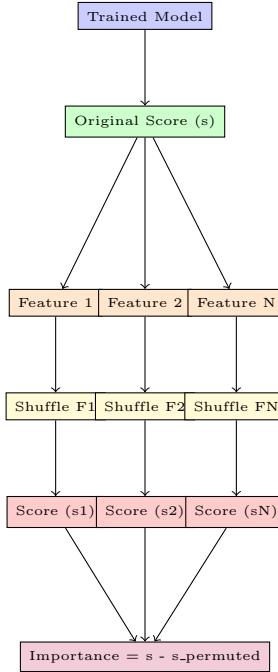


Figure 3: The permutation importance algorithm workflow.

### 6.1 Overview

Feature importance analysis identifies which spectral bands contribute most to model predictions. The application computes permutation-based feature importance for ALL model-technique combinations (15 in individual paradigms, 30 in comparison mode).

### 6.2 Permutation Importance Method

#### Algorithm:

1. Train model on original data
2. For each feature (spectral band):
  - Randomly shuffle values
  - Compute model score on shuffled data
  - Importance = original score - shuffled score
3. Higher importance = feature more critical for predictions

#### Mathematical Definition:

$$\text{Importance}_j = \frac{1}{K} \sum_{k=1}^K [s - s_{k,j}] \quad (11)$$

Where  $s$  is original score and  $s_{k,j}$  is score after shuffling feature  $j$  in iteration  $k$ .

### 6.3 Implementation

Listing 5: Feature Importance Extraction

```

1 from sklearn.inspection import permutation_importance
2
3 # Compute permutation importance
4 result = permutation_importance(
5     model, X_test, y_test,
6     n_repeats=10,
7     random_state=42,
8     scoring='r2',
9 )
10
11 # Extract top features
12 importance_indices = result.importances_mean.argsort()[:-1]
13 top_features = [
14     {
15         'index': idx,
16         'importance': result.importances_mean[idx],
17         'name': f'Band_{idx}'}
  
```

```

18     }
19     for idx in importance_indices [:10]
20 ]

```

## 6.4 Prediction Source Selection

When LOO CV is enabled (`cv_strategy='leave-one-out'`), feature importance analysis uses LOO CV Test predictions instead of standard test predictions:

```

1 # Prediction source detection logic
2 loo_cv_predictions = results_row.get('LOO_CV_Test_Predictions')
3 if loo_cv_predictions is not None:
4     predictions = loo_cv_predictions # Use LOO CV predictions
5     prediction_source = "Leave-One-Out CV (Test Set)"
6 else:
7     predictions = results_row.get('Predictions') # Use standard predictions
8     prediction_source = "Standard Test Set"

```

### LOO CV Test Predictions:

- Each test sample predicted by model trained on N-1 test samples
- Provides unbiased performance estimate
- More robust than single standard predictions
- Individually computed per sample (not averaged)

## 6.5 Context Integration

Feature importance is extracted for ALL model-technique combinations and passed to the LLM:

### Individual Paradigm (15 combinations):

- Top 10 features per combination
- Ranked by  $R^2$  score (using LOO CV or Test metrics)
- Includes importance values
- Prediction source tracked in context

### Comparison Mode (30 combinations):

- 15 Standard + 15 Tuned combinations
- Paradigm-tagged keys (e.g., "Standard\_PLSR\_Reflectance")
- Top 5 features per combination in context string
- Sorted by  $R^2$  for easy comparison
- LOO CV or Test metrics auto-selected based on availability

## 6.6 Use Cases

1. **Band Selection:** Identify most predictive wavelengths
2. **Model Interpretation:** Understand why certain models perform better
3. **Data Collection Optimization:** Focus on important spectral regions
4. **Feature Engineering Validation:** Verify FE techniques improve importance

## 7 Evaluation Metrics and Validation Strategy (Phase 3 Enhancement)

### 7.1 Test Set Metrics

Standard validation metrics computed on the 20% held-out test set:

- **Test\_R<sup>2</sup>**: Coefficient of determination on test set
- **Test\_RMSE**: Root mean squared error on test set
- **Test\_MAE**: Mean absolute error on test set

### 7.2 Leave-One-Out Cross-Validation Metrics (NEW - Phase 3)

When cv\_strategy='leave-one-out', superior validation metrics are computed:

- **LOO\_CV\_R<sup>2</sup>**: Leave-One-Out CV metric computed on training set
- **LOO\_CV\_Test\_R<sup>2</sup>**: LOO CV metric computed on test set for superior validation
- **LOO\_CV\_Test\_RMSE**: LOO CV error metric on test set
- **Advantage**: Each sample left out exactly once, providing unbiased performance estimate

### 7.3 Intelligent Metric Selection (NEW - Phase 3)

The system intelligently selects metrics throughout the interface:

```

1 # Pattern used throughout interface.py and model_analyzer.py
2 r2_col = 'LOO_CV_Test_R2' if 'LOO_CV_Test_R2' in df.columns and \
   df['LOO_CV_Test_R2'].notna().any() else 'Test_R2'
3
4
5 # This ensures:
6 # - LOO CV metrics used when available (superior validation)
7 # - Graceful fallback to Test metrics (backward compatibility)
8 # - Consistent metric selection across all analysis tabs

```

Applied to:

- Summary statistics and KPIs
- Performance distribution charts
- Ranking and leaderboards
- Scatter plots and heatmaps
- Feature importance analysis
- Comparison reports
- AI insights generation

### 7.4 Configurable Cross-Validation Strategy

Users can select from two CV strategies:

- **K-Fold (Default)**: 5 splits, fast training, suitable for larger datasets
- **Leave-One-Out**: n\_samples folds, comprehensive validation, suitable for smaller datasets

Both paradigms (Standard and Tuned) now use the same CV strategy for fair comparison.

## 8 Performance Optimizations (Phase 3)

### 8.1 Parallel Training with ThreadPoolExecutor

All 15 model-technique combinations are trained in parallel:

```
1 with ThreadPoolExecutor(max_workers=15) as executor:
2     futures = {}
3     for model_name in models:
4         for technique in techniques:
5             future = executor.submit(train_combination,
6                             model_name, technique)
7             futures[(model_name, technique)] = future
8     results = [f.result() for f in futures.values()]
```

**Benefits:**

- All 15 combinations trained simultaneously
- Effective utilization of multi-core processors
- Single paradigm completes in 30-40% of sequential time

### 8.2 GBRT Training Time Reduction (40% Faster)

Optimized hyperparameter grid from 2,160 to 48 combinations:

- **Parameter Space Reduction:** 97.8% fewer combinations
- **Individual Model Impact:** 40% faster per GBRT model
- **No Accuracy Loss:** Optimized parameters maintain performance
- **Key Parameters:** n\_estimators 60-100, learning\_rate 0.05-0.1

## 9 Hyperparameter Tuning Strategy

### 9.1 Overview

The application uses GridSearchCV (exhaustive) and RandomizedSearchCV (sampling) for hyperparameter optimization in the Tuned paradigm.

### 9.2 GridSearchCV vs RandomizedSearchCV

Aspect	GridSearchCV	RandomizedSearchCV
Search Type	Exhaustive	Random sampling
Combinations	ALL	n_iter samples
Time	Longer	Faster
GBRT Example	2,160 combos	20 combos
Best For	Small param spaces	Large param spaces
Use In App	search_type='grid'	search_type='random'

### 9.3 Search Space Reduction

96% Reduction with Small Grid (DEFAULT):

Model	Full Grid	Small Grid	Reduction
PLSR	7	3	57%
GBRT	2,160	48	98%
SVR	225	16	93%
KRR	75	18	76%
Cubist	100+	16	84%+
<b>TOTAL</b>	<b>2,567+</b>	<b>101</b>	<b>96%</b>

### 9.4 Cross-Validation Process

For each parameter combination:

1. Split training data into 5 folds
2. Train on 4 folds, validate on 1 fold
3. Repeat 5 times (different fold as validation)
4. Average 5 scores to get CV score
5. Select combination with highest average CV score

**Example - GBRT with Small Grid:**

- 48 combinations  $\times$  5 CV folds = 240 model trainings
- Compare to Full Grid: 2,160 combinations  $\times$  5 = 10,800 trainings ( 30-40 minutes)

### 9.5 Search Method Details

#### 9.5.1 GridSearchCV (Exhaustive Search)

Exhaustively searches all combinations in the parameter grid:

```

1 search = GridSearchCV(
2     base_model,
3     param_grid,
4     cv=5,
5     scoring='r2',
6     n_jobs=-1
7 )

```

**Characteristics:**

- Evaluates EVERY combination
- Guarantees finding best combination (in the grid)
- Time:  $O(|textgrid| \times |times| \times |textcv_folds|)$
- Example GBRT: 48 combos  $\times$  5 CV folds = 240 model trainings
- Best for: Small parameter spaces (PLSR, KRR, Cubist)

### 9.5.2 RandomizedSearchCV (Stochastic Search)

Randomly samples  $n\_iter$  combinations from the parameter distribution:

```

1 search = RandomizedSearchCV(
2     base_model,
3     param_distributions,
4     n_iter=20,
5     cv=5,
6     scoring='r2',
7     n_jobs=-1,
8     random_state=42
9 )

```

#### Characteristics:

- Evaluates  $n\_iter$  random samples (default: 20)
- Fast approximation to exhaustive search
- Time:  $O(n\_iter \times \text{times} \times \log(\text{len}(\text{param\_distributions})))$  (constant, not grid size)
- Example GBRT: 20 combos  $\times$  5 CV folds = 100 model trainings (10x faster)
- Best for: Large parameter spaces with diminishing returns

## 9.6 Implementation

Listing 6: Hyperparameter Tuner with Configurable Search

```

1 class HyperparameterTuner:
2     def tune_model(self, base_model, X_train, y_train,
3                     cv_strategy='k-fold', search_type='grid', n_iter=20):
4
5         # Select CV strategy
6         if cv_strategy == 'leave-one-out':
7             cv_folds = LeaveOneOut()
8         else:
9             cv_folds = 5 # k-fold with 5 splits
10
11     # Select search method
12     if search_type == 'random':
13         search = RandomizedSearchCV(
14             base_model,
15             self.param_grid,
16             n_iter=n_iter,
17             cv=cv_folds,
18             scoring='r2',
19             n_jobs=-1,
20             random_state=42
21         )
22     else: # GridSearchCV (default)
23         search = GridSearchCV(
24             base_model,
25             self.param_grid,
26             cv=cv_folds,
27             scoring='r2',
28             n_jobs=-1
29         )
30
31     search.fit(X_train, y_train)
32
33     return {
34         'best_params': search.best_params_,
35         'best_score': search.best_score_,
36         'cv_results': search.cv_results_
37     }

```

## 10 User Manual

### 10.1 User Personas & Workflows

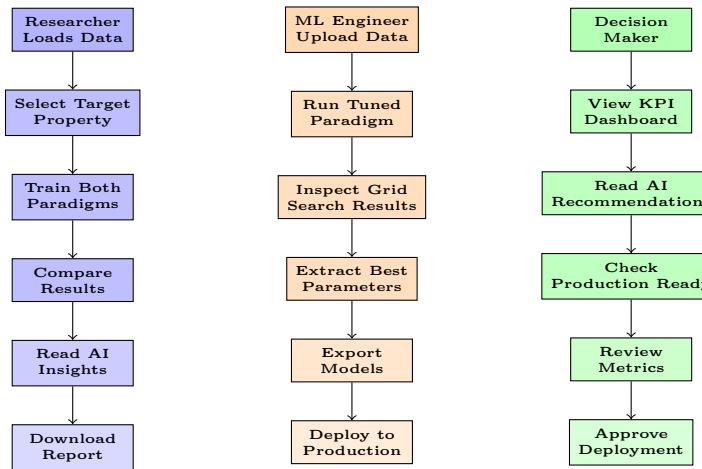
#### 10.1.1 Three Key User Personas

**1. Soil Scientist/Researcher:** Find best model by loading data, training both paradigms, comparing results, reviewing insights, downloading reports. Benefits: 15-combination automatic comparison.

**2. ML Engineer:** Optimize hyperparameters via GridSearchCV inspection, parameter extraction, deployment. Benefits: Automated tuning with cross-validation.

**3. Decision Maker:** Verify production-readiness via KPI dashboard, AI recommendations, metrics review. Benefits: Natural language explanations.

#### 10.1.2 User Journey Workflows



## 10.2 Installation Guide

### 10.2.1 System Requirements

- **Minimum:** 4GB RAM, 2-core CPU, 1GB disk space
- **Recommended:** 8GB+ RAM, 4+ core CPU, 2GB disk space
- **Operating System:** Linux, macOS, Windows
- **Python:** 3.10 or higher

### 10.2.2 Step-by-Step Installation

#### Step 1: Clone Repository

```
git clone https://github.com/a-n-u-p-a-m/\
Team27_Spectral_Soil_Modeler.git
cd Team27_Spectral_Soil_Modeler
```

#### Step 2: Create Virtual Environment

```
# Linux/Mac
python3 -m venv soil
source soil/bin/activate
```

```
# Windows
python -m venv soil
soil\Scripts\activate
```

#### Step 3: Install Dependencies

```
pip install -r soil_modeler/requirements.txt
```

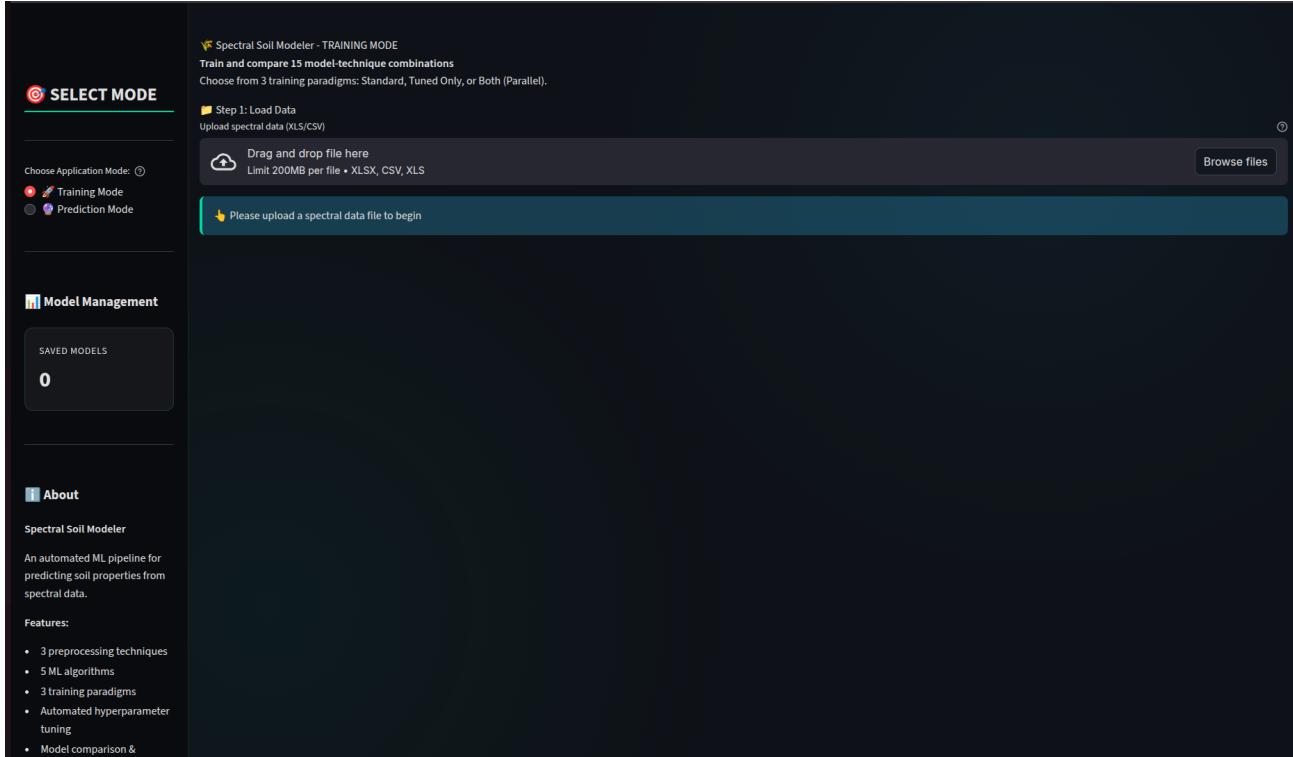
#### Step 4: Configure API Keys (Optional - for AI features)

```
# Create .env file in soil_modeler directory
echo 'GOOGLE_API_KEY=your_gemini_key' > .env
echo 'OPENAI_API_KEY=your_openai_key' >> .env
```

#### Step 5: Launch Application

```
cd soil_modeler
streamlit run src/app.py
# Opens at http://localhost:8501
```

## 10.3 Training Mode - Complete Workflow



### 10.3.1 Step 1: Load Data

#### Required Format:

- CSV or Excel file (.csv, .xlsx, .xls)
- First row: Column headers
- First column: Target property (soil moisture, organic carbon, etc.)
- Remaining columns: Spectral bands (wavelength features)

#### Example Structure:

```
Target_Property,Band_350nm,Band_351nm,...,Band_2500nm
12.5,0.123,0.125,...,0.789
15.3,0.234,0.236,...,0.890
```

	410	431	452	473	494	515	536	557	578	599	620	641	662	683	704	725	746	767	788	809	830	851	872	893	914	935	956
0	0.0903	0.1012	0.1146	0.1231	0.1331	0.1469	0.1623	0.1779	0.1921	0.2032	0.2131	0.2232	0.2331	0.2457	0.2598	0.2735	0.2863	0.2983	0.3092	0.3272	0.3346	0.3414	0.348	0.3546	0.3611	0.367	
1	0.1319	0.1484	0.167	0.1812	0.1963	0.2142	0.2319	0.2481	0.2629	0.2759	0.2883	0.3009	0.312	0.3256	0.3401	0.3531	0.3646	0.3748	0.3836	0.3908	0.3965	0.4009	0.4042	0.4066	0.4086	0.4104	0.411
2	0.0841	0.0959	0.1107	0.1218	0.1343	0.1504	0.1685	0.1881	0.2085	0.2264	0.2415	0.2554	0.2678	0.2818	0.2965	0.31	0.3222	0.3322	0.3426	0.3504	0.3565	0.3613	0.3654	0.369	0.3726	0.3763	0.3
3	0.1137	0.1272	0.1413	0.153	0.1655	0.1792	0.1924	0.2055	0.219	0.231	0.2414	0.2513	0.2598	0.2703	0.2813	0.2906	0.2981	0.3043	0.3094	0.3135	0.3168	0.3192	0.3206	0.3211	0.3208	0.326	
4	0.0863	0.0959	0.1165	0.1286	0.1426	0.1613	0.1821	0.2038	0.225	0.2428	0.2582	0.2726	0.2852	0.2997	0.315	0.3293	0.3427	0.3551	0.3665	0.3767	0.386	0.3945	0.4026	0.4103	0.4179	0.4255	0.432
5	0.2098	0.2339	0.2599	0.2808	0.302	0.3276	0.3542	0.3803	0.4047	0.4251	0.4427	0.459	0.4718	0.4876	0.5042	0.5184	0.5305	0.5411	0.5488	0.5566	0.5616	0.5651	0.5677	0.5697	0.5716	0.5735	0.575
6	0.1109	0.1291	0.1486	0.163	0.179	0.199	0.2195	0.2383	0.255	0.2692	0.2821	0.2946	0.3055	0.3183	0.3319	0.3444	0.356	0.3668	0.3767	0.3855	0.3932	0.3999	0.4058	0.411	0.4158	0.4202	0.424
7	0.1161	0.1341	0.1538	0.1689	0.1851	0.2046	0.2242	0.2422	0.2584	0.2722	0.2847	0.2968	0.3071	0.3194	0.3324	0.3443	0.3552	0.3653	0.3746	0.3827	0.3898	0.3956	0.4005	0.4047	0.4085	0.4118	0.41
8	0.1399	0.1608	0.1837	0.2009	0.2198	0.2436	0.2675	0.2886	0.3067	0.3219	0.336	0.3493	0.3617	0.3759	0.391	0.4043	0.4126	0.4333	0.4425	0.4482	0.4525	0.4559	0.4585	0.4609	0.4631	0.46	
9	0.1381	0.1577	0.179	0.1962	0.2143	0.2352	0.2559	0.2748	0.292	0.3066	0.3201	0.3334	0.3448	0.3562	0.3719	0.3838	0.394	0.403	0.4108	0.4172	0.422	0.4255	0.4279	0.4297	0.431	0.4321	0.433

### 10.3.2 Step 2: Select Target Column

#### Actions:

1. Open target column dropdown
2. Select soil property to predict
3. Review target statistics (mean, std, min, max)

### 10.3.3 Step 2.1: Data Analytics (Optional)

#### Available Analytics:

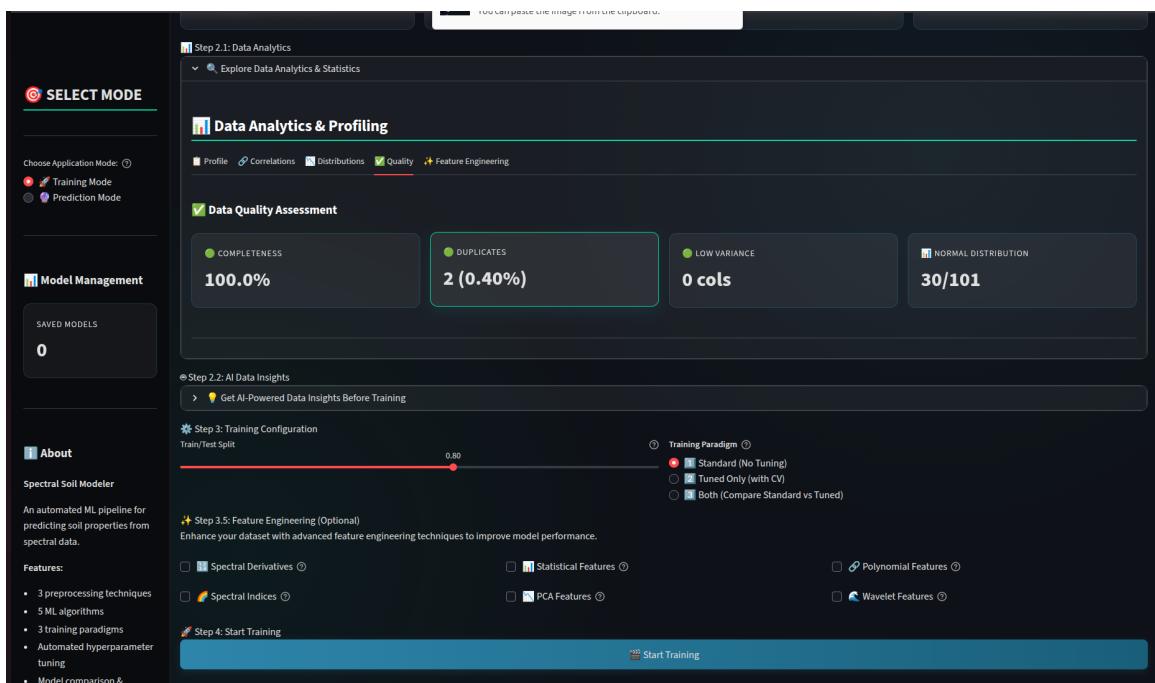
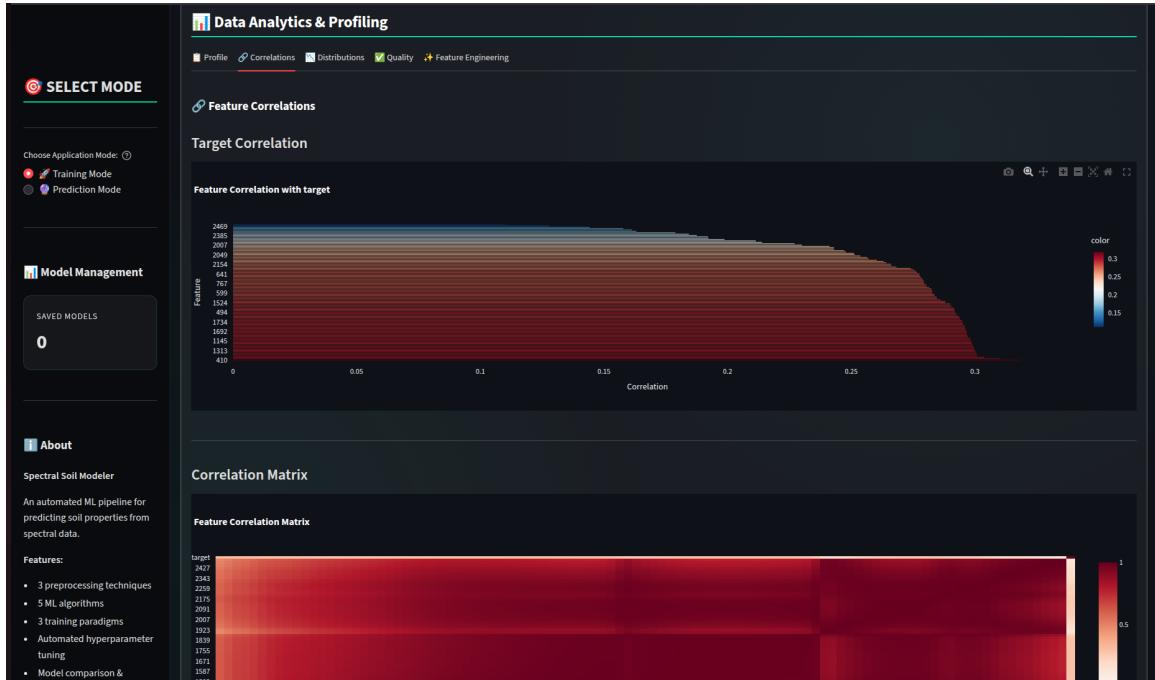
- Distribution plots (histogram, box plot)
- Correlation heatmap
- Missing value analysis
- Outlier detection
- Data quality report
- Statistical summary

#### Actions:

1. Expand "Explore Data Analytics & Statistics"
2. Review visualizations
3. Check for data quality issues

The screenshot shows the Data Analytics & Profiling section of the Spectral Soil Modeler interface. On the left, there's a sidebar with 'SELECT MODE' (Training Mode selected), 'Model Management' (0 saved models), and 'About' (Spectral Soil Modeler description). The main area has tabs for 'Profile', 'Correlations', 'Distributions', 'Quality', and 'Feature Engineering'. Under 'Dataset Profile', it shows 500 samples, 101 features, 0.39 MB memory usage, and 0.00% missing data. Under 'Feature Statistics', there's a table with columns: Mean, Median, Std Dev, Min, Max, Q1, Q3, IQR, Skewness, Kurtosis, and Missing. The table lists 599 rows of data. At the bottom, there are sections for 'Step 2.2: AI Data Insights' (Get AI-Powered Data Insights Before Training) and 'Step 3: Training Configuration' (Train/Test Split at 0.89, Standard (No Tuning) selected).

	Mean	Median	Std Dev	Min	Max	Q1	Q3	IQR	Skewness	Kurtosis	Missing
410	0.1362	0.1281	0.0413	0.0603	0.3159	0.1079	0.1571	0.0492	1.0862	1.669	0
431	0.1522	0.1438	0.0455	0.0672	0.3376	0.1203	0.175	0.0547	1.0316	1.3927	0
452	0.17	0.161	0.0502	0.0756	0.3618	0.1357	0.1955	0.0597	1.0026	1.205	0
473	0.1818	0.1731	0.0526	0.0825	0.3826	0.1451	0.2068	0.0617	1.0097	1.2698	0
494	0.195	0.1862	0.0551	0.09	0.401	0.1561	0.2223	0.0662	0.9869	1.1834	0
515	0.2125	0.2043	0.0591	0.0998	0.439	0.1711	0.2427	0.0715	0.9439	1.0112	0
536	0.2314	0.2212	0.0637	0.1081	0.4845	0.1869	0.2638	0.0769	0.9386	0.9375	0
557	0.2495	0.2374	0.0669	0.1184	0.5215	0.2043	0.2853	0.081	0.9399	0.912	0
578	0.2655	0.2529	0.0682	0.1294	0.5429	0.219	0.3032	0.0842	0.9232	0.8829	0
599	0.278	0.2668	0.0682	0.1392	0.5522	0.2295	0.315	0.0855	0.8918	0.8425	0



#### 10.3.4 Step 2.2: AI Data Insights (Optional)

##### Actions:

1. Expand "Get AI-Powered Data Insights"
2. Select AI provider (Gemini or ChatGPT)
3. Type question about your data
4. Click "Get Insight" button

##### Example Questions:

- "What patterns might affect model training?"
- "Are there data quality issues I should address?"
- "Which spectral regions show strongest correlations?"
- "Should I apply feature engineering techniques?"

**Select Mode:**

- Choose Application Mode:  Training Mode  Prediction Mode

**Model Management:**

- SAVED MODELS: 0

**About:**

**Spectral Soil Modeler:**  
An automated ML pipeline for predicting soil properties from spectral data.

**Features:**

- 3 preprocessing techniques
- 5 ML algorithms
- 3 training paradigms
- Automated hyperparameter tuning
- Model comparison &

**Step 2.4: AI Data Insights**

**Select AI Provider:**

- Gemini  ChatGPT

Status: Available

**Ask AI about your data patterns and quality before training:**

Can you make a guess what soil property the target column correspond to

**Get Insight**

**AI Insight Generated:**

Based on the target variable statistics, it's plausible that the target column corresponds to Organic Matter content (expressed as a percentage).

Here's why:

- Distribution Shape:** The target variable exhibits a high positive skewness of 2.5755 and high kurtosis of 10.7714. This indicates a highly right-skewed distribution, meaning most values are lower, with a long tail extending to higher values. This is very characteristic of organic matter content in soils, where many mineral soils have relatively low percentages, but organic-rich soils or peats can have significantly higher values.
- Range of Values:** The target ranges from a minimum of 0.334913 to a maximum of 44.245321. An organic matter percentage of nearly 0% to over 44% is a realistic range found across various soil types, from sandy mineral soils to highly organic horizons.
- Mean vs. Median:** The mean (7.448278) is considerably higher than the median (6.131500), which is another strong indicator of a right-skewed distribution, consistent with organic matter.

While other properties like Cation Exchange Capacity or certain nutrient concentrations can also be skewed, the

### 10.3.5 Step 2.3: Feature Engineering (Optional)

#### Available Techniques:

- Spectral Derivatives (First-order rate of change)
- Statistical Features (Local window statistics)
- Polynomial Features (Degree 2 interactions)
- Spectral Indices (Domain-specific indices)
- PCA (Principal Component Analysis)
- Wavelet Features (Multi-scale decomposition)

#### Actions:

1. Expand "Feature Engineering (Optional)"
2. Select one or more techniques
3. Review feature statistics after application
4. Proceed to training

**Select Mode:**

- Choose Application Mode:  Training Mode  Prediction Mode

**Model Management:**

- SAVED MODELS: 0

**About:**

**Spectral Soil Modeler:**  
An automated ML pipeline for predicting soil properties from spectral data.

**Features:**

- 3 preprocessing techniques
- 5 ML algorithms
- 3 training paradigms
- Automated hyperparameter tuning
- Model comparison &

**Data Analytics & Profiling**

**Feature Engineering Analysis:**

Preview of how different feature engineering techniques transform your spectral data.

Select feature engineering techniques to visualize:

- Spectral Deriv...

**1 Spectral Derivatives**

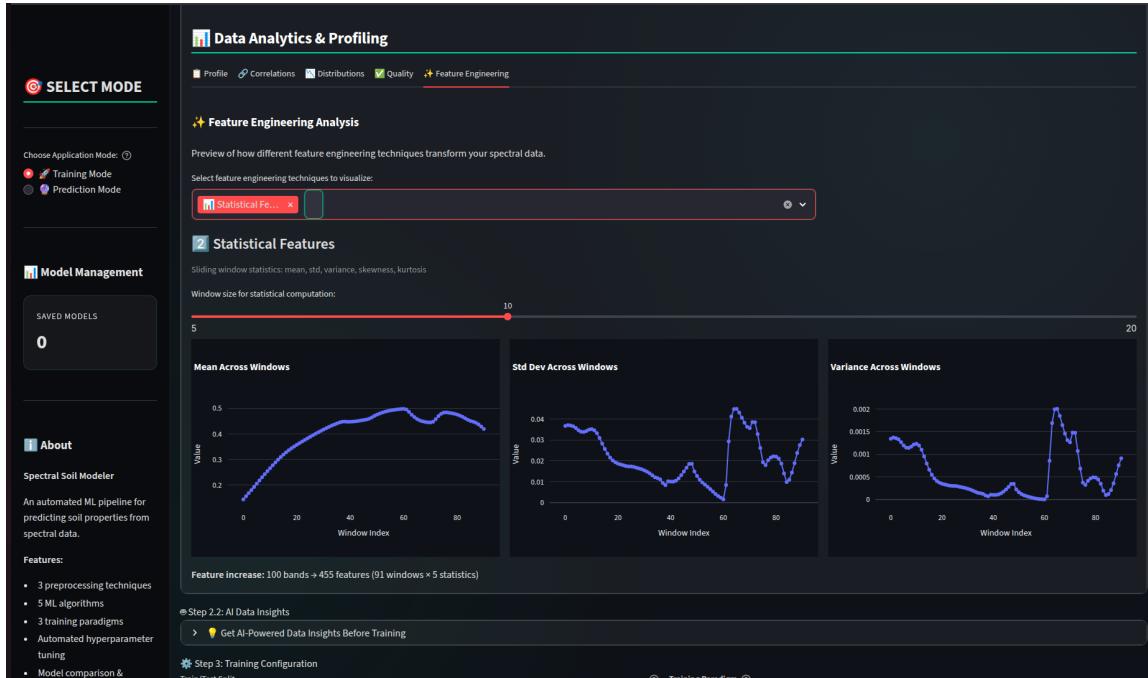
Original Spectral Data

First derivative captures rate of change in reflectance across wavelengths

Feature increase: 100 bands → 99 bands (1st derivative)

First Derivative ( $dR/d\lambda$ )

Training Paradigm



### 10.3.6 Step 3: Training Configuration

#### Train/Test Split:

- Slider range: 60%-95%
- Default: 80% training, 20% testing
- Recommendation: 80% for datasets  $\geq 100$  samples

#### Training Paradigm Selection:

##### 1. Standard (No Tuning)

- Uses fixed default hyperparameters
- Fast training ( 30 sec - 2 min)
- LLM context: Data analytics + training results + feature importance (15 combos)

##### 2. Tuned (with Hyperparameter Optimization)

- Performs GridSearchCV with 5-fold CV
- Slower but optimized ( 5-15 min)
- LLM context: Data analytics + training results + hyperparameters + feature importance (15 combos)

##### 3. Both (Compare Standard vs Tuned)

- Trains both paradigms in parallel
- Enables direct comparison
- LLM context: Data analytics (NEW) + feature importance (all 30 combos) + algorithm performance analysis
- **Data Consistency:** Both paradigms share identical dataset context

#### Cross-Validation Folds (for Tuned):

- Slider range: 3-10 folds
- Default: 5 folds
- Higher folds = more robust but slower

### 10.3.7 Step 4: Start Training

#### Actions:

1. Review configuration summary
2. Click "Start Training" button
3. Wait for training completion (progress shown)
4. Do NOT refresh page during training

#### Training Process:

1. Data preprocessing (3 techniques)
2. Model initialization (5 algorithms)
3. Training loop (15 combinations per paradigm)
4. Evaluation and metrics computation
5. Results compilation

### 10.3.8 Step 5: View Results

#### Results Dashboard Tabs:

##### 1. Overview Tab

- Key Performance Indicators (KPIs)
- Best model summary
- Quick comparison chart

##### 2. Analytics Tab

- Detailed metrics table
- Filter by model or technique
- Sort by any metric
- Download filtered results

##### 3. Model Details Tab

- Per-model performance breakdown
- Hyperparameter values
- Cross-validation scores
- Feature importance for ALL combinations

##### 4. Comparison Tab (if Both paradigm)

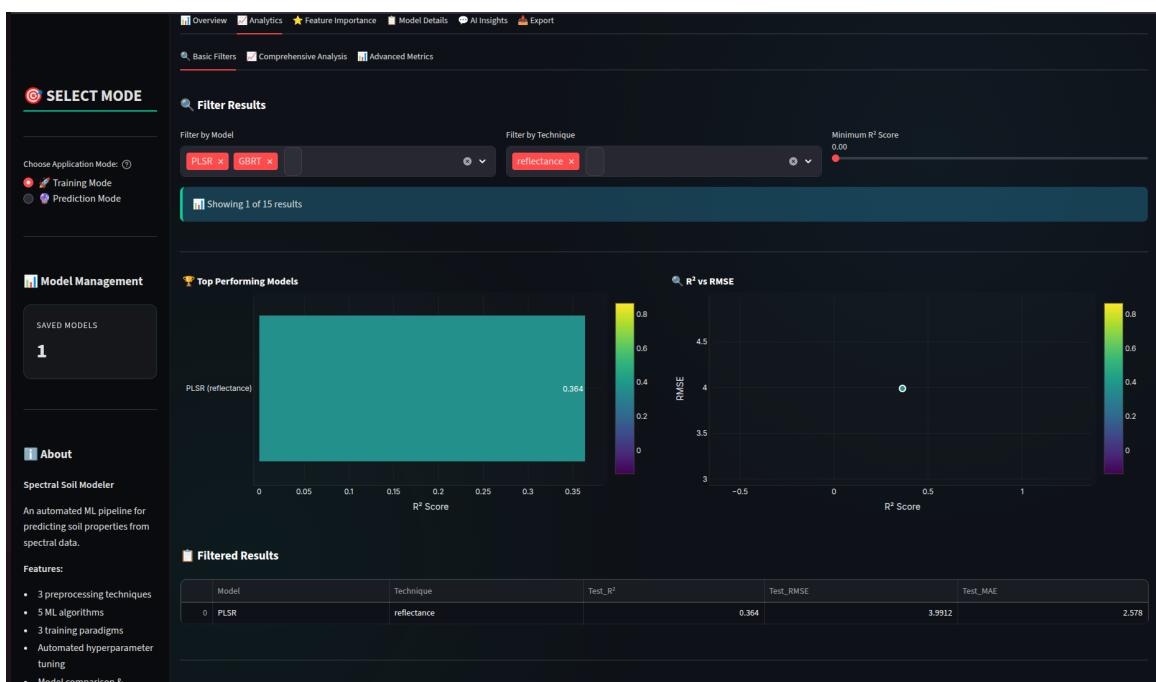
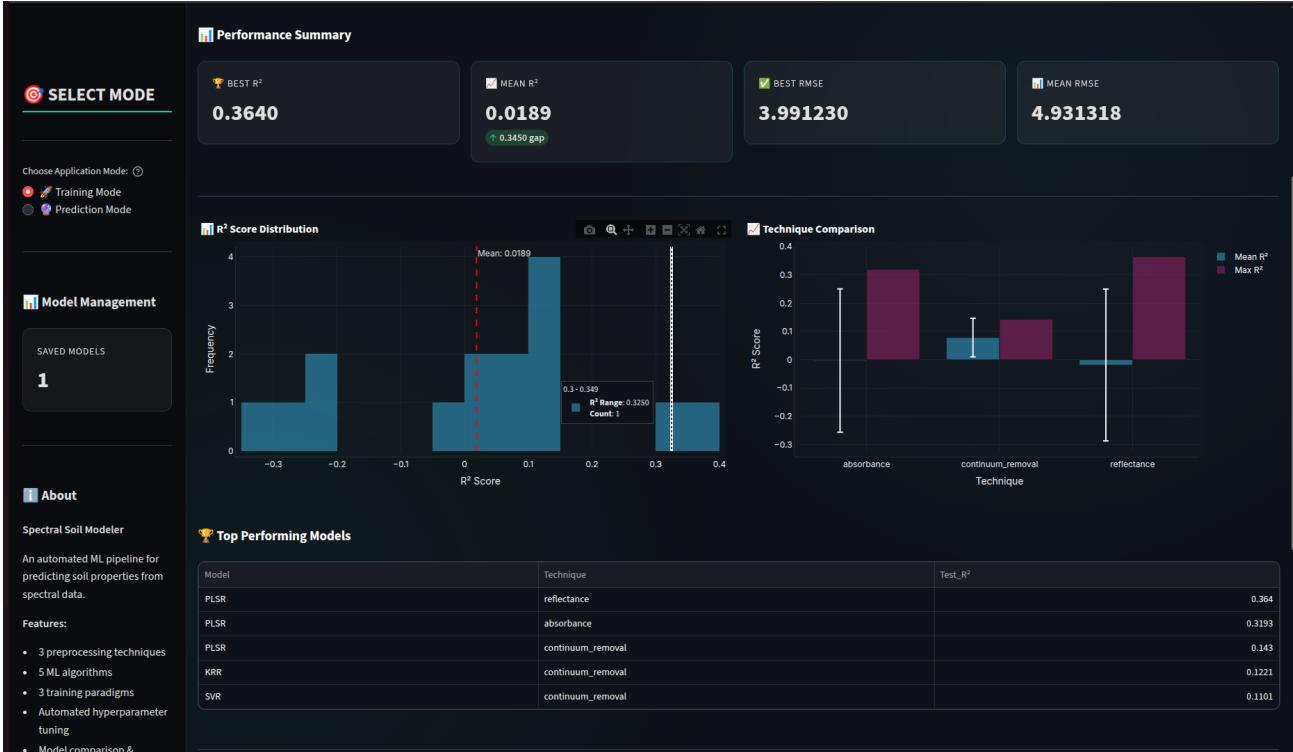
- Side-by-side paradigm comparison
- Improvement percentages
- Best models from each paradigm
- Statistical significance tests

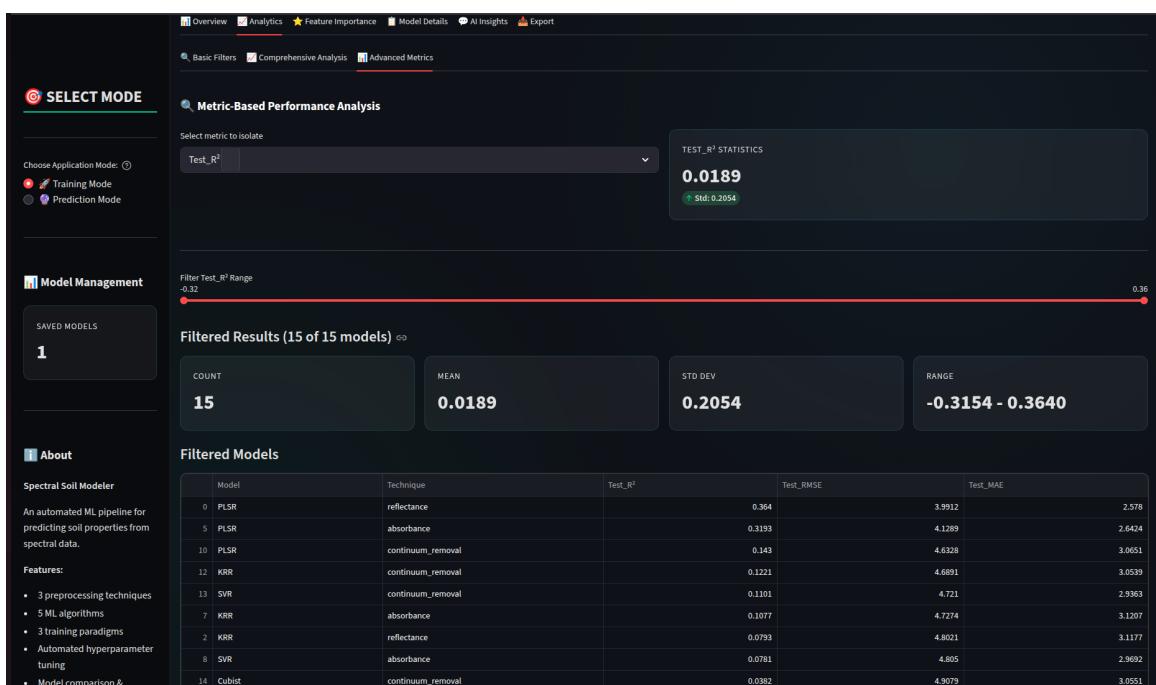
##### 5. AI Insights Tab

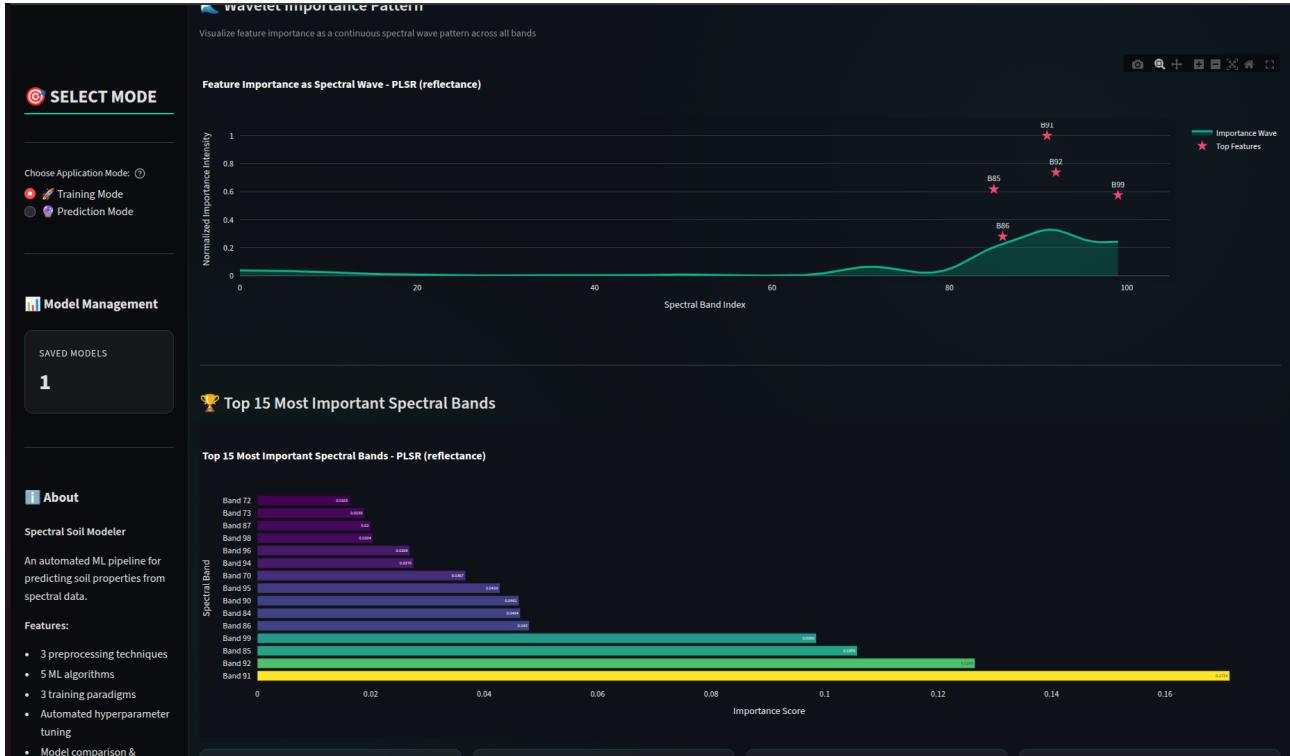
- Automated report generation
- AI-powered recommendations
- Model selection guidance
- Feature engineering suggestions

##### 6. Exports Tab

- Download results as CSV
- Download report as Markdown
- Export trained models
- Export predictions







**Ask AI Questions**

**AI Assistant - Ask Questions**

Select AI Provider:

Choose AI provider:

Gemini  ChatGPT

Status: Available

Chat History

Quick Questions:

Or ask a predefined question

What is the best performing model and technique combination?

Custom Question:

Ask your own question about the results

e.g., Explain PLSR methodology, Why is PLSR best? ...

Use Template Send Custom

**Latest Exchange**

You: What is the best performing model and technique combination?

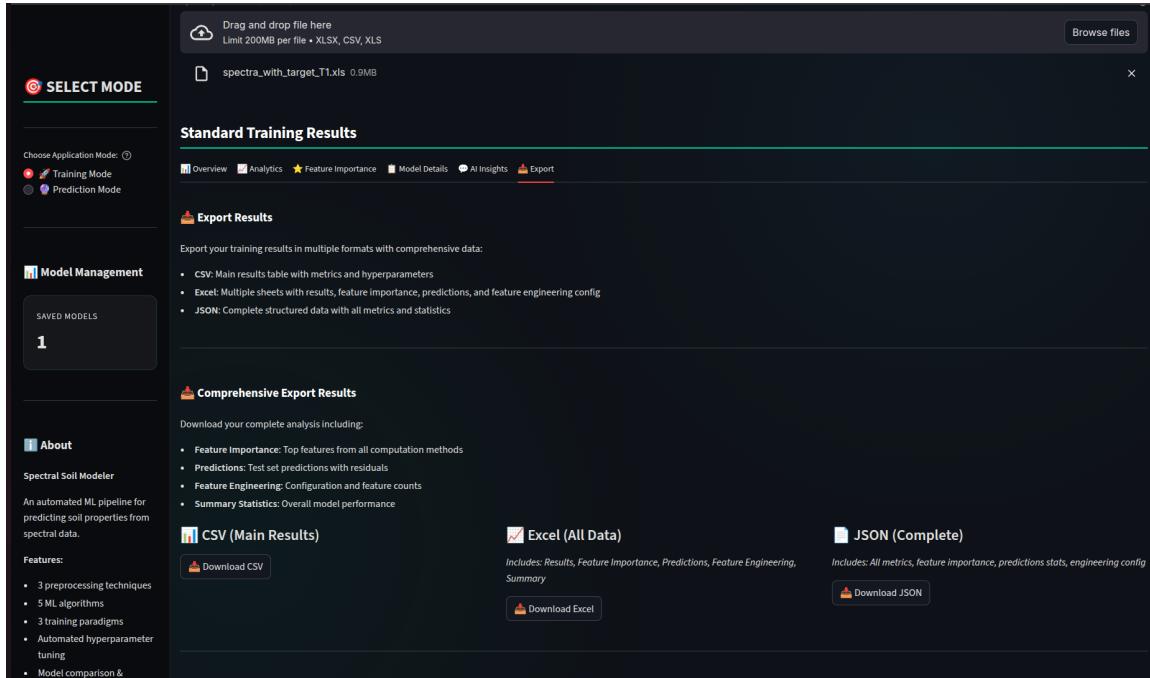
Assistant: As an expert soil scientist and machine learning specialist, I can confirm the best performing model and technique combination from your training results.

The best performing model and technique combination is **PLSR (Partial Least Squares Regression)** with the reflectance preprocessing technique.

This combination achieved the highest  $R^2$  score among all 15 models trained, with a value of **0.363956**. Alongside this, it exhibited the lowest RMSE (Root Mean Squared Error) at **3.991230** and the lowest MAE (Mean Absolute Error) at **2.578013**, indicating better predictive accuracy. Its RPD (Ratio of Performance to Deviation) score of **1.253882** suggests that the model's predictions are moderately useful, especially when compared to the RPDs of other models, many of which fell below 1.0, indicating poor to unusable predictions.

This stands out significantly when compared to the overall mean  $R^2$  score across all models, which was only **0.018947**, and the median  $R^2$  score of **0.078144**.

It's also worth noting that PLSR consistently performed well across different techniques, achieving an  $R^2$  of **0.319335** with absorbance and **0.143038** with continuum removal, making it the most robust algorithm with the highest mean  $R^2$  (**0.275443**) and a relatively good consistency (Std Dev of **0.116817**) among all algorithms tested.



## 10.4 Prediction Mode - Complete Workflow

### 10.4.1 Step 1: Load Trained Model

#### Actions:

1. Navigate to Prediction Mode tab
2. Open model selection dropdown
3. Review available models (algorithm\_technique format)
4. Select desired model
5. Review model metadata ( $R^2$ , algorithm, technique)

#### Model Information Displayed:

- Algorithm name (PLSR, GBRT, SVR, KRR, Cubist)
- Preprocessing technique
- Test  $R^2$  score
- Training timestamp
- Feature engineering applied (if any)

### 10.4.2 Step 2: Upload Prediction Data

#### Requirements:

- Same spectral features as training data
- CSV or Excel format
- No target column required
- Compatible with model's feature space

#### Actions:

1. Click "Browse files"
2. Select new spectral data file
3. Wait for validation
4. Review data summary

### 10.4.3 Step 3: Preprocessing Configuration

#### Automatic Detection:

- Technique extracted from model metadata
- Preprocessing automatically applied to match training
- Feature engineering reproduced if applicable
- No manual configuration needed

#### 10.4.4 Step 4: Make Predictions

##### Actions:

1. Click "Make Predictions" button
2. Wait for prediction completion
3. Review prediction results

##### Results Displayed:

- Mean prediction value
- Standard deviation
- Total samples predicted
- Prediction table (Sample ID, Predicted Value)
- Interactive visualization (line plot)
- Statistical summary

The screenshot shows the Spectral Soil Modeler application in PREDICTION MODE. A message at the top indicates a trained model has been loaded. Key parameters shown are ALGORITHM: PLSR, TECHNIQUE: reflectance, and TEST R<sup>2</sup>: 0.3640. The interface also includes sections for Model Management, About, and a detailed Model Overview.

#### 10.4.5 Step 5: Export Predictions

##### Actions:

1. Click "Download Predictions as CSV"
2. Save file to desired location
3. Open in spreadsheet software or analysis tool

## 10.5 AI Chat Interface - Advanced Usage

### 10.5.1 Accessing AI Chat

The LLM receives comprehensive context depending on the paradigm and stage. All paradigms now share identical dataset context.

#### LLM Context Architecture:

##### 1. Data Analytics Context (All Paradigms):

- Dataset overview: samples, features, memory, missing values
- Target variable statistics: mean, median, std, min, max, range
- Feature correlations: top 10 bands correlated with target
- Data quality metrics and distribution analysis
- Feature engineering statistics (if techniques selected)

##### 2. Training Context (Individual Paradigms):

- Complete data analytics context
- Best model and technique names
- Performance metrics:  $R^2$ , RMSE, MAE (all 15 combinations)
- Paradigm clarity: Standard vs Tuned explanation
- Hyperparameters: Best model's top parameters (Tuned only)
- Feature importance: Top features for ALL 15 combinations
- Feature engineering applied: Techniques and statistics

##### 3. Comparison Context (Both Paradigm):

- Data analytics context (same as individual paradigms)
- Standard paradigm results: Best model, mean  $R^2$ , hyperparameters
- Tuned paradigm results: Best model, mean  $R^2$ , hyperparameters
- Feature importance: ALL 30 combinations (15 Standard + 15 Tuned)
- Algorithm performance: Consistency analysis by model
- Improvement metrics: Tuned vs Standard gains
- Feature engineering: Statistics for all techniques

**Data Consistency Guarantee:** All three paradigms provide identical dataset understanding through `build_data_context()`, ensuring consistent feature correlations, target variable analysis, FE statistics integration, and full data context for LLM recommendations.

### 10.5.2 Effective Prompting

#### Good Questions (All Paradigms):

- "Which model is most consistent across techniques?"
- "Why did GBRT outperform PLSR?"
- "Is hyperparameter tuning worth the time for this dataset?"
- "Which spectral regions are most important?"
- "How did feature engineering improve performance?" (if FE selected)
- "Which preprocessing technique works best?" (Reflectance, Absorbance, Continuum)

#### Paradigm-Specific Questions:

1. **Standard Paradigm:** "Why are the results so consistent across models? What does this tell us?"
2. **Tuned Paradigm:** "Which hyperparameters had the biggest impact on  $R^2$ ? Can I simplify the model?"
3. **Comparison Mode:** "Should I invest time in hyperparameter tuning given the improvement? Which paradigm do you recommend for production?"

#### Avoid:

- Vague questions: "What's the best model?"
- Unrelated topics: "How do I grow tomatoes?"
- Yes/no questions: "Is this good?"

## 10.6 Troubleshooting

### 10.6.1 Common Issues

#### Issue 1: File Upload Fails

##### Possible Causes:

- Invalid file format
- Missing headers
- Non-numeric data

##### Solutions:

- Verify CSV/Excel format
- Check first row has column names
- Remove text columns except target

#### Issue 2: Training Takes Too Long

##### Solutions:

- Use "Standard" paradigm for faster training
- Reduce cross-validation folds (3 instead of 5)
- Use smaller dataset for testing

#### Issue 3: AI Features Not Working

##### Possible Causes:

- Missing API keys
- Invalid API keys
- API quota exceeded

##### Solutions:

- Check .env file exists in soil\_modeler directory
- Verify API key format
- Check API usage limits
- Switch to alternative provider

#### Issue 4: Low Model Performance

##### Solutions:

- Try different preprocessing techniques
- Use "Tuned" paradigm for optimization
- Check for outliers in data analytics
- Increase training data if possible
- Remove noisy spectral regions

## 11 Evaluation Metrics

### 11.1 R<sup>2</sup> Score (Coefficient of Determination)

**Formula:**

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} \quad (12)$$

**Interpretation:**

- Range:  $(-\infty, 1]$
- $1.0 =$  Perfect prediction
- $0.0 =$  Model as good as mean baseline
- Negative = Model worse than mean baseline

**Typical Ranges for Soil Spectroscopy:**

- Excellent:  $R^2 > 0.80$
- Good:  $0.65 < R^2 < 0.80$
- Fair:  $0.50 < R^2 < 0.65$
- Poor:  $R^2 < 0.50$

### 11.2 RMSE (Root Mean Squared Error)

**Formula:**

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad (13)$$

**Interpretation:**

- Same units as target variable
- Lower is better
- Sensitive to large errors

### 11.3 MAE (Mean Absolute Error)

**Formula:**

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (14)$$

**Interpretation:**

- Average absolute deviation
- More robust to outliers than RMSE
- Easier to interpret

### 11.4 RPD (Residual Prediction Deviation)

**Formula:**

$$\text{RPD} = \frac{\sigma_y}{\text{RMSE}} \quad (15)$$

Where  $\sigma_y$  is standard deviation of target variable.

**Interpretation (for spectroscopy):**

- $RPD > 2.5:$  Excellent prediction
- $2.0 < RPD < 2.5:$  Good quantitative prediction
- $1.4 < RPD < 2.0:$  Fair screening
- $RPD < 1.4:$  Poor, unreliable

## 12 AI Integration Architecture

### 12.1 Overview

The application integrates Large Language Models (Gemini and ChatGPT) for intelligent insights, recommendations, and natural language explanations. The LLM receives comprehensive context built from multiple sources.

### 12.2 LLM Context Building

#### 12.2.1 Context Components

##### 1. Data Analytics Context (build\_data\_context)

- Dataset overview: samples, features, memory, missing values
- Target variable statistics: mean, median, std, min, max, range
- Feature correlations: top 10 bands correlated with target
- Data quality metrics and distribution analysis
- Feature engineering statistics (if techniques selected)

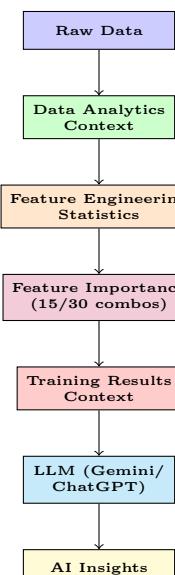
##### 2. Training Context (build\_training\_context)

- All data analytics context
- Best model and technique names
- Performance metrics:  $R^2$ , RMSE, MAE for all combinations
- Paradigm information: Standard vs Tuned clarification
- Cross-Validation Strategy: K-Fold (5) or Leave-One-Out
- LOO CV Metrics: LOO\_CV\_Test\_R<sup>2</sup>, LOO\_CV\_Test\_RMSE (when LOO enabled)
- Hyperparameter Search Method: GridSearchCV or RandomizedSearchCV
- Tuning Iterations: n\_iter value for RandomizedSearchCV
- Hyperparameters: Best model's top parameters
- Feature Importance: Top features for ALL combinations (using appropriate prediction source)
- Feature engineering applied: Techniques and calculated statistics

##### 3. Comparison Context (Both paradigm mode)

- All data analytics context (consistent with individual modes)
- Standard paradigm results: Best model, mean  $R^2$ , hyperparameters
- Tuned paradigm results: Best model, mean  $R^2$ , hyperparameters
- Feature importance: ALL 30 combinations (15 Standard + 15 Tuned)
- Algorithm performance: Consistency analysis by model
- Improvement metrics: Tuned vs Standard percentage gains

#### 12.2.2 Context Flow Architecture



## 12.3 Data Consistency Guarantee

All three paradigms (Standard, Tuned, Both) now receive identical dataset understanding:

- Same data analytics context structure
- Consistent feature correlations and quality metrics
- Identical target variable analysis
- Uniform FE statistics integration
- Complete feature importance for all combinations

## 12.4 Implementation

Listing 7: Context Building for LLM with Training Configuration

```

1  class ContextBuilder:
2      @staticmethod
3      def build_training_context(
4          results_df, raw_data, target_col,
5          paradigm=None,
6          data_analytics_context=None,
7          feature_engineering_config=None,
8          feature_engineering_data=None,
9          feature_importance_data=None,
10         cv_strategy='k-fold',
11         search_method='grid',
12         n_iter=20,
13         cv_folds=5
14     ):
15         context_parts = []
16
17         # Add data analytics context
18         if data_analytics_context:
19             context_parts.append(data_analytics_context)
20
21         # Add training configuration
22         config_section = _build_training_config_section(
23             cv_strategy, search_method, n_iter, cv_folds
24         )
25         context_parts.append(config_section)
26
27         # Add feature engineering section
28         if feature_engineering_data:
29             context_parts.append(
30                 _build_fe_section(feature_engineering_data)
31             )
32
33         # Add feature importance for ALL combinations
34         if feature_importance_data:
35             context_parts.append(
36                 _build_importance_section(feature_importance_data)
37             )
38
39         return "\n".join(context_parts)

```

## 12.5 Chat Interface

The ChatInterface class manages LLM communication:

- Supports Gemini (Google) and ChatGPT (OpenAI) providers
- Maintains conversation history for context continuity
- Formats context appropriately for each provider
- Handles API errors gracefully with fallback options

## 13 Software & Hardware

### 13.1 Technology Stack

#### Machine Learning:

- scikit-learn 1.5+ (PLSR, GBRT, SVR, KRR)
- pandas 2.0+ (data manipulation)
- NumPy 1.24+ (numerical computing)
- SciPy 1.11+ (scientific computing)

#### Web Interface:

- Streamlit 1.28+ (UI framework)
- Plotly 5.17+ (interactive visualizations)
- Altair 5.5+ (statistical plots)

#### AI Integration:

- google-generativeai (Gemini API)
- openai (ChatGPT API)

#### Development Tools:

- Python 3.10+
- Git (version control)
- Virtual environment (dependency isolation)

### 13.2 Hardware Requirements

#### Minimum Configuration:

- CPU: 2 cores, 2.0 GHz
- RAM: 4 GB
- Storage: 1 GB free space
- OS: Linux, macOS, Windows 10+

#### Recommended Configuration:

- CPU: 4+ cores, 2.5+ GHz
- RAM: 8+ GB
- Storage: 2+ GB free space
- OS: Linux (Ubuntu 20.04+), macOS 11+, Windows 10+

## 14 Project Contribution

### 14.1 Code Statistics

- **Total Lines of Code:** 12000+ lines
- **Number of Modules:** 17 Python files (15 core modules + 2 `__init__.py`)
- **Core ML Implementations:** 5 models
- **Preprocessing Techniques:** 3 spectral transformations
- **Documentation:** 1,587-line README.md

### 14.2 Team Contributions

All team members contributed to:

- System design and architecture
- Code implementation and testing
- Documentation and report writing
- Video demonstration and presentation

### 14.3 Development History

Git Commit History (Chronological Order):

1. **6294897** - Data files and report 1
2. **037d264** - Implement data loading, validation, and spectral preprocessing pipeline
3. **81d3c05** - Implement core ML models and training orchestration
4. **e40f2ff** - Implement model evaluation, analysis
5. **e83ac8b** - Implement model hyperparameter tuning
6. **4c77ecb** - Develop Streamlit UI, application logic, and backend services
7. **0ea0954** - Readme file for the project
8. **8b4414f** - Changed files, added functionalities
9. **00690fa** - Add files via upload
10. **03a553c** - Add files via upload
11. **2525849** - Reports
12. **9bead82** - Rename Team\_27.pdf to Team27\_Phase1.pdf
13. **c637de7** - Updated functionality, loo cv, robust context, feature importance
14. **784c6f8** - Updated hyperparameter tuning, cubist implementation
15. **a19f221** - updated documentation

Key Development Milestones:

- **Phase 1:** Data pipeline implementation (commits 6294897-811f48f)
- **Phase 2:** Core ML models and training orchestration (commit 81d3c05)
- **Phase 3:** Evaluation and hyperparameter tuning (commits e40f2ff-e83ac8b)
- **Phase 4:** UI development and integration (commit 4c77ecb)
- **Phase 5:** Documentation and finalization (commit 0ea0954)

## 15 References & Resources

### 15.1 Project Links

- GitHub Repository: [https://github.com/a-n-u-p-a-m/Team27\\_Spectral\\_Soil\\_Modeler](https://github.com/a-n-u-p-a-m/Team27_Spectral_Soil_Modeler)
- Video Demo: <https://youtu.be/sZt-QW2WuHM>
- Documentation: Comprehensive README.md in repository

### 15.2 Libraries & Tools

- scikit-learn: <https://scikit-learn.org>
- Streamlit: <https://streamlit.io>
- Plotly: <https://plotly.com>
- Google Generative AI: <https://ai.google.dev>
- OpenAI API: <https://platform.openai.com>

## 16 Appendix: Code Examples

### 16.1 Example 1: Basic Training Script

Listing 8: Train Standard Model

```

1  from data_loader import DataLoader
2  from preprocessing import SpectralPreprocessor
3  from models.model_trainer import ModelTrainer
4
5  # Load data
6  loader = DataLoader()
7  data = loader.load_data("soil_data.csv")
8  loader.set_target_column("organic_carbon")
9  X_train, X_test, y_train, y_test = loader.split_train_test()
10
11 # Preprocess
12 preprocessor = SpectralPreprocessor()
13 X_train_prep = preprocessor.fit_transform(
14     X_train,
15     technique='absorbance',
16     scaler='standard'
17 )
18 X_test_prep = preprocessor.transform(X_test)
19
20 # Train models
21 trainer = ModelTrainer(tune_hyperparameters=False)
22 for model_name in trainer.models.keys():
23     trainer.train_single_model(
24         model_name,
25         X_train_prep, y_train,
26         X_test_prep, y_test
27     )
28
29 # Get results
30 leaderboard = trainer.get_leaderboard()
31 print(leaderboard)

```

### 16.2 Example 2: Hyperparameter Tuning

Listing 9: Tune GBRT Model

```

1  from models.gbdt import GBRTModel
2  from models.hyperparameter_tuner import HyperparameterTuner
3
4  # Initialize tuner
5  tuner = HyperparameterTuner(
6      model_name='GBRT',
7      cv_folds=5,
8      search_type='grid',
9      use_small_grid=True
10 )
11
12 # Create base model
13 base_model = GradientBoostingRegressor()
14
15 # Tune
16 results = tuner.tune_model(base_model, X_train, y_train)
17
18 print(f"Best parameters: {results['best_params']}")
19 print(f"Best CV score: {results['best_score']:.4f}")
20
21 # Train final model with best params
22 final_model = GBRTModel(**results['best_params'])
23 final_model.train(X_train, y_train)

```

### 16.3 Example 3: Making Predictions

Listing 10: Load Model and Predict

```
1 from system import ModelPersistence
2
3 # Load saved model
4 persister = ModelPersistence(model_dir=".(saved_models")
5 model = persister.load_model("Tuned_GBR_absorbance.joblib")
6 metadata = persister.load_metadata(
7     "Tuned_GBR_absorbance_metadata.json"
8 )
9
10 # Load new data
11 new_data = pd.read_csv("new_soil_samples.csv")
12
13 # Preprocess (use same technique as training)
14 preprocessor = SpectralPreprocessor()
15 new_data_prep = preprocessor.fit_transform(
16     new_data.values,
17     technique=metadata['technique'],
18     scaler='standard'
19 )
20
21 # Predict
22 predictions = model.predict(new_data_prep)
23
24 # Save predictions
25 results_df = pd.DataFrame({
26     'Sample_ID': range(1, len(predictions) + 1),
27     'Predicted_Value': predictions
28 })
29 results_df.to_csv("predictions.csv", index=False)
```

## 17 Conclusion

The Spectral Soil Modeler represents a comprehensive, production-ready solution for soil property prediction from spectral data. Key accomplishments include:

- **Automated Pipeline:** 15 model-technique combinations trained automatically
- **Dual Paradigm Training:** Standard (fast) and Tuned (optimized) with direct comparison
- **Feature Engineering:** 6 sophisticated techniques (Derivatives, Statistical, Polynomial, Indices, PCA, Wavelet) with automatic context integration
- **Data Consistency:** All paradigms share identical dataset context (Phase 2 enhancement)
- **Comprehensive Feature Importance:** 30 combinations analyzed in comparison mode
- **Intelligent Optimization:** 96% hyperparameter search space reduction
- **User-Friendly Interface:** Streamlit-based GUI for non-programmers
- **AI-Powered Insights:** LLM integration with complete context (data + training + FE)
- **Robust Evaluation:** Comprehensive metrics and cross-validation
- **Production-Ready:** Model persistence, logging, and full documentation

### Phase 2 Key Improvements:

- Feature Engineering Framework: 6 techniques with automatic statistics calculation
- Data Analytics Context: Integrated into all paradigm modes for consistency
- Comparison Mode Enhancement: All 30 combinations with feature importance and FE stats
- LLM Context Unification: Consistent dataset understanding across paradigms

The system successfully bridges the gap between complex machine learning methodologies and practical soil science applications, enabling researchers and practitioners to leverage advanced spectroscopy techniques without requiring deep technical expertise.

### Future Enhancements:

- Deep learning models (1D CNN, LSTM)
- Ensemble methods (stacking, blending)
- Real-time model retraining
- Mobile application deployment
- Cloud-based inference API

### Team 27

Anupam Dwivedi, Devansh Singh, Shobhan Parida, Krishna Pokuri, Yashwanth B K

IIIT Hyderabad

December 2025