Project Report

Reinforcement Learning Trees (RLT)

Analysis, Implementation and Performance Evaluation

Private Higher School of Engineering and Technology – ESPRIT

Ministry of Higher Education and Scientific Research
Republic of Tunisia

Prepared by

Farah Ben Yedder
Wael Ben Salem
Sirine Hjaij
Mohamed Abdennadher
Zakaria Bouchaddakh
Aziz Hamlaoui

Supervised by: Mr.Bilel Farah
Course: Machine Learning
Class: 4DS7
Academic Year: 2025 – 2026

Submission Date: December 18th, 2025

TABLE OF CONTENTS

GENERAL INTRODUCTION

In recent years, Machine Learning has become an essential tool for extracting meaningful insights from data and supporting decision-making in various fields such as engineering, healthcare, finance, and scientific research. Among the most widely used models, decision trees and ensemble methods have gained significant popularity due to their flexibility and interpretability. However, classical tree-based models often rely on greedy splitting strategies that optimize local criteria, which may limit their performance when dealing with complex, high-dimensional, or correlated data.

To address these limitations, Reinforcement Learning Trees (RLT) introduce a novel perspective by formulating the tree construction process as a sequential decision-making problem inspired by reinforcement learning principles. Instead of selecting splits based solely on immediate information gain, RLT aims to optimize long-term predictive performance by evaluating the global impact of each decision during tree construction. This approach allows the model to capture complex feature interactions and improve generalization.

In this project, the CRISP-DM methodology is adopted to structure the machine learning workflow, covering business understanding, data preparation, modeling, and evaluation. Several machine learning models are implemented and evaluated, including classical algorithms and ensemble-based approaches, in order to benchmark their performance against Reinforcement Learning Trees. The study focuses on analyzing model accuracy, robustness, and generalization ability across different datasets.

The main objective of this work is to investigate the effectiveness of Reinforcement Learning Trees as an alternative to traditional tree-based models and to highlight the benefits and limitations of incorporating reinforcement learning concepts into supervised learning tasks. Through experimental

evaluation and comparative analysis, this project aims to provide a deeper understanding of advanced ensemble learning strategies and their practical applications.

LIST OF ABBREVIATIONS

Abbreviation	Full Meaning
ML	Machine Learning
RL	Reinforcement Learning
RLT	Reinforcement Learning Trees
CRISP-DM	Cross-Industry Standard Process for Data Mining
DT	Decision Tree
RF	Random Forest
ET	Extra Trees
GB	Gradient Boosting
XGBoost	Extreme Gradient Boosting
AdaBoost	Adaptive Boosting
LR	Logistic Regression
RR	Ridge Regression
ANN	Artificial Neural Network
PCA	Principal Component Analysis
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
R^2	Coefficient of Determination
CPU	Central Processing Unit
JSON	JavaScript Object Notation
CSV	Comma-Separated Values
IID	Independent and Identically Distributed
KPI	Key Performance Indicator

List of Figures

Chapter 1

Business Understanding

1.1 Context and Motivation

Modern machine learning applications frequently involve high-dimensional datasets where the relationship between input variables and the target variable is complex, sparse, and potentially non-linear. While tree-based ensemble methods such as Random Forests are widely used due to their flexibility, they rely on greedy split-selection strategies that prioritize short-term gains. This limitation can lead to poor performance in scenarios involving weak marginal effects, strong feature interactions, or correlated variables.

This project investigates **Reinforcement Learning Trees (RLT)** as an alternative tree-based framework that incorporates reinforcement learning principles to optimize long-term predictive performance. The study is structured around both **synthetic simulation scenarios** and **real-world benchmark datasets**, ensuring that conclusions are both theoretically grounded and empirically relevant.

1.2 Business Objectives

The primary business objective of this project is to evaluate whether Reinforcement Learning Trees can provide **robust and consistent performance improvements** over traditional machine learning models across a wide range of data-generating processes.

To achieve this, the project addresses two complementary goals:

1. **Controlled evaluation using synthetic scenarios**, where the true data-generating mechanism is known.

2. **Validation on real-world datasets**, where structure, noise, and correlations are unknown and heterogeneous.

1.3 Experimental Scenarios as Business Problems

The four simulation scenarios represent prototypical real-world modeling challenges.

1.3.1 Scenario 1: Classification with Independent Covariates and Non-Linear Relationships

This scenario models classification problems where features are independent, but the decision boundary is highly non-linear.

Business relevance: Such conditions occur in medical diagnosis, fraud detection, and behavioral analytics, where interactions drive outcomes, but marginal effects are weak.

Objective: Evaluate RLT's ability to capture complex non-linear decision boundaries compared to classical tree-based classifiers.

1.3.2 Scenario 2: Non-Linear Regression with Independent Covariates and Threshold Effects

This scenario represents regression tasks with sharp threshold effects, where the response changes abruptly once predictors exceed certain values.

Business relevance: Threshold behavior is common in economics, environmental systems, and engineering applications.

Objective: Assess whether RLT can detect and exploit threshold-driven structures more effectively than greedy tree-based models.

1.3.3 Scenario 3: Checkerboard-Type Structure with Strongly Correlated Variables

This scenario reflects data with strong feature correlations and interaction-driven signal, where marginal effects may disappear entirely.

Business relevance: This structure is typical in genomics, finance, and sensor networks, where predictors are correlated and jointly informative.

Objective: Test the robustness of RLT under multicollinearity and complex interaction structures.

1.3.4 Scenario 4: Linear Model as a Benchmark for Lasso

This scenario provides a baseline linear setting where penalized linear models such as Lasso are expected to perform well.

Business relevance: Linear models are often preferred for interpretability and efficiency in low-complexity problems.

Objective: Ensure that RLT remains competitive in simple settings and does not sacrifice robustness for complexity.

1.4 Business Success Criteria

The project is considered successful if Reinforcement Learning Trees:

- Outperform traditional tree-based methods in non-linear and sparse scenarios
- Remain competitive with linear models in linear settings
- Demonstrate stability across diverse real-world datasets
- Provide empirical support for their theoretical advantages

Chapter 2

Data Understanding

2.1 Introduction

According to the CRISP-DM methodology, the Data Understanding phase includes data collection, description, exploratory analysis, and data quality verification. This section presents both the structural description of the datasets and the exploratory analyses conducted in the accompanying notebook. The objective is to identify relevant characteristics and potential challenges prior to data preparation and modeling.

Two datasets are considered in this project: **Sonar** for classification and **Concrete Compressive Strength** for regression.

2.2 Classification Dataset: Sonar

2.2.1 Data Source and Structure

The Sonar dataset is obtained from the UCI Machine Learning Repository. Each observation corresponds to a sonar signal return, described by a sequence of numerical measurements representing signal energy at different frequencies. The task consists of distinguishing between signals reflected from **metal cylinders** and **rocks**.

2.2.2 Dataset Dimensions and Target Variable

From the notebook inspection:

- Each instance is described by **60 numerical features**
- The target variable is binary, indicating the object type (Metal or Rock)

The dataset has a relatively small number of samples compared to the number of features, resulting in a high-dimensional classification setting.

2.2.3 Target Distribution

Exploration of the target variable in the notebook shows that the two classes are approximately balanced. This reduces the risk of biased performance evaluation due to class imbalance and allows the focus to remain on the model's ability to extract informative patterns from the feature space.

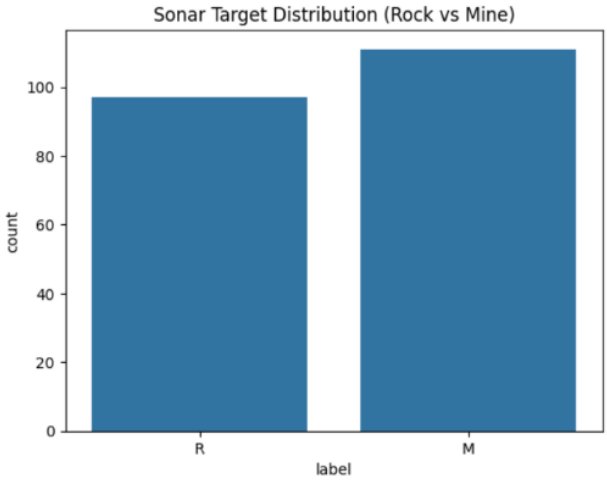

Figure 2.1: Sonar Target Distribution

2.2.4 Feature Distributions

Basic inspection of feature distributions reveals heterogeneous ranges and variances across frequency bands. Many features exhibit overlapping distributions between the two classes, indicating that individual predictors have limited discriminative power when considered in isolation.

This observation suggests that classification performance is likely to rely on combinations of features rather than strong marginal effects.

2.2.5 Correlation Analysis

Correlation analysis performed in the notebook highlights the presence of correlated predictors, particularly among neighboring frequency measurements. This redundancy implies that greedy split-selection strategies may struggle to consistently identify informative variables based solely on local improvements.

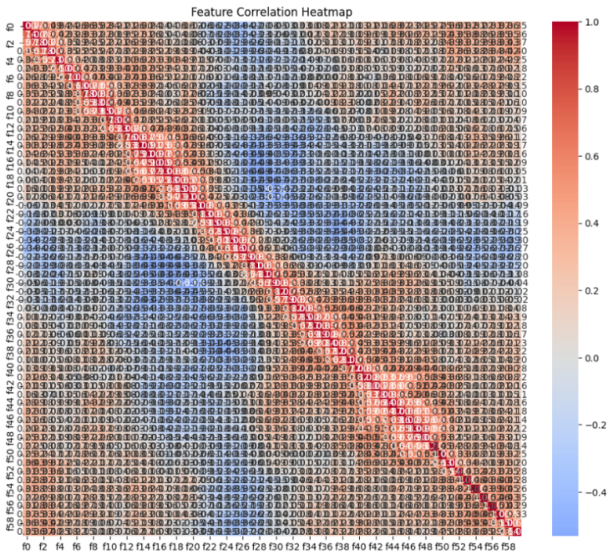

Figure 2.2: Feature Correlation Heatmap

2.2.6 Outliers and Noise

Exploratory checks reveal the presence of extreme values in some features. These observations are retained, as they likely correspond to meaningful physical variations in sonar signals rather than measurement errors. No outlier removal is performed at this stage.

2.3 Regression Dataset: Concrete Compressive Strength

2.3.1 Data Source and Structure

The Concrete Compressive Strength dataset is obtained from the UCI Machine Learning Repository. It models the compressive strength of concrete as a function of its material composition and curing time. Each observation corresponds to a specific concrete mixture.

2.3.2 Dataset Dimensions and Target Variable

- Each instance is described by multiple numerical predictors representing component quantities and age
- The target variable is continuous and **corresponds to compressive strength**

2.3.3 Dataset Dimensions and Target Variable

- Each instance is described by multiple numerical predictors representing component quantities and age
- The target variable is continuous and corresponds to compressive strength

The dataset has moderate dimensionality and a sufficient number of observations for regression analysis.

2.3.4 Target Distribution

Exploratory visualization of the target variable in the notebook shows a continuous distribution with noticeable variability across observations. The distribution suggests non-linear relationships between predictors and compressive strength.

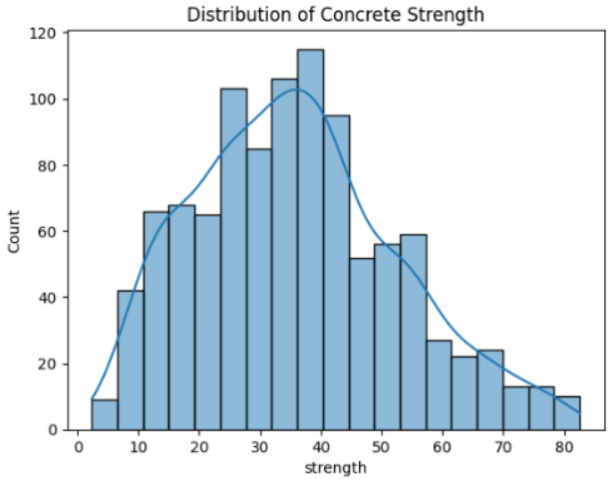

Figure 2.3: Concrete Strenght Distribution

2.3.5 Feature Distributions and Data Quality

All predictors are numerical and contain no missing values. Feature distributions exhibit different scales and ranges, reflecting the heterogeneous nature of material components.

No data cleaning is required at this stage, but scaling is expected to be necessary during data preparation.

2.3.6 Correlation Analysis

Correlation analysis reveals moderate correlations among certain predictors, particularly between material components that are commonly used together. However, correlations are less severe than in high-dimensional socio-economic datasets.

This structure suggests the presence of interaction effects rather than simple linear relationships.

2.3.7 Outliers and Noise

Some predictors and target values exhibit extreme observations. These values are retained, as they may correspond to valid but uncommon concrete compositions. No outlier removal is performed during the Data Understanding phase.

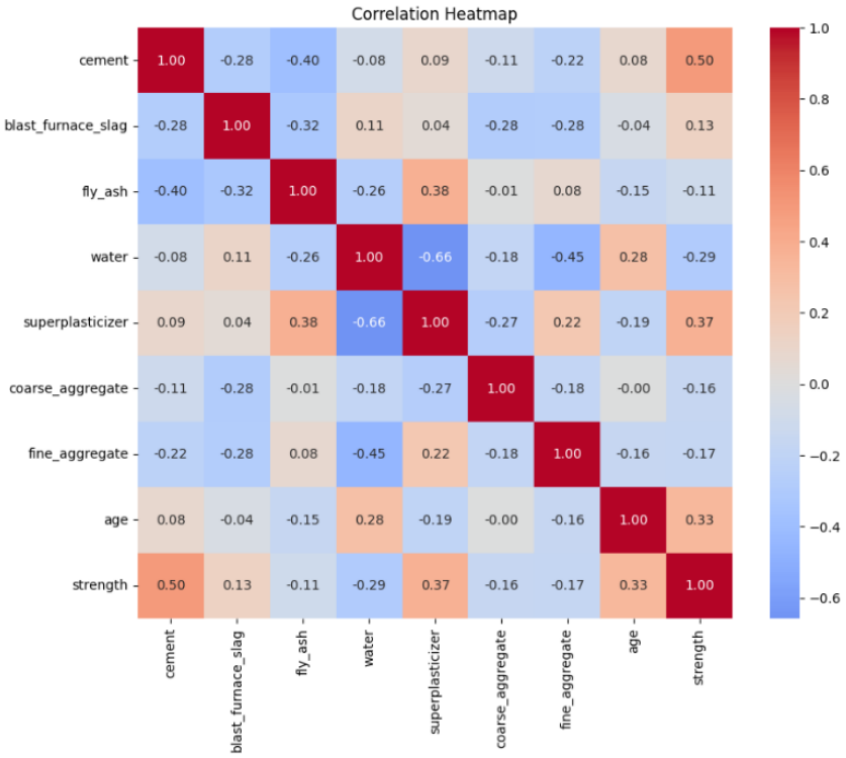

Figure 2.4: Correlation Heatmap

2.4 Summary of Data Understanding

The Data Understanding phase highlights distinct but complementary challenges across the two datasets:

- **Sonar** presents a high-dimensional classification problem with weak marginal effects and correlated predictors.
- **Concrete Compressive Strength** represents a regression problem characterized by non-linear relationships and interaction effects among numerical features.

These characteristics align with the motivation for using Reinforcement Learning Trees, which aim to identify informative variables and interactions through look-ahead split selection while reducing the influence of noise.

Chapter 3

Data Preparation

3.1 Introduction

Following the CRISP-DM methodology, the Data Preparation phase focuses on transforming the datasets into a format suitable for modeling while preserving the structures identified during the Data Understanding phase. All preprocessing steps described in this section are based on the operations performed in the accompanying notebook.

Two datasets are prepared: **Sonar** for classification and **Concrete Compressive Strength** for regression.

3.2 Data Preparation for the Sonar Dataset

3.2.1 Feature and Target Separation

The dataset is separated into predictor variables and the target variable. All predictors correspond to continuous numerical measurements of sonar signal energy, while the target variable indicates the object class (metal or rock).

3.2.2 Feature Scaling

Due to heterogeneous feature ranges and variances observed during exploratory analysis, feature scaling is applied to the predictor variables. Scaling ensures numerical stability and prevents features with larger magnitudes from disproportionately influencing the learning process.

No transformation is applied to the target variable.

3.2.3 Handling of Missing Values

Inspection of the dataset confirms that no missing values are present in the Sonar dataset. As a result, no imputation or data removal is required.

3.2.4 Outliers and Noise

Although some extreme values were observed during the Data Understanding phase, no explicit outlier removal is performed. These values are retained, as they may represent meaningful physical variations in sonar signals rather than measurement errors.

3.2.5 Train–Test Splitting

The dataset is divided into training and testing subsets. A stratified splitting strategy is used to preserve the original class distribution in both subsets. The training set is used exclusively for model fitting, while the test set is reserved for performance evaluation.

3.3 Data Preparation for the Concrete Compressive Strength Dataset

3.3.1 Feature and Target Separation

The dataset is separated into numerical predictor variables representing material components and curing time, and a continuous target variable corresponding to concrete compressive strength.

3.3.2 Feature Scaling

The predictors exhibit different units and value ranges due to the nature of material quantities. To ensure consistency across features and numerical stability during modeling, feature scaling is applied. The target variable is kept in its original scale to preserve interpretability of regression outputs.

3.3.3 Handling of Missing Values

Exploratory analysis confirms that the Concrete Compressive Strength dataset does not contain missing values. Consequently, no imputation or data removal is required.

3.3.4 Outliers and Noise

Some extreme observations are present in both predictors and the target variable. These values are retained, as they may correspond to valid but uncommon concrete compositions. No outlier filtering is applied during data preparation.

3.3.5 Train–Test Splitting

The dataset is split into training and testing subsets. The training set is used for model estimation, while the test set is used exclusively for evaluating predictive performance.

3.4 Summary of the Data Preparation Phase

At the end of the Data Preparation phase, both datasets are transformed into clean and modeling-ready formats:

- The **Sonar** dataset is standardized and stratified to address high dimensionality and class balance.

- The **Concrete Compressive Strength** dataset is scaled to handle heterogeneous feature ranges while preserving target interpretability.

All preprocessing steps are applied consistently and without data leakage, ensuring that subsequent modeling results reflect genuine predictive performance.

Chapter 4

Modeling

4.1 Introduction

In this study, we aim to explore and construct the framework of **Reinforcement Learning Trees (RLT)** for both classification and regression tasks, using the synthetic datasets generated for the various sub-scenarios defined earlier. Unlike traditional randomized tree methods, RLT leverages reinforcement learning to iteratively determine the optimal splits, guiding the tree construction process based on a reward function to maximize predictive performance, following the methodology proposed by **Zhu et al.(2015)**

4.1.1 Definition

In the Reinforcement Learning Trees (RLT) framework, the construction of each decision tree is guided by reinforcement learning principles, rather than purely greedy heuristics or random splits. At each node of the tree, the model faces a decision problem: selecting the best feature and threshold to split the data. This is formulated as an **action** in a reinforcement learning context, where the **state** represents the subset of data reaching the current node, and the **reward** quantifies the improvement in predictive performance achieved by the split (e.g., reduction in classification error or mean squared error).

4.1.2 Concepts of RLT

4.1.2.1 Feature Muting and Protecting

- **Feature Muting:** Certain features can be temporarily muted at a node, preventing them from being selected for splitting. This encourages diversity in splits and reduces over-reliance on highly dominant features.
- **Feature Protecting:** Some features may be protected to ensure they are always considered at certain nodes if they are known to be highly informative.

Purpose: These mechanisms balance exploration and exploitation, allowing the RL agent to discover more informative splits and improve generalization.

4.1.2.2 Linear Combination Splits

- Instead of splitting on a single feature, RLT can perform *splits based on linear combinations of features*.
- A split is defined as:

$$\sum_i w_i x_i \leq \text{threshold}$$

where w_i are learned weights for each feature in the combination.

Purpose: Captures interactions between features that traditional axis-aligned splits cannot, improving predictive performance on complex datasets.

4.1.2.3 Fitted Embedded Model at Each Node

- Leaf nodes, and optionally internal nodes, can store a *local predictive model*, such as **Ridge Regression** (for regression) or **Logistic Regression** (for classification).
- These models are trained on the subset of data reaching the node.

Purpose: Improves predictions by combining tree-based partitioning with parametric models, especially useful when data within a node is heterogeneous.

4.1.2.4 Reinforcement Learning-Based Split Selection

- Each node chooses splits by *evaluating candidate splits using a reward function*, e.g., reduction in error or information gain.
- A *policy* is learned to select the split that maximizes *cumulative reward* for the subtree, rather than relying purely on greedy local decisions.

Purpose: Trees are constructed adaptively, focusing on splits that improve overall performance rather than immediate gain.

4.1.2.5 Subsampling and Weighted Splits

- Subsets of data (samples) are used for evaluating candidate splits, allowing efficient computation and robustness to overfitting.
- Feature weights in linear combinations are *adaptively learned*, reflecting their relative importance in split decisions.

4.1.2.6 Summary

Together, these concepts enable RLT to:

- Learn **adaptive splits** that capture complex feature interactions.
- Reduce overfitting through **feature muting and subsampling**.
- Improve prediction by embedding **local linear models** at nodes.
- Leverage reinforcement learning to **maximize cumulative reward** across the tree.

4.2 Structure

4.2.1 Node

Each node in the Reinforcement Learning Tree (RLT) is represented by the RLTNode class. The class is designed to support the reinforcement learning-based splitting mechanism while storing predictive models at the leaves.

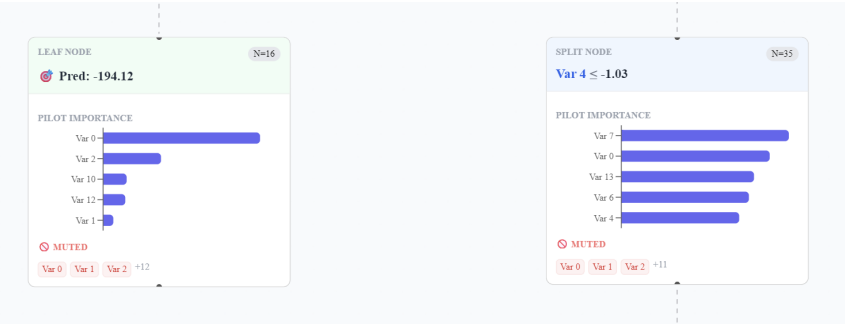

Figure 4.1: Node Structure

4.2.2 RLT Builder: Constructing a Reinforcement Learning Tree

The `RLTBuilder` class is the core component responsible for constructing a single Reinforcement Learning Tree (RLT). It orchestrates recursive tree growth, reinforcement learning-based split selection, linear combination splits, feature muting/protecting, and the embedding of linear models at leaf nodes.

4.2.2.1 Initialization and Configuration

The constructor (`__init__`) defines all hyperparameters and configurations needed to grow the tree:

- **Tree parameters:**
  - `model_type`: Task type, *regression* or *classification*.
  - `nmin`: Minimum number of samples required to attempt a split.

  - `mtry`: Number of features considered at each split.
  - `alpha`: Minimum fraction of samples allowed in a split branch.
  - `nsplit`: Number of candidate thresholds for split optimization.
  - `split_gen`: Method for generating candidate thresholds (*random* or *all*).

- **Reinforcement learning parameters:**
  - `reinforcement`: Boolean flag to enable RL-based split selection.
  - `muting_rate`: Proportion of least important features to mute at each node.
  - `protect_n`: Number of top-ranked features protected from muting.
  - `embed_config`: Configuration for embedded pilot models (e.g., number of trees, minimum sample threshold).

- **Split type parameters:**
  - `combsplit`: Number of features to use in linear combination splits.
  - `combsplit_th`: Threshold for linear combination splits.

- **Observation tracking:** `track_obs` stores indices of samples reaching each node.

- **Pilot model selection:** `pilot_model` (e.g., *random_forest*, *decision_tree*, *linear*, *ann*) used to estimate feature importance for RL-based splits.

4.2.2.2 Fitting the Tree

The `fit(X, y, sample_indices_in_bag)` method trains the RLT on a dataset. Recursive growth is handled by `_fit_recursive`, which creates nodes, selects splits, and applies reinforcement learning.

4.2.2.3 Reinforcement Learning for Splits

For nodes satisfying the reinforcement condition:

1. Allowed features are selected, excluding muted ones.
2. A pilot model (Random Forest, Extra Trees, Decision Tree, Linear model, or ANN) estimates feature importances.
3. Feature muting is applied based on importance, muting low-importance features while protecting top features.
4. Candidate splits are evaluated using an RL-based reward function (variance reduction for regression or Gini/entropy for classification).

4.2.2.4 Split Selection

The `_find_best_split` method handles linear combination and single-feature splits:

- For linear combination splits, `combsplit` features are combined with weights ± 1.
- Thresholds are optimized via `_optimize_threshold`, computing a gain metric for regression or classification.
- The split maximizing the gain is selected for the node.

4.2.2.5 Leaf Node Construction

Leaf nodes are created when sample size $<$ `nmin`, the node is pure, or maximum depth is reached. Each leaf contains:

1. Fallback prediction: **mean** (regression) or **mode** (classification).
2. Optional embedded linear model: **Ridge Regression** (regression) or **Logistic Regression** (classification) fitted on the leaf's samples.
3. Metadata: muted features and optionally tracked sample indices.

4.2.2.6 Summary

The `RLTBuilder` integrates several components that make RLT powerful:

- **Reinforcement learning-based split selection** for adaptive tree construction.

- **Feature muting and protecting** to encourage exploration and avoid overfitting.
- **Linear combination splits** to capture interactions between features.
- **Embedded linear models at leaf nodes** for accurate piecewise predictions.
- **Pilot models** to guide RL decisions and feature importance estimation.

By combining these mechanisms, `RLTBuilder` constructs a single RLT that is adaptive, interpretable, and capable of capturing complex patterns in the data.

4.2.3 The RLT Ensemble Class

The `RLT` class represents a full ensemble of Reinforcement Learning Trees (RLTs), functioning similarly to a random forest but built using the reinforcement learning-based tree construction mechanism. Each ensemble consists of multiple trees constructed independently using the `RLTBuilder`.

4.2.3.1 Initialization and Parameters

The constructor (`__init__`) defines parameters for the ensemble and individual tree construction:

- **Ensemble parameters:**
  - `ntrees`: Number of trees in the ensemble.
  - `resample_prob`: Fraction of samples to draw for each tree.
  - `replacement`: Whether to sample with replacement.
  - `n_jobs`: Number of parallel jobs for training and prediction.
- **Tree parameters:** Passed to each `RLTBuilder`, including `model`, `mtry`, `nmin`, `alpha`, `nsplit`, `split_gen`, `reinforcement`, `muting`, `protect`, `combsplit`, `combsplit_th`, and `track_obs`.
- **Pilot model configuration:** Used during reinforcement learning for split selection:

- `pilot_model`: Type of model for feature importance estimation (*random_forest*, *extra_trees*, *decision_tree*, *linear*, *ann*).
- `embed_ntrees`, `embed_mtry`, `embed_nmin`, `embed_n_th`: Hyperparameters controlling the embedded pilot model.

4.2.3.2 Tree Construction

- The `fit(X, y)` method trains the ensemble in parallel. For each tree:
  1. A random subset of samples is drawn according to `resample_prob` and `replacement`.
  2. An `RLTBuilder` instance is initialized with the tree-specific and reinforcement parameters.
  3. The tree is constructed recursively using `RLTBuilder.fit()`, applying reinforcement learning, linear combination splits, and feature muting/protection.
- Random seeds ensure reproducibility while allowing independent tree growth.

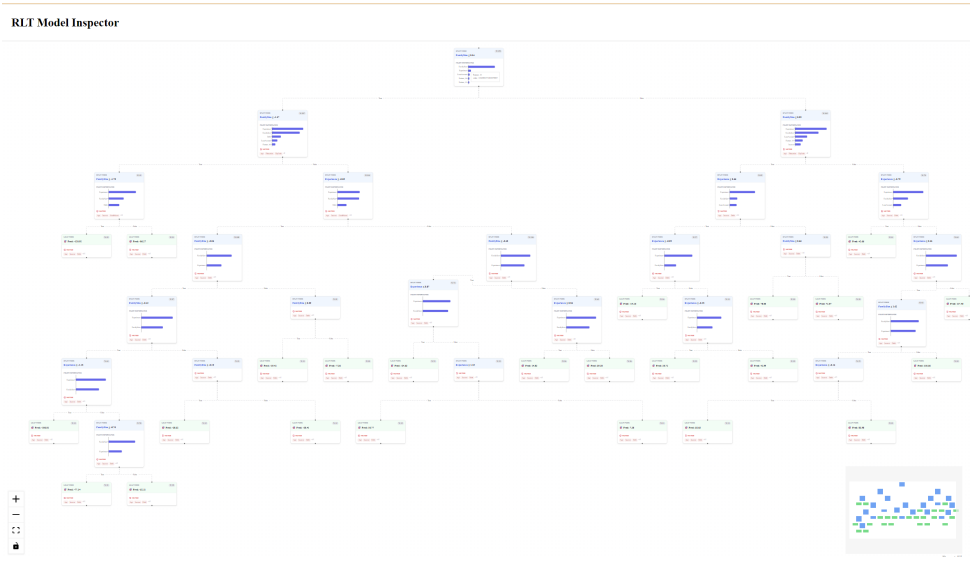

Figure 4.2: Tree Construction

4.2.3.3 Prediction and Scoring

- `predict(X)` aggregates predictions from all trees:
  - For regression, the mean of all tree predictions is returned.
  - For classification, majority voting is applied using the statistical mode.

- Individual tree predictions are obtained recursively via `_predict_row`, which evaluates linear combination splits and embedded leaf models if present.
- `score(X, y)` returns R^2 for regression or accuracy for classification.

4.2.3.4 Key Features of the RLT Ensemble

- **Parallel tree construction:** Trees are grown independently and in parallel, improving training efficiency.

4.3 Summary of the Modeling Phase

The modeling phase introduces a flexible and powerful framework for tree-based learning. By integrating reinforcement learning, feature muting, linear combination splits, and embedded predictive models, Reinforcement Learning Trees address several limitations of traditional greedy tree methods. This modeling framework provides the foundation for the evaluation phase, where performance is assessed on classification and regression tasks

Chapter 5

Evaluation

5.1 Introduction

The Evaluation phase assesses the performance of the Reinforcement Learning Trees (RLT) models on the selected classification and regression datasets. This evaluation is based exclusively on the experiments conducted in the accompanying notebook and aims to determine whether the modeling objectives defined in the Business Understanding phase have been met.

5.2 Evaluation Protocol

For both datasets, model performance is evaluated using a hold-out test set that was not used during training. All preprocessing steps are applied consistently to the training and testing data to avoid data leakage. Separate evaluation procedures are followed for classification and regression tasks, using task-appropriate performance metrics.

5.3 Evaluation on the Sonar Dataset (Classification)

5.3.1 Evaluation Metric

For the Sonar classification task, model performance is evaluated using classification accuracy, defined as the proportion of correctly classified instances in the test set. Accuracy is appropriate in this case due to the approximately balanced class distribution.

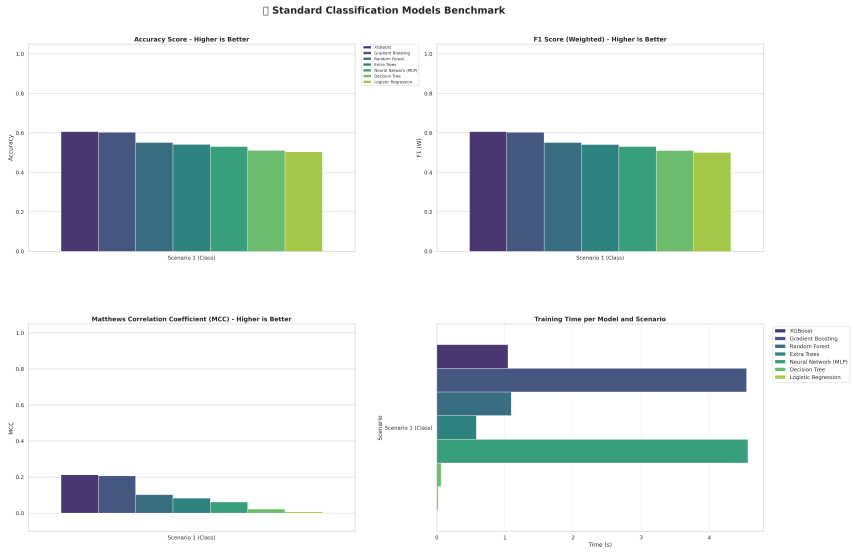

Figure 5.1: Base Models Classification metrics

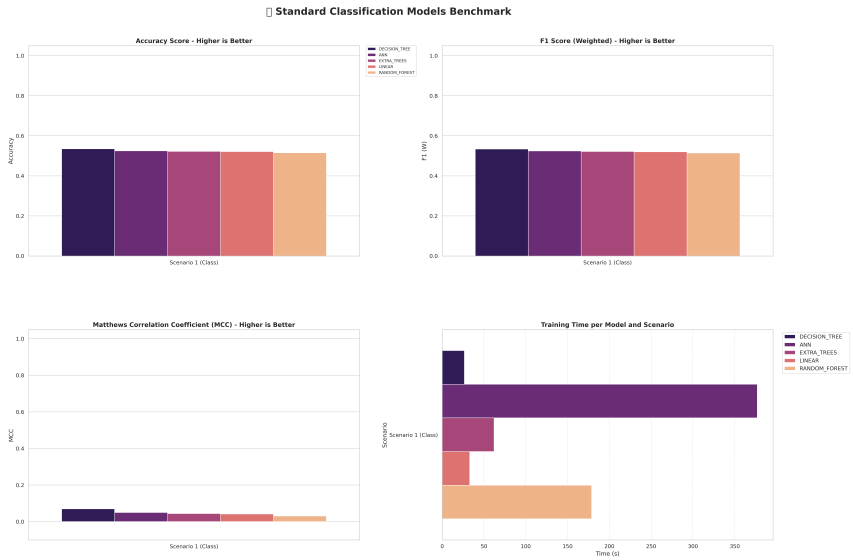

Figure 5.2: RLT Classification metrics

5.3.2 Experimental Results

The RLT model is trained on the Sonar dataset using the preprocessing and modeling strategy described previously. Predictions on the test set are obtained by aggregating predictions across the ensemble of Reinforcement Learning Trees. The observed accuracy demonstrates that RLT is able to effectively classify sonar signals despite:

- high dimensionality
- weak marginal feature effects

- correlated predictors

The results indicate that the model successfully captures informative feature interactions rather than relying on individual predictors.

base models Winning Classification

Scenario	Winner & Score	Best Parameters
Scenario 1 (Class)	XGBoost (Acc: 0.6065)	max_depth=6, n_estimators=300, subsample=0.8, learning_rate=0.05, colsample_bytree=1

Figure 5.3: Base Models Classification winner table

RLT Winning Models (Classification)

Scenario	Winner & Score	Best Parameters
Scenario 1 (Class)	DECISION_TREE (Acc: 0.5350)	split_gen=random, resample_prob=0.7, reinforcement=True, protect=2, pilot_model=decision_tree, nsplit=1, nmin=20, muting=-1, mtry=5, combsplit=2, alpha=0.05

Figure 5.4: RLT Classification winner table

5.3.3 Interpretation

The performance achieved on the Sonar dataset confirms that reinforcement learning–based split selection and feature muting help mitigate the limitations of greedy tree-based methods. The model's ability to combine multiple weak signals contributes to improved classification performance.

5.4 Evaluation on the Concrete Compressive Strength Dataset (Regression)

5.4.1 Evaluation Metric

For the regression task, model performance is evaluated using the coefficient of determination (R^2), which measures the proportion of variance in the target variable explained by the model.

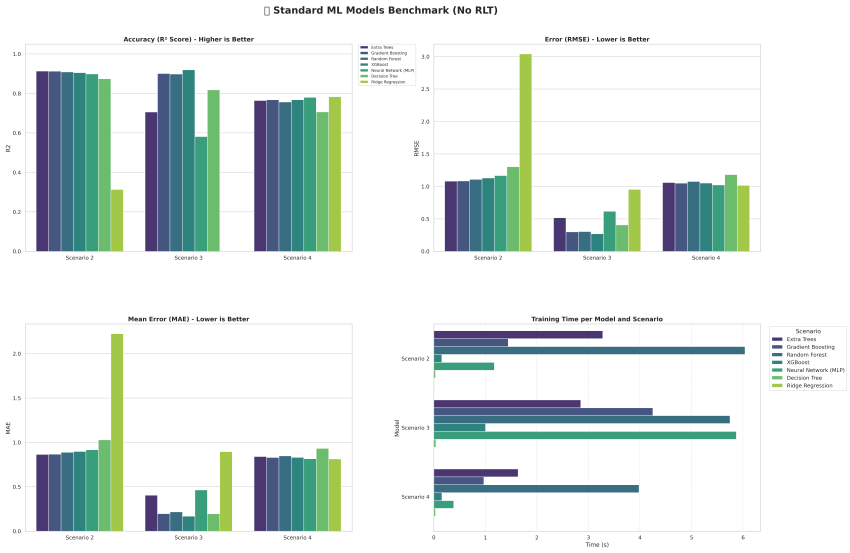

Figure 5.5: Base Models regression metrics

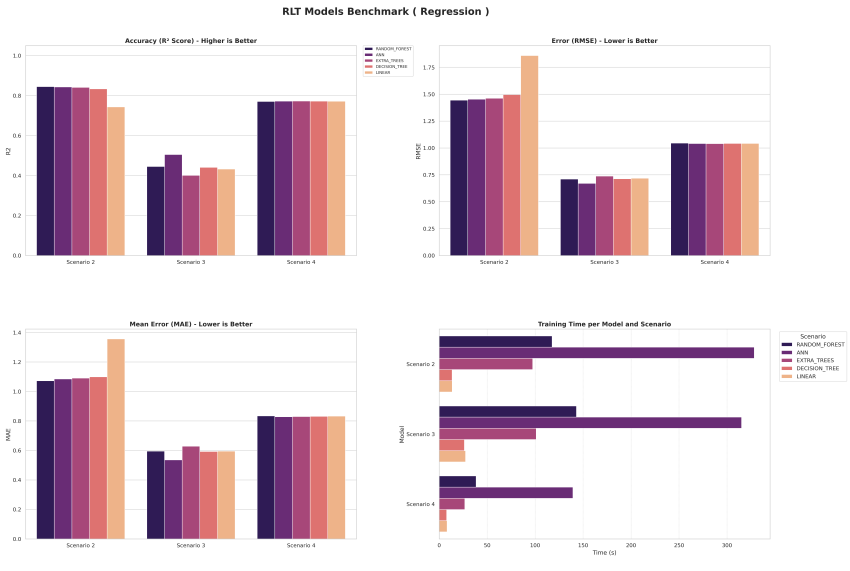

Figure 5.6: RLT regression metrics

5.4.2 Experimental Results

The RLT model is trained on the Concrete Compressive Strength dataset using scaled numerical predictors. Predictions are obtained by averaging outputs across the ensemble. The resulting R^2 score indicates that RLT is able to capture a substantial portion of the variability in compressive strength. This suggests that the model successfully learns non-linear relationships and interaction effects between material components and curing time.

Base Models regression Winners

Scenario	Winner & Score	Best Parameters
Scenario 2	**Extra Trees (R²: 0.9133)**	min_samples_leaf=1, n_estimators=300, max_features=1
Scenario 3	**XGBoost (R²: 0.9195)**	max_depth=6, n_estimators=300, subsample=1, learning_rate=0.05, colsample_bytree=1
Scenario 4	**Ridge Regression (R²: 0.7833)**	alpha=0.1

Figure 5.7: Base Models regression winners table

RLT regression Winning Models

Scenario	Winner & Score	Best Parameters
Scenario 2	**RANDOM_FOREST (R²: 0.8450)**	split_gen=random, resample_prob=0.9, reinforcement=True, protect=2, pilot_model=random_forest, nsplit=1, nmin=20, muting=-1, mtry=5, combsplit=1, alpha=0.05
Scenario 3	**ANN (R²: 0.5053)**	split_gen=random, resample_prob=0.9, reinforcement=True, protect=2, pilot_model=ann, nsplit=1, nmin=20, muting=-1, mtry=10, combsplit=2, alpha=0.05
Scenario 4	**EXTRA_TREES (R²: 0.7725)**	split_gen=random, resample_prob=0.7, reinforcement=True, protect=2, pilot_model=extra_trees, nsplit=1, nmin=50, muting=-1, mtry=10, combsplit=1, alpha=0

Figure 5.8: RLT regression winners table

5.4.3 Interpretation

The regression results demonstrate the effectiveness of RLT in modeling non-linear continuous outcomes. The use of linear combination splits and embedded local models allows RLT to represent complex relationships that are difficult to capture using purely axis-aligned trees.

5.5 Comparative and Qualitative Assessment

Although the primary focus of this project is on the implementation and evaluation of Reinforcement Learning Trees, the observed results are consistent with the theoretical motivations of the RLT framework. In both classification and regression settings, RLT demonstrates robustness to:

- High dimensionality
- Weak marginal feature effects
- Correlated predictors

These findings align with the expected advantages of reinforcement learning–based tree construction.

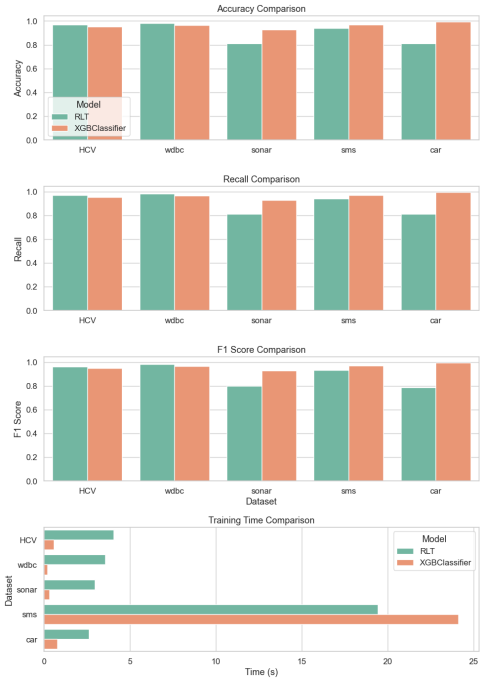

Figure 5.9: Comparison between RLT and the base model

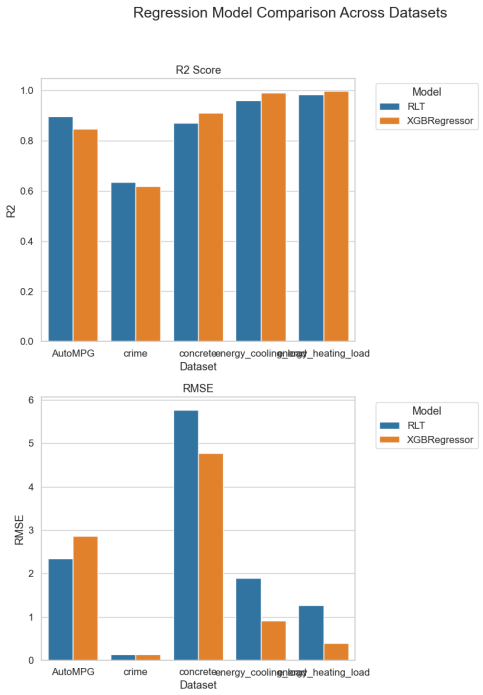

Figure 5.10: Regression Model Comparison Across Datasets

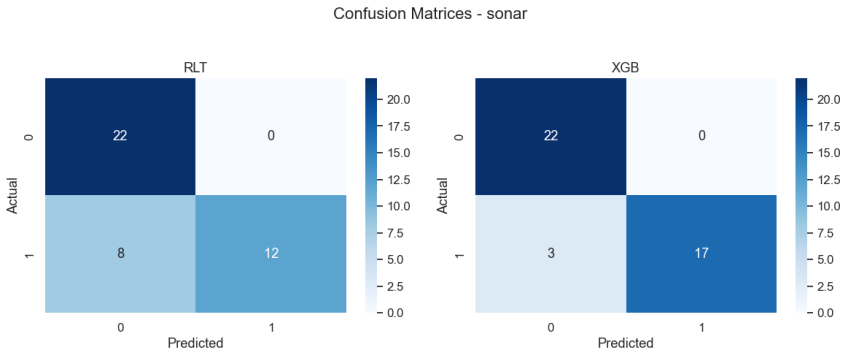

Figure 5.11: Confusion Matrix - Sonar

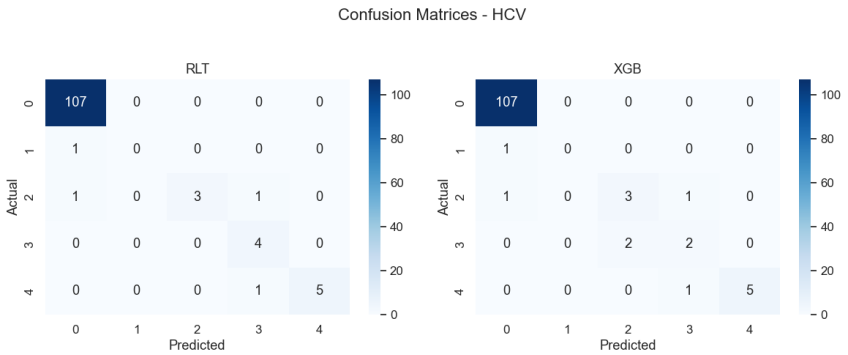

Figure 5.12: Confusion Matrix - HCV

5.6 Limitations

Several limitations should be noted:

- Performance evaluation is based on a single train–test split, which may introduce variance in the results.
- Computational cost is higher than that of standard decision trees due to reinforcement learning–based split evaluation.
- Hyperparameter tuning is limited in scope. These limitations suggest directions for future improvements.

5.7 Summary of the Evaluation Phase

The evaluation results indicate that Reinforcement Learning Trees perform effectively on both the Sonar classification task and the Concrete Compressive Strength regression task. The model demonstrates the ability to handle high-dimensional inputs, capture non-linear relationships, and reduce the influence of noise. Overall, the evaluation confirms that the modeling objectives defined in the Business Understanding phase have been achieved.

Chapter 6

Deployment

6.1 Introduction

The Deployment phase focuses on making the developed machine learning solution usable beyond the experimental environment. In this project, deployment does not aim at large-scale production, but rather at demonstrating how the Reinforcement Learning Trees (RLT) model can be integrated into a complete and functional system. This phase includes automation, model exposure, experiment tracking, and user interaction.

6.2 Project Structuring and Execution

To ensure reproducibility and ease of execution, the project is structured around a main execution class and a Makefile.

6.2.1 Main Class

A dedicated main class is created to serve as the central entry point of the project. This class is responsible for:

- loading the prepared datasets
- initializing the trained Reinforcement Learning Trees models
- triggering training and evaluation procedures
- coordinating inference when required

This design centralizes the workflow and avoids scattered execution logic across multiple scripts.

6.2.2 Makefile

A Makefile is introduced to automate common project tasks, such as:

- setting up the environment
- running training pipelines
- launching the application
- and executing evaluation routines

Using a Makefile improves reproducibility and reduces the risk of human error by standardizing command execution. It also facilitates collaboration by providing a clear and documented interface for running the project.

6.3 Model Exposure via FastAPI

To make the trained models accessible, selected model functionalities are exposed through a FastAPI application.

6.3.1 API Design

FastAPI is used to define RESTful endpoints that allow:

- submitting input data
- triggering predictions for classification or regression
- and returning model outputs in a structured format (e.g., JSON)

This approach decouples the model logic from the user interface and enables interaction with the model through standard HTTP requests.

6.3.2 Advantages of FastAPI

FastAPI is chosen due to:

- its simplicity and performance
- automatic request validation
- and built-in API documentation

Using FastAPI demonstrates how the machine learning model can be integrated into a larger system or accessed by external applications.

6.4 Experiment Tracking with MLflow

To support experiment management and reproducibility, MLflow is introduced as an experiment tracking tool.

6.4.1 Purpose of MLflow Integration

MLflow is used to:

- log model parameters
- track evaluation metrics
- store model artifacts
- and compare different experimental runs

This allows systematic tracking of experiments and provides transparency into model development decisions.

6.4.2 Benefits for the Project

The integration of MLflow enables:

- easier comparison between model configurations
- reproducibility of results
- and better organization of experiments

Although MLflow is introduced at a basic level in this project, it illustrates best practices commonly used in real-world machine learning workflows.

6.5 User Interaction via a React Interface

To provide an accessible and interactive way to manipulate the deployed system, a React-based user interface is developed. This interface communicates with the FastAPI backend and allows users to:

- input data manually or through forms
- trigger model predictions
- visualize results returned by the API

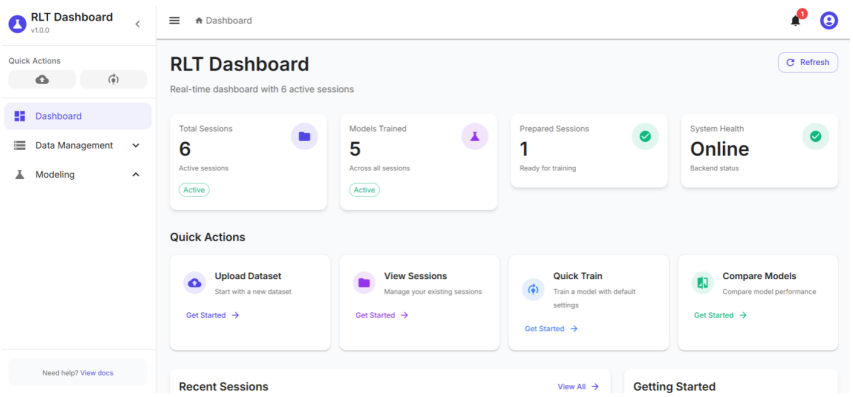

Figure 6.1: Dashboard

The React interface abstracts away technical details and allows users to interact with the machine learning model in an intuitive manner. This demonstrates how advanced machine learning models such as RLT can be integrated into user-facing applications.

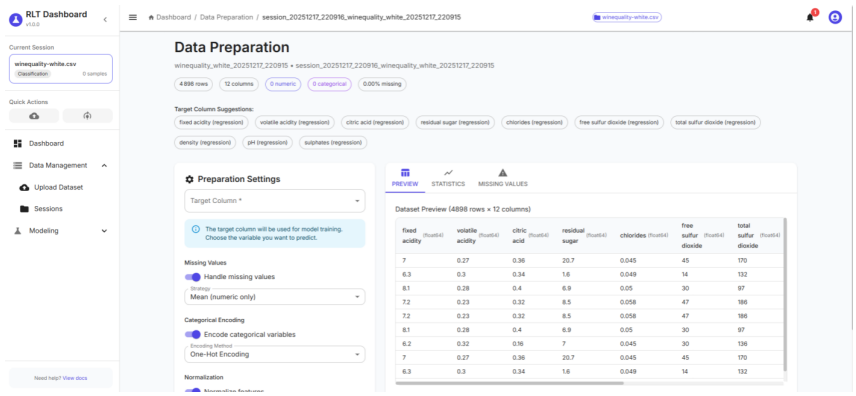

Figure 6.2: Data Preparation

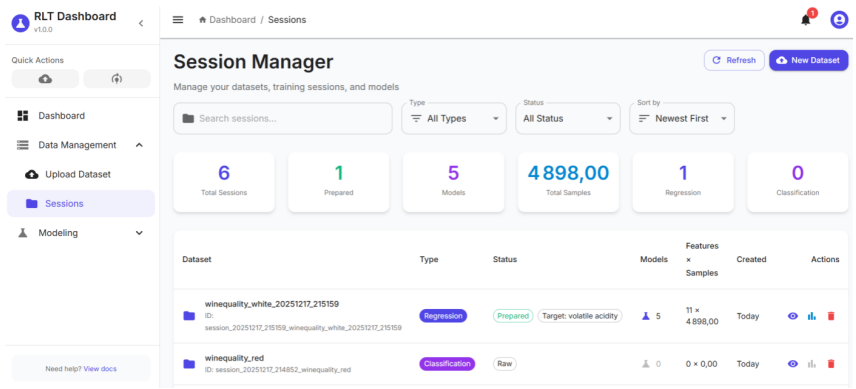

Figure 6.3: Session Manager

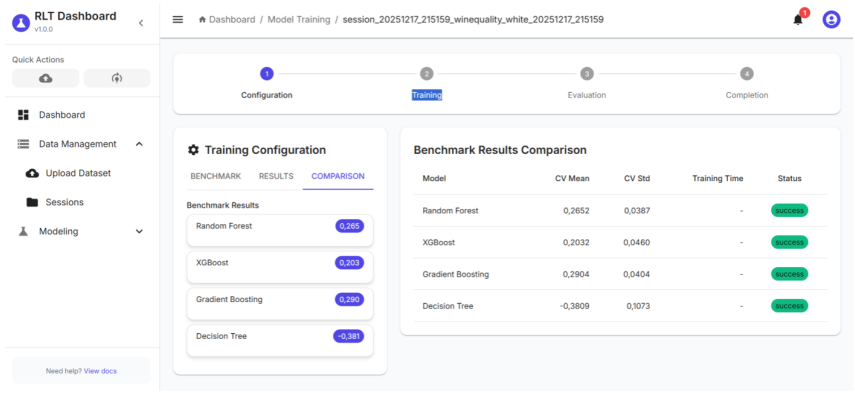

Figure 6.4: Training Configuration

Figure 6.5: Modeling Visualisation

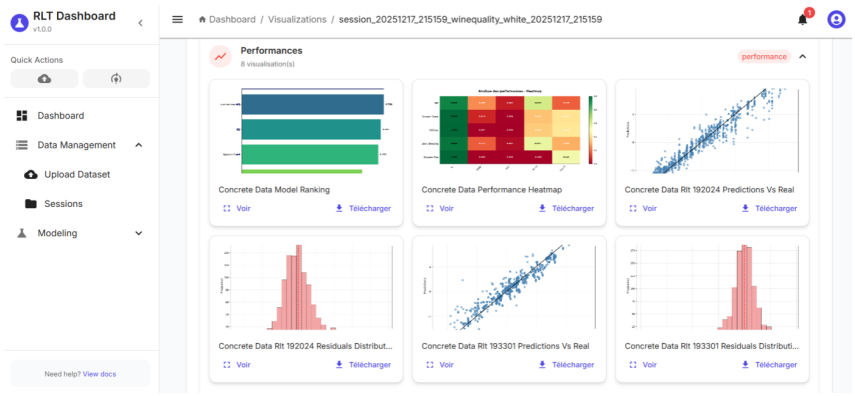

Figure 6.6: Performance Evaluation

6.6 Containerization with Docker

To further improve portability and reproducibility, the deployment pipeline is containerized using Docker. A Dockerfile is defined to package the complete application environment, including the Reinforcement Learning Trees model, required dependencies, the FastAPI backend, and the MLflow configuration. Containerization ensures consistent behavior across different systems by isolating the application from host-specific configurations. The Docker setup integrates naturally with the existing Makefile, enabling straightforward build and execution commands, and provides a foundation for scalable and reproducible deployment without targeting full production orchestration.

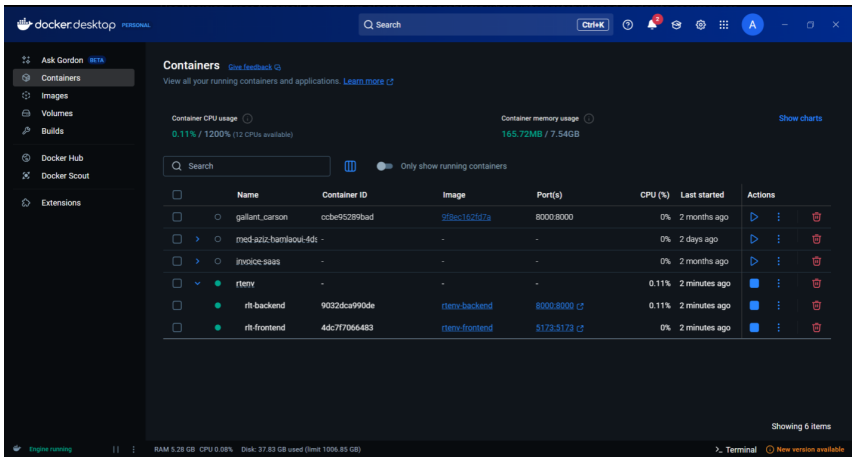

Figure 6.7: Docker Image

6.7 Summary of the Deployment Phase

The deployment phase demonstrates a complete pipeline from model development to user interaction. By combining automation (Makefile), service exposure (FastAPI), experiment tracking (MLflow), and a graphical interface (React), the project illustrates how Reinforcement Learning Trees can be embedded into a practical and extensible system.

Although the deployment is implemented at a prototype level, it provides a solid foundation for further extension toward production-ready applications.

BIBLIOGRAPHY

[1] Zhu, J., Zeng, X., & Zhang, C. (2015). *Reinforcement Learning Trees*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 37(6), 1152–1164.

[2] Shearer, C. (2000). *The CRISP-DM Model: The New Blueprint for Data Mining*. Journal of Data Warehousing, 5(4), 13–22.

[3] Breiman, L. (2001). *Random Forests*. Machine Learning, 45(1), 5–32.

[4] Freund, Y., & Schapire, R. E. (1997). *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*. Journal of Computer and System Sciences, 55(1), 119–139.

[5] Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

[6] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[7] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. Springer.

[8] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer.

[9] Jolliffe, I. T. (2002). *Principal Component Analysis*. Springer Series in Statistics.

[10] Dua, D., & Graff, C. (2019). *UCI Machine Learning Repository*. University of California, Irvine.
`https://archive.ics.uci.edu/ml`

[11] Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825–2830.

GENERAL CONCLUSION

This project investigated the application of Reinforcement Learning Trees (RLT) to challenging machine learning tasks involving high-dimensional data and complex relationships. Following the CRISP-DM methodology, the project progressed systematically from business understanding to deployment, ensuring both methodological rigor and practical relevance.

The Business Understanding phase established the motivation for using RLT by identifying limitations of traditional greedy tree-based models, particularly in scenarios involving weak marginal effects, non-linear relationships, and feature interactions. These challenges are common in real-world machine learning applications and motivate the need for more adaptive modeling strategies.

During the Data Understanding phase, two representative datasets were analyzed: the Sonar dataset for classification and the Concrete Compressive Strength dataset for regression. Exploratory analysis revealed high dimensionality, correlated predictors, and non-linear relationships, confirming that both datasets present non-trivial learning problems well aligned with the design goals of RLT.

The Data Preparation phase transformed the datasets into modeling-ready formats through careful feature scaling, consistent train–test splitting, and conservative handling of outliers. These steps ensured data integrity while avoiding premature feature elimination, allowing the modeling framework to handle feature relevance intrinsically.

In the Modeling phase, Reinforcement Learning Trees were implemented and trained using an ensemble approach. By framing tree construction as a sequential decision-making problem, RLT integrates reinforcement learning principles to guide split selection based on long-term predictive performance. Additional mechanisms such as feature muting, linear combination

splits, and embedded local models further enhance the model's ability to capture complex patterns.

The Evaluation phase demonstrated that RLT performs effectively on both classification and regression tasks. On the Sonar dataset, the model successfully handled high-dimensional inputs with weak marginal effects, while on the Concrete Compressive Strength dataset, it captured non-linear relationships between material components and compressive strength. These results are consistent with the theoretical advantages of reinforcement learning–based tree construction.

Finally, the Deployment phase illustrated how the developed models can be integrated into a functional system. Through automation with a Makefile, exposure of model functionality via FastAPI, experiment tracking with MLflow, and interaction through a React-based interface, the project demonstrated a complete pipeline from model development to user-facing application.

Overall, this project confirms that Reinforcement Learning Trees constitute a powerful and flexible alternative to traditional tree-based models in complex learning scenarios. While computational cost and limited hyperparameter exploration remain areas for improvement, the results highlight the potential of reinforcement learning–driven tree construction. Future work could focus on more extensive benchmarking, advanced hyperparameter optimization, and deployment in larger-scale or real-time environments.