PlantMo: Plant Monitoring

Operational Governance & Versioning Document

Cartago · Tonet · Toso · Zambonini

Document Status. The following document presents a preliminary and evolving version of the operational governance and versioning proposal. The versioning control strategy, model governance, monitoring & maintenance plan described, may be subject to modification as the project advances. Accordingly, this document should be regarded as a working draft rather than a finalized specification.

Contents

1 Version Control Strategy

A well-defined version control strategy is a cornerstone of operational governance in MLOps projects. It enables structured collaboration among multidisciplinary teams, guarantees traceability of changes, and ensures reproducibility across data, code, and models. In this project, version control has been designed as an integrated process covering source code, datasets, machine learning models, and documentation, aligning technical development with governance and accountability requirements.

1.1 Branching Strategy

The project repository is hosted on GitHub and adopts a centralized branching strategy centered on a stable *main* branch. The *main* branch represents the production-ready state of the system and is updated exclusively through controlled merge operations. All development activities are carried out on lateral branches, each associated with specific functional responsibilities.

Component	**Description**
Main Branch	Central and stable branch containing validated, reviewed, and production-ready code, data pipelines, and configurations.
Board | Server	Branch dedicated to core infrastructure components, server-side logic, and system orchestration.
Data | Model	Branch responsible for data ingestion pipelines, preprocessing, feature engineering, and machine learning model development.
Demo	Branch containing demonstration components and prototypes used for validation and stakeholder communication.
Documents	Branch managed by the Project Manager, including technical documentation, governance reports, and project deliverables.

Merge operations are primarily performed by Software Developers and Data Analysts, integrating validated changes from the *Board* |*Server* and *Data* |*Model* branches into the *main* branch. This approach supports parallel development while preserving system stability and enforcing quality control through code reviews and integration checks.

1.2 Board Versioning

Used to systematically track the hardware evolution throughout the development lifecycle. Each board version represents a stage of the system, reflecting functional additions, corrections, or hardware maintenance activities.

Board Version	Description
Board 0.1	Initial prototype featuring only the ambient light sensor, used for basic functionality testing and baseline validation.
Board 0.2	Extension of the initial setup through the integration of additional sensors, enabling multi-parameter data acquisition.
Board 0.3	Introduction of a threshold-based LED indicator to visually signal light intensity levels.
Board 0.4	Replacement of damaged sensors to restore correct operation and ensure measurement reliability.
Board 0.5	Correction of incorrect wiring associated with the watering level sensor, improving system stability and data accuracy.
Board 1.0	Final and fully functional version of the board, incorporating all fixes and enhancements, ready for stable operation and deployment.

1.3 Board Code Versioning

Board Code Version	Description
Board Code 0.1	Initial firmware version implementing the basic control logic and sensor interfacing for early validation and testing.
Board Code 0.2	Minor refinements to the original firmware, improving code organization and preliminary sensor handling.
Board Code 0.3	Extension of firmware functionalities to support additional sensors and enhanced data acquisition routines.
Board Code 1.0	Firmware refactoring involving the replacement of the `DHT_nonblocking` library by ELEGOO with the `DHT` library by ADAFRUIT, alongside the integration of the Adafruit Unified Sensor Driver for improved reliability and standardization.
Board Code 1.1	Stabilized firmware version consolidating all previous changes, optimized for consistent operation and long-term deployment.

1.4 Data Versioning

Data versioning is treated as a first-class concern within the MLOps lifecycle, as datasets directly influence model behavior, performance, and reproducibility. Each dataset is uniquely identified by a precise timestamp that marks the beginning of data registration, ensuring unambiguous temporal traceability.

Attribute	Description
Timestamp Format	Data registration follows the format `YYYY / MM / DD / HH / mm / ss`, enabling exact identification of acquisition time.
Traceability	Timestamp-based identification allows full reconstruction of data lineage for auditing, debugging, and reproducibility.
Chronological Ordering	Ensures consistent temporal alignment between datasets, experiments, and model versions.

In addition to temporal tracking, datasets are grouped into semantic versions that reflect their maturity and intended usage within the project lifecycle.

Data Version	Description
Data 1.0	Synthetically generated data used during early development stages for pipeline validation, architectural testing, and preliminary model experimentation.
Data 2.0	Real-world data collected from operational environments, serving as the definitive input for model training, evaluation, and deployment.

This separation reduces the risk of data leakage and ensures a controlled transition from experimental to production-grade datasets.

1.5 Code Versioning

Code versioning follows the progressive evolution of the system, tightly coupled with changes in data availability and system requirements. Each code version corresponds to a well-defined development phase and reflects increasing levels of robustness and optimization.

Code Version	Description
Code 1.0	Initial implementation based on synthetic data, focused on validating architecture, data pipelines, and baseline functionality.
Code 2.0	Adaptation of the codebase to support real data ingestion and processing, including improved error handling and data validation mechanisms.
Code 2.1	Optimized implementation for real data, incorporating performance improvements, refactoring, and refinements driven by testing and deployment feedback.

1.6 Models Versioning

Model versioning is managed through an experiment tracking platform, enabling systematic comparison and reproducibility of machine learning experiments. Given the temporal nature and complexity of the dataset, linear models were excluded early in the design phase, as they are insufficient to capture nonlinear temporal dependencies.

After consultations with domain experts, a review of the scientific literature, and an analysis of the state of the art, the NHITS model was selected. This architecture is particularly suited for time series forecasting with both historical and future exogenous variables, providing enhanced monitoring and predictive capabilities for the target variable.

Model Version	**Description**
Model 1.0	NHITS model trained on synthetic data, used to validate training pipelines and establish an initial performance baseline.
Model 1.1	NHITS model trained on real data without extensive tuning, serving as a first real-world benchmark.
Model 1.2	Optimized NHITS model trained on real data, incorporating hyperparameter tuning, architectural refinements, and performance improvements.

Each model version is explicitly linked to the corresponding data and code versions, ensuring end-to-end traceability across the entire MLOps pipeline. This integrated versioning framework supports reproducibility, governance, and long-term maintainability of the system.

2 CI/CD Reference

The Continuous Integration and Continuous Deployment (CI/CD) strategy is intentionally not detailed within the present document. A comprehensive description of the envisioned CI/CD workflow is provided in *Document 2: Project Proposal and Development Plan*, Section 2.3. That section clarifies that, at the current stage of the project, a fully automated CI/CD pipeline has not yet been implemented, as the system is still in an exploratory and educational phase where architectural validation and functional correctness are prioritized. Nevertheless, the system has been designed to remain fully compatible with future CI/CD integration. The referenced section also outlines the intended automation approach, including repository-triggered updates of embedded firmware and server-side components, as well as the potential use of orchestration tools such as Jenkins to enable reliable and reproducible deployments.

3 Model Lifecycle Governance

Model Lifecycle Governance defines the set of processes, tools, and controls used to manage machine learning models throughout their entire lifecycle, from experimentation to deployment and continuous monitoring. In this project, governance is designed to ensure reproducibility, traceability, and controlled evolution of models within an automated MLOps pipeline.

3.1 Model Registry

The model registry represents the cloud repository where trained models are stored, versioned, and managed. It enables consistent handling of model artifacts and supports controlled promotion of models across different lifecycle stages.

Aspect	**Description**
Tool Adoption	The model registry is implemented using Weights & Biases (W&B), which provides integrated experiment tracking and model versioning capabilities.
Model Storage	Each trained model is stored as an artifact of a distinct run, uniquely associated with its version, configuration, and training context.
Version Control	Models are saved as immutable versions, ensuring that previously validated models remain accessible and unchanged.
Pipeline Integration	Registered models can be seamlessly reintegrated into the automated pipeline for validation, deployment, or further training.

This centralized registry guarantees governance over model evolution and enables transparent management of model artifacts across development and production environments.

3.2 Reproducibility

Reproducibility is a critical requirement in MLOps, ensuring that any model version can be reliably reconstructed and validated. The project enforces reproducibility by systematically logging all information required to recreate a training run.

Element	Description
Hyperparameters	All hyperparameters used during training are logged for each run, including architectural and optimization-related settings.
Model Version	Each run is explicitly associated with a unique model version identifier, ensuring consistency between experiments and registered artifacts.
Performance Metrics	Key performance metrics are logged and stored, enabling direct comparison across runs and model versions.
Data Association	Model versions are implicitly linked to the corresponding dataset versions used during training, supporting full lineage reconstruction.

3.3 Experiment Tracking

Experiment tracking is used to monitor, compare, and evaluate model behavior across multiple training runs. This process supports both model development and long-term performance monitoring in operational environments.

Tracked Component	Description
Performance Metrics	Metrics such as mean absolute error, loss, are logged for each training run to assess model quality.
Hyperparameters	Training configurations are tracked to enable systematic analysis of their impact on model performance.
Temporal Monitoring	Model performance is monitored over time to detect degradation or changes in behavior.
Experiment Comparison	Logged runs can be compared to identify optimal configurations and performance trends.

A key component of experiment tracking is the detection of model drift.
When a significant degradation in performance is identified, predefined governance actions are automatically triggered.

Event	Governance Action
Drift Detection	The automated pipeline flags a deviation between expected and observed model performance.
Process Interruption	Model inference or deployment is temporarily halted to prevent unreliable predictions.
Model Retraining	The model is retrained using newly collected data to restore performance and adaptability.
Logging Update	New data versions and updated hyperparameters are logged as part of a new experiment run.
Model Re-registration	The retrained model is saved as a new version and reinserted into the automated pipeline.

4 Monitoring & Maintenance Plan

4.1 Data Drift Detection

Given that the system operates on time-series data, it is inherently exposed to seasonal variations and long-term environmental changes. However, the drift detection strategy focuses specifically on *humidity* as the target variable, as it directly influences the watering behavior of the plant and the downstream decision logic.

Environmental changes are reflected in how humidity levels decrease over time, which in turn alters the frequency of watering events. Data drift is therefore assessed by comparing the dominant frequency components of the humidity signal observed during the training phase with those extracted from current prediction data.

From a mathematical perspective, drift detection is performed by applying the Discrete Fourier Transform (DFT) to both the training humidity time series and the live prediction window. The dominant frequency components are identified and compared, and a drift condition is triggered when the relative deviation between these frequencies exceeds a predefined percentage threshold. This approach allows the system to capture structural changes in temporal patterns rather than relying solely on pointwise statistical shifts.

Aspect	**Description**
Target Variable	Humidity, selected due to its direct impact on watering event frequency.
Drift Indicator	Change in dominant temporal frequency of humidity measurements.
Methodology	Comparison of Discrete Fourier Transforms (DFT) computed on training data and current prediction data.
Trigger Condition	Drift is detected when the relative difference between dominant frequencies exceeds a predefined threshold.

4.2 Sensor Drift Detection

In addition to data drift, physical sensors may experience mechanical wear, aging, or environmental degradation, leading to systematic measurement shifts or altered value ranges. Such sensor drift can compromise data reliability even in the absence of genuine environmental changes.

To mitigate this risk, sensor readings are continuously scaled and constrained within the reference ranges defined in the *System Specification Document*, Section 2.1 (Data Sources). Periodically, the system recomputes the observed minimum and maximum values for each sensor and compares them against the expected operational bounds.

Sensor drift evaluation is scheduled every 4 months. This interval represents a trade-off between responsiveness and stability: it is sufficiently long to smooth out short-term noise and transient anomalies, while remaining short enough to detect gradual degradation before it significantly impacts model performance. Upon detecting anomalous deviations, the system prompts the user with a notification requesting manual sensor recalibration, analogous to operating system update reminders that require user acknowledgment and action.

Aspect	Description
Drift Source	Mechanical degradation or aging of physical sensors.
Detection Strategy	Periodic recomputation of observed minimum and maximum sensor values.
Reference Bounds	Expected value ranges defined in the System Specification Document (Section 2.1).
Evaluation Frequency	Every 4 Months, to balance early detection and robustness against short-term fluctuations.
User Interaction	Automatic pop-up notification requesting manual sensor recalibration.

4.3 Maintenance Plan

The maintenance strategy combines automated monitoring routines with scheduled and event-driven interventions to ensure long-term system reliability, security, and performance. Tasks are executed either automatically or conditionally, depending on detected drift signals or infrastructure requirements.

Task	Frequency
Full model retraining	3 Months.
Drift detection	Automatic.
Detection alerts	Automatic.
Dependency updates	As needed.
Security updates	As needed.