

École polytechnique de Louvain

PhenoHive

An intuitive, modular and low-cost phenotyping station

Author: **Adrien MONTARIOL**

Supervisors: **Cristel PELSSER, Laurent FRANCIS**

Readers: **Xavier DRAYE, Sebastien TOUSSAINT**

Academic year 2025–2026

Master [120] in Computer Science

Abstract

PhenoHive is a portable, low-cost station that monitors a single plant continuously. It runs in the LBIR1251 plant biology course at UCLouvain, where about 20 student groups carry out month-long experiments in parallel, most of them in their own homes rather than a controlled lab. A teaching platform like this has to satisfy two demands at once: the sensor data must be accurate enough to interpret in physiological terms, and the hardware has to keep running unattended for weeks. The version I inherited fell short on three fronts. Its codebase was monolithic, so extending or maintaining it was awkward. The sensors were too basic and hadn't been validated against references, which left derived quantities such as vapor pressure deficit and spectral band assignments impossible to trust. And it coped badly with the network and power interruptions that are routine in a home.

In order to fill the gap in measurements, I calibrated the environmental and light sensors in a controlled laboratory. These corrections drop the uncertainty on measurements down to a level that allows the students to be confident in the data and will allow them to interpret the measurements in physiological terms. To achieve this, I characterized the light spectrum across all of its channels by using a monochromator and I calibrated the environmental sensor by using a controlled chamber. The deployment also changed how the station stores and pushes data to the server. It now always writes each measurement locally first in order to not lose any data during potential network outages. Students can also connect the device to their WiFi network through a captive portal instead of a terminal, remote management runs over a private network overlay and every team's dashboard is automatically created the first time their station is connected to the server. My testing on physical hardware validated three critical things. First that a fresh station can start pushing live data to the server within five minutes. Second it that a long disconnection from the server doesn't create any data loss. And third is that a student's data is never accessible to another student from another group.

Acknowledgements

My first thanks go to my supervisors, Cristel Pelsser and Laurent Francis, for their guidance and support throughout this project. Cristel set up the UCLouvain server infrastructure and stayed available all year, both of which were essential to finishing this thesis. Laurent's expertise in sensor methodology and calibration kept the hardware work grounded in sound metrological practice.

Xavier Draye also helped me beyond just reading the final manuscript. As the professor for the LBIR1251 course, he helped me work out what were the requirements for this iteration of the project. So what to improve, what was already fine and which physiological metrics mattered. These conversations helped establish the scope of the project and kept it focused on the things that would make a difference for the students. I am grateful for his time and insights on this matter as well as for his reading of the final manuscript.

Thanks to Sebastien Toussaint for agreeing to sit on my jury and for taking the time to review this work.

Tom Klauner at the WELCOME lab helped me through the sensor calibration sessions and was generous with his time throughout that process, for which I am very grateful.

Alexandre and Arnaud Wery helped me in designing the 3D-printed casing and soldering the station's components. I therefore want to thank them both.

Finally, I want to thank my family: my mother Virginie Goffaux, my father Eric Montariol, and Colin Larcombe, for their careful proofreading and for their support throughout this year.

Contents

Abstract	i
Acknowledgements	ii
List of Abbreviations	x
1 Introduction	1
2 State of the art	4
2.1 Commercial and research phenotyping platforms	4
2.2 Educational platforms for plant biology	5
2.3 Sensor integration and measurement strategy	7
2.4 Synthesis of previous PhenoHive theses	8
3 Requirements and approach	10
3.1 Pedagogical requirements	10
3.2 Hardware and budget constraints	11
3.3 Deployment context and operational resilience	11
3.4 Maintainability and handoff	12
3.5 Data isolation and access control	13
3.6 Requirements summary	13
4 System architecture	14
4.1 Codebase structure and module boundaries	15
4.2 Configuration and runtime composition	16
4.3 Sensor abstraction and hardware binding	16
4.4 Measurement cycle and orchestration	17
4.5 Persistence and remote synchronization	18
4.6 Debug interface and local feedback	18
5 Data acquisition and processing	20
5.1 Sensor suite	20

5.2	Sensor calibration and metrological validation	21
5.2.1	Calibration methodology	22
5.2.2	Environmental sensor calibration	23
5.2.3	Spectral sensor calibration	26
5.2.4	Scale and camera calibration	33
5.3	Data gathering and signal processing	33
5.3.1	Data smoothing and noise rejection	33
5.3.2	Confidence flags and quality metadata	35
5.3.3	Acquisition summary	35
6	Storage, visualization and access control	37
6.1	Edge storage and local caching	37
6.1.1	Flat-file CSV logging	37
6.1.2	Offline JSONL caching queue	38
6.1.3	Synchronization protocol	38
6.2	Central database schema and cardinality control	39
6.2.1	Database schema layout	39
6.2.2	Series cardinality suppression	39
6.3	Multi-tenant access control and security	40
6.3.1	Declarative team-based permissions	40
6.3.2	Query-level isolation	41
6.3.3	Central fleet administration	42
6.3.4	Data privacy	42
7	Deployment and operations	43
7.1	Network topology and security architecture	43
7.1.1	Overlay network rationale	43
7.1.2	Virtual machine security and port isolation	44
7.1.3	Edge security and gateway proxy	45
7.1.4	From sensor to Grafana: the journey of a packet	46
7.2	Station bootstrapping and systemd sequencing	47
7.3	Auto-provisioning service	49
7.3.1	Dynamic station ID reassignment logistics	49
8	Evaluation and results	51
8.1	Test environment	51
8.2	Deployment and network resilience	52
8.3	Runtime behaviour	53
8.3.1	Debug UI and on-device calibration	53

8.3.2	Sensor data quality over a multi-day run	55
8.3.3	Demonstrative plant observation	56
8.3.4	Multi-station isolation and dynamic provisioning	59
8.3.5	Concurrent load test	60
8.4	Validation summary	60
9	Discussion	63
9.1	Revisiting the thesis' original objectives	63
9.2	Limits of the current implementation	63
9.2.1	Calibration validity and reproducibility	63
9.2.2	Scale and multi-station validation	64
9.2.3	Networking and connectivity assumptions	64
9.2.4	Camera pipeline	65
9.3	Pedagogical fitness	65
9.4	Future work	65
9.5	Reflection on the development process	66
10	Conclusion	68
	Original contributions	68
	References	71
A	Technical architecture	74
A.1	Electronic wiring diagram	74
A.2	3D Printed Casing	75
B	Bill of materials and cost analysis	77
C	Project repository	79
D	Station Assembly and Deployment Guide	80
D.1	Prerequisites	80
D.2	Hardware assembly	81
D.3	Prepare configuration and build the image	81
D.4	Flash the image	83
D.5	Wi-Fi setup via comitup	83
D.6	Verify the service	83
D.7	Scale and Camera calibration	84
D.8	Verify data on Grafana	85
D.9	Optional: setting up the server-side infrastructure	85

List of Figures

2.1	Commercial, Open-Source and Educational Phenotyping Reference Systems	5
2.2	Evolution of the PhenoHive Platform	9
3.1	Temperature, Humidity, and Vapor Pressure Deficit	11
3.2	Comparison of Data Persistence Strategies	12
4.1	System Architecture Runtime Data Flow	15
4.2	PhenoHive Module Boundaries	15
4.3	Two-Tier Measurement Timing	18
5.1	PhenoHive Core Sensor Suite	21
5.2	Manual Calibration Workflow	22
5.3	Espec SH-242 Environmental Chamber	24
5.4	SHT35 Calibration Scatter Plot	25
5.5	Spectral Calibration Setup at the WELCOME Lab	26
5.6	TCS3448 Spectral Channel Peak Wavelengths: Datasheet vs. Empirical	28
5.7	Burst Sampling and Signal Processing Pipeline	34
5.8	Signal Cleaning Using MAD Filtering	35
6.1	Edge Node Data Synchronization Flow	38
6.2	Grafana Team-Based Permission Topology	41
7.1	PhenoHive Overlay Network Topology	44
7.2	Secure Data Path from Sensor to Grafana	47
7.3	Systemd Boot Chain	48
8.1	The Comitup Captive Portal	52
8.2	Debug UI Live Sensor Tab	54
8.3	Debug UI Calibration Wizard	54
8.4	Nine Days of Continuous Real Sensor Data	56
8.5	54-Hour Plant Observation	58
8.6	Grafana Admin Panel	59
8.7	Student-Facing Grafana Dashboard	60

A.1	PhenoHive Electronic Wiring Schematic	75
A.2	PhenoHive 3D Printed Casing	76

List of Tables

2.1	Core Phenotyping Metrics and Physiological Interpretation	6
2.2	Phenotyping Platform Families Comparison	7
3.1	System-Level Requirements Summary	13
4.1	PhenoHive Project Structure	16
5.1	SHT35 Calibration Results	24
5.2	TCS3448 Empirical Channel Peak Wavelengths	29
5.3	TCS3448 Deployed Calibration State	30
5.4	TCS3448 Grating 1 Sweep Results (Excerpt)	31
8.1	First-Boot Setup Timing Breakdown	53
8.2	System-Level Validation Test Summary	61
B.1	Bill of Materials for One PhenoHive Station	78

List of Abbreviations

ADC	Analog-to-Digital Converter	MAD	Median Absolute Deviation
BCM	Broadcom GPIO numbering scheme	NAT	Network Address Translation
CIE	Commission Internationale de l'Éclairage	NDIR	Non-Dispersive Infrared
CSI	Camera Serial Interface	NIR	Near-Infrared
CV	Coefficient of Variation	PAR	Photosynthetically Active Radiation
EPL	École Polytechnique de Louvain	RH	Relative Humidity
FDM	Fused Deposition Modelling	RPi	Raspberry Pi
GDPR	General Data Protection Regulation	SBC	Single-Board Computer
I²C	Inter-Integrated Circuit	SSID	Service Set Identifier
IaC	Infrastructure as Code	TA	Teaching Assistant
ISP	Image Signal Processor	TSDB	Time-Series Database
JSONL	JSON Lines (newline-delimited JSON)	UFW	Uncomplicated Firewall
TSM	Time-Structured Merge-tree (InfluxDB)	VPD	Vapor Pressure Deficit
TSI	Time Series Index (InfluxDB)	WELCOME	Wallonia Electronics and Communications Measurements

Chapter 1

Introduction

PhenoHive is a portable phenotyping station built for LBIR1251, the plant biology and physiology course at UCLouvain. One of the things that course tries to do is tie physiological theory to hands-on observation, across topics like water balance, mineral nutrition, photosynthesis and how plants respond to their environment. Traditionally that has meant shared greenhouse space and measurements taken by hand, which caps how much and how freely students can experiment.

Because of the fact that each unit is small enough and is able to run on its own, watching a single plant for weeks and producing data students can relate back to what they see in class. All without the need for a greenhouse or lab bench. A group can design its own experiment, change the conditions, and watch the effects play out close to real time. The limits that are traditionally set by the need to share space and instruments with a large group are lifted. This allows the course to be more flexible, and for students to be more creative than what was previously possible.

I came to this project from a long interest in embedded systems, and in how hardware and software actually behave in a real world environment. PhenoHive encompasses a large amount of the computer science curriculum into one place: low-level sensor communication, software structure, data persistence, networked services, visualising results, and keeping all of it alive in the field. The breadth is what made it hard. It is also what made it worth doing.

Goffinet (2024) moved the OS to DietPi (DietPi, n.d.), added a rolling-median filter for the mass readings, and wrote an automated first-boot routine that bootstrapped a local InfluxDB/Grafana instance straight from a preconfigured SD card, showing that a station based on the Raspberry Pi Zero W could run on its own in a teaching setting. Colinet (2025) took that further. The hardware moved from the Zero W to the faster Zero 2W. The PlantCV pipeline was brought up to version 4.6 with automatic

calibration and shadow detection, and the InfluxDB/Grafana stack was relocated from a local RPi hotspot to a shared UCLouvain server. Colinet also added a resistive soil-moisture probe read through an MCP3008 ADC and a photoresistor for ambient light, and replaced the enclosure with a compact 3D-printed design.

By the time I started, the station already worked end to end: it brought up its hardware, took and timestamped measurements, and pushed them to a central dashboard.

Even so, the prototype I inherited had real limitations for sustained classroom use. The software was heavily monolithic, which made it hard to maintain or extend. Measurement quality was uneven too. Mass tracking was solid enough for evapotranspiration work, and the PlantCV camera pipeline (Gehan et al., 2017; PlantCV Team, n.d.) gave detailed growth indicators, but the air and light sensors were not good enough for any deeper physiological reading. The resistive soil-moisture probe returned a single index with no connection to the atmospheric water-vapor status that a vapor pressure deficit calculation needs. The photoresistor collapsed all incoming light into one lux-like number, so it could not separate the red and far-red bands that drive phytochrome-mediated development. And the platform tended to falter during the long, multi-group home deployments where dropped networks and failing instruments are simply part of normal operation.

My goal was to turn this working prototype into a platform that could be maintained and extended over time. I did not start over. Instead I kept and built on the parts that already worked, and reworked the parts that plainly did not.

The work pursues three objectives.

The first is architectural. The runtime is broken out of its monolithic form into modular components, each responsible for one concern: acquisition, processing, persistence, visualization, and the user interface. Splitting things up this way lowers the maintenance burden and makes it less likely that a teaching assistant (TA) extending the platform will break something elsewhere.

The second concerns sensing quality. Choosing better sensors isn't enough. They have to be calibrated and validated against references. This is why I calibrated the temperature and humidity sensor in a controlled chamber, and characterized the light spectrum with a monochromator. This allows the students to be confident in the data and to interpret it in physiological terms, which is what the course is about.

The third is operational reliability in the field. Any long deployments need the station to be as robust and stable as possible. If a station fails, it compromises the integrity of the student's work. This is why several safeguards were put in place. The station now writes every measurement locally first and now has an offline queue. There

is now also an automated access control system that keeps each group's data separate and private.

Taken together, these changes move the station from a research prototype toward something that can be reused dependably within LBIR1251. Throughout, I have tried to foreground methodology and the reasoning behind each architectural decision, so that the TAs who come after me extend a clear foundation rather than redesign the system. One part is carried over untouched: the camera-based morphological pipeline, the PlantCV leaf-area analysis, comes from Colinet (2025). I did not re-evaluate its segmentation accuracy, and it is not a primary contribution here. Improving the vision side is left as future work in Section 9.4.

The full source code is publicly available at https://github.com/amontariol/Phenohive_2025-26 and is described in Appendix C.

Two linked questions run through the thesis. The first is an engineering research question: *What architectural, metrological, and operational design decisions are required to transform a functional low-cost phenotyping prototype into a platform that delivers calibrated, physiologically interpretable multi-sensor data reliably to 20 concurrent student groups over month-long deployments in uncontrolled environments?* The second turns that into a concrete usability test: *Can a student take a freshly-imaged Raspberry Pi, connect it to any Wi-Fi network, and begin collecting calibrated multi-sensor data on a shared, isolated dashboard within minutes and without any complex setup?* Answering either one takes work at every level of the stack, from calibration through architecture and persistence to deployment, and each level is taken up in the chapters that follow.

Chapter 2

State of the art

Automated plant phenotyping is now increasingly prevalent in both research and education, driven by low-cost sensors, single-board computers and open-source analysis pipelines (Pieruschka & Schurr, 2019). This chapter first reviews the main platform families and related technologies, then positions the PhenoHive project within that field.

2.1 Commercial and research phenotyping platforms

At the higher end of the market, commercial phenotyping systems such as LemnaTec and Phenobox provide fully automated, high-throughput measurements of plant growth and physiological responses (Figure 2.1). These platforms combine multispectral imaging, detailed environmental control and integrated analysis pipelines. They are mainly designed for research institutions, where high cost is justified by throughput and tight experimental control. For teaching programs and individual researchers however, they are usually out of reach because of cost and space requirements.

In the mid-tier and research sector, open-source greenhouse-monitoring platforms provide modular sensor integration, time-series storage and image-analysis workflows that can be assembled and customized. Representative examples include FarmBot (FarmBot Inc., n.d.) (an open-source precision agriculture robot) and community-maintained sensor-logging stacks built on InfluxDB and Grafana. Many of these systems run on Raspberry Pi or similar single-board computers, which makes distributed low-cost deployment realistic. Near-continuous plant monitoring with affordable hardware is increasingly feasible, and educational platforms have begun adopting this model.

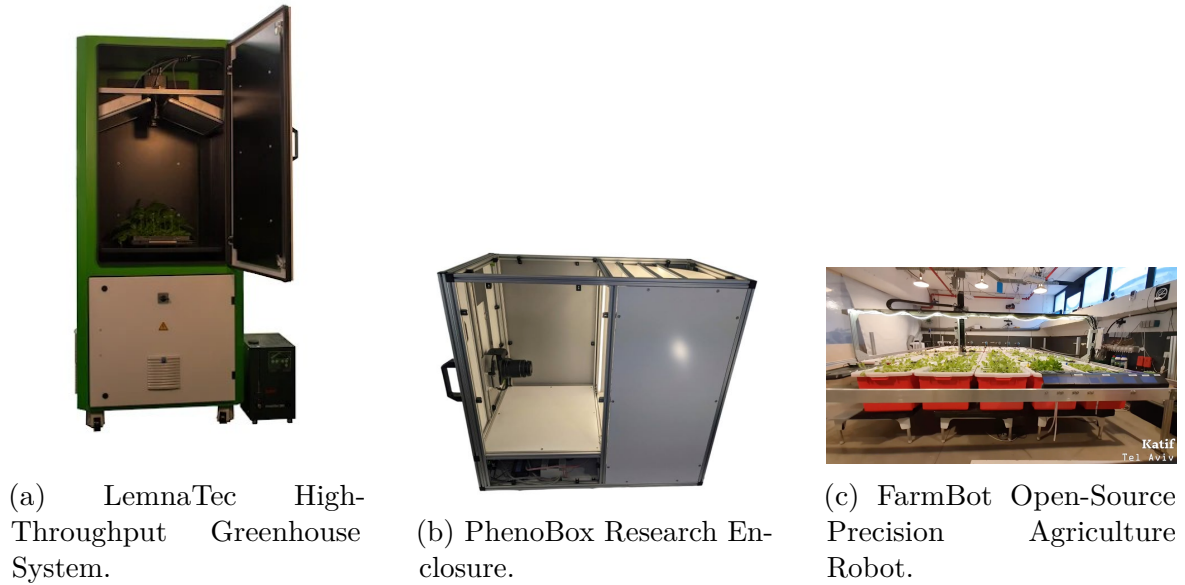


Figure 2.1: Representative phenotyping platforms across market/price tiers. LemnaTec (LemnaTec GmbH, n.d.; Vienna Scientific Instruments, n.d.) and PhenoBox are commercial high-throughput systems that are designed for research institutions. These emphasize throughput and total environmental control. FarmBot (FarmBot Inc., n.d.) is a community-maintained open-source robot that illustrates the modular and affordable end of the spectrum. All three contrast with the portable, educational focus of PhenoHive .

2.2 Educational platforms for plant biology

In academic settings, plant physiology and botany courses often depend on manual measurements, shared greenhouse facilities, or pre-recorded datasets. Several institutions have developed open-source platforms for continuous plant monitoring in teaching and research contexts. Phenotiki (Minervini et al., 2017) is an example of a Raspberry Pi-based platform for image-based phenotyping of rosette plants, designed to lower the barrier to entry for laboratory use. The Growth Monitor Pi (GMPi) (Grindstaff et al., 2019) similarly targets affordable remote monitoring of plant growth in facilities, using single-board computers to enable continuous observation without dedicated infrastructure. The diversity of these approaches reflects different pedagogical goals. Some platforms emphasize instrumentation, while others prioritize biological observation and interpretation. The PhenoHive project follows the second approach, where technology is a tool for observation rather than the primary subject of study.

Autonomous monitoring addresses a timescale mismatch. The physiological processes students are asked to observe acclimation responses, growth allocation, VPD-driven stomatal behavior unfold over days and weeks, not laboratory hours. Continuous

sensors also remove the greenhouse bottleneck, letting twenty groups run parallel experiments with different conditions simultaneously. The reduction in collection overhead shifts student attention from measurement logistics toward experimental design and biological interpretation. The biological signals that make this extended monitoring valuable are grouped into four measurement domains (Table 2.1).

Table 2.1: Core metrics used in plant phenotyping and their physiological interpretation. Environmental, light, mass and morphology signals are not interpreted independently; they are jointly used to reason about transpiration, photosynthesis and growth dynamics over time.

Measurement domain	Primary metrics	Physiological interpretation
Environment	Air temperature, relative humidity, vapor pressure deficit (VPD)	Transpiration behavior, stomatal context, stress exposure
Light	Intensity and spectral composition	Photosynthetic context, acclimation response
Mass	Pot mass trend and short-term variation	Water-balance dynamics, biomass accumulation trajectory
Imaging	Projected leaf area, canopy color, shape descriptors	Growth progression, visual stress signatures

Cross-cutting quality dimensions: temporal continuity, sampling interval and data quality (missing values, outliers, drift).

PhenoHive is specifically designed for this teaching context, where the primary goal is pedagogical understanding and where stations may be deployed in diverse, uncontrolled environments such as student homes. Table 2.2 positions PhenoHive within this field.

Table 2.2: Comparison of plant phenotyping platform families across cost, throughput, modalities, deployment context and teaching access. Commercial systems maximize automation, while open-source and educational platforms prioritize flexibility and affordability.

Criterion	Commercial	Open-source	Educational	PhenoHive (this thesis)
Cost	High	Moderate	Low	Low ($\sim$ €137/unit)
Throughput	Very high	Medium	Low	Low (1 plant/station)
Main modalities	RGB, multispectral, thermal, mass, climate, often gas exchange	RGB, multispectral, thermal, mass, optional custom sensors	RGB, mass, basic climate, optional light/humidity add-ons	Temp./humidity, 14-channel spectral (calibrated), mass, RGB camera
Deployment context	Controlled greenhouse/chamber, dedicated server, high maintenance	Lab/greenhouse, local server or SBC, moderate integration effort	Desk or small enclosure, minimal infrastructure	Student home or desk; single USB supply; Tailscale overlay
Typical users	Industry breeding and research centers	Research labs and student projects	Course cohorts and teaching labs	LBIR1251 student groups ($\sim$ 20 parallel stations)
Teaching accessibility	<i>Low</i>	<i>Moderate</i>	High	High

Synthesis based on literature and prior PhenoHive theses (Colinet, 2025; Goffinet, 2024). PhenoHive sits in the educational cost tier but targets open-source-tier sensing quality through laboratory-validated sensors.

2.3 Sensor integration and measurement strategy

Sensor selection and integration demonstrate a recurring trade-off between price and precision. High-end phenotyping platforms often integrate many sensors and cameras, which provides multispectral, thermal and morphological data at the same time. This completeness comes with higher complexity, cost and data volume. Educational platforms usually favor a more focused approach, selecting a smaller set of sensors that directly serve learning objectives.

Four variables have the most direct pedagogical value for plant physiology. Spectral light quality and illuminance drive photosynthesis and control developmental responses through phytochrome (Franklin & Whitelam, 2005). Air temperature and relative humidity jointly determine vapor pressure deficit, which governs stomatal conductance and evapotranspiration (Allen et al., 1998). Continuous pot mass provides a slow,

integrating water-balance signal; combined with camera images it traces biomass accumulation along two independent lines of evidence.

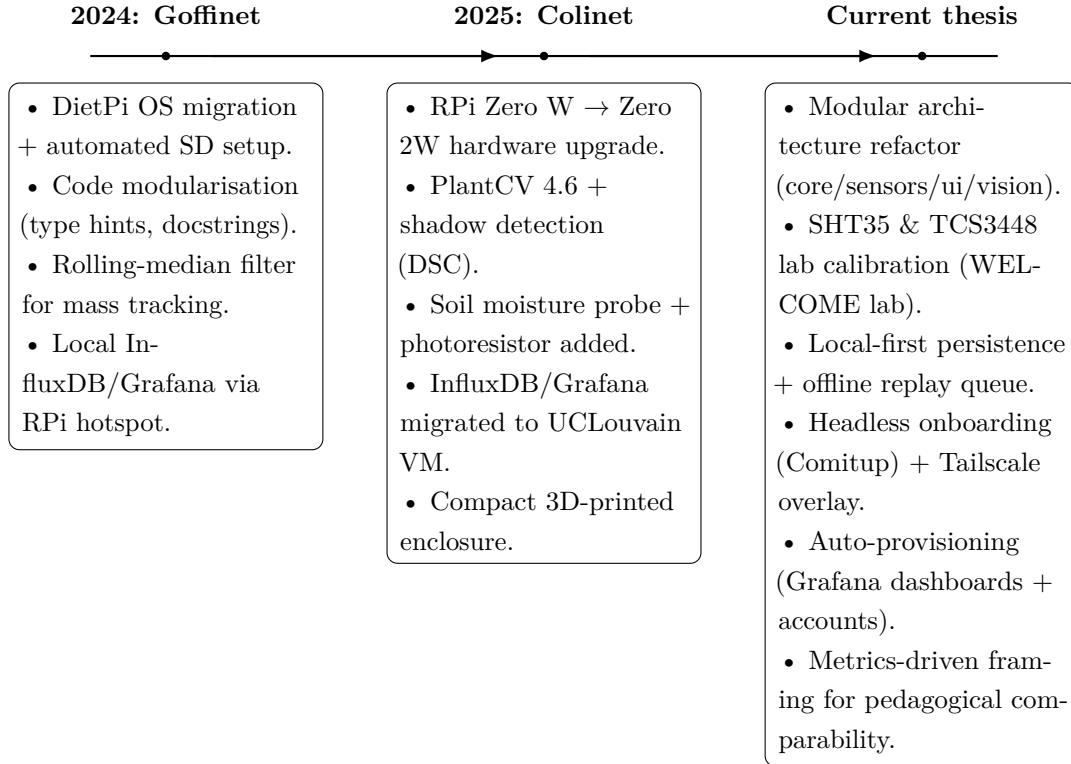
In many existing platforms, sensor choices reflect availability and budget constraints more than strict pedagogical optimization. In the PhenoHive project’s sequence, Goffinet (2024) and Colinet (2025) established a baseline that made continuous environmental and growth observation feasible in an educational context. Their work also showed which variables needed stronger characterization in later iterations.

The VPD and the spectral band ratio carry the most direct pedagogical relevance of the four. Both need precisely calibrated sensors which is what drives the sensor selection in Chapter 5. Not only that but with 20 groups running in parallel, the measurements also have to stay consistent and comparable. If they aren’t, this means that no group can trust the conclusions it draws from its own experiment.

2.4 Synthesis of previous PhenoHive theses

The two previous PhenoHive theses advanced the platform in concrete, distinct steps (Figure 2.2). Goffinet (2024) established the continuous-monitoring feasibility on DietPi, modularised the codebase and provided the first automated deployment pipeline with a self-hosted visualization stack. Colinet (2025) upgraded the hardware, improved the image-analysis pipeline with shadow detection, migrated the visualization infrastructure to an online server, and added first-generation environmental sensors (a resistive soil moisture probe and a photoresistor). Considered together, these works demonstrate a clear progression in platform maturity. What they also reveal is the gap between having a sensor *connected* and having a signal that is *metrologically grounded* and *physiologically informative*. The soil moisture index cannot substitute for air humidity in VPD calculations, and a photoresistor cannot discriminate the spectral bands that drive phytochrome-mediated plant responses.

This review of the state of the art in plant phenotyping platforms, and of the PhenoHive project’s trajectory, all show that for a platform to be effective, it must be reliable, accurate and pedagogically relevant. Also, keeping the platform easily maintainable and extensible is critical for its long-term success in an educational setting. The next chapters will therefore cover detail how the current iteration of PhenoHive addresses these requirements through sensor calibration, software architecture refactor and deployment strategy improvements.



Overall trajectory: feasibility & automation → hardware upgrade & image quality & VM migration → modular architecture & metrological calibration & operational hardening.

Scope note: timeline summarizes thesis-level contributions, not full implementation details.

Figure 2.2: Evolution of the PhenoHive platform across consecutive theses where each one contributes a distinct step to the maturation of the platform as a whole. From initial feasibility to expanded observation and metrics-driven pedagogical framing.

Chapter 3

Requirements and approach

Turning a proof-of-concept into a platform that can actually be deployed means that the first step is to pin down what it has to do. Chapter 1 framed the work around three objectives: architectural clarity (O1), sensing quality (O2), and operational reliability (O3). This chapter translates those into six concrete, verifiable requirements. R1 operationalizes sensing quality (O2); R5 captures the architectural maintainability goal (O1); R3, R4, and R6 together address operational reliability (O3). R2 is a system constraint set by the deployment context — 20 simultaneous stations on a teaching budget — rather than derived from any single objective, and it applies across all three.

3.1 Pedagogical requirements

Requirement 1: The system should provide calibrated sensor data within uncertainty bounds small enough to support physiologically meaningful analysis. More specifically, VPD calculations that are traceable to reference instruments and spectral discrimination at the red/far-red band boundary.

Design approach: This asks the sensors to be able to resolve two things. One is the VPD gradient that drives stomatal response and the other is the red/far-red boundary that triggers phytochrome-mediated responses. Both of these require much more precision than what basic temperature/humidity or RGB sensors can provide (Figure 3.1). Chapter 5 works through the calibration method and the resulting uncertainty budgets.

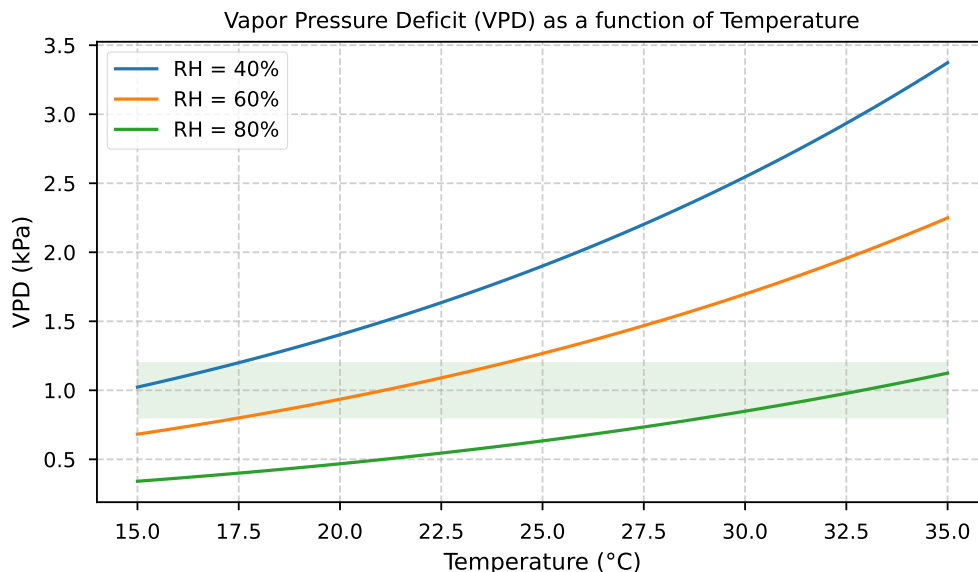


Figure 3.1: The relationship between temperature, relative humidity, and the Vapor Pressure Deficit (VPD). Precise biological modeling of transpiration requires high-precision sensors because subtle errors in humidity translate to significant errors in VPD calculation, increasing at higher temperatures.

3.2 Hardware and budget constraints

Requirement 2: The total hardware cost per station should not exceed €150 as this enables simultaneous deployment of approximately 20 units within a teaching budget.

Design approach: The budget cap motivates each choice of component. Twenty groups each need a station at the same time, so each part has to earn its place on precision against cost. Commercial platforms can rely on more expensive industrial controllers where PhenoHive instead pairs a Raspberry Pi single-board computer with low-cost, I²C-compatible sensors. I kept the Raspberry Pi over a cheaper microcontroller like an ESP32 because of the headroom. It can run a full Linux system, drive the high-resolution camera used for morphological tracking and manage a local persistence stack without needing to rely on a constant cloud connection.

Appendix B contains the full bill of materials. A unit comes to about €137 (Table B.1).

3.3 Deployment context and operational resilience

The stations are meant mainly for students' homes, but they should also work in a traditional greenhouse if need be. A home setting brings its own problems: power can get unplugged by accident, home Wi-Fi may be unstable or get reconfigured, and the surrounding environment is neither controlled nor steady.

Requirement 3 (R3): The system should not lose data during network outages and should recover connectivity autonomously, resuming data transmission without user intervention.

Design approach: The solution is an offline-first persistence model: measurements are written locally first and pushed to InfluxDB whenever it is reachable. When a push fails, the record is added to a local JSONL queue that replays on reconnection (Figure 3.2). Chapters 6 and 8 cover the design and the 48-hour disconnection test.

Requirement 4 (R4): First-boot setup should complete in under five minutes with no command-line interaction, so that students can begin collecting data immediately after connecting the device to Wi-Fi.

Design approach: On first boot, the `comitup` service broadcasts a temporary Wi-Fi hotspot. The student connects to it and enters the home network credentials through a captive portal in a web browser. The station then joins the network and starts pushing data automatically. Chapter 8 measures the end-to-end timing for each phase.

Standard Cloud-First Approach (Fragile)



PhenoHive Local-First Approach (Resilient)

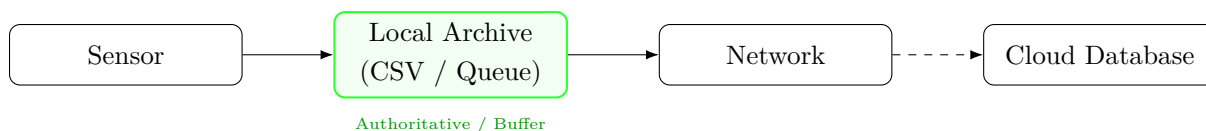


Figure 3.2: Two data-persistence strategies side by side. A standard cloud-first design is fragile because when the link to the cloud database drops during an outage, the readings in transit are lost. PhenoHive first writes to a local archive (CSV and queue) and treats it as authoritative, so an outage interrupts the upload and not the record.

3.4 Maintainability and handoff

Teaching tools at a university change hands constantly so having a tightly coupled, monolithic codebase is a liability because a new TA who wants to add a sensor or change a setting has to touch the acquisition loop and can break it without realising.

Requirement 5 (R5): A new TA with no knowledge of the codebase should be able to add a new sensor and to modify a configuration value without having to modify the main core acquisition loop.

Design approach: The architecture has to be as modular and configuration-driven

as possible. All of the sensor drivers, the core acquisition loop and the persistence therefore need to live each as distinct and testable components that can be changed, swapped out and tested one at a time.

3.5 Data isolation and access control

With 20 groups all writing to one central database at once, there is a real risk of data bleeding across groups. Each student should only be able to see and analyse their own station’s data, so that their experimental hypotheses stay their own.

Requirement 6 (R6): Each student group should only have access to data from their own station. This should be enforced both on the Grafana permission layer and the InfluxDB query layer in order to maintain data integrity and isolation.

Design approach: A `station_id` tag is added to the data at acquisition and stays with it throughout the whole pipeline. Grafana folder permissions and Flux queries filter each block cross-group access on their own. this makes it so that neither layer is a single point of failure. Chapter 6 covers the implementation.

3.6 Requirements summary

Table 3.1: Consolidated system-level requirements with traceability to the three objectives from Chapter 1: 1 architectural clarity, 2 sensing quality, 3 operational reliability. The verification column references the test or evidence that demonstrates compliance in Chapter 8.

ID	Requirement	Verification	Obj.
R1	Calibrated sensor data within research-grade uncertainty for VPD and spectral analysis	§ 5.2.2, § 5.2.3	2
R2	Total hardware cost $\leq$ €150 per station	Appendix B	1–3
R3	No data loss during network outages of $\geq$ 48 h; full replay upon reconnection	48 h disconnection test (§ 8)	3
R4	First-boot setup in < 5 minutes with no command-line interaction	First-boot test (§ 8)	3
R5	New sensor addable without modifying the acquisition loop; configuration-driven extension	§ 4	1
R6	Per-group data isolation enforced at both Grafana permission and InfluxDB query levels	Multi-station isolation test (§ 8)	3

Chapter 4

System architecture

This iteration’s architecture is compact, and each part has a clear job, but it did not get there in one step. I reached the current structure by decoupling responsibilities that had started out too tightly coupled, since the monolithic approach made later changes hard to implement. The system now rests on a small set of components, each with a specific role in the measurement chain: configuration, sensor instantiation, acquisition, persistence, time alignment, and local user feedback. For a teaching platform that division is critical, because the design stays readable and maintainable, and it holds up when one part of the stack fails.

The runtime reads a simple config file, builds the sensor objects, starts the time-sync and debug services and enters the measurement loop (Figure 4.1). Each measurement cycle creates a single flattened record which holds data from every active sensor. That record is written locally first and only then forwarded to the remote time-series backend if it is reachable. The order matters because when the network drops, the dashboard can go stale while acquisition keeps on working.

If this local-first writing approach is not implemented, this would mean that every short connectivity drop would leave a visible gap in the data’s timeline because the records made during the outage would be lost. This approach also means that the queued records are replayed once the link to the server is restored. A 48-hour disconnection test in Chapter 8 confirms this requirement on physical hardware.

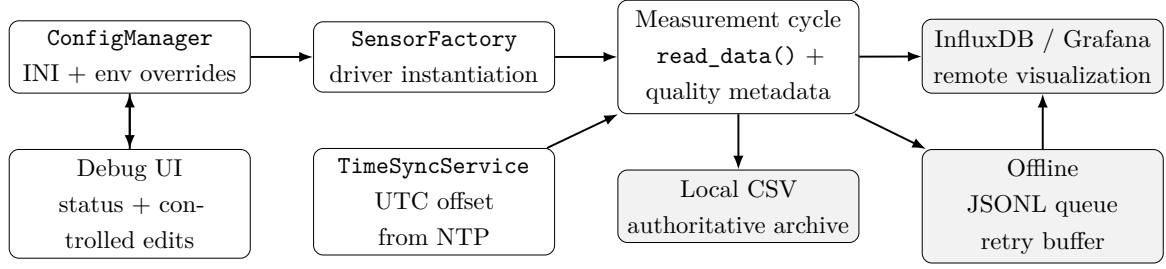


Figure 4.1: Runtime data flow from configuration and sensor acquisition to local persistence, remote synchronization and user feedback.

4.1 Codebase structure and module boundaries

The runtime entry point is in `main.py` and the implementation responsibilities are grouped under `src/`. The `src/core/` package contains orchestration and shared services (configuration management, sensor factory logic, persistence, logging and time synchronization). Hardware-specific drivers are isolated in `src/sensors/`, so sensor changes do not require edits in the runtime loop. The debug dashboard is separated in `src/ui/`, where the dashboard and HTTP service are implemented. Image-processing logic is grouped in `src/vision/` so camera-related code stays independent from numeric sensor acquisition. The structure is completed by `tests/`, which validates core behavior and sensor integration points and `scripts/`, which contains deployment and maintenance scripts. For TAs, this folder layout easily shows where to look when an update is needed (Figure 4.2 and Table 4.1).

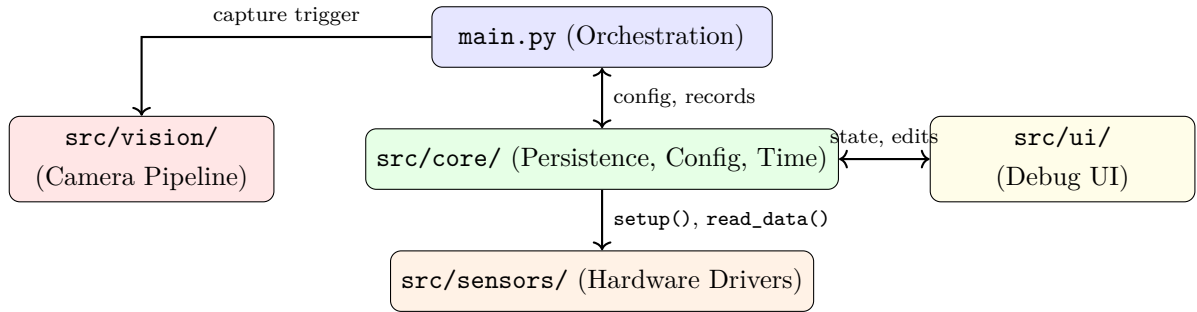


Figure 4.2: Module boundaries and architectural layers of the PhenoHive codebase.

Table 4.1: Project structure used to enforce architectural boundaries and maintenance ownership.

Layer	Project files	Primary responsibility
Runtime entry point	<code>main.py</code>	Application bootstrap, service startup and main acquisition loop.
Core services	<code>src/core/</code>	Configuration loading, sensor factory, persistence logic, time synchronization and shared runtime policies.
Sensor drivers	<code>src/sensors/</code>	Hardware-specific sensor setup and acquisition APIs.
Debug interface	<code>src/ui/</code>	Debug dashboard, local status inspection and controlled runtime configuration updates.
Vision pipeline	<code>src/vision/</code>	Camera capture and image-based phenotyping path separated from numeric sensing.
Validation and operations	<code>tests/</code> , <code>scripts/</code>	Automated checks and maintenance helpers used by TAs.

4.2 Configuration and runtime composition

The configuration layer is intentionally more defensive than just a raw INI parser. The `ConfigManager` wraps `configparser` with typed getters, environment-variable overrides and a uniform fallback strategy. This means that the same configuration source can be used both for normal operation and for deployment on student machines where some values can be injected by the environment.

This means that the runtime stays flexible without turning configuration into a second program. The bootstrap stays simple. It only reads one configuration file, resolves the overrides and then hands a small set of typed values to the rest of the system.

4.3 Sensor abstraction and hardware binding

Sensor construction is handled by a factory instead of through an inline in the main loop. The `SensorFactory` is what holds the mapping from the configuration values to the concrete sensor classes in one place. It's because of this that the mapping is centralised, the runtime has no hardware-specific branching and extending the sensor set stays a controlled and local change that doesn't affect the rest of the system.

The factory currently builds the air temperature and humidity sensor, the light sensor, and the HX711-based scale. Each exposes the same lifecycle: `setup()` to check that it is usable and `read_data()` to return one measurement payload. This shared contract matters more than the specific hardware because it lets the rest of the system treat every sensor as an interchangeable producer of timestamped data.

I intentionally did not reimplement the plant-camera pipeline. It's reused from the previous thesis following the discussions on the requirements choices made before implementation. This fits the methodology I followed throughout the project which is to keep the stable components that already do their job well.

4.4 Measurement cycle and orchestration

The main runtime is centered on a measurement cycle that collects data from all sensors, normalizes it and produces one record for storage. This cycle supports burst sampling and aggregation to reduce short-term variability before interpretation. It also tracks plausibility ranges, success ratios and quality indicators such as coefficient of variation. This means that it keeps uncertainty and outlier sensitivity explicit instead of hiding them in a single final value. I adopted burst sampling over single-read cycles because single reads can produce unstable values that degrade overall data quality. Figure 4.3 illustrates the decoupling between the fast collection interval and the slower publish interval.

Instead of setting the OS clock, the runtime applies a UTC offset in application space through `TimeSyncService`. This makes it a lot simpler to run. If the NTP synchronization fails, the system falls back to the local clock and logs the failure in the diagnostics.

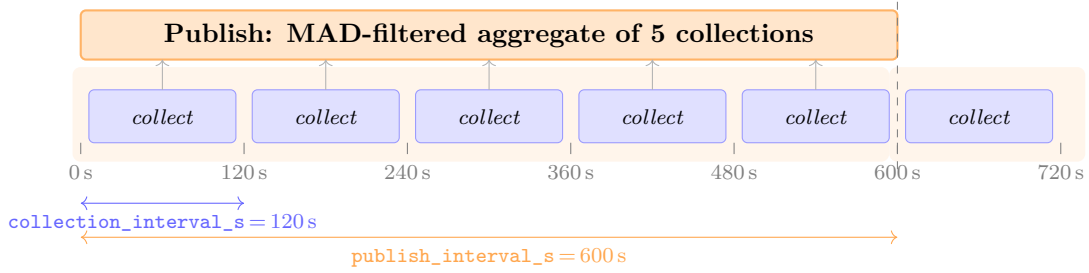


Figure 4.3: Two-tier timing of the measurement loop. Raw sensor bursts (*collect*) fire at the fast `collection_interval_s` (default 120s). Once the `publish_interval_s` window (default 600s) closes, the five collected records are filtered and aggregated into one smoothed record that is persisted and forwarded. Filtering then occurs on two levels. First, a MAD outlier pass within each individual burst’s raw reads. Second, another filter pass across the collection records at publish time (see Section 5.3.1). This decoupling keeps noisy individual reads from corrupting data reads without sacrificing the overall multi-sample noise rejection within each window.

4.5 Persistence and remote synchronization

Persistence is split into a local authoritative layer and a remote synchronization layer. The `DataManager` always appends the record to the local file first. That file is the main archive and it remains usable even if no remote service is reachable. After the local write, the same record is pushed to InfluxDB when it’s able to do so. If that push fails, then the record is added to a JSONL offline queue and retried later.

A network error should not create data loss and a temporary backend failure should not interrupt the station. Keeping local CSV as the first storage step and treating InfluxDB as a synchronization target allows immediate visualization and offline recovery without having to introduce a heavier broker or database layer.

What reaches InfluxDB is purposefully filtered. Only stable numeric fields become measurements and any high-cardinality or free-form strings will stay behind in the CSV archive. This allows the remote schema to stay simple and fast, while at the same time the local archive still holds everything just in case for later analysis or debugging.

4.6 Debug interface and local feedback

The local debug UI exposes the current status, configuration views and limited config-editing capabilities through a lightweight Hypertext Transfer Protocol (HTTP) server. It is not intended to replace a full administration interface. It is a focused tool meant for inspection, diagnosis and controlled parameter changes during deployment by both students and TAs.

The interface has several guards built in. Students cannot change the settings or view safety critical information while TAs can by entering a password. Some options cannot be edited at all because they are safety-critical or tied to the hardware. So the debug UI works more as a local maintenance platform used to supervise the correct functioning of the station instead of it being a general-purpose control plane.

This choice is completely intentional. The platform does not bury its operational complexity under a large management stack. It instead keeps the maintenance path visible and constrained which allows it to answer the requirement of keeping the system easily maintainable and robust for potential future changes.

Chapter 5

Data acquisition and processing

This chapter covers the sensor suite, the laboratory calibration that traces each channel to a known physical reference and the two-stage noise-rejection pipeline that filters raw readings before they reach storage.

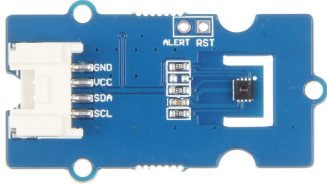
5.1 Sensor suite

The platform combines three numeric measurement domains with one visual observation path:

- **Air conditions:** The SHT35 sensor measures air temperature and relative humidity via the I²C bus (Seeed Studio, n.d.). Its $\pm 1.5\%$ relative humidity (RH) and $\pm 0.1^\circ\text{C}$ accuracy (Sensirion, n.d.) is the technical foundation for the system's VPD calculations (Massmann et al., 2019).
- **Light spectrum:** The TCS3448 (Color 21 Click (MikroElektronika, n.d.)) reads light on 14 separate channels. It has eight narrow-band filters (F1–F8), three broad "Commission Internationale de l'Éclairage" (CIE) XYZ-type channels (FZ $\leftrightarrow$ Z tristimulus, FY $\leftrightarrow$ Y/luminance, FXL $\leftrightarrow$ X-long wavelength), a Near-Infrared (NIR) channel, a clear broadband visible channel (2x_vis_1) and a Flicker Detection (FD) channel that picks apart the frequency of artificial light. Where an ordinary RGB sensor collapses everything into three bands, these 14 channels are enough to reconstruct a spectral power distribution.
- **Plant mass:** A load cell wired to an HX711 24-bit analog-to-digital converter (ADC) (SparkFun Electronics, n.d.) tracks mass. Those 24 bits are what let the station pick up the sub-gram swings evapotranspiration produces in a small potted plant.

- **Visual observation:** A Raspberry Pi Camera Module (v1.3, 5 MP) takes images at fixed intervals, while a dedicated vision service handles focus, exposure and storage. I kept the v1.3 for cost and availability. Its fixed-focus lens resolves leaf area well enough at the short working distances inside this enclosure.

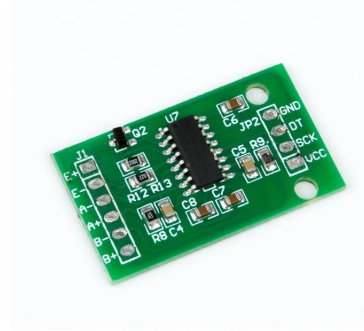
Every sensor sits behind the same (**BaseSensor**) interface. The hardware is shown in Figure 5.1. They all share the same lifecycle. First, an initialization phase (**setup()**), then a data-retrieval phase (**read_data()**). These are what lets the main loop treat all of the sensors as interchangeable data producers. A failed read returns a structured error payload rather than raising an exception. This makes it so in the case of one bad component, it doesn't bring down the whole acquisition loop.



(a) SHT35 (Air conditions).



(b) TCS3448 (Light spectrum).



(c) HX711 (Plant mass).

Figure 5.1: The core numeric sensor suite of the PhenoHive station which were all selected for research-grade precision in an educational context.

5.2 Sensor calibration and metrological validation

A raw sensor reading is of no use for phenotyping until it can be checked, trusted, and traced back to a known physical reference. In the context of an educational platform, this is particularly important because students need to be able to trust the data enough to reason about plant physiology and work with the collected data. Also, TAs need to be able to reproduce or check the calibration without any special expertise. The workflow I built answers both of these requirements by cross-referencing each sensor against traceable laboratory instruments at the WELCOME (Wallonia Electronics and Communications Measurements) lab (UCLouvain, n.d.-b) and writing the resulting corrections straight into the station configuration.

5.2.1 Calibration methodology

The calibration procedure was done manually by using a steady-state reference sweep (Figure 5.2). The target conditions were adjusted directly on the reference laboratory instruments (the ESPEC climatic chamber and the monochromator combined with the optical setup) for each measurement setpoint. Once a stable and steady-state condition was reached, the reference values were recorded. At the same time, the raw readings from the studied sensor of the PhenoHive station was retrieved. By combining these reference measurements with the raw sensor outputs for each setpoint, a dataset was produced from which the calibration offsets and scaling factors were calculated. These offsets and scaling factors were then applied as correction values in the station's config file. This process ensured that each sensor's output was grounded in a known physical reference, allowing for accurate and reliable data collection during phenotyping experiments.

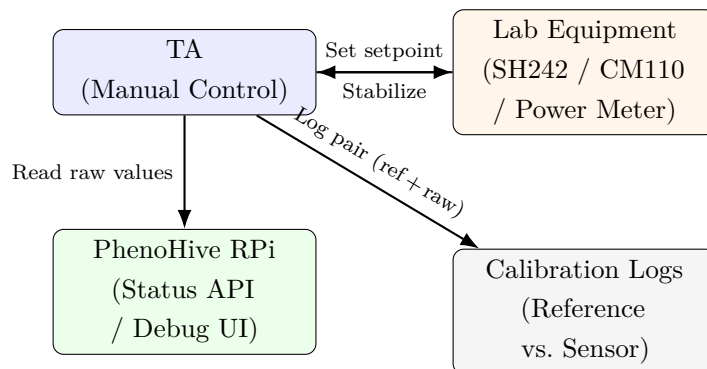


Figure 5.2: Manual calibration workflow. Each setpoint was manually set on the reference equipment after which the raw sensor values were read after stabilization from the RPi API. Each logged calibration pair combines both the reference instrument reading with the corresponding raw sensor output in order to create the calibration dataset.

Throughout the manual calibration the RPi kept on running as it normally would and was not put into a special calibration mode. This shows that the collected data is representative of the actual working conditions of the station.

In order to assist with the data collection, some lightweight Python scripts were used to continuously query the RPi's API. By comparing the live, raw outputs of the sensors against the stabilized laboratory equipment displays, the offsets and scaling factors were then calculated and written to the station's configuration file.

5.2.2 Environmental sensor calibration

The SHT35 temperature and humidity sensor was calibrated against an Espec SH-242 climatic chamber (Espec Corp, n.d.) which is a reference-grade instrument with active temperature and humidity regulation and factory-stated setpoint accuracies of $\pm 0.3^{\circ}\text{C}$ and $\pm 3.0\%$ RH (Figure 5.3). These reference accuracies set a floor on what this sweep can actually resolve. The SH-242’s setpoint tolerances ($\pm 0.3^{\circ}\text{C}$, $\pm 3.0\%$ RH) are in fact coarser than the SHT35’s own datasheet accuracy ($\pm 0.1^{\circ}\text{C}$, $\pm 1.5\%$ RH), so the exercise is best understood as a *verification that the factory calibration still holds* rather than as a correction that improves on it: a chamber cannot certify a sensor to a tighter bound than its own uncertainty. The measured mean deviations ($+0.14^{\circ}\text{C}$, -1.53% RH) are smaller in magnitude than the chamber’s setpoint tolerance, which is the expected outcome of a sensor that agrees with the reference to within the reference’s own uncertainty. The calibration followed a four-point corner sweep made to cover the extremes of the typical operating range. Two temperature levels (20°C and 40°C) were crossed-referenced with two humidity levels (40% and $\approx 80\%$ RH). The chamber stabilized at 83% RH rather than exactly 80% at the 20°C /high-humidity setpoint which is why that condition is reported as 83% in the results. This choice of setpoints spans a bit more than the typical range for indoor plant experiments and captures both the offset and any temperature dependent drift in the humidity reading.



Figure 5.3: The Espec SH-242 used for temperature and humidity sensor validation (Espec Corp, n.d.).

For each setpoint, the chamber was left for 15 minutes before the measurement was made in order for it to stabilize.

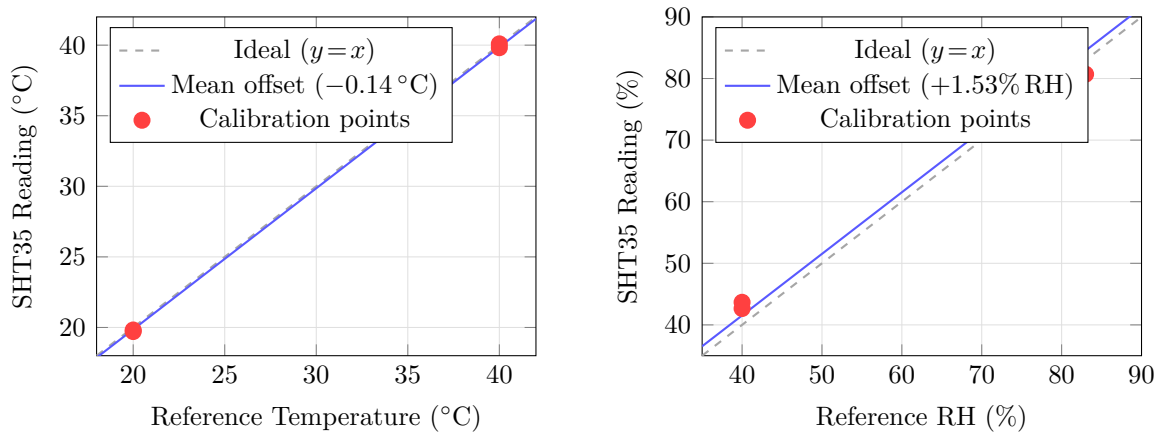
Table 5.1: SHT35 calibration results against the SH242 climatic chamber. Each row corresponds to one stabilized setpoint. The error columns show the difference between the sensor reading and the chamber reference.

Setpoint	Ref. T (°C)	SHT35 T (°C)	Ref. RH (%)	SHT35 RH (%)
20°C / 40%	20.0	19.82 (−0.18)	40.0	43.67 (+3.67)
20°C / 83%	20.0	19.71 (−0.29)	83.0	80.70 (−2.30)
40°C / 40%	40.0	39.82 (−0.18)	40.0	42.66 (+2.66)
40°C / 80%	40.0	40.09 (+0.09)	80.0	82.10 (+2.10)

Mean corrections: $\Delta T = +0.14^{\circ}\text{C}$, $\Delta RH = -1.53\%$. Applied as linear corrections in `config.ini`.

The results (Table 5.1) show that the temperature readings are stable across the entire range with deviations consistently below $\pm 0.3^{\circ}\text{C}$. Humidity shows a slightly larger spread however. The sensor reads high at low humidity and low at high humidity. This is consistent with the accuracy characteristics documented in the sensor datasheet (Sensirion, n.d.). The key metrological observation is that the deviations are small

and consistent across the full operating range, which confirms that the sensor remains within its datasheet specification. The temperature deviation (-0.14°C mean) sits below both the SHT35's own $\pm 0.1^{\circ}\text{C}$ accuracy and the chamber's $\pm 0.3^{\circ}\text{C}$ tolerance, so it cannot be distinguished from reference noise; the $+0.14^{\circ}\text{C}$ correction is therefore applied conservatively and is not claimed to improve accuracy beyond the reference. Humidity shows a larger but repeatable trend (reading high at low RH), so the mean -1.53% offset is applied as a first-order linear compensation. With these corrections, the combined uncertainty stays within the bounds required for meaningful VPD-based transpiration analysis in the target operating range. Figure 5.4 visualises the four calibration setpoints and the fitted mean offsets for both variables.



(a) Temperature: consistent offset of -0.14°C . (b) Humidity: mean offset of $+1.53\% \text{ RH}$ with spread.

Figure 5.4: SHT35 sensor readings put against the climatic chamber reference at the four calibration setpoints. The dashed diagonal represents what the perfect agreement would look like and the solid blue line is the fitted mean offset. Temperature behaviour is linear and consistent because all of the points fall within $\pm 0.3^{\circ}\text{C}$ of the offset line. The humidity points however show an increased residual spread across the range from $+3.67\%$ at $40\% \text{ RH}$ to -2.30% at $83\% \text{ RH}$ which indicates non-linearity that the single global correction ($-1.53\% \text{ RH}$) doesn't fully capture.

5.2.3 Spectral sensor calibration

The TCS3448 spectral sensor was calibrated against a CM110 monochromator (Oceanhood, n.d.) (the wavelength source) and combined with a reference power meter. The monochromator (Figure 5.5) isolates a narrow-band wavelength from a broadband source and directs it toward the sensor, while the power meter provides a traceable irradiance reference at each wavelength step.



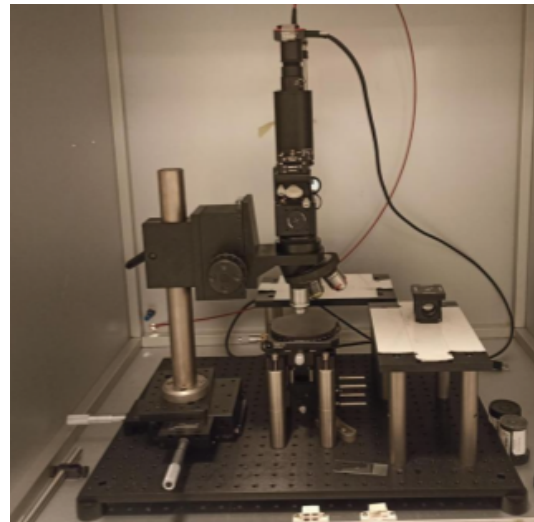
(a) Oceanhood CM110 Monochromator.



(b) Stabilized broadband light source.



(c) CM1210 Monochromator Controller.



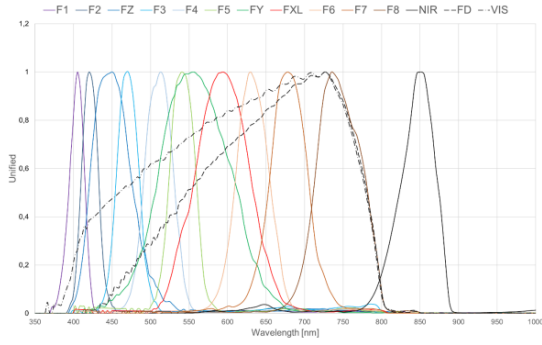
(d) Full optical calibration bench setup.

Figure 5.5: The spectral calibration setup at the WELCOME lab, including the monochromator for wavelength isolation, the power meter, the stabilized light source and the test bench configuration (Oceanhood, n.d.; UCLouvain, n.d.-b).

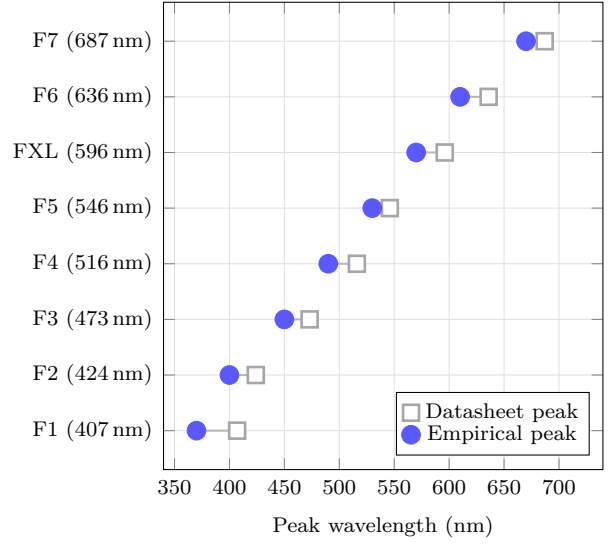
The sweep ran from 300 nm to 940 nm in 20 nm steps. It needed two diffraction gratings as per the test bench configuration. Grating 1 (2400 G/mm, blaze 240 nm)

handled 300–680 nm, and Grating 2 (1200 G/mm, blaze 750 nm) handled 600–940 nm. A grating fans broadband light out into its separate wavelengths. The groove density sets how finely it resolves them, and the blaze angle tilts the diffraction efficiency toward a peak at the design wavelength, which is what keeps the signal-to-noise ratio high across the sweep. At each step I logged the monochromator output power on the reference power meter and, at the time, read the TCS3448 raw counts from the RPi API using the simple script.

The sweep showed that the empirical peak wavelengths of the visible TCS3448 channels (F1–F7) sit consistently below the typical values listed in the manufacturer datasheet (ams-OSRAM, 2024), by about 25 nm on average (individual channels range from ~ 16 nm to ~ 37 nm). The empirical peaks of F8 (datasheet typical 748 nm) and NIR (datasheet typical 855 nm) fall well within the 20 nm resolution of the sweep of their datasheet values, showing no comparable offset. Two non-exclusive mechanisms can explain the visible-channel offset. The first is a normal sensor property which is the manufacturing-lot variation. The second is that a near-uniform offset across all visible channels also indicates a small systematic wavelength-calibration offset in the monochromator itself. The current single benchmarking data cannot fully separate the two causes. Resolving this issue would demand an independent wavelength reference such as a calibrated spectral line source, identified as future work in Section 9.4. For the platform’s current single-station use the practical impact is limited: the offset is consistent and repeatable, so within a given station the channel ordering and the relative spectral structure are preserved (Figure 5.6). The distinction is not irrelevant, however, for *absolute* wavelength labelling: if the shift originates in the monochromator rather than the sensor, the sensor’s true peaks remain near the datasheet values and the empirical mapping would mislabel them by about 25 nm. Settling this requires an independent wavelength reference or a per-station sweep, both identified as future work in Section 9.4. With that caveat, the offset is encompassed by the empirical channel mapping, derived from the wavelengths where each channel showed its maximum response. This is summarized in Table 5.2 and used throughout the platform.



(a) Theoretical responsivity. © ams-OSRAM AG, reproduced from the TCS3448 datasheet for academic comparison (ams-OSRAM, 2024).



(b) Empirical peak wavelengths from the monochromator sweep vs. the datasheet's nominal values. Each row shows one channel where the circle is what's the measured empirical peak and the square is what's the datasheet value. The horizontal gap is the shift of the visible channels. They range from ~ 16 nm (F5) to ~ 37 nm (F1), with a mean of about 25 nm. F8 and NIR are not displayed because they aligned with their datasheet peaks within the sweep resolution.

Figure 5.6: Comparison between nominal peak wavelengths from the TCS3448 datasheet and the empirical peaks determined by the monochromator sweep.

Table 5.2: Empirical peak wavelengths of the TCS3448 channels measured by the monochromator sweep. The “Empirical peak” column reflects the wavelength range at which the channel registered its highest raw count during the sweep (Grating 1 was used for channels peaking below 680 nm, and Grating 2 for channels peaking above 680 nm).

Channel	Datasheet peak (nm)	Empirical peak (nm)	Dominant response range
F1	407	~360–380	UV/Violet
F2	424	~400	Violet
F3	473	~440–460	Blue
F4	516	~480–500	Cyan/Green
F5	546	~520–540	Green
FXL	596	~560–580	Yellow/Orange
F6	636	~600–620	Orange/Red
F7	687	~660–680	Red
F8	748	~720–760	Far Red / NIR edge
NIR	855	~840–900	Near-Infrared

The complete calibration state applied to the physical station is stored in `config.ini`. Table 5.3 lists the values as deployed in each station.

Table 5.3: Deployed TCS3448 calibration state as stored in `config.ini` on the station. Scale factor converts raw ADC counts to calibrated irradiance (mW/m^2) after dark subtraction. The dark offset is the residual count observed with the sensor fully covered.

Channel	Scale factor	Dark offset (counts)	Empirical peak (nm)
F1	0.880	0	$\sim 360\text{--}380$
F2	0.864	0	~ 400
F3	1.063	0	$\sim 440\text{--}460$
F4	1.085	0	$\sim 480\text{--}500$
F5	2.90	0	$\sim 520\text{--}540$
F6	9.81	1	$\sim 600\text{--}620$
F7	9.68	2	$\sim 660\text{--}680$
F8	9.21	2	$\sim 720\text{--}760$
FZ	0.96	0	Broad (CIE Z tristimulus)
FY	2.0	0	Broad (CIE Y / luminance)
FXL	2.36	0	$\sim 560\text{--}580$ (empirical)
NIR	11.54	4	$\sim 840\text{--}900$
2x_vis_1	1.0 (uncal.)	2	Broadband visible
fd_1	1.0 (uncal.)	79	AC-coupled (flicker)

From the sweep data (Table 5.4), a per-channel scaling factor was computed by combining the power-to-count ratio from the sweep with a normalization to spectral irradiance:

$$s_k = \frac{P_{\text{ref},k}}{C_{\text{peak},k}} \cdot \frac{10^{-3}}{A_{\text{eff}}} \quad (5.1)$$

where $P_{\text{ref},k}$ [μW] is the optical power measured by the reference power meter at the wavelength of peak response for channel k , $C_{\text{peak},k}$ is the corresponding peak ADC count from the calibration sweep, and $A_{\text{eff}} \approx 7.0 \times 10^{-8} \text{ m}^2$ is the effective per-channel optical aperture area according to the TCS3448 datasheet (ams-OSRAM, 2024). The 10^{-3} factor is what converts μW to mW . These factors convert raw ADC counts directly into spectral irradiance in mW/m^2 , a range (typically $10\text{--}5000 \text{ mW}/\text{m}^2$ per channel under indoor lighting) that is human-readable on the dashboard and matches the conventions used for photosynthetically active radiation (PAR) measurements.

Table 5.4: Sweep results on Grating 1 for selected channels. For each monochromator wavelength, the power meter reference and the dominant TCS3448 channel response are shown. Channels not listed peaked below the noise floor at the given wavelength.

λ (nm)	Ref. power	Peak channel	Peak counts
380	0.333 μ W	F1	5 404
400	0.471 μ W	F2	7 781
440	0.826 μ W	F3	10 121
480	1.220 μ W	F4	14 184
500	1.319 μ W	F4	17 364
520	1.584 μ W	F5	7 782
560	2.006 μ W	FXL	10 625
600	2.222 μ W	F6	13 397
620	2.205 μ W	F6	24 225
640	2.205 μ W	F6	17 677
660	2.115 μ W	F7	21 720
680	2.031 μ W	F7	20 502

A second calibration step addressed the sensor’s electrical noise floor. Even in complete darkness, each channel produces a small number of counts due to some noise in the silicon photodiodes. These dark counts vary by channel. Longer-wavelength channels (F7, F8, NIR) exhibit higher noise than the visible-range channels. The dark baseline was measured by physically covering the sensor and recording the residual counts. Per-channel offsets were then subtracted before the irradiance scaling is applied, so the station reports exactly zero under no-light conditions.

During the second grating sweep (Grating 2, 600–940 nm), the grating’s higher power output pushed multiple channels into the 16-bit ADC ceiling of 65 535 counts. Those points were saturated so they were dropped from the coefficient calculation. The 600–680 nm overlap between the two gratings is used as a sanity check. I observed that each channel that peaked at a given wavelength peaked there under both gratings which confirms the channel mapping. Because the saturated peak samples were discarded, the long-wavelength scale factors (F6–F8, NIR) were computed from the largest *unsaturated* counts available rather than from the true peak counts; this is why their factors in Table 5.3 are several times larger than the visible-channel factors and cannot be reproduced directly from the unsaturated Grating 1 counts shown in the excerpt of Table 5.4.

Everything the calibration produces is in `config.ini`. Recalibrating, if need be,

only requires editing one flat file and restarting the service, with no code changes at all.

The same precision to accuracy split observed on the SHT35 shows up again here. The TCS3448 channels are precise and under steady light their responses are consistent and repeatable. The systematic wavelength shift is an accuracy problem, there's a fixed gap between where each channel is supposed to peak and where it actually does. The empirical channel mapping and the per-channel scaling factors from the monochromator sweep cancel that gap, so accuracy is restored without touching the sensor's underlying precision.

Measurement uncertainty. The 20 nm sweep increments restrict each channel's peak wavelength to about ± 10 nm. This uncertainty is fine for the broad visible channels (F1–F7) but becomes less ideal for the narrower long-wavelength ones (F8, NIR). The power meter's own accuracy rating also contributes to the scaling-factor uncertainty.

5.2.4 Scale and camera calibration

The load cell and HX711 ADC were carried over from Colinet (2025), who already showed they are adequate for evapotranspiration tracking on PhenoHive, so I ran no full lab sweep on them. The inherited two-point procedure is the right amount of effort for a teaching station. The tare value, the zero-load ADC reading, is taken with an empty platform, and the calibration factor comes from one known mass placed on the scale. Both numbers go into `config.ini` and are applied at read time. A dashboard wizard on the DebugUI webpage steps the TA through the two actions, which cuts down on slip-ups during deployment.

The platform now reports the plant’s growth in real-world units instead of just a plain pixel count which is hard to interpret. For correct calibration, the TA sets a `pixel_to_cm_ratio` through a semi-automated step where an object of known size sits in the field of view. The system then applies that ratio to its skeleton-length figure which means that students can read growth in centimeters. The station can also capture a background image on demand which gives it a clean reference for shadow removal and background subtraction that sharpens the morphological measurements. This was retained from the previous version of the platform, but the calibration step was made more user-friendly by integrating it into the DebugUI.

5.3 Data gathering and signal processing

PhenoHive has to turn physical measurements into biological indicators a student can trust. This means that data quality is the whole point here. If there’s too much noise, the students start doubting the readings instead of studying plant physiology. My work therefore went into two things. First, a standard way to integrate sensors and second, a two-stage MAD filter pipeline that cleans and aggregates the raw signals before they reach storage and visualization.

5.3.1 Data smoothing and noise rejection

Raw sensor data can contain sudden spikes from electrical interference, I²C bus jitter or physical disturbances to the scale. The pipeline applies Median Absolute Deviation (MAD) filtering (Leys et al., 2013) at two stages to reject outliers before they reach the stored record. For a set of n values, the filter computes the median $\tilde{x}$ and discards any value that satisfies the condition:

$$|x_i - \tilde{x}| > k \cdot \text{MAD}, \quad \text{where } \text{MAD} = \text{median}(|x_i - \tilde{x}|) \quad (5.2)$$

with sensitivity factor $k = 3$ by default (configurable). At least three samples are needed for meaningful outlier detection.

Stage 1 — burst-level. A sensor is read n times within one collection event and those reads are filtered even before they become a collection record (Figure 5.7). This allows for fine-grained spike rejection in noisy deployments.

Stage 2 — cross-collection. At the close of each publish window, the five records that were gathered over 600s all pass through the same MAD filter and are then averaged into the one single record that's then persisted locally and forwarded to InfluxDB (Figure 4.3).

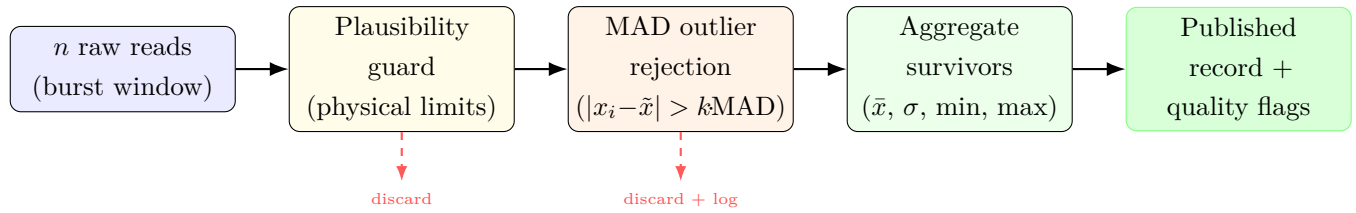


Figure 5.7: Per-collection signal processing pipeline. Each collection event makes n raw sensor reads where the values that are outside of physical plausibility bounds are immediately discarded. The MAD filter then removes each statistical outlier from the remaining burst. This means that the surviving samples are all aggregated into one smoothed value with quality metadata appended for storage and display.

Figure 5.8 shows the cleaning effect on a real temperature trace. Before the MAD filtering stage, the plausibility guards throw out physically impossible values, such as humidity outside 0–100% or negative illuminance, and logs them as invalid readings.

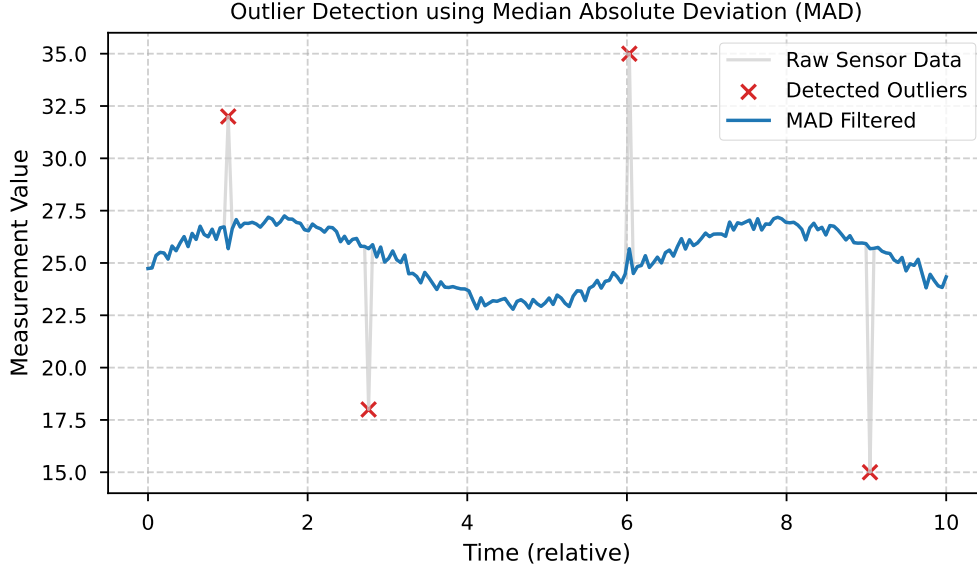


Figure 5.8: Signal cleaning using Median Absolute Deviation (MAD). This two-stage pipeline identifies and removes any transient spikes that are caused by electrical interference or physical disturbances. This prevents them from corrupting any long-term trend analysis.

5.3.2 Confidence flags and quality metadata

When publishing the aggregated record, the system does not only output a single smoothed value. It also includes statistical indicators of the signal’s stability over the window, such as the minimum, maximum and standard deviation. Each sensor reading is also tagged with confidence metadata. The system tracks the ratio of successful reads to attempted reads and computes the coefficient of variation (CV) within the burst:

$$CV = \frac{\sigma}{\mu} \quad (5.3)$$

where the sample mean μ and sample standard deviation σ are calculated by the acquisition pipeline from the set of valid, post-filtering samples collected during the burst window. If the success ratio goes below a configured threshold or the CV exceeds the maximum allowed variation, the record is then flagged with a `low_confidence` marker.

5.3.3 Acquisition summary

Taken together, the acquisition layer pulls back from a hardware-first view. It hides the hardware specifics, filters out physical noise, and holds every channel to a calibration that traces back to laboratory references.

After calibration, the temperature reading is verified to agree with the reference

chamber to within its $\pm 0.3^{\circ}\text{C}$ setpoint tolerance, with the SHT35's own datasheet accuracy ($\pm 0.1^{\circ}\text{C}$) remaining the limiting figure. The humidity measurement holds around $\pm 2.2\%$ RH across the usual 40–80% indoor range. The Spectral channels now are now mapped to physical irradiance units with per-channel dark subtraction. This means that what reaches the student is data they is ready to be read and interpreted biologically.

Chapter 6

Storage, visualization and access control

Two constraints drive the storage and access-control design. The edge nodes run unattended in student homes and face unstable connectivity conditions, so local storage must work independently of network availability. All twenty stations write to the same InfluxDB instance, so the schema must enforce per-group isolation at the query level, not just at the dashboard level. This chapter describes both these things. The local-first persistence model on the edge side, and the InfluxDB schema and multi-tenant access-control model on the server side.

6.1 Edge storage and local caching

The edge runtime keeps two copies. Every measurement first goes into local storage before the station ever tries to push it upstream, so a network outage cannot lose data. This follows the local-buffer pattern that production IoT systems use wherever connectivity is unstable (Wakili & Bakkali, 2026).

6.1.1 Flat-file CSV logging

On the actual device, the long term backup solution is a simple csv file used as a log. The `DataManager` python script is what appends each measurement row to the file. Picking CSV over a local SQLite database was intentional. If a power cut interrupts a write, it can corrupt the database file. And to read a CSV you can just insert the SD card into any computer, with no database tooling or special knowledge in the way.

6.1.2 Offline JSONL caching queue

When the central InfluxDB is offline or unreachable, the device has to buffer its data so none is lost. Because of the fact that an in-memory queue would vanish on power loss, the station instead keeps a persistent retry queue in **JSON Lines (JSONL)** format at `/opt/phenohive/data/offline_queue.jsonl`.

A plain JSON array would need rewriting on every addition which is why a JSONL file was used. This is because it just appends each new record as one line so this means that the write cost stays flat however long the queue grows.

6.1.3 Synchronization protocol

The data lifecycle follows the state machine in Figure 6.1. A new measurement first goes to the local csv file, then to InfluxDB if it's available. If that push fails, then the record is appended to the offline queue and is retried after the next acquisition cycle. This keeps it out of a busy-retry loop which could block the system. On a successful push however, the entire backlog is flushed out in one pass, and the queue now will only contain the valid records that couldn't be pushed.

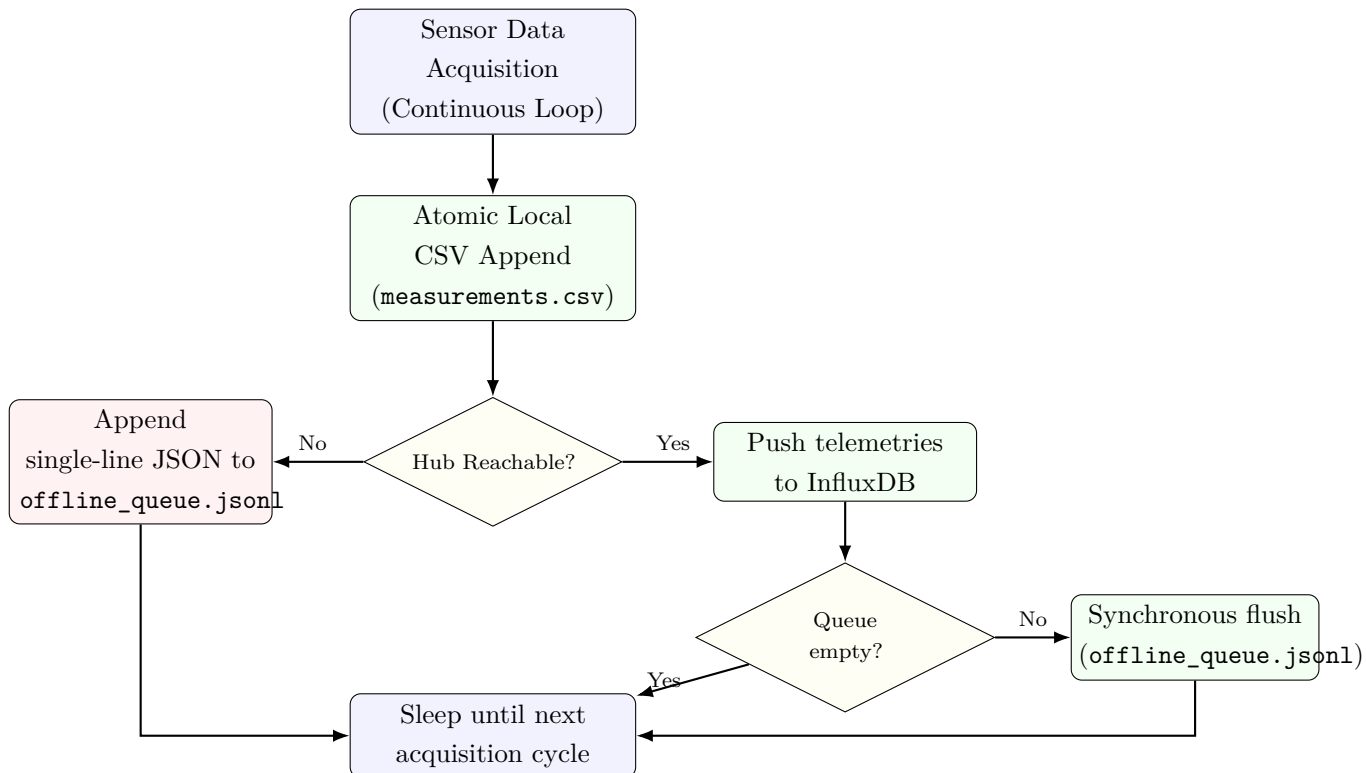


Figure 6.1: Edge node data synchronization flow. Acquired readings are immediately logged to local flat-file CSV for physical resilience, then routed depending on the state of the network connectivity. Unsent records are written to a persistent JSONL queue file and flushed out on connection restoration.

6.2 Central database schema and cardinality control

On the server, InfluxDB handles the centralized time-series storage (InfluxData, n.d.). Instead of using tables like a relational database, it lays points out along a time axis, which is a good fit for the high-frequency concurrent writes and time-range queries a fleet of stations throws at it. This choice is carried over from previous theses as it was deemed to still be the best fit for the use case, and it has proven to be robust and performant in practice.

6.2.1 Database schema layout

All PhenoHive telemetry is stored inside a single InfluxDB bucket named **phenohive** under a unified measurement called **phenohive_measurements**. To optimize indexing and search speeds, the points are organized into tags (which are indexed by InfluxDB's Time Series Index (TSI), while the underlying series data is persisted by the TSM engine, a Time-Structured Merge-tree built on LSM-tree principles) and fields (which represent the raw, unindexed numerical time-series values):

- **Tags (Indexed metadata).**
 - **station_id**: This is the main logical distinguisher (for example 01, 02). It allows for routing student queries and for keeping each group's data apart.
 - **hardware_uuid**: This is a random UUID (RFC 4122 version 4) that's created on first boot and written back to `config.ini`. This is what associates the physical device to it's corresponding **station_id**.
- **Fields (Unindexed metrics).**
 - Climate values: **sht35_air_temperature_c** and **sht35_air_humidity_pct** (calibrated), alongside the raw versions **sht35_raw_temperature_c** and **sht35_raw_humidity_pct**.
 - Spectral readings: the 14 TCS3448 channel fields (**tcs3448_f1**–**tcs3448_f8**, **tcs3448_fz**, **tcs3448_fy**, **tcs3448_fxl**, **tcs3448_nir**, **tcs3448_2x_vis_1**, **tcs3448_fd_1**).
 - Weight: **scale_hx711_weight_g** (calibrated mass in grams) and **scale_hx711_weight_raw** (the raw ADC median).

6.2.2 Series cardinality suppression

A classic way to destroy a time-series database is a series cardinality explosion. In InfluxDB a series key is the measurement name, tag set and field keys taken together.

Storing high-cardinality values as tags, things like free-form strings, dynamic error logs or timestamp markers causes the engine to have to mint a fresh index entry for every point. This leads to memory use ballooning and queries slowing down to a crawl.

In order to try and remove the possibility of a cardinality explosion, the **DataManager** instead will serialize the InfluxDB points by using a strict filtering scheme:

1. It checks the type of each key-value pair in the telemetry payload.
2. Only the pre-approved keys (`station_id` and `hardware_uuid`) are written as indexed tags. All other keys are written as unindexed fields.
3. Boolean values are cast to floats.
4. Any unstructured diagnostic texts and high-cardinality strings are explicitly discarded from the database write-path. this is done because they are already written to the local CSV on the SD card, so they remain available for offline forensics without cluttering the central database.

6.3 Multi-tenant access control and security

Keeping the groups apart is critical to the educational aspect. A student in Group 1 shouldn't be able to view or query Group 2's station. The TAs, on the other hand, need to be able to see the whole fleet and have access to every dashboard and all the data.

A two-layer security model enforces this isolation. It does this by working on both the Grafana permission layer and the InfluxDB query layer.

6.3.1 Declarative team-based permissions

Grafana's built-in team and folder permission structures (Grafana Labs, n.d.) are used as the primary authorization layer (Figure 6.2). This configuration is defined through simple CSV configuration files located in the repository under `grafana/access-control/`:

- `teams.csv`: This lists the available teams (e.g., `phenohive_station_01`, `phenohive_assistants`).
- `users.local.csv`: This maps out the individual students and TA logins to their respective teams.
- `dashboard_permissions.csv`: This is what links each specific dashboards to their permitted teams.

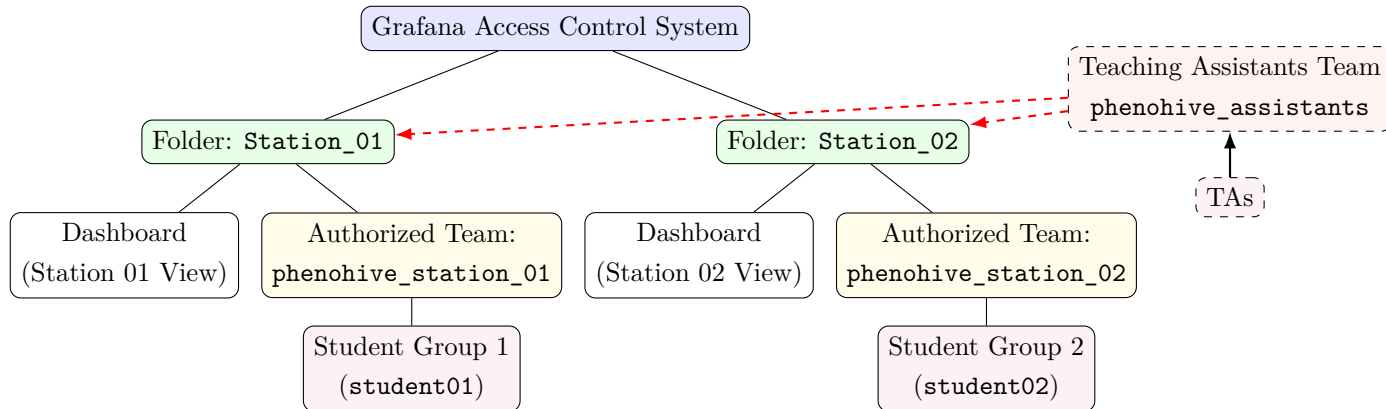


Figure 6.2: Logical topology of Grafana team-based permission boundaries. Each student group maps to one station folder with one assigned user. The Teaching Assistants team has fleet-wide dashboard access.

A dedicated Python script called `sync_grafana_access.py` reads these CSV files, calls the Grafana HTTP API, and reconciles the live system to match. It runs every 60 seconds. This allows to add a student to `users.local.csv` and will lead to the script provisioning the account with an initial password derived from the station ID (intended to be changed by the student on first login), assigning them to their team and finally removing the default viewer rights from every other station folder.

6.3.2 Query-level isolation

A misconfigured or bypassed Grafana permission rule could allow a student to reach another station's dashboard. To prevent any permission bypass from leaking data, a second isolation layer is enforced at the query level.

Every student dashboard is generated as an independent JSON template instance. During the provisioning phase, the auto-provisioning engine clones the master template and hardcodes the station identifier into every Flux query's filter block:

```
|> filter(fn: (r) => r.station_id == "01")
```

Because the dashboard doesn't have any global variables or dynamic dropdown menus allowing students to select different IDs, the browser has no way to request data from other stations. Even if a student inspects the network traffic and captures the raw Flux queries, the credentials associated with their user role restrict them from altering the query parameters on the server preventing any cross-station data access at the query level.

6.3.3 Central fleet administration

While students operate in isolated groups, TAs require a centralized administration panel to be able to oversee the entire station fleet. The **Grafana Admin Panel** dashboard provides TAs with:

- **Heartbeat indicators.** A color-coded status panel showing “UP” or “DOWN” for each station. The status is driven by a Flux query that checks whether the station’s `hardware_uuid` tag has produced any measurement in the preceding 60-minute window:

```
from(bucket: "phenohive")
  |> range(start: -1h)
  |> filter(fn: (r) => r._measurement == "phenohive_measurements")
  |> filter(fn: (r) => r.hardware_uuid == "<uuid>")
  |> count()
```

A non-zero count maps to “UP” (green) while zero or no series maps to “DOWN” (red). This one-hour window is a deliberate choice because it tolerates the default 10-minute publish interval plus possible network errors without generating false alerts, while still flagging potetial problematic stations.

- **Raw logging table.** This is just one single unified table that shows the latest records from every active station. Each row contains the station ID, hardware UUID, last-seen timestamp and the last published sensor metrics(temperature, humidity, light and weight). This allows a TA to get a health check on the entire fleet of stationswithout having to open up a single dashboard.

The two panels are a deliberately minimal (MVP) admin interface. They are enough for a TA to confirm each station is alive and data is flowing.

6.3.4 Data privacy

The data PhenoHive collects falls under GDPR (European Parliament and Council of the European Union, 2016). It is nothing but environmental sensor readings tagged with a station identifier, with no personal identifiers and no location data. It lives on the UCLouvain VM, is reachable only by the TAs and the group it belongs to and is processed under Article 6(1)(e) for educational research.

Chapter 7

Deployment and operations

Getting a prototype working is different from getting it deployed at classroom scale. A student with no command-line experience or deep computer knowledge needs to be able to reach a live dashboard within minutes. Also, a TA needs to be able to add a station without needing to interact with the backend. On top of that, twenty stations must push to a shared server without interfering with each other. This chapter describes the deployment architecture that addresses each of these.

7.1 Network topology and security architecture

Each PhenoHive station must push data to the UCLouvain server from wherever a student happens to live, through whatever network is available. The main constraint is that the university VPN requires specific certificates that are difficult to manage in an automated setting. The WireGuard-based overlay and server-side security model described in this section address this.

7.1.1 Overlay network rationale

The UCLouvain VPN (UCLouvain, n.d.-a) isn't designed for devices such as this one because it relies on certificate-based authentication which makes it unsuitable for a fleet of Raspberry Pis that need to be able to connect to it without user interaction. They are also difficult to manage at scale. Another reason is that the University's VPN policy is that it's reserved for university and research staff only. What this means is that even if the technical barriers were overcome, it would not be in accordance with the university's acceptable use policy for students to connect their home devices to the VPN.

To bypass these limitations, this iteration of the PhenoHive project uses Tailscale, a WireGuard-based overlay mesh network (Donenfeld, 2017; Tailscale, n.d.). Tailscale

authenticates via pre-provisioned keys, so the Raspberry Pi that runs it has it as a native `systemd` service called (`tailscaled`) that reconnects silently on boot without any user interaction. It also uses NAT traversal to go through home routers and establish direct peer-to-peer tunnels (Figure 7.1). While Tailscale is used as the proof-of-concept coordination server due to its ease of setup, the architecture is fully compatible with Headscale which is an open-source, self-hosted alternative for long-term, cost-free production deployment that relies on the same underlying technology (Headscale, n.d.).

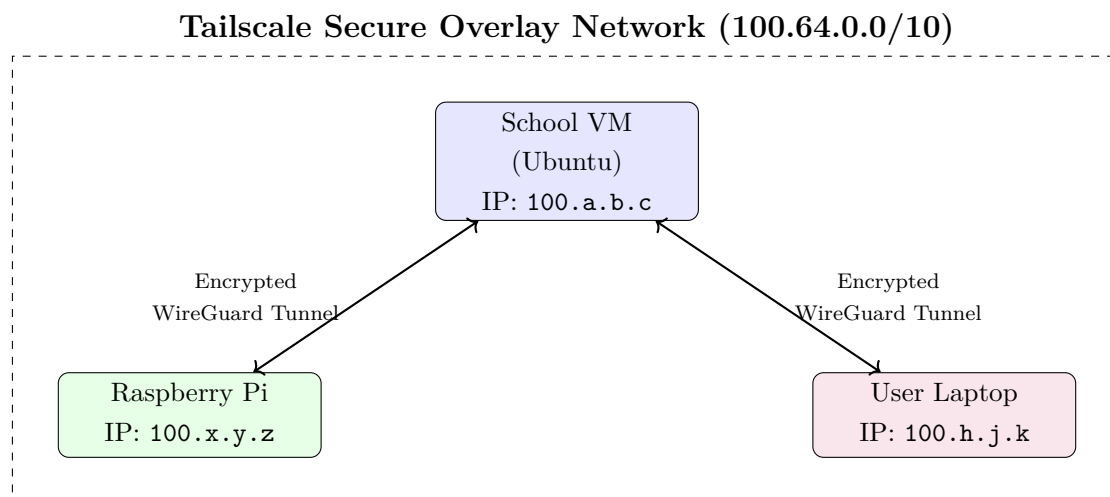


Figure 7.1: The PhenoHive overlay network topology. Tailscale establishes a secure WireGuard-based mesh that connects edge devices together with the central VM and user laptops across restrictive networks.

7.1.2 Virtual machine security and port isolation

There are three independent layers that protect the central hub VM. First, the UCLouvain firewall filters inbound traffic to the VM’s public interface. Then, a host-level UFW ruleset denies by default. And third, Docker port bindings that determine which services are reachable at all.

Public ingress and network-level security The UCLouvain’s perimeter firewall filters any inbound traffic to the VM’s public interface. While port 22 (Secure Shell, SSH) stays open to the internet for remote administration, it’s hardened at the OS level and since password authentication is off, this means that only public keys can get in. There is also a `fail2ban` service that runs at the same time and it blocks any IPs that look like they may be trying to execute a brute-force attack. The final defense is that operational ports like 3000 (Grafana) and 8081 (Nginx Ingest Proxy) are blocked outright by the university’s outer firewall for anything off campus.

In order to not rely solely on the university firewall alone, the VM also runs its own

host-level firewall, **UFW** (Uncomplicated Firewall), set to deny by default. It explicitly permits SSH on the public interface and pushes container traffic through the Tailscale adapter (`tailscale0`).

Docker network isolation and loopback bindings Inside the host OS, Docker's port socket bindings keep each service walled off from one another.

```
services:
  influxdb:
    ports:
      - "127.0.0.1:8086:8086" # Bound strictly to VM localhost

  influx_write_proxy:
    ports:
      - "0.0.0.0:8081:8081" # Open to local interfaces (secured by Tailscale)

  grafana:
    ports:
      - "0.0.0.0:3000:3000" # Open to local interfaces (secured by Tailscale)
```

Binding the InfluxDB socket to the loopback interface (127.0.0.1) keeps the database off every external network adapter, the Tailscale overlay interface included. A direct external query to port 8086 is dropped at the host kernel. The only ways in are local sockets, such as administrative CLI commands, or the Nginx write proxy, which runs in a different container on the same host network bridge.

The Nginx write proxy and Grafana are bound to 0.0.0.0 inside the VM so they can get packets off the overlay. Both of the firewalls still drop public traffic to ports 3000 and 8081 in such a way that only devices that are authenticated on the Tailscale network can reach those interfaces.

7.1.3 Edge security and gateway proxy

By letting a field-deployed device talk straight to the database could create a real risk. If someone physically tampers with a Raspberry Pi, they could pull the database credentials off it and gain full read, write, and delete rights.

In an effort to contain that risk, the Nginx Write Proxy sits on port 8081 and acts as a secure API gateway. It normalises the request formats and only accepts `POST` requests to `/api/v2/write` or `/api/v2/query`. Everything else gets a 403 Forbidden error and is therefore rejected.

```
# Nginx write-proxy excerpt (simplified)
location /api/v2/write {
    proxy_pass http://influxdb:8086;
}
location /api/v2/query {
    proxy_pass http://influxdb:8086;
}
location / {
    return 403;
}
```

So even with a compromised edge token the malicious user can only either append data reads to the **phenohive** bucket or perform queries. It cannot reach InfluxDB’s administrative and management APIs and cannot reconfigure the server.

7.1.4 From sensor to Grafana: the journey of a packet

Figure 7.2 traces the complete journey of a single packet. Sensor data is formatted into InfluxDB Line Protocol on the RPi, encrypted by WireGuard through the Tailscale overlay, proxied by Nginx into InfluxDB, and finally queried by Grafana for visualization.

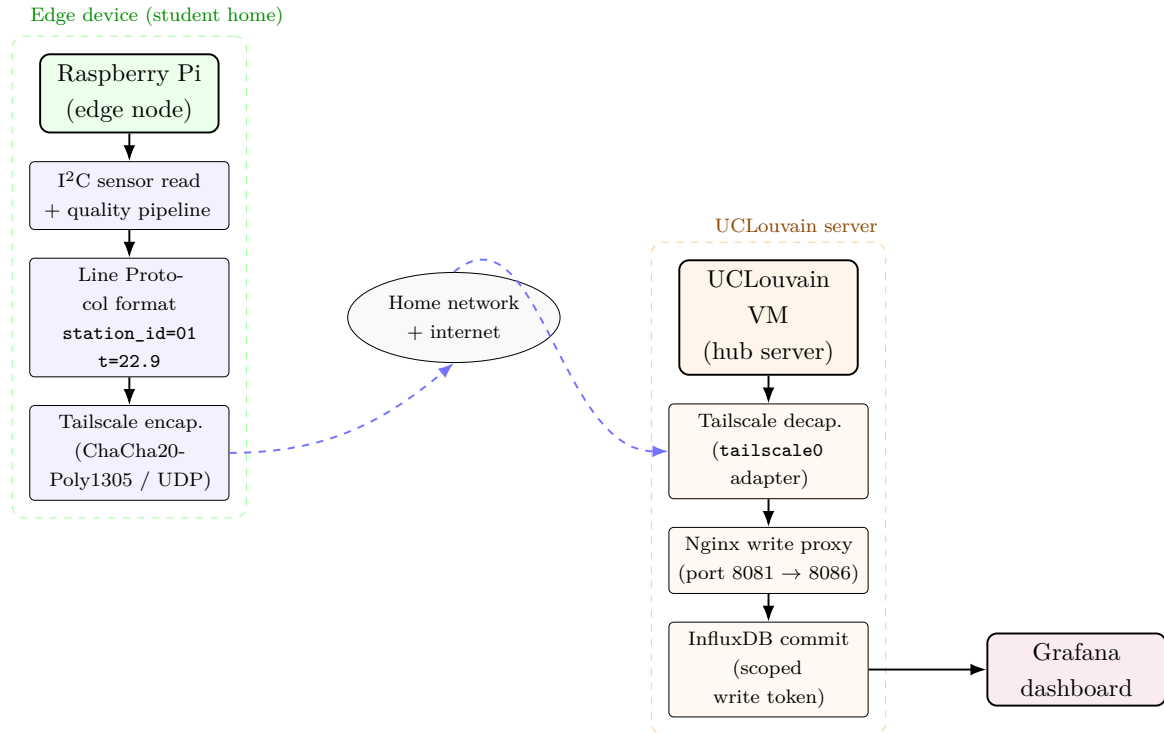


Figure 7.2: The complete path of a single telemetry packet from sensor read to dashboard. The edge device formats data as InfluxDB Line Protocol, encrypts it via Tailscale (WireGuard / ChaCha20-Poly1305), and transmits it over any available internet connection. On the server side, the Nginx write proxy restricts incoming requests to append-only writes before forwarding to InfluxDB, preventing a compromised edge token from reaching administrative endpoints. The grafana dashboard finally queries the database for visualization, completing the data journey.

7.2 Station bootstrapping and systemd sequencing

Each station is identified by a unique `station_id` string, set in its local `config.ini` file. This identifier is attached as a tag to every InfluxDB measurement and establishes the foundation for all downstream data isolation.

```
[general]
station_id = 01

[influxdb]
enabled = True
url = http://<SERVER_IP>:8081
token = phenohive-local-dev-token
org = uclouvain
bucket = phenohive
```

The station identity is deliberately kept minimal: a two-digit string is enough for classroom-scale deployments, and adding a new station requires only flashing the image, setting a unique `station_id` and the server ip in `config.ini`, and starting the service.

To ensure the station boots reliably without any command-line work, the main Python runtime is controlled by a **systemd service** with a network pre-check (as illustrated in Figure 7.3). The service unit blocks runtime execution until the network and clock are ready:

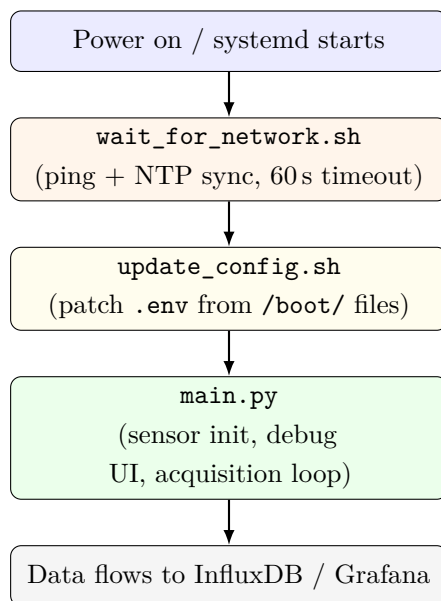


Figure 7.3: The systemd boot chain from power-on to live data acquisition. Two `ExecStartPre` directives sequence the pre-start steps. First the network and clock are confirmed as being ready, then the runtime configuration is patched from the boot partition.

```
# systemd service pre-start sequencing
[Service]
ExecStartPre=/bin/bash /opt/phenohive/scripts/wait_for_network.sh
ExecStartPre=/bin/bash /opt/phenohive/scripts/update_config.sh
ExecStart=/opt/phenohive/.venv/bin/python main.py
```

The first pre-start script, `wait_for_network.sh`, executes a loop that blocks service start until an internet connectivity is confirmed via a ping to `8.8.8.8` and until NTP clock synchronization is completed. This makes sure that all data points carry exact UTC timestamps.

The second and final pre-start script, `update_config.sh`, checks the `/boot` partition for two optional plain-text files: `phenohive_server.txt` (containing the hub's IP address) and `phenohive_token.txt` (containing the InfluxDB write token). If either file is present, then the script sets the corresponding values into `/opt/phenohive/.env`

before the main runtime starts. Because of the fact that the `/boot` partition is accessible from any operating system without needing specific mounting tools, a TA can pre-configure a bunch of stations just by writing these files to the SD card from any computer. On first boot, each station is then able to self-configure its server target and token without requiring any SSH access or manual edits to hidden Linux system files.

7.3 Auto-provisioning service

Creating teams, users, dashboards and permissions by hand for each new station is error-prone and time-consuming. The server includes an **auto-provisioning service** (`auto_provision_grafana.py`) that detects new station identifiers in InfluxDB and provisions the Grafana resources automatically.

The service executes a continuous loop with a 60-second cycle:

1. **Discovery.** It queries InfluxDB using a Flux query to find any and all distinct `station_id` tags that are currently present in the database.
2. **Provisioning.** For each discovered station ID that does not yet have a corresponding Grafana dashboard associated, the service automatically:
 - Creates a new Grafana team (`phenohive_station_XX`).
 - Creates a student user account called (`studentXX`) with an initial password that's derived from the station ID.
 - Clones the master dashboard template and injects the station-specific query filters into it.
 - Configures folder and dashboard permissions for the new team.
3. **Synchronization.** It finally runs the full access-control sync so that any manual changes that the admin made to Grafana are reapplied.

Deploying a new station takes no interaction with Grafana at all. The TA only needs to edit the station's `config.ini` and starts the service. Within two minutes of the first data push, the auto-provisioning service sees the new station ID in InfluxDB, builds every Grafana resource it needs and the student can then log in.

7.3.1 Dynamic station ID reassignment logistics

In the event of a problem, moving a station from one group to another comes down to a single `station_id` change in `config.ini`. The auto-provisioning service picks up

the new ID and builds the new student environment, while the old dashboard keeps its history, and again nobody has to touch Grafana. This can also be done using the debug UI if logged in as admin but it doesn't allow to change the station ID to an existing one.

Chapter 8

Evaluation and results

While previous chapters described the design, calibration, and deployment of PhenoHive at the component level, this chapter answers the main question: “Can a student take a fresh Raspberry Pi, connect it to any Wi-Fi network, and start collecting sensor data on a shared dashboard without opening a terminal?”

8.1 Test environment

Every physical test in this chapter ran on a Raspberry Pi Zero 2 W with DietPi 9.x, Python 3.11, and the PhenoHive service set up as in Chapter 7. On the server side, the UCLouvain VM ran InfluxDB 2.7 and Grafana 10.x, reached over the Tailscale overlay. Unless noted otherwise, the station communicated with the server over an ordinary residential Wi-Fi connection.

There are two limitations that apply to all tests. First is that the timing values in Table 8.1 come from three to five informal manual observations. This means that they are representative of typical conditions but are not statistically rigorous measurements and that the actual durations may shift with network latency and NTP server response time. The second limitation is that the 20-station concurrent load test (Section 8.3.5) used one physical Raspberry Pi station and 19 software-emulated ones all running as independent Python processes on a development machine. Those emulated stations all share a single Tailscale tunnel and therefore can’t reproduce independent residential network variability or hardware-level failures. This means that the test only validates server-side write capacity and auto-provisioning latency. End-to-end residential network behaviour still has to be confirmed in a live deployment.

8.2 Deployment and network resilience

What was tested. Headless first-boot onboarding: a student connects a fresh SD card, powers the Pi, and reaches live data on the shared Grafana dashboard with no terminal access.

On first boot, the `comitup` service creates a temporary hotspot. Opening `http://comitup.local` after connecting to it brings up a captive portal (Figure 8.1) where the student is able to pick their Wi-Fi network and type in their password (Steele, n.d.). The Pi will then reboot and joins the network if the credentials are correct. From there the systemd chain will then initialize the sensors and start pushing data.

Result. The time from power-on to live data on Grafana took under five minutes with no command-line interaction at any point. Table 8.1 breaks the timing down by phase.

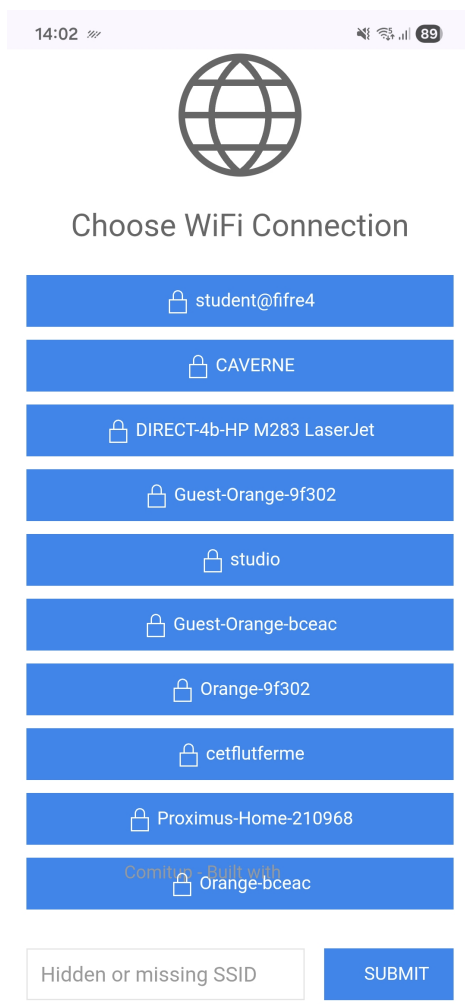


Figure 8.1: The `comitup` captive portal as seen from a smartphone. The student selects the target Wi-Fi network and enters the password. No SSH or terminal access is required.

Table 8.1: Approximate timing of each phase in the first-boot setup sequence, from power-on to live data visible on the student Grafana dashboard. Values are representative estimates from testing on a Raspberry Pi Zero 2 W; actual times depend on network speed and NTP server response latency.

Phase	Milestone	Description	Approx. duration
T0 → T1	Power-on	Boot sequence to <code>comitup</code> hotspot broadcast	≈ 45 s
T1 → T2	Portal access	Student connects to hotspot; captive portal loads in browser	≈ 15 s
T2 → T3	Network join	Wi-Fi credentials submitted; Pi reboots and joins target network	≈ 90 s
T3 → T4	Pre-start checks	<code>wait_for_network.sh</code> : internet ping and NTP synchronization confirmed	≈ 30 s
T4 → T5	Sensor init	Concurrent <code>setup()</code> calls across all sensors (15 s global timeout)	≈ 15 s
T5 → T6	First write	First measurement record successfully written to InfluxDB	≈ 10 s
T6 → T7	Provisioning	Auto-provisioning detects new <code>station_id</code> ; Grafana dashboard and student account created	< 60 s
T0 → T7	Total	Power-on to live data on student dashboard	< 5 min

8.3 Runtime behaviour

8.3.1 Debug UI and on-device calibration

What was tested. That the on-device web UI gives students access to live sensor readings and guided calibration without any terminal interaction.

The debug UI is a single-page web app that’s hosted on port 8080 of the RPI and it has no external dependencies. Its live tab (Figure 8.2) refreshes on its own and shows temperature, humidity, weight and the 14 TCS3448 spectral channel intensities. The calibration tab (Figure 8.3) walks the student through the scale calibration and the camera setup. The admins can also change settings from within this webpage instead of editing the `config.ini` file directly and see some relevant debugging information.

Result. Every calibration and diagnostics operation ran successfully through the

browser, with no command-line interaction needed.



Figure 8.2: The Debug UI live tab showing real-time readings from all sensors. The spectral bar chart (bottom) visualizes the live data from the 14 TCS3448 channels. The temperature, humidity and weight live values are displayed as large numeric cards. The interface refreshes every few seconds via Asynchronous JavaScript and XML (AJAX) polling.

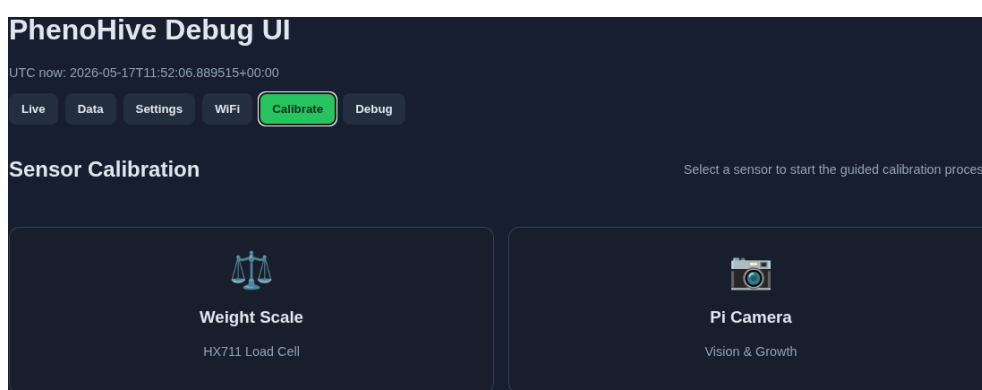


Figure 8.3: The guided scale and camera calibration wizards in the Debug UI. Students can calibrate the scale and camera from this easy to understand interface through a guided step-by-step process where no command-line interaction is needed.

8.3.2 Sensor data quality over a multi-day run

What was tested. That the full pipeline — burst sampling, MAD outlier filtering, smoothed aggregation, and InfluxDB write — captures continuous, clean environmental data over multiple days.

Figure 8.4 shows nine days of data from the physical station (2026-05-18 to 2026-05-27) across three representative channels: air temperature, relative humidity, and TCS3448 spectral channel F6.

Result. All three channels recorded continuously without any gaps in the data. This demonstrates the long term stability of the platform as a whole.



(a) Air temperature ($^{\circ}\text{C}$), 2026-05-18 to 2026-05-27. A transient spike is visible around 2026-05-22; the two-stage MAD pipeline removes it before the record is persisted.



(b) Relative humidity (% RH) which was taken during the same period. The trace is clean and continuous with no visible artefacts.



(c) TCS3448 channel F6 intensity (counts, empirical peak $\sim 600\text{--}620\text{ nm}$). It shows a clear day and night cycle that reflects the station's lighting environment.

Figure 8.4: These are nine days of continuous sensor data from the physical PhenoHive station, as visualized on the Grafana student dashboard. The three channels that are shown here (air temperature, relative humidity, and one TCS3448 spectral channel) demonstrate the pipeline's ability to capture multi-day environmental dynamics with the temporal resolution and data quality required for plant-physiology interpretation. They also show the platform's stability long term.

8.3.3 Demonstrative plant observation

What was tested. That the calibrated sensor suite does indeed detect the physiological signals that the course targets when monitoring an actual potted plant. These are cumulative water loss via the pot-mass channel, atmospheric evaporative demand derived from the calibrated temperature and humidity, and light-cycle structure from

the spectral channel.

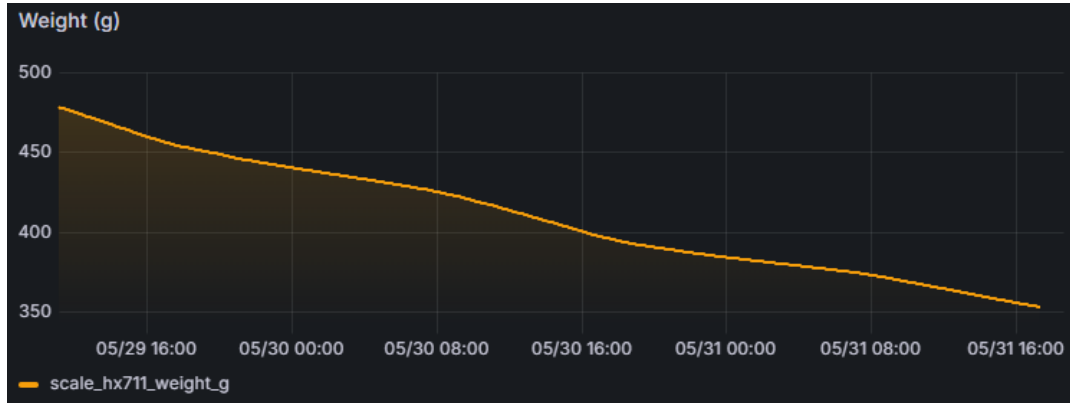
A potted plant was placed on the station and monitored from 2026-05-29 to 2026-05-31 without watering. This allowed an uninterrupted trace across all channels to be made (Figure 8.5).

The plant mass (Figure 8.5a) declined linearly from 478 g to 353 g over 54 hours which represents a total loss of 125 g or (≈ 55 g/day). Two features of this trace are notable. First is that the decline is nearly linear with no day-night acceleration. This strongly suggests that a load-cell drift under sustained load is contributing to a constant offset alongside the real water loss. The HX711 ADC is known to drift with temperature changes over hours-to-days timescales. The calibration that was carried over from Colinet (2025) provides no drift correction. Second, slope changes over shorter windows (1–2 hours) remain qualitatively interpretable. A periodic re-taring protocol which is identified as future work in Section 9.4 would restore quantitative multi-day accuracy.

Temperature (Figure 8.5b) however remained in the 29.5–31 °C range with a ≈ 1 °C diurnal amplitude. This is warmer than typical indoor conditions but was due to the weather conditions at the time of recording. Relative humidity (Figure 8.5c) also rose from ~ 42 % RH to 45–46 % RH over the test cycle with a step increase overnight on May 30th and 31st which is consistent with the plant humidifying the air that surrounds the sensor.

The TCS3448 F4 channel (Figure 8.5d empirical peak ~ 480 –500 nm, cyan-green) shows a clear natural day-night light cycle. The intensity rises from near zero at night to about 600–730 mW/m² during the afternoon. The day-to-day variation in peak intensity reflects the changing outdoor illumination instead of just some sensor artefact.

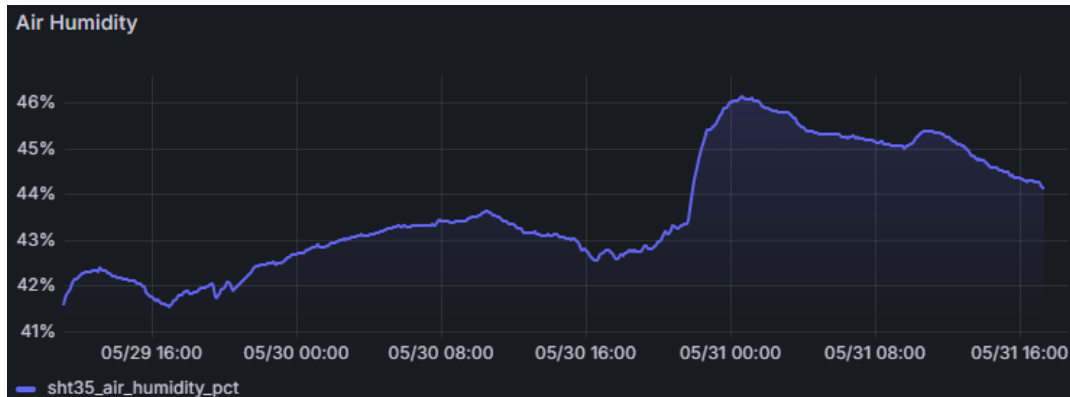
Result. The temperature, humidity and spectral channels functioned continuously and captured the expected physiological structure. The mass channel however revealed a drift artefact that limits absolute multi-day rate comparisons but does not affect short-term trend analysis. This is due to a calibration-maintenance issue, not an architectural one. Together the four channels demonstrate that the platform manages to capture the signals that the course requires and that the calibrated environmental data can already fully support the stomatal-response reasoning that the course expects.



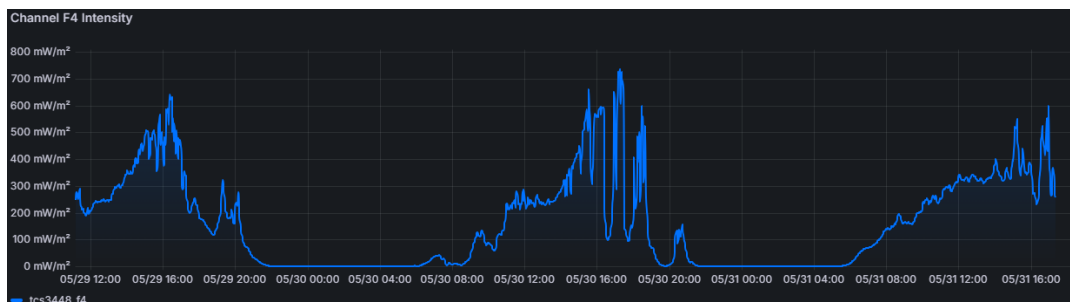
(a) Pot mass (g), 2026-05-29 to 2026-05-31. The nearly linear decline from 478 g to 353 g over about 54 h (≈ 55 g/day) combines real evapotranspiration and load-cell drift. But due to the absence of day/night rate variation, it confirms that sensor drift dominates.



(b) Air temperature ($^{\circ}\text{C}$) during the same period. Warm and stable ($29.5\text{--}31^{\circ}\text{C}$) with $\approx 1^{\circ}\text{C}$ diurnal amplitude.



(c) Relative humidity (% RH) with again the same period. The rise from $\sim 42\%$ to $45\text{--}46\%$ is consistent with the plant humidifying the nearby air.



(d) TCS3448 channel F4 (mW/m^2 , with an empirical peak at $\sim 480\text{--}500$ nm). Natural day and night cycles are clearly visible. With a near zero value at night and $600\text{--}730$ mW/m^2 at afternoon peak. The day-to-day variation in peak intensity reflects changing outdoor illumination.

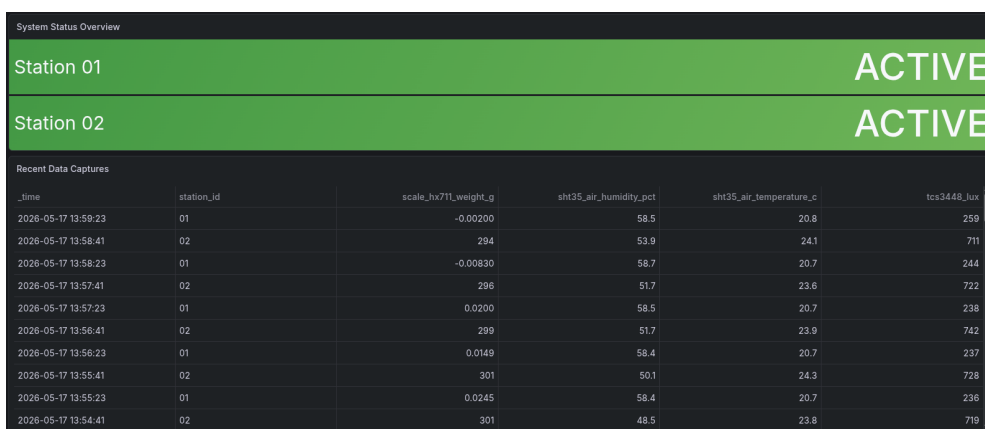
8.3.4 Multi-station isolation and dynamic provisioning

What was tested. That data is isolated per student and that reassigning a station ID triggers correct re-provisioning without data loss.

Running a physical Station 01 at the same time as a Docker-based mock Station 02 confirmed isolation at both the query and permission levels. Each student login returned only its own station’s data as expected(Figures 8.6–8.7).

To test out the dynamic provisioning path, Station 01’s ID was changed to 04 through the debug UI.

Result. The auto-sync service built the new Grafana environment (dashboard and user account) in about two minutes. The original Station 01 dashboard retained its full history but stopped receiving new data once the ID was changed. No data loss occurred during the transition.



The screenshot displays the Grafana Admin Panel. At the top, under 'System Status Overview', there are two green status bars: 'Station 01' and 'Station 02', both labeled 'ACTIVE'. Below this is a section titled 'Recent Data Captures' containing a table with the following data:

_time	station_id	scale_hx711_weight_g	sht35_air_humidity_pct	sht35_air_temperature_c	tcs3448_lux
2026-05-17 13:59:23	01	-0.00200	58.5	20.8	259
2026-05-17 13:58:41	02	294	53.9	24.1	711
2026-05-17 13:58:23	01	-0.00830	58.7	20.7	244
2026-05-17 13:57:41	02	296	51.7	23.6	722
2026-05-17 13:57:23	01	0.0200	58.5	20.7	238
2026-05-17 13:56:41	02	299	51.7	23.9	742
2026-05-17 13:56:23	01	0.0149	58.4	20.7	237
2026-05-17 13:55:41	02	301	50.1	24.3	728
2026-05-17 13:55:23	01	0.0245	58.4	20.7	236
2026-05-17 13:54:41	02	301	48.5	23.8	719

Figure 8.6: The Grafana Admin Panel showing two active stations. Station 01 runs on physical hardware and Station 02 is a Docker-based mock. Both report the “UP” status with their most recent measurements visible in the unified table.



Figure 8.7: The student-facing Grafana dashboard for Station 01 which shows the time-series panels for temperature, humidity, spectral irradiance and plant weight. Logging in as `student02` would display an identical layout but with only Station 02 data, as explained to establish query-level isolation.

8.3.5 Concurrent load test

What was tested. Server-side write capacity and auto-provisioning latency with 20 stations writing concurrently (1 physical + 19 software-emulated).

The 19 emulated stations all ran as independent Python processes on a development machine. They were all writing through to the same Nginx proxy to the shared InfluxDB instance. The run lasted a total of 3 hours at an accelerated 30s/60s collection and publish rate to simulate real-world usage.

Result. Not a single station produced an offline queue file. The auto-provisioning service had all 19 dashboards up within 60–90s of each station’s first write which confirms that the server-side write capacity and that the provisioning latency hold up at classroom level scale. As noted in Section 8.1, the emulated stations shared a single Tailscale tunnel and could not reproduce independent residential network variability. This means that the end-to-end home-network behaviour still needs validation in a live deployment.

8.4 Validation summary

Table 8.2 lists all validation scenarios and their outcomes, with traceability to the original requirements.

Table 8.2: Summary of system-level validation tests and their traceability to the original requirements. All tests were performed on physical hardware (Raspberry Pi Zero 2 W) except the 20-station load test (1 physical + 19 software-emulated).

Test scenario	Expected outcome	Outcome	Req.
Headless Wi-Fi setup (fresh SD card)	Station broadcasts hotspot, accepts credentials via captive portal, begins data acquisition	Pass (~5 min first-boot)	R3, R4
Network relocation (new Wi-Fi)	Falls back to hotspot, accepts new credentials, resumes without data loss	Pass	R3
48-hour server disconnection	All measurements buffered locally and replayed in order upon reconnection	Pass (no data loss observed)	R3
Power-cycle reboot	Automatic reconnection and uninterrupted data resumption	Pass	R3
Calibration persistence across code updates	Scale tare and factor survive deployment script execution	Pass	R3, R5
Camera concurrency	Sensor polling continues at 10s collection interval during image capture (interval reduced from 120s for test visibility)	Pass	R3
Sensor failure recovery	Acquisition loop continues for other sensors, and failed sensor data resumes automatically upon reconnection	Pass	R3
I ² C bus recovery under service restart	TCS3448 and SHT35 initialize correctly after a service restart that follows an ungraceful process termination; the in-process recovery mechanism (<code>SensorFactory._run_i2c_recovery()</code>) clears the stuck bus state before each sensor setup retry	Pass	R3
Persistent sensor failure watchdog	SHT35 or TCS3448 remaining in <code>ERROR</code> state for 30 min triggers an automatic service restart; both sensors recover within the first retry cycle of the new process	Pass (recovered in 1 retry cycle)	R3

(Table 8.2 continued from previous page)

Test scenario	Expected outcome	Outcome	Req.
Station ID conflict detection	Attempting to claim an ID owned by another device is rejected	Pass	R6
Multi-station data isolation	Each student login sees only their station's data	Pass	R6
Dynamic ID reassignment	New Grafana environment created within 2 minutes, old dashboard retains history	Pass (<2 min provisioning)	R5, R6
20-station concurrent load (1 physical + 19 mock)	All write requests acknowledged, no offline queue files, all 19 dashboards auto-provisioned within 90 s	Pass (0 queued, 19 dashboards in <90 s)	R3, R6

The validation campaign confirmed that the current implementation meets all six original requirements. Headless onboarding, multi-day network resilience, sensor-failure recovery, multi-station isolation, and concurrent 20-station operation were each demonstrated on physical hardware or a close emulation. The result that matters most in practice is the five-minute first boot: a fresh station with no prior configuration reaches live data on the shared dashboard with no command-line interaction, which is exactly the usability bar a teaching platform has to clear.

Chapter 9

Discussion

This objective of this chapter is to examine the implementation against its stated objectives, identifies what holds up under scrutiny, where the gaps remain, and what aspects require more attention.

9.1 Revisiting the thesis' original objectives

All three original objectives were met. The architecture became much clearer once the code was rewritten into four bounded source packages behind a factory-based sensor abstraction. Sensing quality came from the WELCOME-lab calibrations, which corrected a roughly 25 nm TCS3448 channel shift and brought down environmental factor uncertainty down. And deployment resilience held up on real hardware, the station handles out 48-hour disconnections without losing data and reaches live Grafana data in under five minutes from first boot.

9.2 Limits of the current implementation

9.2.1 Calibration validity and reproducibility

The environmental and spectral calibrations were only performed once on a single station at the WELCOME lab. For a platform that will eventually include around 20 units, that creates a few problems.

The TCS3448's spectral response is not perfectly uniform across all manufacturing batches. This means that the roughly 25 nm empirical shift that's documented in Chapter 5 could differ between individual sensors, but the current calibration applies

the same per-channel scale factors to every unit. For rigorous spectral comparisons across student groups, each sensor would need its own monochromator sweep.

The SHT35 calibration also only relied on a single reference which means that it carries the same risk. The factory calibration is very good. Still, the 0.1–0.3 °C offset seen during validation hints that some units may drift further than others. A batch calibration against a high-precision reference like the one used in the WELCOME lab would ideally confirm that every station are within an acceptable margin.

9.2.2 Scale and multi-station validation

The initial evaluation ran with two stations. One physical station with real hardware and one Docker-based mock. The classroom-scale load test in Section 8.3.5 pushed that to the full targeted number by using one physical plus 19 mock stations. The write proxy, InfluxDB, and the auto-provisioning service all were able to handle the 20 concurrent stations pushing data with no data loss and no provisioning failures.

Two questions however still remain open. The first is how the auto-provisioning service performs under sustained load, which fires on the order of 100 Grafana API calls per 60-second sync cycle at full cohort size. The second is cardinality over longer horizons: 14 spectral channels per station at a 10-minute publish interval add up to roughly $20 \times 6 \times 24 \times 30 = 86\,400$ data points per channel per month across the cohort (stations $\times$ writes/hour $\times$ hours/day $\times$ days/month). The 60 s publish rate in the load test stood in for heavier write pressure than the default 600 s, and nothing went wrong during the test window, but a full month of real data from 20 stations has not been benchmarked at production intervals.

9.2.3 Networking and connectivity assumptions

The station currently relies on the Tailscale VPN to reach the central server through restrictive networks. Tailscale is a commercial service, so if it goes down, stations stay offline until it recovers. The offline replay queue keeps the data safe, but students would lose live dashboard updates during such an outage. For a large deployment, like the one this thesis aims to support, cost is a factor too. The self-hosted alternative, Headscale, settles both concerns, but requires an infrastructure configuration to be able to run it.

9.2.4 Camera pipeline

The vision pipeline is inherited from Colinet (2025) and was not re-evaluated here. A threading fix resolved the blocking regression introduced by the single-threaded capture design, but the PlantCV segmentation and its accuracy under varied indoor lighting was not re-tested.

9.3 Pedagogical fitness

The central pedagogical question (whether a student in the LBIR1251 course can take the numbers from their dashboard and use them to reason about stomatal conductance, phytochrome signalling, or water balance) is not answered by this thesis. PhenoHive has not yet been deployed in a real LBIR1251 class, and no student learning outcomes have been measured.

The sensor suite meets the preconditions that are necessary for the type of analysis conducted in the LBIR1251 course, but whether students can reliably translate the data into correct physiological conclusions is an open empirical question.

What remains a hypothesis (to be tested). Three things stay hypothetical. First is whether students can read this data and draw correct physiological conclusions. Second is whether the better data quality changes how they design experiments. And third is whether the lower hardware friction buys them more time on analysis instead of troubleshooting. Settling them needs a structured evaluation with real student groups, ideally comparing the quality of experimental analysis between groups. This as the single most important next step.

9.4 Future work

Per-station spectral calibration. Calibrating the TCS3448 sensor against the monochromator took about two hours including the time it took to setup. Automating that sweep with the DOPES library (WELCOME Lab, UCLouvain, n.d.), which already provides a Python interface for the CM110 and power meter, and writing a script to compute the per channel offsets from the resulting dataset would greatly reduce the calibration time. Each station would then be able to carry its own characterization instead of sharing a single one, making the spectral comparisons across student groups even more defensible and precise.

Automated calibration drift monitoring. A lightweight software check on the scale baseline could flag when the resting mass drifts more than what normal plant growth or evaporation would explain. This would allow the TA to see a warning before the student even notices the readings are off.

Migration to Headscale. Swapping out the Tailscale platform for a self-hosted Headscale instance on the UCLouvain servers allows for removing the commercial dependency without touching the station firmware. This would be a one-time infrastructure job that would make the platform considerably more self-reliant and secure.

Soil moisture and temperature sensing. Soil moisture and temperature were dropped from this thesis for time and scope reasons, even though these metrics both feed straight into the course’s material. Adding a sensor of that class through the existing factory pattern is very manageable. The real work would rely in calibrating it and presenting the data at the right level for the class.

CO₂ monitoring. A CO₂ sensor would provide a direct view of net photosynthetic activity that no current channel gives. The main factor that keeps it from being added is cost as a calibrated Non-Dispersive Infrared (NDIR) sensor that stays reliable in an uncontrolled environment costs more than any of the I²C sensors already in the suite.

Month-long evaluation. PhenoHive has yet to run through an actual LBIR1251 class. A structured evaluation, measuring student engagement, report quality and conceptual understanding against a comparison group on the traditional protocols, would supply the evidence needed to judge whether the platform really changes and improves how students learn. That study should follow the first real deployment.

9.5 Reflection on the development process

Working on the third generation meant deciding which open problems were worth the remaining effort. The earlier theses had already proved that the hardware was feasible and that the data is useful. The main danger this time was scope creep, adding more and more features onto a codebase that was already hard to maintain would’ve been problematic.

Refactoring first paid off in a concrete way when the monochromator sweep turned up the channel-mapping error in the TCS3448, the fix was pretty easy as it only

required a simple configuration-file edit, because the sensor abstraction was already clean. The 48-hour disconnection test was a similar story. The local-first persistence model behaved exactly as designed, and the buffered records flushed cleanly once the network connectivity returned. Both outcomes confirmed that the choices that were made were the right ones.

One configuration fault ended up proving that point as well. The InfluxDB write URL was mistakenly pointing at the station's loopback address instead of the VM's Tailscale address. This opened up a one-hour data gap. The offline queue caught every record during that time. After fixing the URL over SSH, the system restored the write path at once and the local archive had worked as a real safety net under an actual configuration error because all the data that was gathered during the outage was pushed at the very next cycle.

Altogether, the development process was a good example of how to balance refactoring and feature development, and how to design for failure in a way that makes the system resilient to the inevitable mistakes and oversights that happen during development.

Chapter 10

Conclusion

The approach this thesis took was not a rebuild from scratch, but more a careful look at what the platform was actually doing, and an effort to improve or correct what was not working. The work was organized around three objectives. These were defined as architectural clarity, sensing quality, and deployment resilience. The previous iterations had established what the concept is and the hardware choices are. What was left to do was consolidating them into a platform that could be trusted in uncontrolled environments by students who should not need to think about the measurement system and the quality of the data. Two components are explicitly carried over from the previous theses and weren't reevaluated here. These are the PlantCV vision pipeline and HX711 mass-channel calibration from Colinet (2025) and the DietPi deployment model from Goffinet (2024).

Original contributions

The three objectives decompose into eight specific contributions, each of which was absent from both preceding PhenoHive theses:

1. **Modular software architecture.** Refactoring the codebase from a simple monolithic script into an easily maintainable codebase with four Python source packages (`core`, `sensors`, `ui`, and `vision`) under `src/`, with a `scripts/` directory of operational helpers, and a factory-based sensor abstraction (`BaseSensor`) that allows for new hardware to be added without needing to modify the acquisition loop.
2. **SHT35 metrological calibration.** A four-point corner sweep against the WELCOME lab's Espec SH-242 climatic chamber, which produced traceable

linear corrections (+0.14 °C, −1.53 % RH) which were applied to every reading in the platform, allowing the student to trust the numbers and receive precise measurements that will allow them to reason more carefully about the environmental conditions.

3. **TCS3448 spectral calibration and channel mapping.** A full 300–940 nm monochromator sweep with two diffraction gratings, which exposed a systematic shift of roughly 25 nm across the visible channels and fixed per-channel dark offsets and irradiance scale factors (counts to mW/m²). The empirical mapping it produced replaces the nominal values that were provided by the datasheet everywhere in the platform. This means the student can now read the spectral data in physical units, and reason about the light environment in terms of photosynthetically active radiation instead of raw counts. It means they can also trust the readings provided by the platform and be able to more easily figure out what affects their plant's growth.
4. **Local-first persistence with offline replay.** A CSV-first append model keeps the authoritative archive independent of network availability. So it being paired with a JSONL offline queue that buffers records and replays them to InfluxDB on reconnection is a robust solution. A deliberate 48-hour disconnection test confirmed it. Students can therefore truly trust that their data is being recorded even when the network drops and that it will still be there when they come back.
5. **Headless Wi-Fi onboarding.** Integration of the Comitup captive-portal service which lets a student set up the network on first boot with no command line at all. It allows for power-on to live data on Grafana that was measured in at under five minutes. A student gets the station online and running without needing to ask for help or to know anything about system configuration.
6. **Tailscale overlay network and Nginx write proxy.** An encrypted peer-to-peer VPN in place of the certificate model, together with a restrictive write-only proxy that stops a compromised edge token from ever reaching InfluxDB's administrative endpoints. This secures the network without needing to manage certificates, and allows the station to be deployed on any network that allows outbound VPN connections, without needing to ask for permission or have prior knowledge in networking.
7. **Auto-provisioning service.** Continuous InfluxDB polling that picks up new `station_id` values and builds isolated Grafana teams, dashboards, and student accounts on its own, cutting per-station setup to zero Grafana interaction. Adding

a station to the deployment no longer means working through tedious configuration tasks.

8. **Two-stage MAD signal-processing pipeline.** Burst-level and cross-collection Median Absolute Deviation filtering, with explicit quality metadata (`low_confidence`, coefficient of variation, success ratio) attached to every published record. This means that the student can filter out bad data with confidence, and that they can understand why a particular reading is flagged as low-confidence.

On physical hardware, the evaluation chapter answered the engineering half of the research question. A student can take a freshly-imaged Raspberry Pi, bring it online through the Comitup captive portal, and reach calibrated multi-sensor data on a private Grafana dashboard in under five minutes, never touching a command line. The pedagogical half is the part this thesis cannot answer yet. Whether students will really be able to use the data to reason more carefully about stomatal conductance, phytochrome signalling or water balance. Only a real deployment will reveal the answer, and the platform is now in a state where that's possible. The sensor suite is metrologically validated, the network holds up, provisioning is fully automated, and the station recovers from the disconnections and surprise reboots that are routine in a home. Together, these move PhenoHive from a prototype to a fully fledged system that can be deployed.

References

- Allen, R. G., Pereira, L. S., Raes, D., & Smith, M. (1998). *Crop evapotranspiration: Guidelines for computing crop water requirements* (tech. rep. No. FAO Irrigation and Drainage Paper 56). Food and Agriculture Organization of the United Nations. Retrieved May 27, 2026, from <https://www.fao.org/3/x0490e/x0490e00.htm>
- ams-OSRAM. (2024). *Tcs3448: 14-channel multi-spectral sensor datasheet*. Retrieved May 16, 2026, from <https://look.ams-osram.com/m/1c24b057e65ee61e/original/TCS3448-14-Channel-multi-spectral-sensor.pdf>
- Colinet, X. (2025). *Improving phenohive: Growth analysis and sensor integration* [Master’s thesis]. UCLouvain. <https://thesis.dial.uclouvain.be/entities/masterthesis/facc0e5e-c881-4f58-b877-1f709c625ec9>
- DietPi. (n.d.). *Dietpi: Lightweight justice for your single board computer*. Retrieved May 19, 2026, from <https://dietpi.com/>
- Donenfeld, J. A. (2017). WireGuard: Next generation kernel network tunnel. *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Retrieved May 27, 2026, from <https://www.wireguard.com/papers/wireguard.pdf>
- Espec Corp. (n.d.). *Bench-top type temperature & humidity chamber sh-242*. Retrieved May 16, 2026, from https://espec.com/na/products/model/sh_242
- European Parliament and Council of the European Union. (2016). GDPR Article 6 – Lawfulness of Processing. Retrieved May 31, 2026, from <https://gdpr-info.eu/art-6-gdpr/>
- FarmBot Inc. (n.d.). *Farmbot phenotyping*. Retrieved May 29, 2026, from <https://farm.bot/pages/phenotyping>
- Franklin, K. A., & Whitelam, G. C. (2005). Phytochromes and shade-avoidance responses in plants. *Annals of Botany*, 96(2), 169–175. Retrieved May 24, 2026, from <https://pmc.ncbi.nlm.nih.gov/articles/PMC4246865/>
- Gehan, M. A., Fahlgren, N., Abbasi, A., Berry, J. C., Callen, S. T., Chavez, L., Doust, A. N., Feldman, M. J., Gilbert, K. B., Hodge, J. G., Hoyer, J. S., Lin, A., Liu, S., Lizárraga, C., Lorence, A., Miller, M., Platon, E., Tessman, M., & Sax, T. (2017).

- PlantCV v2: Image analysis software for high-throughput plant phenotyping. *PeerJ*, 5, e4088. <https://doi.org/10.7717/peerj.4088>
- Goffinet, R. (2024). *Phenohive: A portable pedagogical phenotyping station* [Master's thesis]. UCLouvain. <https://thesis.dial.uclouvain.be/entities/masterthesis/24894194-9347-4d5a-9460-fef04090a871>
- Grafana Labs. (n.d.). *Grafana documentation*. Retrieved May 19, 2026, from <https://grafana.com/docs/>
- Grindstaff, B., Mabry, M. E., Blischak, P. D., Quinn, M., & Pires, J. C. (2019). Affordable remote monitoring of plant growth in facilities using Raspberry Pi computers. *Applications in Plant Sciences*, 7(8), e11280. <https://doi.org/10.1002/aps3.11280>
- Headscale. (n.d.). *Headscale: An open-source, self-hosted implementation of the tailscale coordination server*. Retrieved May 18, 2026, from <https://headscales.net/stable/>
- InfluxData. (n.d.). *Influxdb documentation*. Retrieved May 19, 2026, from <https://docs.influxdata.com/>
- LemnaTec GmbH. (n.d.). *Greenhouse solutions for high-throughput phenotyping*. Retrieved April 29, 2026, from <https://www.lemnatec.com/>
- Leys, C., Ley, C., Klein, O., Bernard, P., & Licata, L. (2013). Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4), 764–766. Retrieved May 24, 2026, from https://dipot.ulb.ac.be/dspace/bitstream/2013/139499/1/Leys_MAD_final-libre.pdf
- Massmann, A., Gentine, P., & Lin, C. (2019). When does vapor pressure deficit drive or reduce evapotranspiration? *Journal of Advances in Modeling Earth Systems*, 11(10), 3305–3320. <https://doi.org/10.1029/2019MS001790>
- MikroElektronika. (n.d.). *Color 21 click: Multi-channel color sensor*. Retrieved April 29, 2026, from <https://www.mikroe.com/color-21-click>
- Minervini, M., Giuffrida, M. V., Perata, P., & Tsaftaris, S. A. (2017). Phenotiki: An open software and hardware platform for affordable and easy image-based phenotyping of rosette-shaped plants. *The Plant Journal*, 90(1), 204–216. <https://doi.org/10.1111/tpj.13472>
- Oceanhood. (n.d.). *Cm110 compact monochromator*. Retrieved May 16, 2026, from <https://en.oceanhoodtw.com/>
- Pieruschka, R., & Schurr, U. (2019). Plant phenotyping: Past, present, and future. *Plant Phenomics*. <https://doi.org/10.34133/2019/7507131>
- PlantCV Team. (n.d.). *PlantCV: Plant image analysis software (version 4.6, software archive)* [Software archive; Gehan et al. (2017) is the peer-reviewed primary publication]. Retrieved May 19, 2026, from <https://plantcv.org/>

- Raspberry Pi Foundation. (n.d.). *Raspberry pi zero 2 w*. Retrieved May 19, 2026, from <https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/>
- Seeed Studio. (n.d.). *Grove - i2c high accuracy temp&humi sensor (sht35)*. Retrieved April 29, 2026, from <https://www.seeedstudio.com/Grove-I2C-High-Accuracy-Temp-Humi-Sensor-SHT35.html>
- Sensirion. (n.d.). *Datasheet sht3x-dis: Humidity and temperature sensor*. Retrieved May 16, 2026, from https://sensirion.com/media/documents/213E6A3B/63A5A569/Datasheet_SHT3x_DIS.pdf
- SparkFun Electronics. (n.d.). *Sparkfun load cell amplifier - hx711*. Retrieved May 19, 2026, from <https://www.sparkfun.com/sparkfun-load-cell-amplifier-hx711.html>
- Steele, D. (n.d.). *Comitup: Zero-configuration wi-fi bootstrap service*. Retrieved May 19, 2026, from <https://davesteele.github.io/comitup/>
- Tailscale. (n.d.). *Tailscale: Fast, reliable, and secure virtual private overlay networks*. Retrieved May 18, 2026, from <https://tailscale.com/>
- UCLouvain. (n.d.-a). *VPN access – UCLouvain*. Retrieved May 31, 2026, from <https://uclouvain.atlassian.net/wiki/spaces/CDSMEXTP/pages/565444682/Access+VPN+-+UCLouvain>
- UCLouvain. (n.d.-b). *Welcome (wallonia electronics and communications measurements) lab*. Retrieved May 16, 2026, from <https://app.uclouvain.be/PTech/Home/WELCOME>
- Vienna Scientific Instruments. (n.d.). *Shoot phenobox: Automated plant phenotyping*. Retrieved April 29, 2026, from <https://www.vienna-scientific.com/products/shoot-phenobox/>
- Wakili, A., & Bakkali, S. (2026). Ruraledgehealth: An offline android mobile edge AI reference architecture for IoT sensor classification in resource-constrained environments. *Array*. <https://doi.org/10.1016/j.array.2026.100912>
- WELCOME Lab, UCLouvain. (n.d.). *Dopes: Device and optics programming and experiment software*. Retrieved May 24, 2026, from <https://forge.uclouvain.be/welcome/dopes>

Appendix A

Technical architecture

A.1 Electronic wiring diagram

Figure A.1 provides the complete electronic wiring schematic for assembly and troubleshooting reference.

The station is centered around a Raspberry Pi Zero 2 W (Raspberry Pi Foundation, n.d.). Three bus distribution headers consolidate the 3.3 V, 5 V, and Ground rails to reduce wire congestion. The environmental (SHT35) and multi-spectral (TCS3448) sensors both share a single I²C bus communicating over General Purpose Input/Output (GPIO) 2 (SDA) and GPIO 3 (SCL). The HX711 24-bit analog-to-digital converter is connected via a digital interface over GPIO 5 (DOUT) and GPIO 6 (SCK). The KY-019 relay module controlling the 5 V COB LED strip is connected to GPIO 23 (BCM). The camera module is connected via a CSI interface directly to the Raspberry Pi.

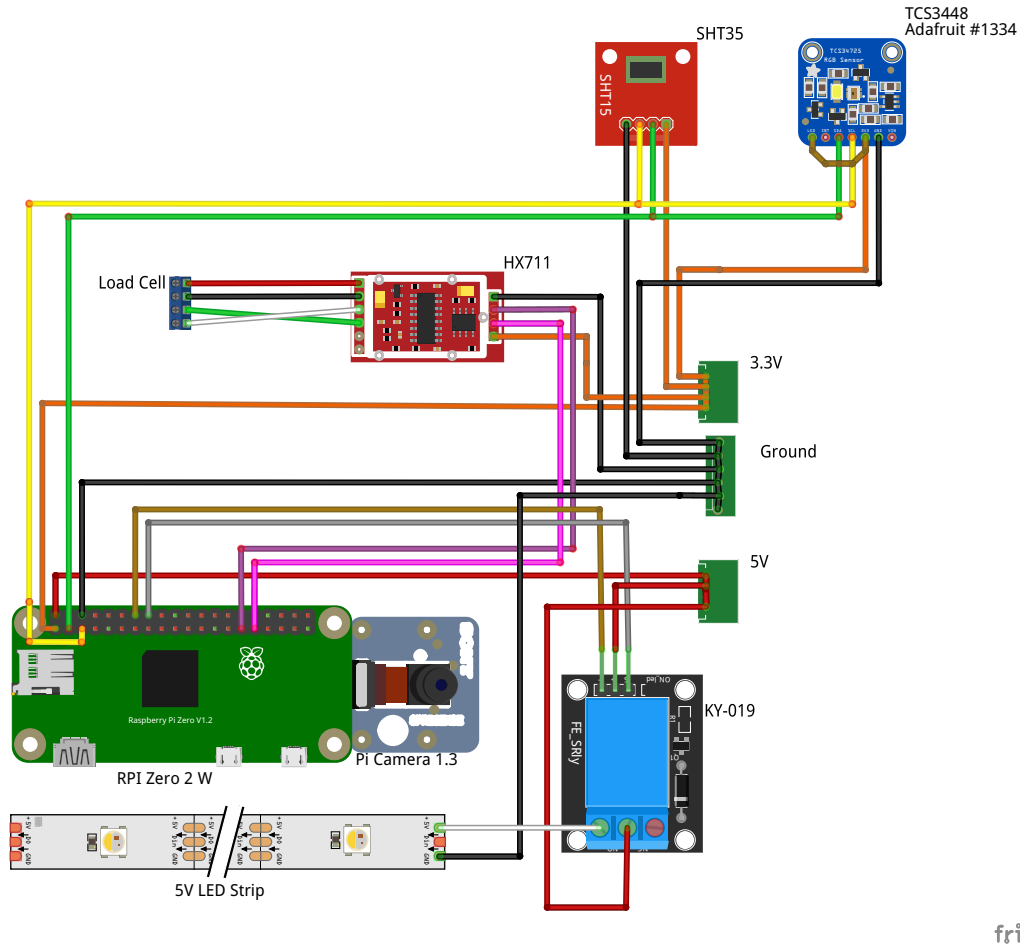
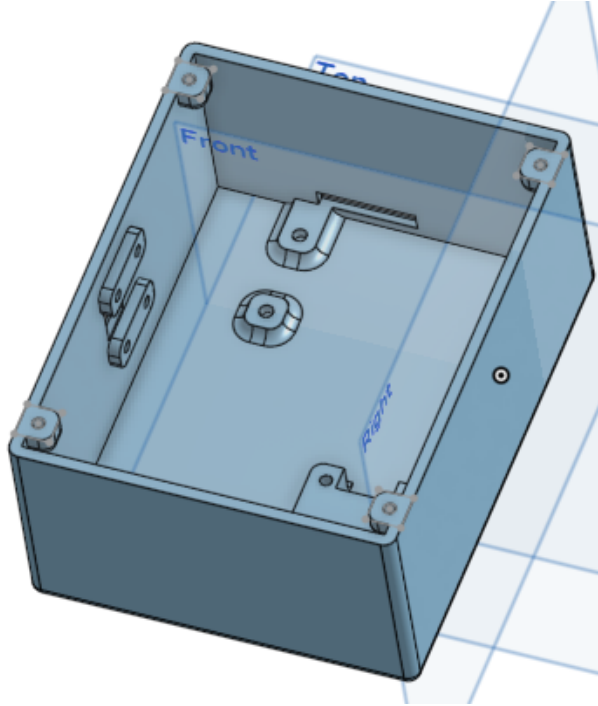


Figure A.1: Complete electronic schematic of the PhenoHive edge sensor station. The Raspberry Pi Zero 2 W is the central compute unit, connected to the SHT35 and TCS3448 over a shared I²C bus, the load cell via an HX711 ADC, the LED relay on GPIO 23 (BCM), and the camera via CSI.

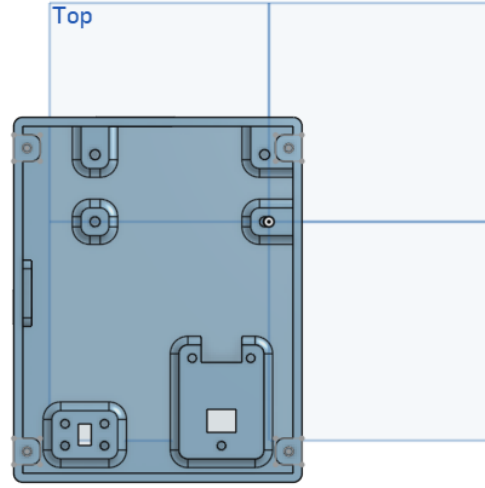
A.2 3D Printed Casing

The PhenoHive station is housed in a custom designed casing made in CAD and fabricated with a standard 3D printer (Figure A.2). The design is available in the project repository under `casing/`.

The casing is a single continuously printed box with rounded outer corners and four insert holes at the corners for a lid attachment. The interior floor and sides have dedicated mounting pads for each electronic component.



(a) Isometric CAD render showing the open-top enclosure, the camera bracket (left wall), and the circular TCS3448 sensor mount on the floor.



(b) Top-down view that shows the corner screw bosses, the circular sensor seat, the cable routing slot and the breakout board pad.

Figure A.2: CAD renders of the PhenoHive 3D-printed casing. The design enforces a fixed sensor geometry across all stations. All of the features are dimensioned to match the breakout boards listed in the bill of materials (Table B.1).

Appendix B

Bill of materials and cost analysis

A primary objective of the PhenoHive project is maintaining a strict hardware budget to allow multi-station laboratory deployments (20+ units). The components were selected to balance precision with cost-efficiency. Table B.1 details the pricing and sourcing of all components required to build one edge station. By using standard I²C sensors and a Raspberry Pi Zero 2 W, the total hardware cost stays well within the 100–150€ target.

Table B.1: Bill of Materials (BOM) for a single PhenoHive edge sensor station.

Component Description	Supplier	Approx. Cost (€)
Raspberry Pi Zero 2 W with pre-soldered headers	Standard Retailer	18.00
Raspberry Pi Camera Module V1.3 (5MP)	Standard Retailer	15.00
MicroSD Card (32GB, Class 10)	Standard Retailer	8.00
Micro USB Power Supply (5V, 3A)	Standard Retailer	10.00
SHT35 Temperature & Humidity Sensor Breakout	Adafruit / Seeed	15.00
TCS3448 Spectral Sensor (MikroE Color 21 Click)	MikroElektronika	30.00
TAL226 Load Cell (or equivalent generic strain gauge)	Generic Retailer	10.00
HX711 24-bit ADC Breakout Board	SparkFun / Generic	2.00
AZDelivery KY-019 Single-Channel Relay Module (5 V)	AZDelivery / Generic	3.00
Luktix 5 V COB USB LED Strip (1.5 m, 4000 K)	Generic Retailer	8.00
Wago 221-415 Splicing Connectors (for power distribution)	Hardware Store	3.00
Prototyping wires and 3D printed chassis	Local fabrication	15.00
Total Estimated Cost		$\approx$ 137.00

Appendix C

Project repository

The complete source code for this thesis is publicly available on GitHub:

https://github.com/amontariol/Phenohive_2025-26

The repository contains the full station software, deployment scripts, calibration tools, Grafana provisioning scripts, and the L^AT_EX source for this report.

Appendix D

Station Assembly and Deployment Guide

This appendix is a step-by-step reference for a TA or developer who needs to build and commission a new PhenoHive station from scratch. It covers hardware assembly, image build, and first-boot verification. The UCLouvain server-side infrastructure (InfluxDB, Grafana, auto-provisioning) is assumed to already be running; a brief setup reference is included at the end.

D.1 Prerequisites

Before starting, the following items should be ready:

- All components listed in the Bill of Materials (Appendix B).
- A microSD card reader.
- A Linux machine to build the golden image (the build script requires `losetup`, `chroot` and `qemu-user-static`).
- SSH access to the UCLouvain VM and an admin InfluxDB token.
- A Tailscale account with admin right and a reusable pre-auth key generated from the Tailscale admin console.
- A 3D-printed enclosure (STL files in `docs/casing/` of the repository).
- The PhenoHive repository cloned locally.

D.2 Hardware assembly

Wire the components following the schematic in Figure A.1 (Appendix A.1). The key connections are summarised below.

- **SHT35 and TCS3448** both share the I²C bus on GPIO 2 (SDA) and GPIO 3 (SCL). They both run at 3.3 V.
- **HX711 ADC** DOUT on GPIO 5, PD_SCK on GPIO 6 and the breakout is powered at 3.3 V.
- **KY-019 relay** (LED strip control) DATA goes on GPIO 23 (BCM numbering) and is powered at 5 V.
- **Camera module**: connected via the CSI ribbon cable to the Raspberry Pi Zero 2 W.
- **Power input**: a 5 V / 3 A Micro-USB power supply to the Raspberry Pi micro-USB power port.

Use Wago connectors to distribute the 3.3 V, 5 V and GND rails cleanly without needing to solder anything. All breakout boards can be set on their mounting pads and then close the enclosure lid.

D.3 Prepare configuration and build the image

All station-specific values are integrated into the image at build time. This means that no SD card editing or SSH access is required after flashing.

Create the configuration files In the `dietpi/` directory of the repository, create three plain-text files (one value per file, no trailing whitespace):

File	Content
<code>dietpi/phenohive_server.txt</code>	Tailscale IP of the UCLouvain VM (e.g. 100.117.27.12)
<code>dietpi/phenohive_token.txt</code>	InfluxDB write token (<i>gitignored — do not commit</i>)
<code>dietpi/phenohive_station_id.txt</code>	Unique station identifier (e.g. 01, 02)

The build script reads these files and writes the values directly into `/opt/phenohive/.env` and `config.ini` inside the image.

Requirements The build script (`dietpi/build_image.sh`) requires a Linux host with:

- `qemu-user-static` and `binfmt` support (for cross-architecture chroot).
- Standard utilities: `losetup`, `parted`, `e2fsck`, `resize2fs`, `rsync`.
- Internet access from within the chroot (for `apt-get` and the Tailscale repository).

Obtain the base image Download the latest DietPi ARMv8 image for Raspberry Pi Zero 2 W from <https://dietpi.com/> and place it in the `dietpi/` directory. The script auto-detects any `DietPi*.img.xz` file there.

Run the build From the repository root:

```
sudo ./dietpi/build_image.sh --tailscale-key tskey-auth-XXXXX
```

Pass a reusable pre-auth key from the Tailscale admin console. The script performs the following steps inside a chroot of the base image:

1. Installs the system packages: `NetworkManager`, `comitup` (captive-portal Wi-Fi onboarding), `Tailscale`, `python3-picamera2`, I²C tools and `Avahi`.
2. Enables `NetworkManager`, `comitup`, `avahi-daemon`, `tailscaled` and `phenohive` all as `systemd` services.
3. Bakes the `PhenoHive` source tree into `/opt/phenohive/`, creates a Python virtual environment and installs all dependencies.
4. Writes `.env` with the server IP and token read from the local configuration files, and `config.ini` with the station ID.
5. Embeds the Tailscale pre-auth key for automatic first-boot authentication.
6. Enables I²C and the camera module via `config.txt` overlays and configures a fully unattended DietPi first boot.

The output is called `dietpi/phenohive.img`, is approximately 5 GB larger than the base image and build time is 20–40 minutes depending on network speed and host CPU.

D.4 Flash the image

Flash `dietpi/phenohive.img` to the microSD card. On Windows use Raspberry Pi Imager or Balena Etcher. On Linux:

```
dd if=dietpi/phenohive.img of=/dev/sdX bs=4M status=progress
```

Insert the card into the Raspberry Pi and power it on. No further preparation of the SD card is needed.

D.5 Wi-Fi setup via comitup

Because no Wi-Fi credentials are stored in the image, the station cannot reach any network on first boot. The `comitup` service handles this automatically.

1. Power on the station. After approximately 30 seconds, a Wi-Fi hotspot named `comitup-XXXX` appears (where `XXXX` is derived from the device MAC address).
2. On a phone or laptop, connect to that hotspot. A captive portal opens automatically. If it does not, open a browser and navigate to `http://comitup.local`.
3. Select the target Wi-Fi network from the list, enter the password and confirm. The hotspot disappears and the station joins the network.
4. The station reboots automatically. On the second boot it authenticates to Tailscale using the baked-in key and the PhenoHive service starts.

Subsequent network changes: if the station cannot find its saved network, `comitup` automatically re-broadcasts the hotspot. Repeat steps 2–3 to re-configure.

D.6 Verify the service

Once the station is on the network and Tailscale is authenticated, confirm the service is running. There are two ways to do this. You can either check the logs directly on the Pi via SSH, or monitor the incoming data on Grafana or the debug UI (next step).

To SSH into the Pi (via its Tailscale IP, available in the Tailscale admin console):

```
ssh root@<PI_TAILSCALE_IP>
Ctrl+C to stop the live log stream.
systemctl status phenohive
journalctl -u phenohive -n 60 --no-pager
```

The log should show:

- Sensor initialisation messages (`SHT35 OK`, `TCS3448 OK`, etc.).
- An NTP clock synchronisation confirmation.
- A `Collection cycle started` line after the first collection interval.
- A `Publishing aggregate record` line after the first publish interval.

If a sensor fails to initialise, the service continues and retries on every subsequent poll cycle. Check the wiring against Figure A.1 if a sensor does not recover.

If the service cannot reach the server, check:

- `tailscale status` where both the VM and the station should appear as connected.
- `/opt/phenohive/.env` verify `INFLUX_URL` and `INFLUX_TOKEN` were applied correctly.
- That the provisioning files were placed on `/boot` before the first boot (they are only processed once, if this wasn't done, then reflash the compiled image to the sd card, there is no need to rebuild it entirely).

D.7 Scale and Camera calibration

Open the debug dashboard from any device on the same Tailscale network:

`http://<PI_TAILSCALE_IP>:8080`

Navigate to the *Calibration* tab to perform both the calibrations:

For the scale:

1. Place the station on its final surface. Do not put anything on the scale and click on *Confirm: Scale is empty*.
2. Place an object with a known mass on the scale, then enter that mass in the *Reference weight (g)* field and click *Calibrate*. The reading should now match the reference.
3. You can choose to add an offset correction to the calibration to take the pot's weight into account.
4. Remove the calibration weight and place the plant (pot and soil) on the scale. Confirm the reading is plausible.

For the camera:

1. Ensure that the station is in its final position and the camera has a clear view of where the plant will be but without the plant in the frame and click on *Capture background*.
2. Use an item in the captured image to compute the pixel-to-cm scale factor then enter the real-world size in cm and its value in pixels on the image and click *Calibrate*. The computed scale factor is applied to all subsequent recordings and is written to `config.ini`.

This captures the current view as Calibration values (`tare` and `calibration_factor`) are written to `config.ini` immediately and persist across reboots and software updates.

D.8 Verify data on Grafana

Within a few minutes of the first successful data push, the auto-provisioning service on the VM detects the new `station_id` tag in InfluxDB and automatically creates:

- A Grafana team (`phenohive_station_XX`).
- A student user account (`studentXX`) with an initial password derived from the station ID.
- A dashboard cloned from the master template and filtered to this station.

Log in to Grafana at `http://100.117.27.12:3000` as `usr: admin & pwd: admin12345` and confirm the new dashboard appears. Share the student credentials (`studentXX` / `phenohive_XX`) with the corresponding group.

D.9 Optional: setting up the server-side infrastructure

If the backend stack is not yet running (e.g. deploying a new course edition), set it up as follows.

1. SSH into the VM and clone the repository.
2. Copy `.env.example` to `.env` and set `INFLUXDB_INIT_ADMIN_TOKEN`, `GRAFANA_ADMIN_PASSWORD` and `INFLUXDB_INIT_PASSWORD`.
3. Start the production Docker stack:

```
docker compose up -d
```

4. Create a scoped InfluxDB write token (append-only to the **phenohive** bucket) via the InfluxDB web UI on port 8086 (accessible locally on the VM). This is the token that goes into **phenohive_token.txt** on each station's SD card.
5. Start the auto-provisioning service:

```
python scripts/auto_provision_grafana.py
```

Register it as a systemd unit for persistence across VM reboots.

6. Install Tailscale on the VM and join the PhenoHive tailnet:

```
tailscale up --auth-key=<TSKEY-AUTH-...>
```

Once the VM stack is running, return to Step 2 to build and flash stations.

Appendix E

Use of artificial intelligence tools

In accordance with UCLouvain’s policy on the use of generative artificial intelligence and academic transparency standards, the following AI tools were used during the development of this Master’s thesis for the following purposes:

- **Code assistance and debugging - Claude Code:** Was used as a pair-programming assistant throughout development to accelerate the writing of acquisition scripts and boilerplate code, to help debug I²C driver issues and to help propose fixes to implementation problems. All of the suggested code was reviewed and tested on the physical hardware and was manually validated before integration into the codebase.
- **Mathematical assistance - Claude:** Was used to verify the calibration offset calculations and check various mathematical formulas. All results were independently and manually checked.
- **LaTeX formatting - Claude Code:** Was used to generate TikZ diagram scaffolding and tabular structure templates as well as general formatting assistance. All generated markup was reviewed, fact checked and adapted to match the thesis content.
- **Writing and proofreading - Claude:** Was used to correct grammar and spelling errors and to improve general clarity. All of the text in this manuscript reflects my own analysis, methodology and conclusions. There are no AI-generated passages that appear word-for-word in the final manuscript that weren’t reviewed and revised.

All AI-assisted contributions were reviewed, tested where applicable and validated to ensure technical accuracy and scientific integrity.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | www.uclouvain.be/epl