SmartPlantMonitor: A Self-Watering Plant Monitor with a Serverless AWS Backend and AI Analytics Platform

Kyaw Phone Lwin
Independent project, Singapore · https://smartplantmonitor.com

***Abstract*—SmartPlantMonitor is a general-purpose plant monitor that looks after a potted plant with no one watching. It is currently in service on an okra plant in a Singapore HDB corridor. A small computer board (an ESP32) measures the soil, air, light and water tank every minute, waters the plant when the soil is dry, and points a camera at it. The board makes every watering decision itself and keeps working when the internet is down. A cloud backend on Amazon Web Services (AWS) stores the measurements, runs a live website, and can ask the board to stop watering; the board must confirm every such request. Plain statistics code turns the measurements into trends, forecasts and alerts. Google's Gemini AI model then writes short daily notes and a weekly report about them, and every number it writes is checked against the measured data before anyone sees it. A study of 48,582 records from 27 September to 5 November 2026 found 53 crashes that fixed themselves and 26 stops that needed someone to unplug the board. Each was traced to its cause and fixed. The system has been in service since 27 September 2026; it holds 53,436 records and costs about S$8–12 a month to run.**

***Index Terms*—ESP32, Internet of Things, AWS, serverless, reliability, large language models, Gemini.**

Project at a glance	
Built by	One person, end to end: wiring, firmware, cloud, website and AI
Plants	Any potted plant; currently an okra plant
In service	Since 27 Sep 2026; 53,436 records so far
Hardware	2 ESP32 boards, 4 sensors, a pump, 2 servo motors, a camera
Firmware	C++ on both boards, each recovering from faults by itself; the main board waters on its own behind 9 safety checks
Cloud	11 Amazon Web Services (AWS) services; 3 AWS Lambda functions in Python 3.13; 25 web routes
Analytics	5 statistical models in plain Python, no ML libraries
AI	Google Gemini for 5 jobs; every number it writes is checked
Reliability	79 field failures studied, 13 root causes found and fixed
Cost	About S$356 to date; AWS about S$8–12 a month

I. Introduction

Most home plant monitors either water on a timer or let a cloud service decide. A timer ignores how wet the soil actually is. A cloud-controlled waterer stops working whenever the internet, the cloud service or its settings fail, and the plant is the one that pays. Adding an AI model that writes about the plant brings a third risk: confident text quoting numbers that were never measured.

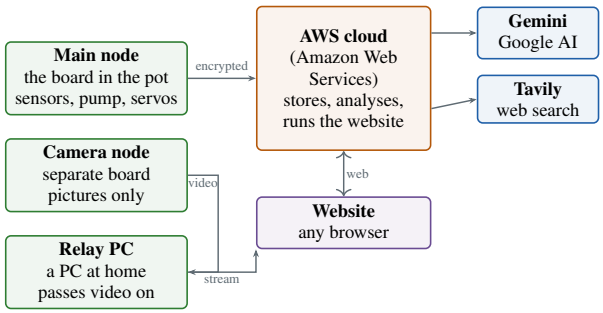


Fig. 1. The parts of the system. The main board measures and waters on its own; the AWS cloud stores, analyses and displays; the AI model (Google Gemini) and web search (Tavily) are outside AWS and are called only from the cloud; video reaches the website through a PC at home.

This project was built around three rules, which the rest of the report follows:

1. **The board decides.** The device in the pot makes every watering decision and keeps its own memory, so the plant is safe with or without a connection.
2. **The cloud watches and asks.** It stores, analyses and displays data, and it may *request* a change, but it can never override a safety check on the board.
3. **Code computes; AI explains.** Every number a person sees comes from a measurement or from ordinary code. The AI model may describe those numbers but may not invent any, and a checker enforces this.

The report follows the data: hardware first (Section III), then the cloud (Section IV), then the analytics and AI (Section V), then the problems met in the field and how they were solved (Section VI), the design decisions (Section VII), and the cost (Section VIII). Table I explains the technical terms used; each is also explained in the text where it first matters.

II. How the System Works, in One Minute

Fig. 1 shows the parts, and the following steps show what happens in a typical minute:

1. The **main board** reads its sensors: soil moisture, air temperature and humidity, light, and the water level in the tank.
2. It checks nine safety conditions (Table III). If the soil has been dry for three readings in a row and every check passes, it runs the pump for 5 s. No connection is needed for this.
3. It sends the reading to AWS over Wi-Fi as an encrypted MQTT message.
4. AWS stores the reading in the database, and the website (Fig. 2) shows it to anyone viewing.

TABLE I
TERMS USED IN THIS REPORT

Term	Meaning
ESP32	A low-cost microcontroller (a small computer on one chip) with built-in Wi-Fi
Firmware	The program that runs on the board
GPIO	A numbered pin on the board that can read a sensor or switch something
Strapping pin	A pin the chip reads at power-up to choose how to start; wiring it carelessly can stop the board from starting
Flash (NVS)	Memory on the board that keeps its contents through resets and power cuts
Watchdog	A timer that restarts the board if the program stops responding
AWS	Amazon Web Services, Amazon's cloud platform; all cloud parts of this project run on it
MQTT	A lightweight messaging protocol: devices publish short messages to named topics
AWS IoT Core	Amazon's service that receives device messages and routes them
Lambda	Code in the cloud that runs only when triggered, billed per run ("serverless")
DynamoDB	Amazon's database, used here as one table of all readings and settings
Telemetry	The stream of measurements and status reports the board sends
Deterministic	Ordinary code: the same input always gives the same output
LLM	Large language model, an AI that writes text (here Google Gemini)
Grounding	Checking that every fact in AI-written text comes from supplied evidence

5. Every 10 minutes, statistics code in the cloud updates trends, a soil forecast and any alerts.
6. Three times a day, Gemini writes a short note about the plant from those numbers, and a checker confirms every number in it before the note is saved.
7. If the owner switches automatic watering off on the website, the cloud sends the board a numbered request. The board applies it and reports back, and only then does the website show the change as done.

Nothing in the hardware, firmware or cloud is tied to one kind of plant. What is plant-specific (the ideal ranges for soil moisture, temperature and light, and the plant's name in a few AI instructions) is kept in one plant profile, currently set for okra, so moving to a different plant means changing that profile and the watering threshold, not the system.

A second board, the **camera**, streams video through a small relay program on a PC at home and takes a photograph of the plant at 06:00 and 18:00 for a growth journal.



Fig. 2. The live website. Top: soil gauge and the pump card, showing the board's own watering state (on, waters below 30 %). Middle: both boards online with signal strength and uptime, then the live sensor cards. Bottom: the last 24 hours. The "time until dry" figure is a known display fault (a dry-down rate of 60 %/h taken from spikes while the probe was being handled); it does not affect watering, which only the board decides.

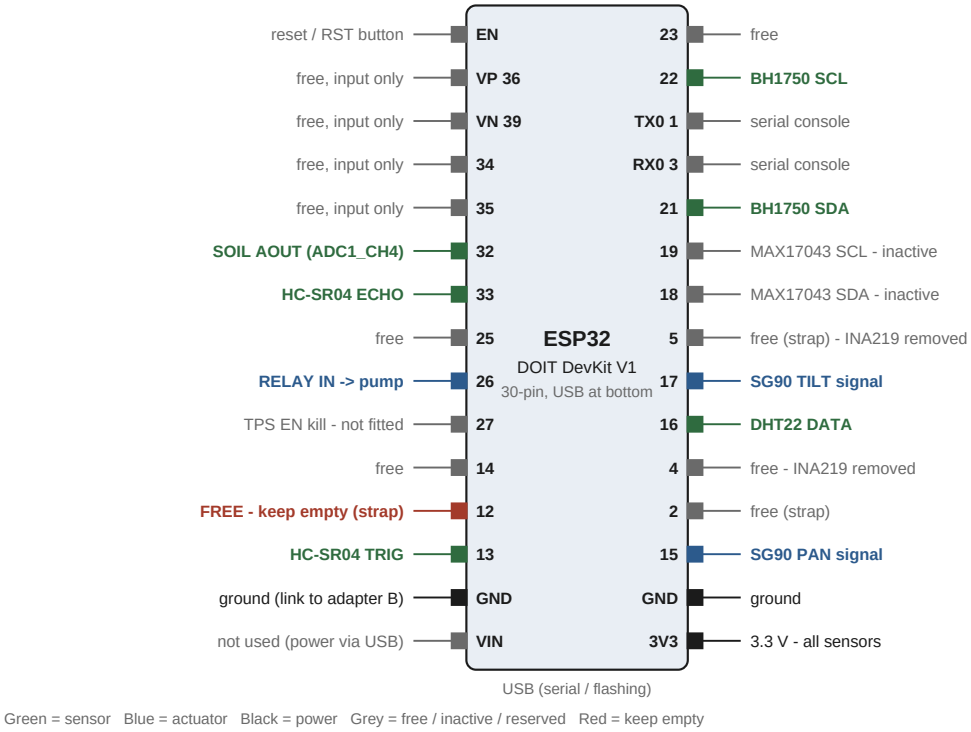


Fig. 3. Which pin of the ESP32 DOIT DevKit V1 (30-pin board, USB connector at the bottom) is wired to what. Coloured labels are in use; grey ones are free. GPIO12 is deliberately empty because it is a strapping pin; the tank sensor's echo wire was moved from GPIO12 to GPIO33 after the start-up failures in Section VI.

III. Hardware

A. The main board and its sensors

The main board is an ESP32 DOIT DevKit V1 [1], seated on an expansion board that provides 3.3 V power pins for the sensors. Table II lists what is connected and why, and Fig. 3 shows exactly which pin each part uses. Two details in that drawing came from failures described in Section VI. Pin GPIO12 is left empty on purpose: it is a strapping pin, and a sensor holding it at the wrong level during a restart stopped the board from starting. The soil probe sits on a pin that can switch on a weak internal pull in either direction, so the board can tell a disconnected probe from dry soil by comparing the two readings.

B. Power and grounding

The build was first run from a power bank and then from a battery pack (four 18650 lithium cells with a voltage converter and a charge gauge). Both caused trouble: the pump, the servo motors and the camera draw current in sudden bursts, and every voltage dip, servo twitch and camera dropout traced back to a small converter feeding those bursts. The system now runs on mains through three ordinary 5 V USB phone chargers: one for the main board, one for the servos and pump, and one for the camera (Fig. 4). The battery kit is kept for demonstrations and power-cut tests.

One subtle point matters. The servos and pump get their power from the second charger, but the main board tells them what to do, and a control signal only means something relative to a shared ground. One wire therefore joins the second charger's ground to the main board's ground. On 7 November the camera's tilt servo jittered, worse when the chargers were plugged into different wall sockets. Each charger's output floats at a small voltage set by its socket, so with all three on one power strip they float together and the single ground wire no longer carries the difference. All three now share one power strip.

TABLE II
What is connected to the main board

Measures / does	Part	Pin	How the firmware makes it reliable
Air temperature, humidity	DHT22	16	Own reading routine that cannot freeze the board
Light	BH1750	21, 22	Middle of three readings; unjams its shared wire
Soil moisture	Capacitive probe v1.2	32	Settles, then takes the middle of 20 samples; detects a missing probe
Water in tank	HC-SR04 echo sensor	13, 33	Three echoes must agree; no pumping if the tank is empty
Watering	5 V pump via a 3.3 V relay switch	26	5 s per run; at least 60 s between runs
Camera aiming	Two SG90 servo motors	15, 17	Limited angles; power released after each move
Pictures	ESP32-S3 camera board	—	Separate board with its own recovery (Section III-D)

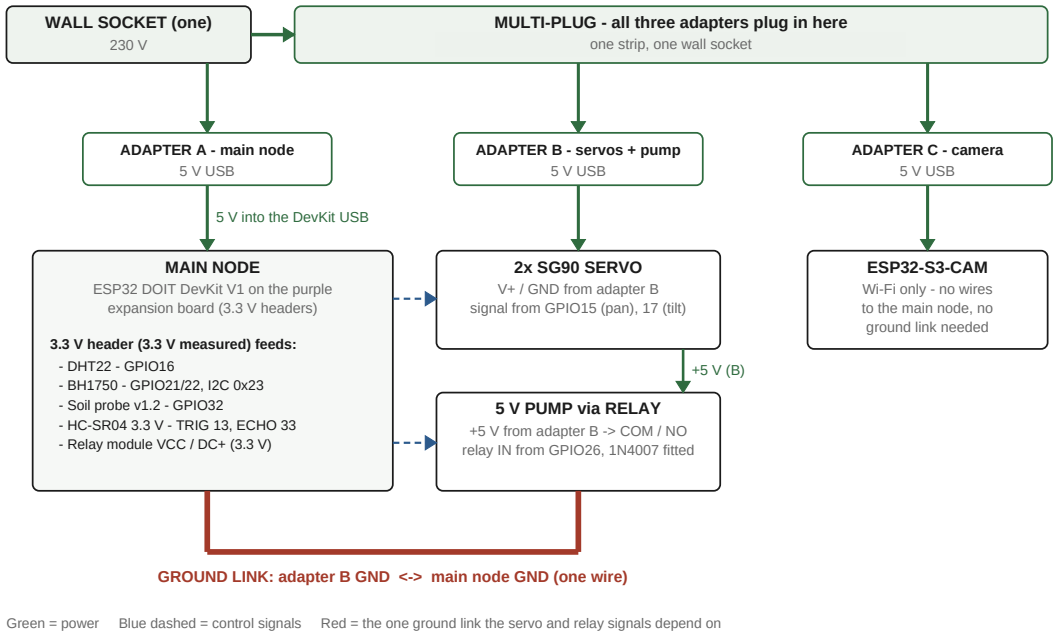


Fig. 4. Power and ground. Green lines carry power, blue dashed lines carry control signals, and the red line is the single ground link that the servo and pump-relay signals depend on.

C. The firmware

The firmware is written in C++ (Arduino framework on Espressif's ESP-IDF 5.3.2, built with PlatformIO). On mains it runs the chip at full speed with Wi-Fi always on. Every 60 s it reads the sensors, runs the safety checks of Table III, and sends a reading. Because nobody watches the board, it is built to recover from faults by itself and to explain afterwards what went wrong:

- **Watchdog:** if the main loop stops for 120 s, the board restarts instead of hanging.
- **Crash report:** after a crash, the next start-up sends the crash type and the exact program location. A standard tool (`addr2line`) then turns that location into a line of source code.
- **Breadcrumbs:** the board notes which step it was in (reading the temperature sensor, sending, connecting) in memory that survives a restart, so a crash report says where it happened.
- **Graded recovery:** if the connection drops, the board first restarts its radio, then restarts itself at widening intervals (6, 15 and 30 minutes, then hourly), and keeps measuring and watering in between.
- **Connection log:** every Wi-Fi drop is recorded with the reason the router gave.

New firmware is installed only over a USB cable, by the owner's choice, so each version is first tested in a simulation on a PC.

D. The camera

The camera board streams pictures over a WebSocket (a connection that stays open for live data) to a small relay program on a PC at home. The relay exists for two reasons: the camera cannot encrypt a cloud connection and send video quickly at the same time, and the relay is the only part that knows whether anyone is watching, so it wakes the camera for viewers and lets it rest otherwise. A Cloudflare tunnel gives the relay a fixed public web address (Section IV-G). At 06:00 and 18:00 the cloud asks the relay for a photograph, which is saved for the growth journal.

Like the main board, the camera recovers by itself. It:

- restarts if no picture is sent for 40 s, or its link to the relay is dead for 40 s while streaming (10 min when idle);
- checks the link every 5 s and reopens a dead one within about 15 s;
- retries Wi-Fi every 8 s and restarts after 10 failures;
- drops to a smaller picture size after 8 failed captures in a row, instead of freezing;
- clears a stuck "streaming" state after 10 s.

TABLE III
THE NINE CHECKS BEFORE EVERY AUTOMATIC WATERING

#	All must be true; none needs the internet
1	Automatic watering is switched on (remembered in flash)
2	Not paused (daily limit reached, last runs had no effect, or the cloud asked to pause)
3	The soil reading is valid and the probe is not faulty or missing
4	Soil below 30 % for 3 readings in a row
5	The tank is not empty
6	At least 30 minutes since the last watering, and 60 s since any pump use
7	Fewer than 6 automatic waterings today
8	The last automatic watering raised the soil by at least 5 points (two misses pause)
9	If the board has lost its clock (power cut with no internet), at most 2 runs

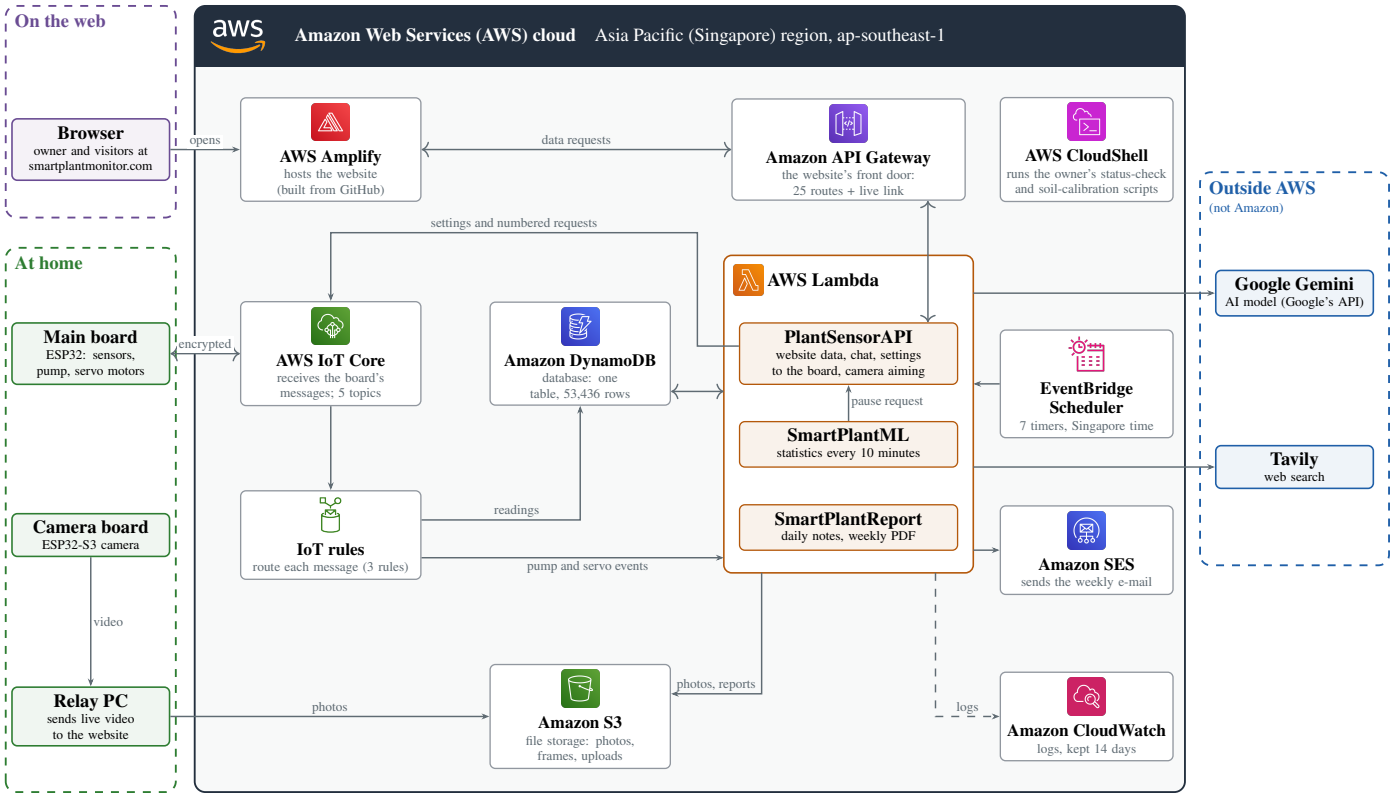


Fig. 5. The cloud side. Everything inside the dark frame is an Amazon Web Services (AWS) service, shown with its AWS console name and icon. Google's Gemini AI model and the Tavily search service are separate providers outside AWS. Arrows show what starts, reads or writes to what. Only the main board can start the pump: the cloud reaches it through AWS IoT Core as settings messages carrying numbered requests, which the board confirms.

IV. The Cloud: Amazon Web Services (AWS)

All of the project's cloud services run on Amazon Web Services (AWS), Amazon's cloud platform, in its Singapore region; none run on Google Cloud, Azure or another provider. Outside AWS, Google's Gemini AI model and the Tavily web-search API are called only from AWS; the domain is registered at Namecheap, and Cloudflare runs the domain's DNS and the camera relay's tunnel. Fig. 5 shows the AWS services and how they connect, Figs. 6 and 7 show how the website and the live video reach a viewer, and Table IV explains each AWS service.

A. How a reading travels

The board sends its messages to AWS IoT Core, Amazon's device-messaging service, on named topics: `plant/sensors` for the minute readings, `plant/pump-event` for pump runs and watering-state changes, and `plant/config`, which carries settings and requests in the other direction. A topic marked "retained" keeps its last message, so the board receives the current settings the moment it reconnects. Small routing rules inside IoT Core ("IoT rules") then pass each message on: readings go straight into the database, and pump events go to the main cloud program. One lesson from this design: the database rule lists every field by name, so a new field added to the firmware is silently dropped until the rule is updated. That cost one round of missing crash reports.

All data lives in a single Amazon DynamoDB table. Each row is identified by a device name and a time, and a few special rows at time zero hold the current settings, the pump status, the latest forecast and the details of the growing season. On 8 November the table held 53,436 rows (43 MB).

B. How the boards connect to AWS

Each board has its own identity in AWS IoT Core, called a *thing*: `SmartPlantMonitor` for the main board and `SmartPlantCam` for the camera. Each thing has its own device certificate, an electronic ID card created in IoT Core, and a matching private key. Both are stored in the board's firmware, in a secrets file that is never shared, and the key never leaves the board. Both certificates are active and valid until 2049.

When a board connects, it opens an encrypted TLS connection to the project's IoT Core address on port 8883, and the two sides check each other before any message is sent. The board checks AWS's certificate against the Amazon root certificate it carries; AWS checks the board's certificate and that the board holds the matching key. The main board sets its clock from the internet first, because these checks fail with a wrong date. Only then does the board log in to MQTT under its thing name and subscribe to its topics (Table V). The camera uses this link only for small control messages; its video goes to the relay instead (Section III-D).

What a board may do is set by a policy attached to its certificate. Both certificates share one policy,

TABLE IV
EVERY AWS SERVICE USED (AWS CONSOLE NAMES): WHAT EACH DOES, WHAT STARTS IT AND WHAT IT AFFECTS

AWS service (what it is)	In this project	Started by / reads	Writes / calls
AWS IoT Core (message hub for devices)	Each board's identity and certificate; the main board's 5 topics	Both boards, over encrypted MQTT	The rules below; settings back to the boards
IoT rules, part of IoT Core (message routing)	Readings → database; pump events → Lambda; servo status → Lambda (3 of 5 rules in use)	Each message on its topic	DynamoDB, PlantSensorAPI
AWS Lambda (code run on demand)	PlantSensorAPI (2048 MB, up to 8 min)	Website requests, IoT rules, photo schedule	DynamoDB, S3, settings to the board via IoT Core; Gemini and Tavily (outside AWS)
AWS Lambda	SmartPlantML (512 MB)	Schedule: every 10 minutes	Forecast, alerts and daily summary; pause requests
AWS Lambda	SmartPlantReport (1024 MB)	Schedule: 08:00, 16:00, 23:45 and weekly; chat requests	Notes and weekly PDF; SES; Gemini and Tavily (outside AWS)
Amazon API Gateway (web front door)	25 web routes plus a live WebSocket connection	The website and the chat	PlantSensorAPI
Amazon Cognito (sign-in)	The owner's login for the website	The owner, on the website	Checked by PlantSensorAPI before owner changes
Amazon DynamoDB (database)	One table of all readings and settings; pay per use	All three Lambda functions	—
Amazon S3 (file storage)	Three buckets: camera frames, journal photos, chat uploads (deleted after a day)	Relay PC, PlantSensorAPI	Read by the website, chat and report
Amazon EventBridge Scheduler (timers)	7 schedules on Singapore time	—	Starts the photos, statistics, notes and report
Amazon SES (e-mail sending)	Sender on the project's domain	SmartPlantReport	Weekly PDF report by e-mail
AWS Amplify (website hosting)	Builds the website from GitHub	The owner's code push	Serves smartplantmonitor.com
Amazon CloudWatch Logs (logs)	Kept 14 days per function	All functions	Used for debugging; no alarms yet
AWS CloudShell (terminal in the browser)	Owner's scripts: system status report, soil calibration, setup snapshot	The owner	Reads the database; writes calibration settings

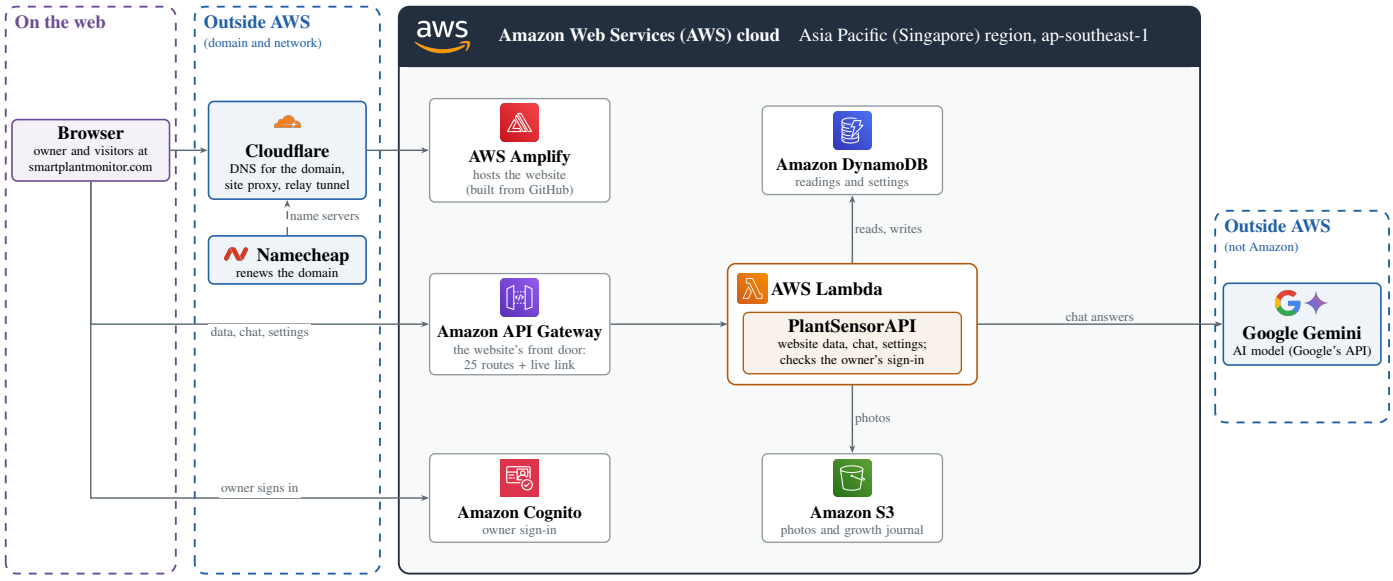


Fig. 6. How the website is reached. The domain's DNS is on Cloudflare; sign-in and data go straight to AWS.

`SmartPlantMonitorPolicy`, which lets them connect, publish and subscribe on any topic. IoT Core has five routing rules, and three are in use: readings go to DynamoDB, and pump events and servo confirmations go to PlantSensorAPI. The other two are older camera rules; both are switched off, and nothing has sent to their topics in the last 30 days. Messages to the boards are sent by PlantSensorAPI with its own AWS permissions, not with a device certificate.

The website never talks to a board directly. Every 5 s the page asks API Gateway for the latest data, which PlantSensorAPI reads from DynamoDB, and the live WebSocket link pushes changes, such as a pump run or a new camera angle, as

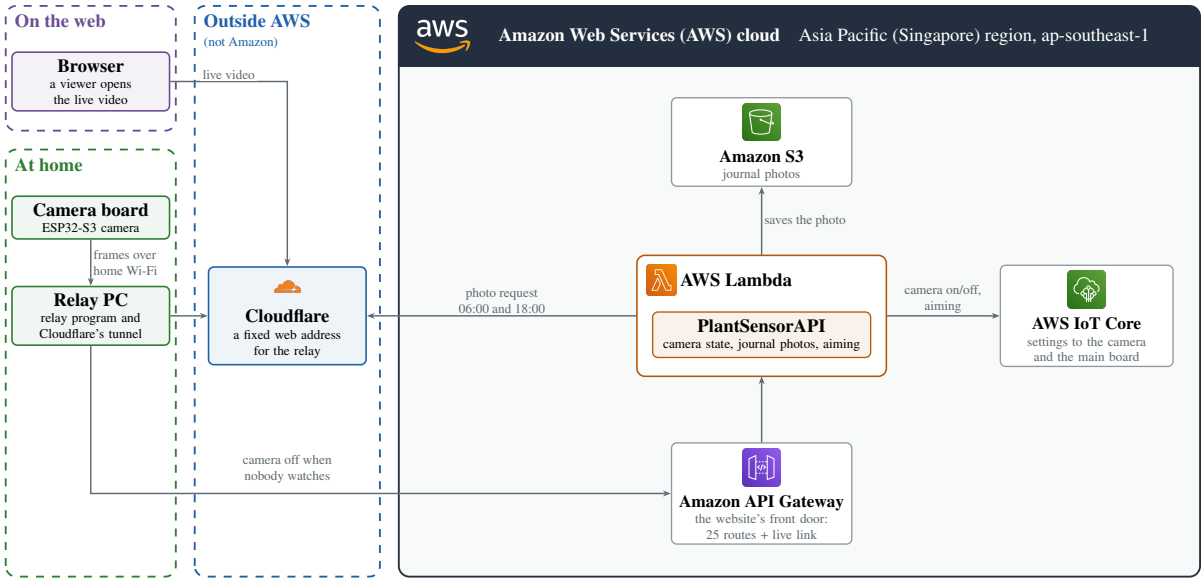


Fig. 7. How the live video reaches a viewer. The relay PC opens a tunnel to Cloudflare from inside the home.

TABLE V
MQTT TOPICS USED BY THE TWO BOARDS

Topic	From → to	Carries
`plant/sensors`	Main board → DynamoDB	Readings every 60 s
`plant/pump-event`	Main board → PlantSensorAPI	Pump runs and watering state
`plant/servo-status`	Main board → PlantSensorAPI	The camera angle reached
`plant/config` (retained)	PlantSensorAPI → main board	Settings and numbered requests
`plant/servo`	PlantSensorAPI → main board	Camera aiming
`SmartPlantCam/power`, `sleep`, `settings`, `reboot`	PlantSensorAPI → camera	Camera on/off, rest, picture settings, restart
`SmartPlantCam/status`	Camera → IoT Core	Camera status (not stored)

soon as PlantSensorAPI handles them. A button press travels the other way, from API Gateway through PlantSensorAPI and IoT Core to the board. The board's confirmation then comes back on its own topic, and the page shows the change once it arrives. The camera's state on the website comes from the relay, not from the camera's status messages, which no rule stores.

C. The AWS services and how they connect

Table IV lists each AWS service under its name in the AWS console, says in plain words what it is and what it does here, and shows what starts it and what it writes or calls. The work itself is done by three AWS Lambda functions, small programs that AWS runs only when something triggers them. *PlantSensorAPI* runs the website's data requests, the chat assistant and the camera aiming. *SmartPlantML* runs the statistics every 10 minutes. *SmartPlantReport* writes the daily notes and the weekly report. The setup was checked on 8 November with a read-only script that exports every rule, function, schedule, table and storage bucket, and compares the code running in AWS byte for byte with the project files.

D. Keeping the board and the cloud in agreement

Until 7 November the board kept its count of the day's waterings in memory that is wiped on every restart, and the cloud enforced the daily limit only after a run had finished. On 7 November the two flaws together let seven automatic waterings happen against a limit of six. The fix makes the board the single owner of watering, using numbered requests, much like ticket numbers at a counter:

- the board keeps the on/off switch, any pause and its reason, today's count, the time of the last run, the number of the last request it accepted, and up to eight runs it has not yet reported, all in flash memory that survives restarts and power cuts;
- every reading carries the board's watering state, and a full status report goes to the cloud every 60 s and whenever something changes;
- when the owner flips the switch, or the statistics layer asks for a pause, the cloud issues a request with the next number; the board accepts a request only if its number is higher than the last one it accepted, so an old, repeated message can never switch watering back on;
- the website shows *Waiting for the board to confirm* until the board's report matches, and *Board not reporting* if it goes silent for 3 minutes;
- each watering run carries its own start time from the board, and runs made while offline are sent once the connection returns.

Turning watering back on lifts a "no effect" or cloud pause but never the daily limit, which only the board's own midnight clears. Older firmware does not announce the new protocol, and

the cloud then falls back to the previous behaviour by itself, so rolling back the firmware needs no cloud change.

E. The website and day-to-day operation

The website (Fig. 2) is a single page with four areas: live readings, watering, camera and journal, and the analytics. It stores nothing itself; everything shown is a stored value or is calculated from one on the spot. Updates follow a fixed order: cloud first, board last. Each upload is verified by a checksum (a fingerprint of the file's contents) rather than by its file size, and each cloud change is made so that it still works with the firmware currently running.

F. How the website is reached

The project's domain is registered at Namecheap, which renews it and forwards any e-mail sent to it. Its name servers point to Cloudflare, so Cloudflare holds every DNS record for the domain: the website, the camera relay, and the records that let Amazon SES send the weekly e-mail from it.

Fig. 6 shows what happens when someone opens the site. The browser connects to Cloudflare, which passes the request on to AWS Amplify, and Amplify sends the page. From then on the page talks to AWS directly. The owner signs in with Amazon Cognito, Amazon's sign-in service, and the page asks API Gateway for readings, settings and chat answers. PlantSensorAPI fetches these from DynamoDB, S3 or Gemini, and checks the owner's sign-in with Cognito before it accepts any change. Table VI lists the services outside AWS.

G. How the live video reaches a viewer

The camera board sends its pictures over the home Wi-Fi to the relay PC (Section III-D). The relay also runs Cloudflare's tunnel program, which opens a connection from inside the home out to Cloudflare. Cloudflare then gives the relay a fixed web address, relay.smartplantmonitor.com, and no port has to be opened on the home router (Fig. 7).

A viewer's browser connects to that address for the live video. At 06:00 and 18:00 PlantSensorAPI uses the same address to ask the relay for a photograph, which it saves in S3 for the growth journal. The website sends camera on/off and aiming requests to API Gateway, and PlantSensorAPI passes them to the camera and the main board through AWS IoT Core. When nobody has watched for a short while, the relay itself asks for the camera to be switched off, so it rests between viewers.

TABLE VI
SERVICES OUTSIDE AWS

Service	What it does here
Namecheap	Registers and renews the domain; forwards its e-mail
Cloudflare	The domain's DNS, the site proxy, the relay tunnel
Google Gemini	AI model for the notes, report, chat and aiming
Tavily	Web search for the weekly report
GitHub	The website's code, which Amplify builds

V. ANALYTICS AND AI

A. Statistics first

Every 10 minutes the *SmartPlantML* function cleans the latest data and updates its models. It uses ordinary code only, with no AI and no external libraries. Two cleaning rules matter most. A reading outside the physically possible range is treated as missing, never forced into range, because a forced value would be a made-up measurement. And only readings the board itself marked as healthy are used to train models or to judge watering. Each model stays silent until it has enough clean history (Table VIII).

The same function also judges every automatic watering. Thirty minutes after a run it checks whether the soil rose by at least 2 points. One miss gives a warning, two slow watering to once an hour, and three ask the board to pause. That pause request is a numbered request like any other, and the board decides whether to apply it.

The forecast is judged honestly against the simplest possible guess, "the soil will stay as it is now". Over the 12 days to 22 August the forecast was off by 3.45 points on average, yet it did worse than that simple guess. The reason was the data, not the model: the probe had spent that period in a dry cup and then in a glass of water, where "no change" is almost always right. This report therefore claims no forecast accuracy until the model has been retrained on clean data from the probe in its final pot.

B. Warning signs from the plant and the sensors

The website's *Plant intelligence* card turns all of this into a short list of warnings, which come from two layers. First, the main board checks every sensor each minute and names its state, in steps that run from "does anything answer on the wire?" to "does the value still change when the world changes?". A sensor that fails a later step is plugged in, powered and broken, which needs a different fix from one that is not plugged in at all. Second, SmartPlantML reads those verdicts together with the readings and adds signals of its own: faults that affect the whole board, and signs that the plant itself is not doing well.

Table IX shows how each sensor failure is told apart. The soil probe has the most checks because it alone can start the pump: unattended watering runs only while the board rates the probe as working, so every soil fault in the table also stops automatic watering. A suspect number is usually kept and flagged rather than deleted, so the cloud can see what the pin was doing, but it is never used for training or for watering.

Table X lists the signs that the plant or its watering needs attention. Each signal has a severity, and the card groups them by kind, with *Not connected* first, because nothing else about a sensor can be known until it is plugged in. Signals that depend on a broken sensor are hidden, so one unplugged probe gives one warning instead of six. Every signal is advice only, except the watering pause, which is a numbered request that the board itself decides on (Section IV-D).

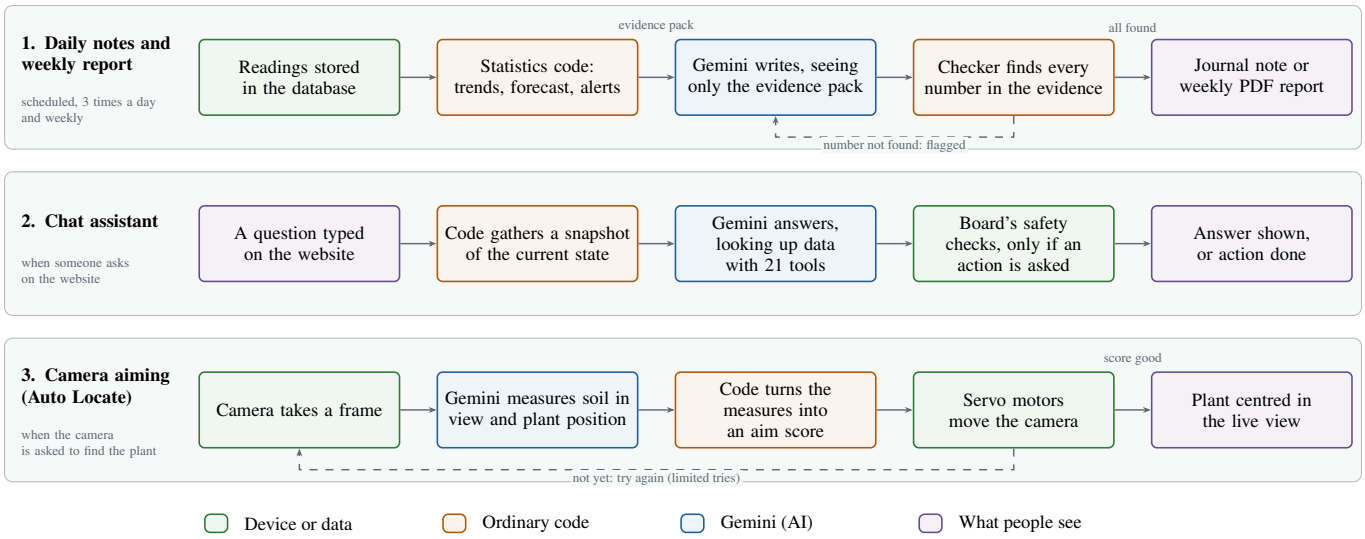


Fig. 8. Where the AI model is used. Each row is a separate feature, read left to right. In every row, ordinary code produces the numbers or makes the decision; Gemini only writes or measures. Nothing Gemini says can run the pump without passing the board's own checks.

TABLE VII
THE FIVE JOBS OF THE AI MODEL

Job	Model	What it is given	What it produces	What keeps it honest
Daily notes (08:00, 16:00, 23:45)	gemini-3.5-flash; 3.1 Flash Lite for small steps	An evidence pack: the day's rule-written summary, the readings, pump runs, alerts, photo descriptions	A note added to the growth journal, never replacing the rule-written text	A checker traces every number; web facts are labelled
Weekly report	same	The week's evidence, plus up to 5 web results (Tavily search) on what the plant needs at its current growth stage	A PDF report sent by e-mail	Same checker; web lines labelled with their source
Chat assistant on the website	3.1 Flash Lite; 3.5 Flash for harder questions	A snapshot of the current state, and 21 "tools" it can call to look up data	Answers; it can start actions such as watering	Actions pass through the same board checks as the website
Photo captions	3.1 Flash Lite	One journal photo	A caption, saved once	Supporting evidence only
Camera aiming (Auto Locate)	Gemini vision	One camera frame	Measurements: how much soil is in view, where the plant is	Code turns the measurements into a score and the next servo move

TABLE VIII
STATISTICAL MODELS AND THE HISTORY EACH NEEDS BEFORE IT IS USED

Model (in plain words)	What it produces	Needs (days / rows)
Rules and trends	"Above or below threshold" and direction of change	0 / 5
Alert rules	Alerts that must persist 2 checks to start and 3 to clear	0 / 30
Tank trend (a robust line fit)	When the tank will run dry	2 / 200
Weighted formula (ridge regression [2])	Soil moisture in 30, 60 and 180 minutes	10 / 2000
Odd-one-out detector (isolation forest [3])	Readings unlike anything in the last 14 days	14 / 5000

C. *What Gemini does*

Google's Gemini model has five jobs (Table VII, Fig. 8). In none of them does it produce a number that code has not already produced.

D. *How the AI is kept to the facts*

For the daily notes and the weekly report, Gemini never reads the database itself. Code first builds an *evidence pack* of the day's numbers, and the model sees nothing else. Its instructions set nine rules, among them: use only numbers that are in the pack, with no new calculations, not even an average; say plainly when evidence is missing; raise alerts only from the alert log; label anything taken from the web with its source; and write times as local clock times. The rules are then enforced by code: a checker pulls every number out of the finished text and looks for it in the pack, allowing for rounding. A number it cannot find is flagged instead of being published quietly. The judgement in a note (what matters today, what to watch) is the model's own; the figures behind it are not.

TABLE IX
HOW EACH SENSOR FAILURE IS TOLD APART

Sensor	Failure	How it is recognised	Shown on the website as
Soil probe	Not connected	The pin floats in the pull-up/pull-down test (a difference of 3000 counts or more) and the reading sits on a rail, on two tests in a row	*Not connected*; the reading is removed
	Impossible reading	Raw value below 100 or above 4000 counts	*Connected but reading impossible values*
	Out of the soil	Raw value at or above the probe's dry-air level	*Connected but out of the soil*
	Frozen	The same raw value for 60 readings (1 h) with no noise at all, or barely any movement for an hour in the cloud's view	*Connected but frozen*
	Dead input or short	All 20 samples of one reading within 2 counts of each other	*Connected but frozen*
	Loose contact	Samples spread over 400 counts or more, and at least 10 % of recent readings lost	*Probe placement and contact*; only watched if the readings keep arriving
	Sudden jump	A step of 300 counts or more within one minute	Logged as an event (probe moved, watering landed), never a fault
Temperature and humidity (DHT22)	Not connected	No answer to the start pulse	*Not connected*
	Data corrupt	Answers, but the frame fails its checksum	*Connected but the data is wrong*
	Impossible reading	Below −10 or above 80 °C, or humidity outside 0–100 %	*Connected but reading impossible values*
Light (BH1750)	Wiring fault	The I^2C lines are held low at start-up	*Wiring fault*
	Not connected	Nothing answers on the I^2C bus	*Not connected*
	Data wrong	A dropout, or the reading fails to scale when the sensor's sensitivity is changed, three times running	*Connected but the data is wrong*
	Impossible reading	Above 200,000 lux	*Connected but reading impossible values*
Water level (HC-SR04)	Not connected	The echo never starts	*Not connected*
	Hung	The echo starts and never ends	*Connected but frozen*
	Noisy	Three pulses disagree by more than 2 cm	*Connected but the data is wrong*
	Impossible reading	An echo closer than 2.5 cm	*Connected but reading impossible values*
	Tank empty	No echo from within the tank's depth	*Working – tank empty*; not a fault
Whole board (cloud)	No data	Nothing for 30 min, or for three times the usual interval if that is longer	*Nothing reported*
	Several sensors down	Three or more readings missing, or exactly zero, at the same time	*Node-wide fault*: power or ground is more likely than weather
	Blackout	Every reading missing	*Node-wide fault*

The chat assistant works differently. It starts each conversation with a snapshot of the current state (latest readings, the board's own health verdicts, watering status, recent trends, the forecast, the daily summaries and a description of the system), and it can call 21 lookup tools for more. Those tools answer to the minute for any time range, by day for any date, and exactly for records such as "hottest day so far", and they do not slow down as the data grows. The assistant can trigger a few actions, such as watering or a photo, but each one is a message the board may refuse.

Camera aiming follows the same rule. The AI only reports measurements of each frame, and code turns them into an aiming score. An earlier scoring rule simply rewarded more soil in view and drove the camera into the pot; the score now peaks at a target amount of soil and falls off on both sides.

TABLE X
SIGNS THAT THE PLANT OR THE WATERING NEEDS ATTENTION

Signal	Raised when	Severity
Dry, not watered	Soil below 30 % for 15 min and no pump run in 2 h, with auto-watering on	High
Dry, manual mode	The same, with auto-watering off	Medium
Watering had no effect	Soil rose less than 2 points 30 min after a run	Medium, then high
Watering paused	The daily limit of 6 runs was reached, or the board paused after runs with no effect	Medium or high
Tank low / empty	Water below 10 % / 5 % of the tank	Medium / high
Temperature stress	Outside 18–40 °C while the sensor itself is healthy	Medium
Unusual pattern	The mix of readings is among the most unusual 2 % of the plant's own history (after 100 clean readings)	Medium
Daily note	The day's soil, temperature and humidity compared with the plant profile (okra: soil 40–70 %, 24–35 °C, 50–75 % RH)	Text only

TABLE XI
PROBLEMS, THEIR REAL CAUSES AND THE FIXES

What was seen	What was actually wrong	Fix	Version
Hours offline until someone unplugged the board	Battery-era recovery code put the board into deep sleep on mains, where no watchdog can wake it	Restart the chip instead of sleeping; recover in widening steps without blocking	v6.6.5, v6.6.9
Watchdog crashes in the temperature step	The sensor library paused the chip's background work for an unlimited wait; confirmed by tracing the crash location	New reading routine that pauses for at most 8 ms and gives up cleanly	v6.6.7
Reset button failed; unplugging worked	The tank sensor's echo wire was on GPIO12, a strapping pin that decides how the chip starts	Echo moved to GPIO33; GPIO12 left empty	v6.6.7
Crash reports missing from the database	The routing rule lists fields by name and dropped the new ones	Every new field ships with a rule update	rule, 6 Nov
Soil readings drifted dry after an update	Reading the chip's temperature used the same converter as the soil probe (likely; not proven)	Chip-temperature reading removed	v6.6.8
Seven automatic waterings with a limit of six	Count kept in memory wiped by every restart; the cloud limit acted only after a run; the website switch re-enabled watering	Board keeps state in flash, checks the limit first, and only accepts numbered requests	v6.6.9, cloud v33
A watering run missing from history	Each run's end message reused the stored start time and overwrote the run before it	The board sends each run's own start time	cloud v33
Waterings that changed nothing	Water reached a part of the pot the probe was not in	Check the effect: the board pauses after two runs with no effect	v6.6.8, ML
No watering while offline	Start-up waited for Wi-Fi; restarts repeated	Start without waiting; restarts spaced out	v6.6.9
Bright-light spikes in the middle of the night	One corrupted light reading passed straight through	Middle of three readings; unjam the shared wire	v6.6.5
Forecast worse than "no change"	Calibration-era data (probe in a cup, then in water) in the training set	Use only healthy readings; retrain on clean data	ML
Camera servo jitter	Chargers on different wall sockets	All chargers on one power strip	7 Nov
Website switch showed OFF though nobody touched it	One setting was used both for the owner's switch and the automatic limit	The board reports the pause reason separately	cloud v33

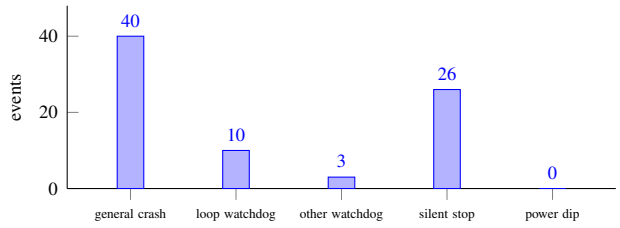


Fig. 9. What happened in the 48,582 records of the field study. Crashes and watchdog restarts recovered by themselves within about 2 minutes; every silent stop (116 h offline in total) needed someone to unplug the board. The power never dipped.

VI. Problems in the Field and How They Were Solved

A. What the data showed

Because the board reports its own restarts, crash locations and connection drops, its history can be studied after the fact. Over 48,582 records from 27 September to 5 November 2026 (Fig. 9) the power supply never dipped. There were 53 crash restarts (40 general crashes, called panics, plus 13 watchdog restarts), each recovering by itself in about 2 minutes. There were also 26 silent stops, 116 hours offline in total, and every one ended only when someone unplugged the board. Wi-Fi signal and free memory looked normal before the stops, which ruled out a weak connection or a memory leak early on.

B. Causes and fixes

Table XI lists the problems found between August and November 2026, the actual cause of each, and the fix. Three needed evidence the system did not collect at first:

- **The watchdog crashes** were pinned on the temperature sensor only after the breadcrumbs showed its name and a crash location was traced back to the source. The sensor's standard library paused all of the chip's background work while waiting, with no time limit, for a reply that sometimes never came.
- **The silent stops** were explained when pressing the board's reset button revived it. Recovery code written for battery operation had put the board into deep sleep "to save the battery", even on mains, and a sleeping chip cannot be rescued by its watchdog.
- **Crash reports went missing** because the database rule dropped every field it did not name.

VII. Design Decisions

Table XII lists the main design choices, the alternative each replaced, and the reason. Most were forced by a problem in Table XI rather than planned from the start.

VIII. Cost

Table XIII gives the cost to date in Singapore dollars. Cloud figures come from the AWS bill: October is exact, and earlier months are read from the billing console's monthly chart to the nearest S$0.5. Hardware prices are typical Singapore retail

TABLE XII
DESIGN DECISIONS AND WHY

Choice (instead of)	Reason
Board decides watering (cloud decides)	The plant stays safe offline, and a cloud bug cannot flood it
Memory in flash (memory wiped on restart)	Counts and settings must survive restarts and power cuts
Numbered requests (a simple on/off flag)	An old repeated "on" message re-enabled watering after the limit
Restart the chip (deep sleep)	A sleeping chip cannot be rescued by its watchdog
Three phone chargers (battery, power bank)	Voltage dips traced to small converters feeding bursty loads
One database table, pay per use	Any question is one simple lookup; cost follows usage
Serverless functions (a server running all day)	The load is a reading a minute plus a few scheduled jobs
Plain statistics (heavy ML frameworks)	Small, fast to start, and every result can be explained
Code computes, AI explains	Prevents invented numbers; a checker enforces it
AI measures, code scores the aim	Repeatable aiming; stopped the camera diving into the pot
Search service, labelled (the AI's own search)	The model's built-in search was refused on the free keys
Relay PC for video (straight to the cloud)	The camera cannot encrypt and stream fast at once
USB-only updates (updates over Wi-Fi)	The owner's security choice; costs one box opening per update

TABLE XIII
PROJECT COST TO DATE (S$)

Item	S$
Hardware in use (estimates)	
ESP32 DevKit V1 + expansion board	15
ESP32-S3 camera board	25
Temperature, light, soil and tank sensors	19
Relay switch, pump and tubing	12
Two servo motors and pan–tilt bracket	10
Three USB chargers, cables, power strip	40
Wiring, small parts, enclosure, pot, tank	30
Subtotal	*151*
Kept or replaced (estimates)	
Battery kit (4 cells, protection, converter, gauge, level shifter)	65
Power bank; INA219 current sensor (removed)	41
Spare ESP32 board and spare soil probe	13
Subtotal	*119*
Cloud and services (billed)	
AWS, June–October 2026 (about US$49.6)	64
AWS, 1–8 November 2026 (US$2.20)	3
Domain name, paid up front (US$15)	19
Gemini, Tavily, Cloudflare, GitHub (free tiers)	0
Subtotal	*86*
Total to date	**≈356**

TABLE XIV
AWS COST BY SERVICE, OCTOBER 2026 (US$)

Service	US$	Share
Amazon DynamoDB (database)	2.58	40 %
IoT Core (device messages)	1.42	22 %
AWS Amplify (website hosting)	0.84	13 %
Amazon API Gateway (web front door)	0.77	12 %
EC2-Other (not itemised)	0.77	12 %
All other services	0.02	<1 %
Total	6.40	

prices, not receipts. The PC that runs the camera relay was already owned and is not counted. US dollars are converted at about S$1.29.

Running costs are small, and most of the money goes to storing and moving data. In October the whole AWS bill was US$6.40, about S$8.30 (Table XIV). The database was the largest item, because every minute's reading is written to it and the website and statistics read from it. The device messages, website hosting and web front door made up most of the rest. The Lambda functions, file storage, e-mail, schedules and logs together cost at most two cents, which is the case for running the analytics as scheduled functions rather than on a server that runs all day. AWS forecasts US$9.50 for November. Monthly spend from June to October ranged from about US$5 to US$16.5. All AI and web-search calls run on free tiers. That works because the scheduled calls are few and the chat assistant rotates through several API keys; a paid AI plan would be the first new cost if usage grew. Electricity for the three chargers, a few watts in total, is well under S$1 a month.

IX. LIMITATIONS AND FUTURE WORK

The system looks after one plant in one place, and so far it has run only with the okra profile. Several things are not yet proven. The soil forecast has no established accuracy until it is retrained on clean data from the probe in its final pot, and the probe still needs calibrating in that pot. Calibration means recording its readings in the driest and wettest soil the plant should see, and it can now be done from the cloud without touching the board. Firmware v6.6.9 went into service on 8 November 2026, and its two-day reliability check was still running when this report was written. The camera was last recorded restarting itself about every 80 s when idle, so one of its recovery checks is firing when it should not; the cause has not yet been traced. With the current sensor removed, power figures date from the battery period. On the cloud side, the database has no point-in-time backup, the routing rules have no error handling, and no alarms are set. Both boards share one IoT policy that allows any topic; giving each certificate only its own name and topics, and deleting the two switched-off camera rules, would limit the damage if a key ever leaked. The weekly report's web sources are whatever the search returns, so a short list of trusted sites is the next step. Hardware work still to do: a raised and ventilated enclosure that keeps water out, a sealed soil probe (DFRobot SEN0308), and remounting the tank sensor over open water.

X. Conclusion

Three rules shape the system: the board decides and keeps its own memory; the cloud watches and asks through numbered requests that the board confirms; and the AI may explain or measure, but code computes every number and makes every decision. The field study showed how much one developer can learn from records the board keeps about itself. Each long-standing failure was closed at its cause, not worked around.

References

[1] Espressif Systems, *ESP32 Series Datasheet* and *ESP32 Technical Reference Manual*. [Online]. Available: `https://www.espressif.com`

[2] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[3] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, 2008, pp. 413–422.

[4] Amazon Web Services, *AWS IoT Core*, *AWS Lambda*, *Amazon DynamoDB* and *Amazon EventBridge Scheduler* developer guides. [Online]. Available: `https://docs.aws.amazon.com`

[5] Google, *Gemini API documentation: function calling and vision*. [Online]. Available: `https://ai.google.dev`

[6] Espressif Systems, *ESP-IDF Programming Guide v5.3: watchdogs, non-volatile storage, sleep modes*. [Online]. Available: `https://docs.espressif.com`