Simulation-Based Evaluation of an ESP32 Irrigation Controller with Safety Lockout and Internet of Things Monitoring

Rabee Hamza Ahmed Gharib[1]

[1]College of Economics and Political Science - Surman, University of Sabratha, Libya

Corresponding Author: E-mail: rabee@sabu.edu.ly

ORCID: 0009-0000-4276-1914

Abstract:

Irrigation controllers must switch reliably and respond safely to faults before they are connected to pumps and valves. This study tests an ESP32 irrigation controller in the Wokwi simulator, with remote monitoring through the ThingSpeak cloud service. A potentiometer provided an uncalibrated input index, a slide switch represented rain, and two LEDs showed the pump command and a safety lockout. The control rules started irrigation below 30%, stopped it at or above 40%, blocked it during rain, and enforced a latched 90-second limit on continuous operation. Separate simulator traces recorded a threshold stop at 40.76%, a rain-triggered shutdown, and a timeout exactly 90.000 seconds after start. After the input validator was corrected, an injected non-finite value stopped the pump command at 10.006 seconds, and the command restarted at 15.016 seconds once a valid value returned. In a 604-second cloud trace, 30 snapshots were scheduled, 25 uploads were confirmed and 25 records were stored. Four intermediate snapshots never reached the server, consistent with the latest-only queue design, and the final snapshot was still pending when logging ended. The stored records cover 86.21% of the 29 eligible sampling instants. Because these results come from separate development-stage runs rather than one regression suite on a frozen release, they support only the tested command transitions and cloud reporting. They do not demonstrate field performance, water savings or resilience to network outages. Machine-learning forecasting is identified as future work; no predictive model was trained or evaluated.

Keywords: Irrigation control; simulation; safety lockout; Internet of Things; cloud telemetry

1. Introduction

Automated irrigation systems combine soil measurements, weather data and decision rules to reduce the need for manual operation. Reliable agricultural setpoints, however, depend on calibrated sensors and crop-specific agronomic validation (Allen et al., 1998; Jones, 2004). Wireless instrumentation and remote monitoring for irrigation have been studied extensively (Gutiérrez et al., 2014; Kim et al., 2008). A simulation-only prototype is a practical way to check decision logic before real sensors and hydraulic equipment are connected, but it cannot by itself demonstrate efficient water use.

This study asks whether the implemented controller shows hysteretic switching, rain override, shutdown on invalid input and a latched runtime limit while it reports its state to a remote service. Its contribution is a limited functional evaluation based on time-stamped events, together with an account of which cloud samples were retained. The work does not propose a new agronomic control law, and the tests do not show that the controller outperforms existing irrigation controllers.

The study builds on the author's earlier Arduino UNO R4 WiFi prototype tested in Surman (Gharib, 2026). That prototype used an FC-28 soil sensor with a two-point relative-index calibration and verified local 30%/40% hysteresis and rain override. These thresholds are reused here and are not presented as new. The new elements are the ESP32 simulation, a latched runtime limit, handling of software-injected invalid inputs, and an analysis of how many cloud samples were retained. The soil calibration and hardware results of the earlier study do not carry over to the present potentiometer-based simulation. Table (1) summarizes these differences.

Table (1): Incremental contribution relative to the author's prior Arduino UNO R4 WiFi study (Gharib, 2026)

Dimension	Earlier study (Gharib, 2026)	Present ESP32 simulation
Control thresholds	30%/40% hysteresis and rain override	Same rules retained; not claimed as novel
Measurements/actuation	FC-28 calibrated index; real relay and DC pump; soil sample	Potentiometer-based uncalibrated index; LED command only
Additional safety	No 90-s latched runtime / injected-NaN result reported	90-s latched runtime; corrected non-finite-input shutoff
IoT evidence	Local prototype verification; cloud integration proposed	25 accepted eight-field ThingSpeak records; queue-retention analysis
Validation scope	Four irrigation cycles; sensor stability checks	Independent simulator traces; one 604-s cloud trace; no field efficiency claim

The analysis uses the original simulation logs and the private ThingSpeak feed exported on 7–8 November 2026. No field trials were carried out. Every percentage described as a "simulated index" refers to the potentiometer stimulus, not to calibrated volumetric soil moisture.

The work is therefore positioned as an implementation-level verification case rather than a new irrigation policy or a proven gain in crop performance. Relative to the earlier Arduino study (Gharib, 2026), this simulation documents a latched 90-s command timeout, shutdown on a software-injected NaN after the validator was corrected, and ThingSpeak retention measured against the scheduled source snapshots. Current IoT irrigation platforms (Saputri et al., 2025; Yellaboina & Bhowmik, 2025) already provide cloud monitoring and automation, so the use of an ESP32, a DHT22 sensor, ThingSpeak and 30%/40% hysteresis is not novel in itself. What distinguishes this study is an auditable, event-level record and a transparent count of accepted and omitted telemetry snapshots. It is a case study and includes no matched comparative experiment.

2. Methodology (Materials and Methods)

A. Simulation and circuit configuration

An ESP32 DevKit v1 board running the Arduino framework was simulated in Wokwi (Wokwi, 2026). A potentiometer on GPIO34 (ADC1) supplied the input index, and a slide switch on GPIO27 simulated rain (LOW = rain). A DHT22 sensor was read on GPIO15 with a 10-kΩ pull-up resistor. The pump-command and lockout LEDs were driven from GPIO26 and GPIO25 through nominal 220-Ω resistors. The model contains no relay, pump, soil column, water meter or hydraulic plant. Air temperature and relative humidity are logged as telemetry only and play no part in the control rule. The circuit is shown in Figure (1), and the components and monitored signals are listed in Table (2).

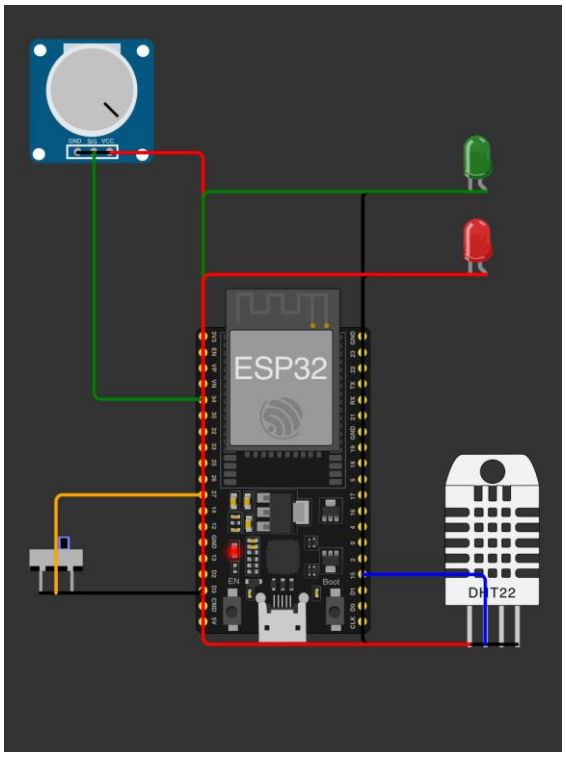


Figure (1): Wokwi simulation circuit of the ESP32 controller, showing the potentiometer, DHT22 sensor, rain switch, pump-command LED and lockout LED. The LEDs indicate commands only and do not represent hydraulic actuation.

Table (2): Simulated components and monitored signals

Item	Pin / field	Function
Potentiometer	GPIO34 / F1–F2	Raw 12-bit ADC; uncalibrated 0–100 index
DHT22	GPIO15 / F3–F4	Ambient temperature and relative humidity
Rain slide switch	GPIO27 / F5	Rain detected when LOW
Green LED	GPIO26 / F6	Commanded pump state only
Red LED	GPIO25 / F7	Latched timeout status
Timestamp	ThingSpeak F8	Source sample timestamp in simulator milliseconds

The simulated index I was obtained by normalizing the raw 12-bit reading of the analog-to-digital converter, ADC_{raw}, as shown in Equation (1).

$$I\ (\%) = 100 \times \frac{ADC_{raw}}{4095} \quad (1)$$

where ADC_{raw} lies between 0 and 4095 and I between 0 and 100%. The mapping was not calibrated against soil water content.

B. Control policy and safety ordering

The local control loop ran nominally every 100 ms, and temperature, humidity and serial status were logged every 2 s. The control module stored the pump state, the time of the last start and a lockout flag held in RAM. Each update first checked the runtime limit; invalid analog values and rain then stopped an active pump; an active lockout blocked new starts; and finally the 30%/40% hysteresis rules were applied. An invalid-input stop cleared automatically once a valid reading returned, whereas a timeout lockout remained until the board was reset. Table (3) lists these decisions in priority order, and Algorithm (1) gives the complete sequence.

Table (3): Control decisions evaluated on each update

Priority / condition	Controller action / event
Pump active for ≥ 90 s	Stop, latch lockout; STOP_SAFETY_TIMEOUT
Input non-finite or outside 0–100	Stop if active; STOP_INVALID_INPUT
Rain detected	Stop if active; STOP_RAIN_OVERRIDE
Lockout active	Suppress new pump starts
Pump ON and I ≥ 40%	Stop; STOP_TARGET_REACHED
Pump OFF and I < 30%	Start; START_LOW_MOISTURE

Algorithm (1): Ordered local control decision policy (evaluated approximately every 100 ms)

1. If pump == ON and uint32(now − started) ≥ 90000 ms: set pump = OFF; locked = true; return STOP_SAFETY_TIMEOUT.
2. Else if input is non-finite or outside [0, 100]: if pump == ON set pump = OFF and return STOP_INVALID_INPUT; otherwise retain OFF/state.
3. Else if rain == true: if pump == ON set pump = OFF and return STOP_RAIN_OVERRIDE; otherwise retain state.
4. Else if locked == true: suppress starting the pump (remain OFF).
5. Else if pump == ON and index ≥ 40.0%: set pump = OFF; return STOP_TARGET_REACHED.
6. Else if pump == OFF and index < 30.0%: set pump = ON, started = now; return START_LOW_MOISTURE.
7. Else: preserve pump command. Note: rain-transition log entries are emitted separately by the sketch.

The revised input validator inspects the exponent bits of the 32-bit floating-point value to reject NaN and infinity, and also rejects values outside 0–100. This change followed an earlier simulation in which a simple finite-value check failed to stop the pump command. The TC-06 result therefore applies only to the corrected firmware, not to the earlier version.

C. Cloud architecture, logging and dataset

Firmware version 3.3 connected to the Wokwi-GUEST wireless network and used a separate FreeRTOS task to upload eight fields to a private ThingSpeak channel (MathWorks, 2026). A single-slot queue held only the newest sample, so an older snapshot could be overwritten while the sender was busy. The target cloud rate was one snapshot every 20 s, independent of the faster local control loop. An upload was logged with result code 200 only if the HTTP status was 200 and the response body contained a positive ThingSpeak entry number. Each CLOUD_RESULT event recorded the timestamp of its source sample. The network task never wrote to GPIO26 or GPIO25. The receipt check is shown in Listing (1).

Listing (1): ThingSpeak receipt validation (shortened from ESP32 v3.3)

```
int status = https.POST(payload);
if (status == 200) {
  String body = https.getString(); body.trim();
  code = body.toInt() > 0 ? 200 : -401;
} else { code = status; }
```

The simulation used encrypted transport via WiFiClientSecure, but server certificate verification was disabled. The server's identity was therefore not authenticated, and this configuration cannot be taken as evidence of secure deployment. Authentication credentials are omitted from the manuscript.

The control loop placed a new snapshot in the queue every 20 s, while the transmission task waited at least 20 s after finishing one attempt before starting the next, and then sent the newest snapshot waiting in the queue. Because each network attempt itself took time, the two rates did not match, and a queued snapshot could be replaced without any HTTP request failing. The source logs show omissions consistent with this behaviour, but they contain no overwrite counter, so individual overwrite events were not measured directly.

Four sources of evidence were used: (i) local functional tests of switching and rain response; (ii) a separate 90-s timeout trace; (iii) a five-second injected-NaN trial run after the validator correction; and (iv) a continuous 604-s serial trace of ESP32 v3.3 paired with the 25-row ThingSpeak export (feeds.csv). Each

result is attributed to the run that produced it; the logs are not treated as one continuous experiment. Because the runs come from different development stages, they do not form a complete regression test of one frozen firmware release. The cloud trace is identified as v3.3, and the corrected invalid-input trial is reported separately. The available evidence does not show that all earlier unit tests were repeated on the final v3.3 build, so the results are reported as separate, supported scenarios.

D. Reproducibility and version provenance

The 604-s cloud trace identifies the firmware explicitly as v3.3 (BOOT_V3_3_CLOUD_SIMULATION). A supplementary archive contains the sanitized files sketch.ino, control.h, diagram.json and libraries.txt, representative local logs and the exported ThingSpeak feeds.csv; a separate manifest lists a SHA-256 digest for each key-free source file. These digests identify the distributed source files, not the ESP32 binary used in the original runs. One earlier build log reports ESP32 Arduino core 3.3.12 and Xtensa GCC 14.2.0, but this does not prove that the same toolchain was used for every experiment, and the DHTesp library version was not pinned. Exact source-to-binary provenance and reproduction of the build environment therefore remain unverified. Because the live Wokwi project can change, a tagged archive with locked dependencies is recommended before the repository is published.

E. Analysis metrics and scope

Cloud snapshots were counted from CLOUD_SNAPSHOT events, confirmed uploads from CLOUD_RESULT events, and server retention from the unique sample timestamps in ThingSpeak field 8. The snapshot-to-result interval was computed as $t_{result} - t_{sample}$ from simulator timestamps. Completeness over the 20–580-s sampling grid was computed as $N_{stored} / N_{expected} = 25/29$. This ratio is neither a packet-loss rate nor the probability that an HTTP request succeeds. No formal timing benchmark against a known physical input change was performed. Table (4) lists the evidence available for each scenario and the tests that remain outstanding.

Table (4): Evidence inventory and unresolved reviewer-requested tests

Scenario	Observed evidence	Status / qualification
Low-index start	Observed under 30%	Recorded in local logs
Hysteresis middle state / upper stop	36.26% ON; 40.76% OFF	Recorded once
Rain override	STOP_RAIN_OVERRIDE	Recorded
Runtime lockout	Exactly 90.000 s; RAM lock	Recorded in separate run
Injected NaN	STOP_INVALID_INPUT after code correction	Recorded in separate run
Cloud receipt	25 confirmed server entries	Recorded in v3.3 run
Exactly 30.0/40.0 input	No controlled boundary trial	Not tested
Infinity / out-of-range / physical sensor disconnect	No independent stimulus log	Not tested
Simultaneous timeout and rain	No explicit coincident-condition trial	Not tested
Wi-Fi forced offline / repeated 10-min sessions	No controlled trials	Not tested

3. Results and Discussion

A. Hysteresis and rain override

In the local hysteresis trial, the pump command started at an index of 1.47% (t = 0.515 s), stayed ON at 36.26% (t = 20.006 s), and stopped with STOP_TARGET_REACHED at 40.76% (t = 20.215 s). A later low stimulus of 25.91% restarted it (t = 45.215 s). Rain then triggered STOP_RAIN_OVERRIDE at t = 54.815 s, with the index at 0%. These transitions were produced by stimuli chosen by the operator and are not repeated random trials. Separate start and stop thresholds prevent repeated switching while the input stays between 30% and 40%; immunity to field noise was not assessed.

B. Runtime lockout and injected invalid-input response

In the dedicated timeout trace, START_LOW_MOISTURE was logged at 35,015 ms and STOP_SAFETY_TIMEOUT at 125,015 ms, exactly 90,000 ms apart. The command then remained OFF with lockout = 1 until the last logged row at 320,006 ms. This confirms that the lockout persisted during the logged interval, but not that it would survive a power loss.

In the final NaN-injection test on the corrected firmware, the pump command turned ON at 515 ms, STOP_INVALID_INPUT was logged at 10,006 ms, and START_LOW_MOISTURE followed at 15,016 ms once a valid index of 10.35% was restored, giving an OFF interval of 5.010 s. The fault was injected in software and says nothing about behaviour under a real disconnection, a floating input or sensor drift. The automatic restart after a fault clears is a weakness for field use and should be compared with a policy requiring manual acknowledgement. Table (5) summarizes the functional events observed across the independent runs.

Table (5): Observed functional events from independent simulator runs

Test	Observed event(s)	Assessment
TC-01/02/03	36.26%: ON; 40.76%: STOP_TARGET_REACHED; 25.91%: restart	Pass in tested sequence
TC-04	STOP_RAIN_OVERRIDE at 54.815 s	Pass
TC-05	35.015 → 125.015 s; lockout persists through 320.006 s	Pass in logged window
TC-06	STOP_INVALID_INPUT 10.006 s; restart 15.016 s	Pass after validator correction
TC-07/08	Wi-Fi connected; HTTP 200; eight fields stored	Pass for confirmed transfers
Network-outage trial	Not performed in simulator	Not evaluated

C. ThingSpeak telemetry: transmission and completeness

The 604-s v3.3 trace contains 30 CLOUD_SNAPSHOT events spaced about 20 s apart, from 20.000 s to 600.001 s. It also contains 25 CLOUD_RESULT records, each with Wi-Fi connected = 1 and result code 200 (HTTP 200 plus a positive entry number in the response body). The separate ThingSpeak export holds 25 stored rows whose field 8 values match the confirmed uploads, so server receipt is confirmed for these 25 samples. The final snapshot, at 600.001 s, had no result before logging stopped; it is treated as censored rather than as a failed upload.

Of the 29 sampling instants expected between 20 and 580 s, the server retained 25 (86.21% schedule completeness). Four snapshots, at 140, 260, 400.001 and 520.001 s, appeared in the local log but not in the server feed. The single-slot, latest-only queue, combined with the mandatory 20-s wait after each attempt, can replace older pending samples even when every request succeeds. The serial trace shows successful requests carrying newer samples, which supports this explanation, but it does not show a failed HTTP request for any of the four missing samples. All 25 completed requests returned status 200; this conditional 25/25 success should not be read as 100% network reliability. Table (6) accounts for all 30 scheduled snapshots.

Table (6): Accounting for the 30 scheduled source snapshots in the 604-s cloud trace

Group	Scheduled source instants (simulator s)	Observed server status
Confirmed retained	25 instants between 20 and 580 s	25 CLOUD_RESULT accepted codes and 25 matching stored feed rows
Intermediate omitted	140; 260; 400.001; 520.001	No corresponding stored entry; consistent with superseded pending snapshots
Right-censored	600.001	CLOUD_SNAPSHOT recorded; no result before log ended

The difference between each source-sample timestamp and its CLOUD_RESULT ranged from 3.873 s to 23.302 s, with a mean of 12.915 s and a median of 12.322 s (n = 25). This interval includes application scheduling and any wait for the communication task, so it is not a physical network round-trip time. Likewise, ThingSpeak created_at timestamps cannot be compared directly with the simulator's millis() clock to measure end-to-end latency. Because only one continuous cloud run was recorded, no confidence interval or long-term reliability can be estimated. Figure (2) shows the retained samples, Figure (3) shows the snapshot-to-result delays, and Table (7) summarizes the telemetry statistics.

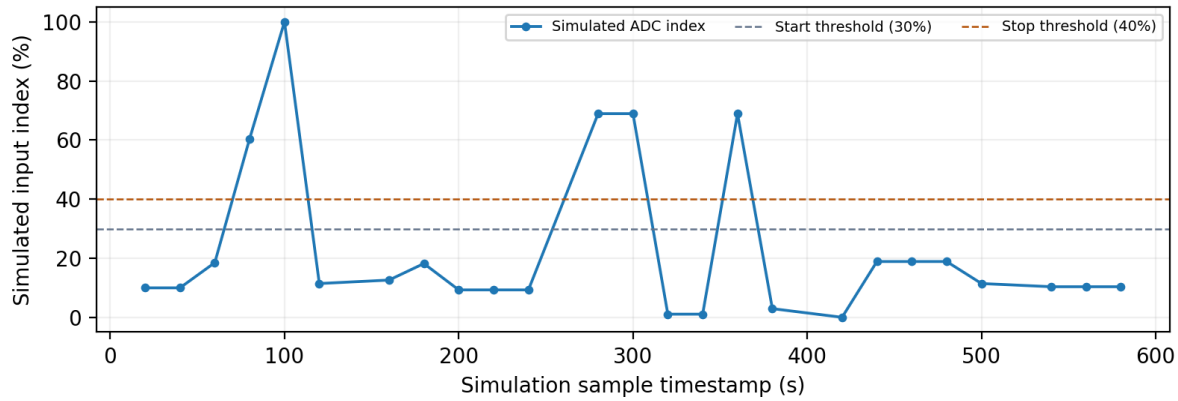


Figure (2): The 25 ThingSpeak samples retained by the server, plotted against ESP32 simulator source time (s). The vertical axis is the uncalibrated 0–100% potentiometer index, not volumetric soil water content. The scheduled snapshots at 140, 260, 400.001 and 520.001 s are missing from the stored series.

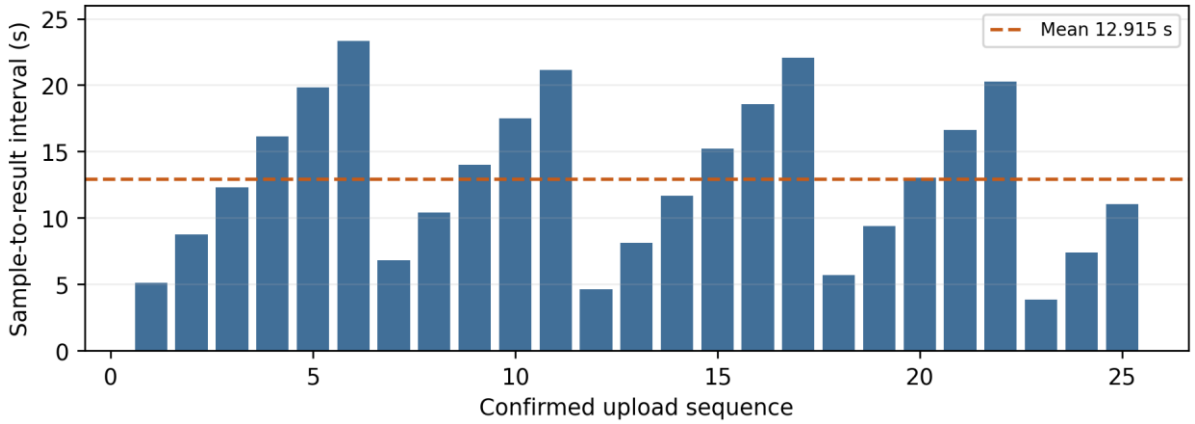


Figure (3): Delay (s) from each source snapshot to its CLOUD_RESULT, computed from simulator timestamps for the 25 confirmed samples. The delay includes task scheduling and networking and is not a measured network round-trip time.

Table (7): Ten-minute cloud telemetry statistics (ESP32 v3.3)

Measure	Observed value / interpretation
Trace coverage	0–604 s; 30 snapshots scheduled at 20–600 s
Cloud results / retained server entries	25 results (HTTP 200) / 25 ThingSpeak entries
Intermediate snapshot omissions	4 (140, 260, 400.001, 520.001 s)
Final source sample	600.001 s; result pending at log end
Completeness (20–580 s)	25/29 = 86.21%; not HTTP success probability
Result-code success in completed records	25/25 HTTP 200; conditional on completed records
Snapshot-to-result mean / median	12.915 s / 12.322 s (simulator clock)
Snapshot-to-result min / max	3.873 s / 23.302 s (simulator clock)

D. Independence of control from reporting and limitations

Local state changes continued while telemetry was running, including a STOP_SAFETY_TIMEOUT at 227.205 s followed by confirmed uploads of the locked state. This shows that control and cloud reporting can run together while the network is connected, but not that control is independent of the network during a deliberate Wi-Fi failure. A connectivity-interruption test was not possible in the available simulation workflow; it remains untested and is not a hidden positive result.

The experiments did not measure volumetric soil water content, irrigation volume, energy use, crop response, physical response time, relay wear, real sensor failure or resilience to network loss. DHT22 inputs were adjusted by hand, and the LED shows the command rather than feedback from a pump. The NaN rule lets the pump restart automatically once a valid low index returns, and the runtime lockout is held in volatile RAM, so a reset or power interruption clears it. Neither behaviour should be assumed fail-safe in an unattended field installation, which may need manual acknowledgement, pump feedback, a hardware cut-off and persistent fault storage. Because certificate verification was disabled, the v3.3 TLS link must not be described as an authenticated secure channel. Soil-sensor calibration and repeated tests are required before any field deployment.

Several cases remain untested: transitions exactly at 30.0% and 40.0%, retention of the OFF state inside the hysteresis band after a stop, simultaneous rain and runtime expiry, injected positive or negative infinity, finite out-of-range values, and controlled disconnection or floating analog inputs. No results are claimed for these cases. The tests come from separate development-stage runs rather than one regression suite on a frozen build. Several independent 10-minute cloud runs and an explicit queue-overwrite counter are needed to estimate run-to-run variation and to separate overwrites from other causes of omission. No deliberate Wi-Fi interruption test was carried out. These gaps are methodological limitations that editing alone cannot remove.

E. Future integration of artificial intelligence

Future work will examine a cloud-based machine-learning model that forecasts soil moisture from past moisture readings, air temperature, relative humidity, rainfall status and pump records. Such forecasts could trigger early irrigation alerts while the ESP32 keeps local control and safety functions. Developing the model will require calibrated sensor data covering several irrigation and drying cycles. The forecast horizon should be fixed before evaluation, and the data split chronologically into training, validation and held-out test sets to avoid temporal leakage. Mean absolute error and root mean squared error should be compared with a persistence baseline that predicts the most recent moisture value. The hand-varied potentiometer inputs and 25 cloud records in this study do not form a soil-moisture forecasting dataset, and no artificial-intelligence model was trained or evaluated here.

F. Comparative interpretation and boundary of novelty

The earlier hardware study (Gharib, 2026) used a real pump and soil sample but did not record a v3.3-style eight-field telemetry retention trace. The present ESP32 simulation, in contrast, gives clearer software event timestamps and safety-state ordering but less hydraulic realism. The 25/29 schedule completeness reflects an application-level design trade-off between freshness and complete history, and it should not be compared with a network packet-loss rate. Previous IoT irrigation studies (Gutiérrez et al., 2014; Kim et al., 2008; Saputri et al., 2025; Yellaboina & Bhowmik, 2025) show that remote monitoring is already well established; this paper adds a small, reproducible case documenting what the controller was observed to do and which samples the cloud trace failed to keep. Any claim of better performance, lower cost or higher water-use efficiency would need a matched experimental baseline and lies outside this study.

The findings therefore support a laboratory simulation report, not a field deployment. The comparator study (Gharib, 2026) already tested the 30/40 switching rule with soil samples and a physical pump; the present work adds tests of the ESP32-specific timeout, invalid-index safety and cloud retention. Superiority, water saving, robustness to physical sensor disconnection and resilience during a deliberate Wi-Fi outage

were not evaluated. A future deployment should enforce certificate verification, count dropped queue entries, calibrate real soil sensors, check actuator feedback and repeat all tests on a single firmware version.

4. Conclusion

In the Wokwi simulation, the ESP32 controller started the pump command at low index values, followed the hysteresis thresholds, stopped when rain was detected, enforced a 90-second runtime lockout and, after the validator correction, shut down on an injected NaN. During a 604-second serial record, ThingSpeak stored eight-field data from 25 confirmed transfers. Four intermediate samples were missing from the server, consistent with pending samples being replaced under latest-only queueing, although individual overwrites were not measured. The final sample was still pending when recording stopped. The 86.21% (25/29) completeness describes how much of the schedule was retained, not a network failure rate. These pilot results justify further testing of the implementation but apply only to the reported simulator runs. They do not show water savings, field readiness or resilience to network outages. Machine-learning forecasting of soil moisture remains a proposed extension that needs new data and independent evaluation. Exact-threshold tests, multi-run variance estimates, pinned toolchain versions, real sensor disconnections and a single-version regression series are still outstanding.

Data Availability

The evidence for this study comprises simulator source files, time-stamped local logs and an exported ThingSpeak feed from a private channel. No public archival repository for the complete versioned evidence package is cited, which limits independent reproduction. Authentication keys are excluded. A sanitized supplementary archive with the source code, representative logs, reproducibility notes and a file-level SHA-256 manifest is provided separately; the manifest does not certify that the original executable was compiled from exactly these files. The Wokwi project is available at https://wokwi.com/projects/480068360727109633 (accessed 8 November 2026). Because this live link is not an immutable snapshot, its availability and firmware revision should be checked before reproduction, and a verified content hash with pinned dependencies is recommended for archival release.

References

Allen, R. G., Pereira, L. S., Raes, D., & Smith, M. (1998). *Crop evapotranspiration: Guidelines for computing crop water requirements* (FAO Irrigation and Drainage Paper No. 56). Food and Agriculture Organization of the United Nations.

Gharib, R. H. A. (2026). Smart irrigation using Arduino UNO R4 WiFi in Surman, Libya. *International Science and Technology Journal*, *39*. https://mail.stcrs.com.ly/istj/en/article-details.php?id=1629

Gutiérrez, J., Villa-Medina, J. F., Nieto-Garibay, A., & Porta-Gándara, M. A. (2014). Automated irrigation system using a wireless sensor network and GPRS module. *IEEE Transactions on Instrumentation and Measurement*, *63*(1), 166–176. https://doi.org/10.1109/TIM.2013.2276487

Jones, H. G. (2004). Irrigation scheduling: Advantages and pitfalls of plant-based methods. *Journal of Experimental Botany*, *55*(407), 2427–2436. https://doi.org/10.1093/jxb/erh213

Kim, Y., Evans, R. G., & Iversen, W. M. (2008). Remote sensing and control of an irrigation system using a distributed wireless sensor network. *IEEE Transactions on Instrumentation and Measurement*, *57*(7), 1379–1387. https://doi.org/10.1109/TIM.2008.917198

MathWorks. (2026). *ThingSpeak documentation*. Retrieved November 8, 2026, from https://www.mathworks.com/help/thingspeak/

Saputri, F. R., Linelson, R., Salehuddin, M., Nor, D. M., & Ahmad, M. I. (2025). Design and development of an irrigation monitoring and control system based on Blynk Internet of Things and ThingSpeak. *PLOS ONE*, *20*(4), Article e0321250. https://doi.org/10.1371/journal.pone.0321250

Wokwi. (2026). *ESP32 simulation*. Retrieved November 8, 2026, from https://docs.wokwi.com/guides/esp32

Yellaboina, M. P., & Bhowmik, B. R. (2025). IoT-based smart irrigation system for optimized water usage in agriculture. In *2025 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)* (pp. 1–6). IEEE. https://doi.org/10.1109/DISCOVER66922.2025.11258968

Appendix A. Invalid simulated index validation (ESP32 v3.3)

```
inline bool validSoilIndex(float value) {
  uint32_t bits;
  memcpy(&bits, &value, sizeof(bits));
  bool finite = (bits & 0x7F800000UL) != 0x7F800000UL;
  return finite && value >= 0.0f && value <= 100.0f;
}
```