GreenWatch: An Edge-AI CPS Prototype for MQTT-Based Greenhouse Monitoring

Cayleigh Goberman, Roman Huerta Manrique, and Pablo Rivas

Marist University, Poughkeepsie, NY, USA
{cayleigh.goberman1, roman.huertamanrique1, pablo.rivas}@marist.edu

Abstract. Greenhouse care depends on timely sensing and interpretation of plant stress before environmental or visible symptoms become severe. Small growers therefore need systems that combine low-cost sensing with clear operator guidance, rather than raw telemetry alone. GreenWatch addresses this need through an edge-oriented greenhouse architecture that links environmental sensing, message-based event transport, plant-specific monitoring rules, and local language-model advice. The broader architecture also includes visual plant inspection and access monitoring. Here we show that the environmental path can collect temperature, humidity, and light readings over serial input, publish them through MQTT, confirm sustained threshold violations, and produce local advice records after alerts. An instrumented run produced regular sensor cycles, confirmed light-condition alerts for the succulent profile, and saved structured advice outputs from the local model. These results show that a compact cyber-physical stack can transform greenhouse telemetry into operator-facing guidance while preserving inspectable runtime traces. Future work extends the same architecture to camera-based plant inspection, access monitoring, dashboard storage, notification delivery, and longer greenhouse deployments.

Keywords: Cyber-physical systems · Embedded systems · Greenhouse monitoring · MQTT · Edge AI

1 Introduction

Greenhouse monitoring is a practical cyber-physical systems (CPS) problem because plant health depends on both environmental variables and visible plant condition. Temperature, humidity, and light affect plant stress, while wilted leaves, discoloration, blights, pests, and unexpected movement near plants can require rapid human attention. Prior smart-greenhouse work often follows a sensor-gateway-application pattern: low-cost sensors collect environmental readings, a gateway moves the data, and software reports abnormal conditions or supports remote supervision [4]. IoT farming surveys also note that camera-based computer vision can support pest and disease monitoring when image data are available [2].

GreenWatch targets this mixed monitoring problem. The system design uses Arduino nodes for greenhouse sensing, Raspberry Pi gateways for aggregation

and MQTT messaging, an AI camera path for plant inspection, and an LLM layer for plain-language summaries. In the full design, one Arduino reports temperature, humidity, and light; a second Arduino monitors door state and nearby motion; a vision Raspberry Pi analyzes plant images; and a hub stores data, serves a dashboard, and sends daily or urgent summaries.

GreenWatch combines environmental sensing, MQTT-based event transport, plant-profile monitoring, and local language-model advice generation into an end-to-end greenhouse monitoring path. In the environmental path, an Arduino-based sensing node sends JSON records to a Raspberry Pi publisher, the publisher routes readings through MQTT, a monitor checks plant-specific thresholds and confirms sustained anomalies, and a local language-model service produces operator-facing advice after confirmed alerts. The broader GreenWatch architecture also includes visual inspection and access monitoring. MQTT fits this setting because its publish/subscribe model decouples sensing devices from subscribers [6]. Edge gateways also let part of the analysis remain close to the sensors, though resource limits constrain compact CPS deployments [1].

We assess GreenWatch through an instrumented run of the environmental path that exercises sensing, MQTT communication, alert confirmation, and local language-model advice generation. The run contains 427 complete sensor cycles, 197 out-of-range monitor events, 15 confirmed alerts, and 13 local advice records. All confirmed alerts concern light conditions for the succulent profile. Results show completion of the sensor-to-advice workflow for one greenhouse-monitoring case, while agronomic validation, longer deployments, broad plant coverage, camera-based pest detection, dashboard operation, and notification delivery remain outside this run.

The paper makes three contributions:

1. A CPS design frame for greenhouse monitoring that separates environmental sensing, visual inspection, MQTT transport, rule-based confirmation, and LLM-based operator summaries.
2. A compact implementation of the environmental-monitoring path using Arduino serial JSON, Raspberry Pi MQTT publication, plant-profile threshold checks, and local advice generation.
3. A prototype evaluation of the environmental path, including telemetry counts, alert behavior, advice-record structure, and defined scope boundaries.

2 Background and Related Work

2.1 IoT Greenhouse Monitoring

Greenhouse monitoring is a common CPS setting because its variables are concrete and can be sensed at low cost. Prior smart-greenhouse and smart-agriculture systems measure values such as temperature, humidity, light, and moisture, then send readings to a gateway, cloud service, or dashboard [4,5]. GreenWatch follows this pattern for environmental telemetry, but its system design also includes image-based plant inspection and plain-language feedback.

2.2 Vision and Plant-Health Monitoring

Environmental readings alone cannot identify all greenhouse problems. Pest activity, wilting, discoloration, and disease symptoms often require visual inspection. Reviews of smart-farming systems report that IoT deployments use camera data and computer vision for crop monitoring, plant disease detection, and pest identification [2]. GreenWatch includes this vision layer through an AI camera path, while the prototype run reported here centers on environmental monitoring and advice generation.

2.3 MQTT-Based CPS Messaging

MQTT is widely used in IoT systems because publish/subscribe messaging separates producers from consumers and routes data through broker-managed topics [6]. This model fits GreenWatch: the publisher sends device-scoped sensor messages, while the monitor subscribes without reading the serial stream directly. Public-broker operation is convenient during early prototyping, but prior MQTT security work notes the need for authentication, message protection, and broker trust [3]. GreenWatch uses MQTT as transport; broker hardening and authenticated deployment are treated as future operational work.

2.4 Edge AI and Local Advice

Edge intelligence places compute or decision support near the data source [1]. GreenWatch uses this idea in a limited way: numerical rules detect alerts, and the local LLM service is called afterward for operator-facing advice. This split keeps detection auditable from logs. Work on edge LLMs also shows why advice-quality claims must be careful: model size, memory, compute, and latency remain practical limits [7].

2.5 Evaluation Design

GreenWatch is assessed through an instrumented run that exercises serial sensing, MQTT publication, threshold monitoring, alert confirmation, local language-model advice generation, and record storage. This evaluation design isolates the environmental-monitoring path while preserving the broader system model for visual inspection, monitoring, dashboarding, and notification delivery.

3 Problem Setting and Requirements

3.1 Greenhouse Monitoring Problem

GreenWatch addresses greenhouse care as a cyber-physical monitoring problem. A grower must track plant-specific temperature, humidity, and light conditions while also watching for visible stress signals such as bugs, wilted leaves, discoloration, or blight. The useful state of the greenhouse is distributed across the

Table 1. GreenWatch requirements and confirmed prototype scope.

ID	Requirement	Confirmed scope	Paper role
R1	Sense temperature, humidity, and light in the greenhouse.	Implemented with Arduino firmware, DHT11, LDR, and serial JSON.	Runtime core.
R2	Publish environmental records through MQTT.	Implemented through Raspberry Pi publication to device-scoped topics.	Runtime core.
R3	Apply plant-specific environmental thresholds.	Implemented through plant-profile monitoring; evaluated with succulent.	Runtime core.
R4	Confirm sustained abnormal readings before alerting.	Implemented with repeated-reading confirmation, forgiveness, and hysteresis.	Runtime core.
R5	Generate plain-language advice after confirmed alerts.	Implemented with a local `hailo-ollama` service using `qwen2:1.5b`.	Runtime core; quality bounded.
R6	Inspect plants visually for bugs, wilting, discoloration, or blight.	Defined as an AI-camera module.	Architecture extension.
R7	Monitor greenhouse access or nearby movement.	Defined as a door/motion sensing module.	Architecture extension.
R8	Store records, show dashboard views, and send daily or urgent notifications.	Defined as hub services.	Architecture extension.

physical space, so the system needs sensing at the plant environment, transport to a hub, event interpretation, and human-facing summaries that turn raw readings into actions.

This problem has two sensing modes. The environmental mode measures temperature, humidity, and light with an Arduino sensor node and sends structured records to the Raspberry Pi gateway. The visual and access-monitoring mode extends the same greenhouse watch concept with an AI-camera node for plant inspection and a second Arduino path for door or motion events. The hub role is to collect records, apply plant-specific logic, maintain outputs, and generate daily or urgent summaries for the operator.

The prototype presented here realizes the environmental sensing-to-advice path: serial sensor input, MQTT publication, plant-profile threshold monitoring, alert confirmation, and local language-model advice generation. Camera-based plant inspection, door and motion events, persistent dashboard views, and operator notification are broader GreenWatch modules that share the same system architecture but are outside the instrumented run analyzed in this paper.

3.2 Requirements

Table 1 separates the runtime core from broader GreenWatch modules. This distinction gives the paper a precise systems boundary: the measured environmental path demonstrates end-to-end operation, while visual inspection, access monitoring, dashboarding, and notification delivery remain coherent extensions of the same greenhouse-monitoring architecture.

3.3 Design Scope

GreenWatch is a monitoring and advice system, not a closed-loop greenhouse controller. It does not actuate fans, pumps, lights, or irrigation in the prototype run. The measured contribution is the environmental path that converts Arduino sensor readings into MQTT events, plant-profile decisions, confirmed light alerts, and local advice records. Plant-care correctness, pest diagnosis from images, and long-term greenhouse reliability require later studies with calibrated sensors, controlled plant scenarios, and expert review.

Reproduction of the measured path is supported at the interface level. The Arduino firmware defines the serial JSON stream, the publisher forwards device-scoped MQTT messages, the monitor applies plant-profile thresholds, and the output records preserve alert and advice fields. These interfaces make the environmental path inspectable while allowing the larger GreenWatch architecture to add image analysis, access monitoring, dashboard storage, and notification services without changing the core telemetry path.

4 System Design

GreenWatch is organized as a layered greenhouse CPS with sensing, gateway, messaging, monitoring, advice, and operator-output layers. Figure 1 shows this structure. The solid path is the environmental sensing-to-advice path analyzed in this paper. The dashed modules show the broader GreenWatch architecture: visual inspection, monitoring, dashboard storage, and operator notification.

4.1 Environmental Sensing Path

The environmental sensor node uses an Arduino with a DHT11 sensor for temperature and humidity and an analog light-dependent resistor path for light. The firmware emits newline-delimited JSON over USB serial at 115200 baud. Each sample period emits separate telemetry records for temperature, humidity, and light, with units of Celsius, percent relative humidity, and raw ADC value. The firmware samples approximately every two seconds and can also emit local threshold events with a per-sensor cooldown.

Light values require explicit interpretation because the LDR circuit is inverse: low ADC values correspond to brighter light, and high ADC values correspond to darker light. The system therefore maps low ADC threshold violations to bright-light conditions and high ADC threshold violations to low-light conditions. This convention is preserved in the monitor-side condition labels used during evaluation.

4.2 Gateway and Messaging Path

The Raspberry Pi publisher bridges serial sensing to MQTT. It receives line-oriented JSON records from the Arduino, filters malformed messages, and publishes valid sensor, status, and event records under a device-scoped topic hierarchy. The MQTT layer uses a public HiveMQ broker for networked operation

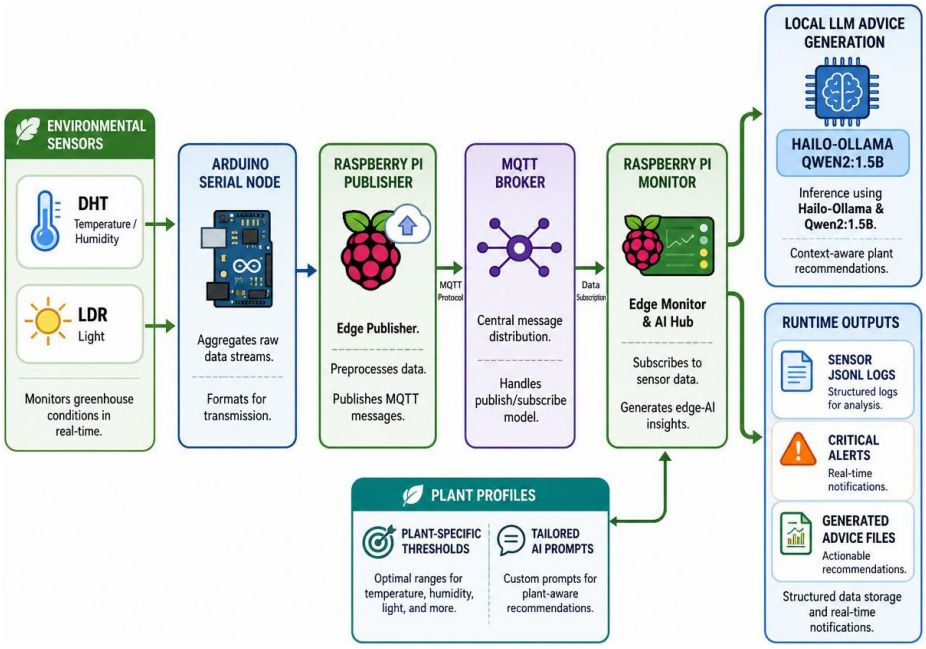

Fig. 1. GreenWatch greenhouse-monitoring architecture. The solid path is the instrumented environmental sensing-to-advice path; dashed modules indicate visual inspection, access monitoring, dashboard storage, and notification services that extend the same architecture.

and a local Mosquitto broker as a fallback. This broker abstraction keeps the Arduino firmware independent of the monitor, dashboard, and advice layers.

4.3 Monitoring and Advice Path

The Raspberry Pi monitor subscribes to the device topics, filters sensor records, and applies plant-profile thresholds. It uses a confirmation policy before raising an alert: repeated out-of-range readings in the same direction are required, short in-range interruptions are forgiven, and hysteresis limits rapid re-firing near a threshold. This design keeps alert generation deterministic and auditable before any language-model output is requested.

After a confirmed alert, the monitor builds an advice prompt from the plant profile, triggering sensor, condition label, latest value, and recent-window statistics. The prompt is sent to a local language-model service through `hailo-ollama` with model `qwen2:1.5b`. The model output is requested as structured JSON with summary, condition, action, priority, and follow-up fields. The monitor saves the advice output and publishes a report message.

Table 2. Implementation profile of the GreenWatch environmental path.

Layer	Component	Implementation detail
Sensing	Arduino firmware	Reads DHT11 temperature/humidity and LDR light; emits newline-delimited JSON over USB serial at 115200 baud.
Gateway	Publisher process	Bridges serial telemetry to device-scoped MQTT topics and records publication events.
Transport	MQTT broker	Uses HiveMQ as the confirmed broker path, with local Mosquitto support in the software design.
Runtime	Edge software	Runs on Debian with Python 3.12; separates publication, monitoring, and advice generation into distinct processes.
Profiles	Plant-profile config	Defines plant-specific thresholds; the runtime study uses the succulent profile.
Analysis	Monitor process	Applies threshold checks with five-reading confirmation, two forgiven in-range readings, and hysteresis.
Advice	Local model service	Uses `ollama-hailo` with `qwen2:1.5b`; saves JSON/TXT advice outputs after confirmed alerts.

4.4 Broader GreenWatch Modules

The larger GreenWatch design extends the environmental path with two additional sensing services. First, an AI-camera module on a Raspberry Pi with an AI HAT is intended to inspect a plant for visible stress, including bugs, wilting, discoloration, and blight. Second, an access-monitoring Arduino is intended to report greenhouse-door and motion events. Both services fit the same messaging model: local sensing produces structured events, MQTT transports them to the hub, and the hub can include them in operator-facing summaries.

The same architecture also supports database storage, dashboard views, and operator notifications. These services extend the environmental path into a fuller greenhouse operations platform while preserving the central systems contribution of this paper: working environmental telemetry, alert confirmation, and local advice generation.

5 Implementation

The implementation uses a two-process Python architecture that separates sensing and transport from monitoring and advice generation. The publisher process bridges Arduino serial telemetry to MQTT topics, while the monitor process consumes those topics, applies plant-profile thresholds, confirms alerts, and records local language-model advice. This separation keeps the sensing interface stable while allowing the monitoring policy and advice layer to evolve independently. Table 2 summarizes the implemented environmental path.

The Arduino firmware streams newline-delimited JSON over USB serial at 115200 baud. Each sampling period emits temperature, humidity, and light records with a shared sequence number and explicit units: Celsius for temperature, percent for humidity, and raw ADC for light. The firmware uses a DHT11 sensor for temperature and humidity and an LDR on analog input A0 for light. Because the LDR response is inverse, lower ADC values indicate brighter light

and higher ADC values indicate darker conditions. This explicit serial contract is the main reproduction interface for the sensing node.

The gateway runtime uses Debian with Python 3.12, HiveMQ as the MQTT broker path, `ollama-hailo` for local model serving, and `qwen2:1.5b` for advice generation. These choices support the paper's edge-AI framing: sensing, transport, monitoring, and language-model advice all run as a local greenhouse pipeline rather than as a cloud-only service.

The monitor loads plant thresholds from `plant_profiles.json`. The succulent profile is the formal case in the runtime study, with monitor-side thresholds of 10–35 °C for temperature, 15–40% for humidity, and 500–900 ADC for light. Alert records also store threshold values active at alert time, which preserves configuration history when thresholds are tuned. The evaluation therefore uses the threshold fields saved with each alert when reporting alert behavior.

For reproduction, the system is specified through its stable interfaces: the Arduino serial schema, MQTT topic structure, plant-profile configuration, alert-record fields, and advice-record format. These interfaces are the essential scientific content of the implementation because they make the sensing-to-advice path inspectable and portable across equivalent hardware.

6 Prototype Evaluation

This section reports an instrumented run of the GreenWatch environmental-monitoring path. The run exercises serial sensing, MQTT publication, plant-profile monitoring, alert confirmation, local advice generation, and runtime record storage. The evaluation addresses three questions: whether GreenWatch completes the environmental sensing-to-advice path, how confirmed alerts are produced from monitored sensor streams, and whether generated advice records preserve enough structure for later inspection.

6.1 Runtime Measurements

Table 3 summarizes the runtime results. The publisher log contains 1,281 sensor records across 427 complete cycles, with matched counts for temperature, humidity, and light. The observed temperature-record interval is 2.004 s on average, matching the 2-second sampling cadence configured in the Arduino firmware. The monitor log contains 197 out-of-range events and records confirmation and hysteresis behavior, showing that alerts are produced by sustained sensor conditions rather than single-sample threshold crossings.

6.2 Alert and Advice Behavior

The strongest alert result comes from the succulent run. All 15 confirmed alerts involve the light sensor: 12 `light_high` and 3 `light_low`. This result is consistent with the inverse LDR behavior: low ADC values correspond to bright conditions and high ADC values correspond to dark conditions. The runtime

Table 3. Runtime summary for the GreenWatch environmental sensing-to-advice path.

Runtime item	Value	Interpretation
Publisher log	1,440 rows; 1,281 sensor records	MQTT publication path ran across the recorded session.
Sensor cycles	427 complete cycles	Temperature, humidity, and light streams remained aligned.
Sensor interval	Mean 2.004 s for temperature records	Runtime cadence matches the Arduino 2-second sampling design.
Monitor log	1,352 rows; 197 out-of-range events	Threshold processing and confirmation logic executed.
Confirmed alerts	15 light alerts for succulent	Repeated light anomalies were confirmed and saved.
Advice files	13 JSON and 13 TXT files	Local advice generation ran and produced inspectable outputs.
Alert-to-advice timing	Median 9.98 s; range 8.64–37.58 s	Internal monitor trace for the recorded run.
Advice-quality notes	2 empty actions; 4 non-enum condition values; several temperature mentions	Directs future schema and prompt hardening.

records therefore show that GreenWatch can convert light-level telemetry into plant-profile alerts using a transparent confirmation policy.

The monitor produced 13 local advice records after confirmed alerts. Each JSON advice file contains the expected top-level structure, which confirms that the local language-model path was invoked and persisted in a machine-readable format. The advice content also reveals useful design guidance: two records contain empty action text, four include at least one non-enum condition value, and several summaries mention temperature even though the triggering alerts were light alerts. This finding clarifies the design boundary: deterministic rules remain responsible for alert detection, while local model output should be checked, templated, or reviewed before plant-care recommendations are shown to operators.

The runtime records include one timing observation for the local advice path. The internal monitor-log gap between the embedded alert timestamp and the advice-event timestamp has a median of 9.98 s and a range of 8.64–37.58 s across 13 advice events. This value characterizes the recorded run and is best read as an operational trace, not as a hard latency guarantee.

6.3 Evaluation Result

Results show that GreenWatch completed the environmental sensing-to-advice path: the system collected periodic sensor readings, published them through

MQTT, applied plant-profile monitoring, confirmed repeated light anomalies, saved 15 alert records, and generated 13 local advice files. The measured contribution is therefore a working greenhouse telemetry-to-advice pipeline with explicit data contracts and inspectable runtime records.

7 Discussion

GreenWatch demonstrates a practical design pattern for edge-AI CPS prototypes: keep sensing and transport simple, keep alert detection deterministic, and use local language models for explanation rather than for the safety-critical decision. This organization matters because each stage remains inspectable. The Arduino defines a stable serial interface, MQTT decouples the publisher from the monitor, plant profiles define the monitored operating envelope, and alert records preserve the state that led to each advice request.

The most important design strength is the separation between alert confirmation and advice generation. GreenWatch does not ask the language model to decide whether an abnormal condition exists. Numerical monitoring produces the alert first; the language model then converts the alert context into operator-facing text. This split makes the system easier to test, easier to audit, and easier to improve: threshold logic, prompt design, and advice validation can be advanced independently.

The runtime records also strengthen the systems contribution. Publisher logs, monitor logs, alert records, and advice files reconstruct the path from physical sensing to generated advice. For small IoT deployments, this logging discipline is also a method for making field prototypes scientifically inspectable. The same structure supports later extensions because camera events, door or motion events, dashboard records, and notification records can reuse the MQTT and logging pattern already exercised by the environmental path.

The measured run is centered on the environmental path. Camera-based plant inspection, access monitoring, database-backed dashboarding, and operator notification remain architecture modules for the next implementation stage. This framing keeps the paper focused on a well-defined sensing-to-advice pipeline that forms the core of the larger GreenWatch architecture.

8 Scope and Future Work

GreenWatch is evaluated here through the environmental sensing and advice path. This focus gives the study a clean systems boundary: serial sensing, MQTT transport, plant-profile monitoring, alert confirmation, and local language-model advice are exercised end to end. The same architecture leaves clear extension points for camera-based plant inspection, access monitoring, dashboard storage, and notification delivery.

The runtime study uses one plant profile and one instrumented run. This is appropriate for validating the pipeline structure and alert flow, while broader greenhouse claims require repeated runs across plant species, lighting conditions,

humidity ranges, and sensor placements. Future experiments should include calibrated measurements, fixed test scenarios for each sensor, and hardware-in-the-loop runs that combine environmental, visual, and access events.

The advice layer is already separated from alert detection, which is a strength for future validation. Threshold checks remain deterministic, and generated advice is stored for review. Future versions can compare local models, refine prompts, enforce stricter output schemas, add sensor-specific templates, and include expert review before advice is presented as plant-care guidance. Operational hardening should add authenticated MQTT, encrypted transport, broker health checks, and CPU, memory, power, and network traces.

9 Conclusion

This paper presented GreenWatch, an edge-AI CPS prototype for greenhouse monitoring. The implemented environmental path connects Arduino sensing, MQTT publication, plant-profile monitoring, confirmed light alerts, and local language-model advice records. In the succulent run, GreenWatch produced 1,281 sensor records, 1,352 monitor records, 15 confirmed light alerts, and 13 saved advice records. These results support a focused systems contribution: GreenWatch turns greenhouse telemetry into inspectable alerts and operator-facing advice through a modular edge pipeline. The next stage is to extend the same architecture to camera-based plant inspection, monitoring, dashboarding, notifications, repeated plant-profile studies, and stronger advice validation.

References

1. Jin, W., Xu, R., Lim, S.: Dynamic inference approach based on rules engine in intelligent edge computing for building environment control. Sensors **21**(2), 630 (2021). https://doi.org/10.3390/s21020630
2. Navarro, E.d.M., Costa, N., Pereira, Á.: A systematic review of iot solutions for smart farming. Sensors **20**(15), 4231 (2020). https://doi.org/10.3390/s20154231
3. Sahmi, I., Abdellaoui, A., Mazri, T.: Mqtt-present: Approach to secure internet of things applications using mqtt protocol. International Journal of Electrical and Computer Engineering **11**(5), 4577–4586 (2021). https://doi.org/10.11591/ijece.v11i5.pp4577-4586
4. Shanto, S.S., Rahman, M., Oasik, J.M.: Smart greenhouse monitoring system using blynk iot app. Journal of Engineering Research and Reports **25**(2), 94–107 (2023). https://doi.org/10.9734/jerr/2023/v25i2883
5. Simó, A., Dziţac, S., Badea, G.E.: Smart agriculture: Iot-based greenhouse monitoring system. International Journal of Computers Communications & Control **17**(6) (2022). https://doi.org/10.15837/ijccc.2022.6.5039
6. Tareq, R.W., Khaleel, T.A.: Implementation of mqtt protocol in health care based on iot systems: A study. International Journal of Electrical and Computer Engineering Systems **12**(4), 215–223 (2021). https://doi.org/10.32985/ijeces.12.4.5
7. Zheng, Y., Chen, Y., Qian, B.: A review on edge large language models: Design, execution, and applications. ACM Computing Surveys **57**(8), 1–35 (2025). https://doi.org/10.1145/3719664