AN IoT-BASED FLOOD ALERT SYSTEM UTILIZING INTELLIGENT CACHING AND LIGHTWEIGHT AI ON A MICROCONTROLLER UNIT

Vu Van Thanh*, Huynh Viet Thang*

The University of Danang – University of Science and Technology, Vietnam

*Corresponding author: vvthanh@dut.udn.vn; thanghv@dut.udn.vn

(Received: April 29, 2025; Revised: June 15, 2025; Accepted: June 18, 2025)

DOI: 10.31130/ud-jst.2025.23(9D).552E

Abstract - In flood-prone areas, the connection infrastructure is often poor, hindering real-time data transmission in IoT early warning systems. This study proposes a smart store-forward and data prioritization mechanism, implemented on the ESP32 microcontroller combined with SIM7600CE to ensure stable information transmission via the MQTT protocol. The system uses a lightweight artificial intelligence model (TinyML), compressed by quantization techniques, to help sensor nodes classify data into three warning levels: Normal, Warning and Danger. Based on the priority level, the device decides whether to store the data locally or forward it to the server, contributing to reducing network congestion and saving energy. Actual testing shows that the system achieves a 100% data transmission rate, reduces memory usage by 25%, and significantly improves warning performance compared to the traditional method. This solution is suitable for hard-to-reach areas, supporting the improvement of disaster response capacity.

Key words – Flooding; artificial intelligence; embedded device; Edge Impulse; SIM7600CE; early alert; early warning

1. Introduction

Flooding poses increasing threats globally, especially in developing nations like Vietnam. IoT-based early warning systems are effective in mitigating flood damage. However, their implementation often faces poor telecommunications infrastructure in flood-prone regions, causing unstable real-time data transmission and lowering alert system effectiveness.

Additionally, limitations in energy, memory, and processing power in embedded devices make it challenging to continuously collect, process, and transmit sensor data. Existing solutions often fail to address data loss, network congestion, and energy inefficiency. While prior studies have explored flood monitoring using satellite imagery, LoRa communication, or edge intelligence, none focus on integrating a priority-based store-and-forward mechanism with optimized TinyML models on embedded systems for flood alerts.

In addition, the limitations of energy, memory and processing capacity of embedded devices are also major challenges for continuous processing and transmission of sensor data. Traditional solutions have not completely solved the problem of data loss, network congestion or unnecessary energy consumption. There have been many domestic and foreign studies focusing on handling early flood warnings, but there is no research on the smart storage-forwarding mechanism and data prioritization integrated in embedded devices that combine the optimized TinyAI model. In [1], the authors used Sentinel-1 satellite image data combined with machine learning algorithms to monitor and map flooded areas in the Mekong Delta. The study did not pay attention to the issues of data storage, transmission and hazard warning performance. The work in [2] presents an IoT application based on LoRa technology to monitor and manage rural facilities, which may be related to flood monitoring. The novelty is the use of LoRa to expand the scope, not dependent on existing telecommunications infrastructure, but the transmission speed is slow and there is no solution to solve the problem of data loss. A hybrid deep learning model combined with edge intelligence was proposed in [3] to predict the flood process, helping to improve early warning and risk management capabilities. The authors of this study also did not address the issue of data transmission and storage performance. Research in [4] proposed a "Smart-Edge-CoCaCo" architecture that combines intelligent computing, storage and communication at the edge of the network in a heterogeneous IoT system. This architecture uses AI to optimize data processing, storage and communication, to reduce latency and energy consumption. Research in [5] presented a model-based approach to support AI development on IoT edge devices through TinyML. This approach helps design and deploy machine learning models on embedded devices with limited resources, improving system maintainability and scalability. However, the research results are general and not applicable in the specific field of flood or inundation warning.

In this context, this paper proposes an intelligent store-forward and data prioritization mechanism that optimizes the processing, storage, and transmission of warning information in unstable network conditions. This mechanism combines a lightweight AI model deployed directly on the ESP32 microcontroller, allowing on-site classification of flood levels and selection of data transmission or storage actions based on priority, thereby improving the reliability and performance of the warning system.

2. System Architecture

The proposed system includes the following components: water level sensor and rainfall sensor, microcontroller unit (MCU) running the optimized TinyML model; local memory with micro SD card, 4G network communication via SIM7600CE module, data transmission to the cloud via MQTT protocol in real time. Figure 1 describes the detailed block diagram of the

proposed system model, the system is built on the ESP32-WROOM-32 microcontroller (dual-core Xtensa LX6, 240 MHz, 4 MB flash, 520 KB SRAM) [5], integrating the Gamicos GLT500 sensor [10] to measure water level (range 1m - 200m) and the ES-RAINF-01 rain sensor [11] to measure rainfall intensity (sensitivity 0.1 mm). Each IoT node is equipped with a SIM7600CE module, supporting LTE Cat-1 bands (B1/B3/B5/B8/B20), allowing data transmission at a maximum speed of 10 Mbps and wide coverage, even in remote areas [7].

To ensure sustainability, the system uses a 30W solar panel and 6 3.7V 2000mAh LiPo rechargeable batteries connected to 7.4V 4000mAh, with a TP4056 charging circuit to regulate the current (maximum current 1A). An LM2596 voltage regulator stabilizes the output voltage at 3.3V for the ESP32 and 4.2V for the SIM7600CE. The ESP32 is programmed with ESP-IDF, using FreeRTOS to manage multitasking (sensor reading, data transmission, AI processing).

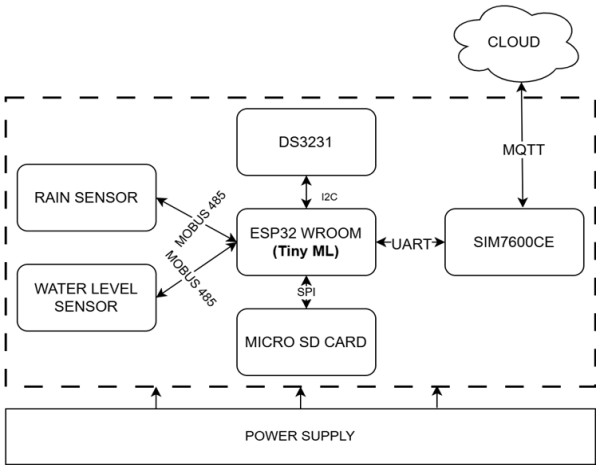

Figure 1. *Block diagram of the proposed system*

Each sensor node operates independently with the following procedures:

- Reading sensor data periodically;

- Assessing importance using the TinyML model;

- If important: send immediately if there is a network, or save and mark it as a priority;

- If not important: save temporarily to transmit when there is a network.

Sensor data is stored locally on a 4GB SD card (about 1 million records, each record is 32 bytes) when the 4G connection is lost, using the SPI interface of the ESP32. The data forwarding algorithm prioritizes data based on the warning level: Danger is sent immediately, Warning is sent next when bandwidth allows, and Normal is stored or sent when there is no more priority data. The data is sent via the SIM7600 via the MQTT protocol to the Mosquitto broker on the server. The MQTT data packet is structured in 12 bytes: 2 bytes for the node ID, 2 bytes for the water level, 2 bytes for the rainfall intensity, 2 bytes for the warning level (0: Normal, 1: Warning, 2: Danger), and 4 bytes of timestamp. When there is a 4G connection, the data is synchronized with the server.

3. Deployment of lightweight AI model on ESP32

The Tiny AI model is developed based on the MLP (Multi-Layer Perceptron) neural network with a structure of 3 hidden layers (64, 32, 16 neurons) [6], using the ReLU activation function and softmax output to classify data into three warning levels:

- Normal (water level ≤ 7m, rain ≤ 30mm/h),

- Warning (7m < water level ≤ 10m or 30mm/h < rain ≤ 50mm/h),

- Danger (water level > 10m or rain > 50mm/h).

The Tiny AI model is deployed on ESP32 via the https://studio.edgeimpulse.com/ platform. Sensor data is uploaded to Edge Impulse Studio, where the model is trained and optimized with integrated tools such as EON Compiler. After training, the model is exported as a C++ library and integrated into the ESP32 code via Arduino IDE, ensuring optimal computational performance.

The dataset consists of 4,300 samples from sensors (water level, rainfall intensity) collected at Dong Tam hydrological station on Gianh River, Tuyen Hoa district, Quang Binh province in 6 months (from September 2024 to February 2025) with a sampling period of 1 hour, with a split ratio of 83% training and 17% testing. The training process uses a batch size of 32, a learning rate of 0.001, and is optimized using the Adam algorithm in 100 epochs, achieving an accuracy of 96.3% on the training set and 95.09% on the testing set.

To optimize model size and performance, a compression technique is applied in the training process, which is the quantization technique. Quantization is an optimization technique that reduces model size and improves computational efficiency when deploying machine learning models on resource-constrained embedded devices, such as the ESP32. This method converts the model's weights and activations from 32-bit floating-point format (float32) to 8-bit integer format (int8), thereby reducing memory size and speeding up inference. The quantization process consists of three main steps: determining the range of weights, calculating the scaling factor, quantizing and de-quantizing [8], [9].

3.1. Lightweight AI model training

Figure 2 is a graph showing the training process of the lightweight AI model on ESP32. During the training of the lightweight deep learning model (TinyML), the loss and accuracy values on both the training set and the validation set show a clear convergence trend, reflecting the good learning ability of the model. The training graph (Figure 2) shows that the loss value on the training set decreases steadily from 2.2466 to about 0.3176 after 100 epochs, while the validation loss also drops sharply from 1.3910 to 0.3355. The small difference between the training loss and validation loss in the last epochs (≈ 0.02) shows that the model is not overfitting and has good generalization ability.

In terms of accuracy, the model achieved a significant increase from 26.25% in the first epoch to 95.92% on the training set, and from 23.72% to 97.21% on the test set.

The highest validation accuracy value was recorded at epoch 97, with an accuracy of 97.21%, while the training accuracy at this point was 93.35%, reflecting the stable performance and effective learning ability of the model.

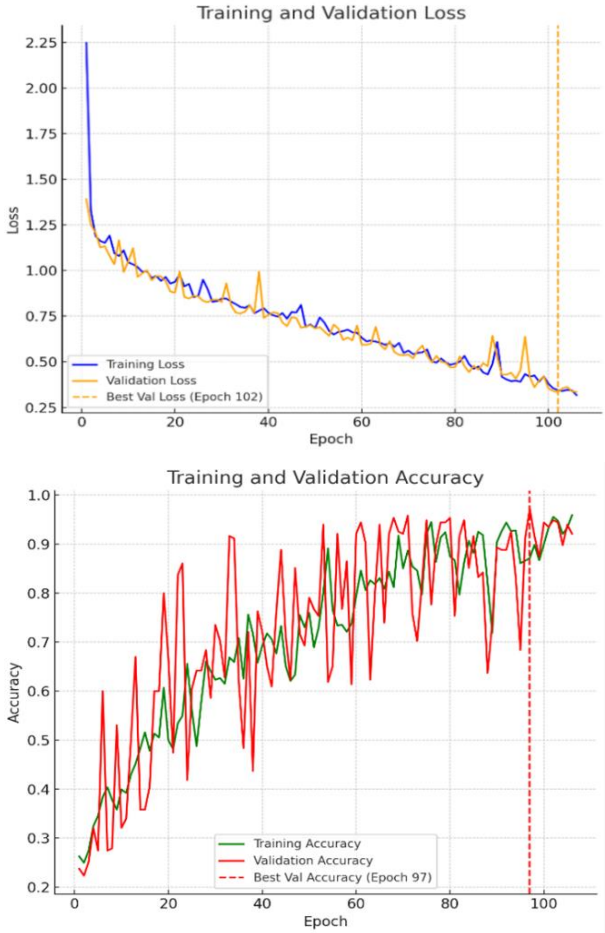

Figure 2. *Training of a lightweight AI model on ESP32*

Notably, the accuracy and loss indices did not fluctuate strongly in the last epochs, indicating that the model had reached the convergence threshold. This also suggests that further training is not necessary as it will not significantly improve performance and may lead to overfitting.

Thus, the trained TinyML model not only achieves very high accuracy on both datasets but also ensures stability and generalization ability, allowing efficient deployment on resource-limited embedded devices such as ESP32.

3.2. Performance of Lightweight AI Model

The model after being quantized to int8 format - suitable for resource-constrained microcontrollers such as the ESP32 - still maintains very high performance on the validation set, demonstrating its robustness to real-world deployment.

The overall results show that the model achieves an accuracy of 96.3% and a loss of 0.32, only slightly lower than the original unquantized version. This proves that the quantization process does not cause significant performance degradation, while saving memory and speeding up inference on embedded hardware.

Table 1. *Confusion matrix*

Confusion matrix (validation set)	danger	normal	warning
Danger	**98.3%**	0%	1.7%
Normal	0%	**93.3%**	6.7%
Warning	0%	2.5%	**97.5%**
F1-score	0.99	0.95	0.95

Table 2. *Cycling power estimation results*

Metric	Value
Area under ROC Curve	1.00
Weighted average Precision	0.96
Weighted average Recall	0.96
Weighted average F1 score	0.96

Table 1 is the confusion matrix showing that the model very accurately classifies the three classes “Danger”, “Normal” and “Warning”. The confusion matrix shows that the model very accurate in the classification of the three classes “Danger”, “Normal” and “Warning”, detailed as follows:

- The Danger class is correctly classified at a rate of 98.3%, with only 1.7% being mistaken for “Warning”.

- The Normal class achieved 93.3% accuracy, with 6.7% being mistaken for “Warning”.

- The Warning class achieved 97.5% accuracy, with 2.5% being mistaken for “Normal”.

Also, the F1-scores are very high: 0.99 for the “Danger” class and 0.95 for the other two classes. This shows that the model is not only accurate but also has a good balance between recall and precision for all classes.

Table 2 summarises the metrics used in this study. The results in Table 2 can be explained as follows:

- Area Under Curve (AUC) = 1.00 is the area under the Receiver Operating Characteristic (ROC) curve, indicating the model’s ability to distinguish between classes. The value of 1.00. Is the maximum value, meaning that the model distinguishes perfectly between classes. All positive points are correctly ranked higher than negative points. This is a very ideal result, rarely achieved in real models. This may be due to clear and less noisy training data, or the model has been very well optimized.

- Weighted Average Precision = 0.96 is the ratio of the number of correctly predicted samples to the total number of correctly predicted samples. High Precision (0.96). The model rarely predicts false positives. For each warning given (for example “Danger” or “Warning”), it is almost certainly correct.

- Weighted Average Recall = 0.96 is the ratio of the number of correctly predicted samples to the total number of actually correct samples. High Recall (0.96). The model rarely misses false positives. That is, almost all true cases of “Danger”, “Warning”, “Normal” are correctly identified.

- Weighted Average F1 Score = 0.96, F1 is the average of Precision and Recall. F1 is high (0.96). Showing that the model achieves a near-perfect balance between precision and coverage.

The post-quantization classification model maintains very high accuracy, with almost no performance loss compared to the float32 model. The Precision, Recall and F1 indices are all close to the maximum, showing that the model is suitable for early warning applications in IoT systems, where errors in warnings are unacceptable.

4. Experimental setup

4.1. Experimental lightweight AI model

Table 3 presents statistics for the lightweight AI model deployed on Edge Impulse tool, clearly showing the difference between two model versions before and after applying compression technique. Applying quantization helps reducing RAM usage by 11.1%, FLASH usage by 19.7% and inference latency by 50%. Although the accuracy is slightly reduced (about 0.89%), the resource saving effect is significant, allowing easy deployment on memory-limited devices such as ESP32 (with 4 MB Flash and 520 KB RAM). This is especially suitable for embedded applications such as IoT-based flood warning systems.

Table 3. *Statistics for lightweight AI model on ESP32*

Parameter	Float32 (Unoptimized)	Int8 (Quantized)
Latency	2 ms	1 ms
RAM	1.8 KB	1.6 KB
FLASH	22.8 KB	18.3 KB
Accuracy	95.98%	95.09%

With high accuracy, low error and practical deployment after quantization, the TinyML model has proven to be effective and suitable for embedded applications in the field of IoT disaster warning. This is a major step towards a lightweight, intelligent and low-cost early warning system.

4.2. Real-world sensor node experimental setup

We deployed two pilot sensor nodes in March 2025, forming a two-node network, operating completely independently and powered by a combination of solar panels and lithium batteries. Each node is built on the ESP32 platform, integrating a SIM7600 module (for 4G connectivity), water level and rainfall sensors, and an RTC real-time clock to mark the sampling time.

The sensor nodes are configured to collect data periodically every 30 minutes and process locally to classify the warning level (Normal, Warning, Danger) through an embedded machine learning model (TinyML) deployed with Edge Impulse Studio, then make a decision to transmit or store data depending on the 4G network status. 1 station operates with integrated caching + forwarding mechanism to ensure that data is not lost in case of temporary connection loss, 1 station operates with traditional non-caching mechanism. We use the following criteria to evaluate system performance:

- Data transmission success rate: number of data packets successfully sent to the MQTT server compared to the total number of packets generated.

- Memory usage: RAM and flash capacity used for programs, machine learning models and buffer data (in KB).

- Energy consumption: Average power consumed in an operating cycle (mW).

- Alarm latency: Time from alarm detection to successful MQTT packet sending (seconds).

5. System performance evaluation

Evaluation results are aggregated from both nodes after more than 1,400 sampling cycles: one node running with *"no caching"* strategy, and one node running with *"caching and forwarding"* strategy.

5.1. Data transmission success rate

Table 4 summarizes the data transmission success rate. The system achieves a high transmission success rate even when the network is very weak. If warning level is "Danger", all the important data is transmitted. This improvement comes from the use of MQTT protocol along with *Caching + Forwarding* strategy, which helps increase the ability to resist packet loss in areas with unstable 4G coverage as shown in Table 4.

Table 4. *Performance evaluation*

Signal level	RSSI value	Signal estimated (dBm)	Quality rating	No Caching	Caching + Forwarding
Very weak	0–9	-113 to -95	Insufficient data transmission	10%	36%
Weak	10–14	-93 to -85	Connection intermittent	30%	64%
Average	15–19	-83 to -75	Transmission possible but unstable	50%	84%
Good	20–25	-73 to -63	Connection stable	80%	98%
Very good	26–31	-61 to greater than -51	Connection very stable	100%	100%

5.2. Memory usage

Memory usage is reduced by 25%, thanks to the optimization of the control program with FreeRTOS, using a compressed binary data format (32 bytes per packet) and quantizing the machine learning model (int8), in which, the flash part is mainly used to store the model and data processing program, while RAM is for data queues and SPI buffers.

5.3. Power consumption

Table 5 presents the power consumption for one sensor node. As can be seen, the average power consumption per node is higher than the LoRa study, due to the use of the SIM7600 which has a higher power consumption than the LoRa technology. However, power saving techniques including turning off sensors when not needed, using the ESP32 deep sleep mode, and only activating the SIM7600 module when data transmission is needed have improved the system's power consumption.

Table 5. *Power consumption of sensor node*

Mode	Time (s)	Current (mA)	Power (mW)	Energy (mWh)
ESP32 + SIM7600 + AI	5	300	1260	1.75
Deep Sleep	1795	1	4.2	2.09
Total per 30 minutes	1800	-	-	3.84

From Table 5, we can estimate the average energy consumption per hour (corresponding to 2 sending times) as follows: 3.84 mWh × 2 = 7.68 mWh per hour. The estimated number of days of operation of the sensor node when using a Li-Ion battery with a nominal voltage of 7.4V–4000mAh is about 160 days.

5.4. Alert delay

For the "Danger" level alerts, the average latency from detection to successful alert delivery is 0.2 seconds. This outperforms the LoRa system (around 1.3 seconds) [12] due to the high transmission speed of the 4G network and the protocol used for data transmission and reception is MQTT. This is an important advantage when deploying the system in an area where latency can determine the effectiveness of the response.

5.5. TinyML performance on ESP32 microcontroller

ESP32 performs alert classification based on water level and rainfall data using a 3-layer quantized machine learning model. The average inference time is ~20ms, but the power consumption increases slightly (~10%) in areas with weak 4G signal because the SIM7600 module needs more power to maintain the connection. This shows the need for a flexible adjustment mechanism according to network conditions.

6. Conclusion

In conclusion, the experimental results show that the proposed system meets the requirements of reliability, latency, and energy efficiency well in practical deployment conditions. The integration of edge machine learning models on ESP32 and the flexible use of 4G networks allow the system to adapt to harsh environmental conditions while maintaining high performance. Compared with the traditional flood warning solution, our system shows obvious improvements in accuracy, response speed, and resource efficiency.

REFERENCES

[1] L.C. Nguyen. S. Niculescu and S. Bengoufa. "Monitoring and Mapping Floods and Floodable Areas in the Mekong Delta (Vietnam) Using Time-Series Sentinel-1 Images, Convolutional Neural Network, Multi-Layer Perceptron, and Random Forest*", Remote Sensing for the Study of the Changes in Wetlands,* vol.15, no.8, pp.2001, 2023. https://doi.org/10.3390/rs15082001.

[2] A. Fainerman *et al*., "An IoT application based on LoRa for monitoring and managing a rural establishment", *in Proc. 2024 Latin American Computer Conference (CLEI).* Buenos Aires, Argentina, 2024, pp. 1–5. doi: 10.1109/CLEI64178.2024.10700563.

[3] C. Chen, J. Jiang, Y. Zhou, N. Lv, X. Liang, and S. Wan, "An edge intelligence empowered flooding process prediction using Internet of things in smart city*", Journal of Parallel and Distributed Computing,* vol. 165, pp. 66–78, 2022. [Online]. Available: https://doi.org/10.1016/j.jpdc.2022.03.010.

[4] Y. Hao, Y. Miao, L. Hu, M. S. Hossain, G. Muhammad, and S. U. Amin, "Smart-Edge-CoCaCo: AI-Enabled Smart Edge with Joint Computation, Caching, and Communication in Heterogeneous IoT", *IEEE Network*, vol. 33, no. 2, pp. 58–64, Mar./Apr. 2019, doi: 10.1109/MNET.2019.1800235.

[5] Espressif Systems, "ESP32 Technical Reference Manual", *Espressif.com,* 2023, [online] Available https://www.espressif.com/en/products/socs/esp32/resources, [Accessed April 08, 2025].

[6] P. Warden and D. Situnayake, *"TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers"*, First Edition, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2019.

[7] SIMCom, "LTE Modules SIM7600E & SIM7600E-H Passed the GCF Certification", *Simcom.com,* Retrieved December 12, 2022, [Online] Available: https://www.simcom.com/product/SIM7600X.html [Accessed July 03, 2024].

[8] Edge Impulse. "Optimizing models using quantization", *Docs.edgeimpulse.com,* 2024, [online] Available https://docs.edgeimpulse.com, Accessed April 08, 2025].

[9] TensorFlow Lite, "Post-training quantization", *Tensorflow.org,* 2024, [online] Available https://www.tensorflow.org/lite/performance/post_training_quantization, Accessed April 08, 2025]

[10] Gamicos, "Pressure-Level-Sensor GLT500", G*amicos.com*, 2025, [online] Available https://www.gamicos.com/Products/GLT500-Pressure-Level-Sensor, Accessed April 08, 2025].

[11] Epcb, "rainfall sensor", *Epcb.vn,* 2023, [online] Available https://epcb.vn/products/cam-bien-do-luu-luong-mua-rainfall-es-rainf-01-rs485-modbus-rtu, [Accessed April 01, 2025].

[12] Semtech, "LoRa Calculator", *Semtech.com*, 2025, [online] Available https://www.semtech.com/design-support/lora-calculator [Accessed April 08, 2025].