การประชุมวิชาการทางวิศวกรรมไฟฟ้า ครั้งที่ 48
The 48th Electrical Engineering Conference (EECON-48)
วันที่ 19-21 พฤศจิกายน 2568 ณ โรงแรมฟูราม่า จังหวัดเชียงใหม่

A Lightweight Object-Oriented MicroPython Firmware for LINE Beacon Broadcasting on ESP32

Supakit Nakpomchin[1], Pichaya Sookplung[1], Somkiat Chormuan[2] and Suphitcha Chanrueang[2*]

[1]Department of Information Technology, Faculty of Science and Technology, Nakhon Pathom Rajabhat University, Thailand, supakrid@webmail.npru.ac.th pichaya@webmail.npru.ac.th

[2]Department of Software Engineering, Faculty of Science and Technology, Nakhon Pathom Rajabhat University, Thailand, tko@webmail.npru.ac.th suphitcha@webmail.npru.ac.th*

Abstract

This paper presents a lightweight, object-oriented MicroPython firmware framework for broadcasting LINE Beacon packets on ESP32. The framework enhances modularity, reusability, and maintainability compared with conventional C/C++ implementations, which are often complex and time-consuming. The design encapsulates BLE advertising management and LINE Beacon payload generation into two reusable classes—BLEManager and LineBeacon—ensuring protocol compliance with the 3-AD format. Experimental validation confirmed stable operation, with advertising intervals consistently maintained at ~63 ms, packet accuracy above 97%, and RSSI values around −60 to −65 dBm at distances of 5–10 m. Memory analysis showed a modest overhead of 10–11 KB, acceptable for ESP32-based IoT deployments. These results demonstrate that the proposed firmware framework offers a practical and efficient foundation for proximity-based services, balancing ease of development with reliable performance.

Keywords: MicroPython, ESP32, Bluetooth Low Energy (BLE), Firmware Framework, LINE Beacon, IoT

1. Introduction

The Internet of Things (IoT) has transformed embedded systems by enabling real-time sensing, actuation, and communication in domains such as healthcare, smart cities, and industrial automation [1,2]. The ESP32 microcontroller has emerged as a popular platform for IoT prototyping due to its integrated Wi-Fi and Bluetooth Low Energy (BLE) modules, dual-core processor, and cost efficiency [3]. Traditionally, firmware for such platforms is developed in C/C++, which—while performant—often results in longer development cycles, limited modularity, and a steep learning curve.

MicroPython, a lightweight implementation of Python for microcontrollers, offers a more accessible and iterative development environment [4,5]. When combined with object-oriented programming (OOP) principles, it enables reusable, scalable, and maintainable firmware designs [6]. However, few existing works leverage MicroPython and OOP to create modular BLE broadcasting frameworks for real-world applications.

Meanwhile, LINE Beacon, a BLE-based proximity protocol by LINE Corporation, has gained adoption in Southeast Asia for location-aware services in retail, transit, and tourism [7,8]. Despite its potential, there is a lack of lightweight, open-source firmware specifically tailored to broadcast LINE Beacon packets on low-cost microcontrollers using high-level languages.

This work addresses that gap by presenting a lightweight, object-oriented MicroPython firmware framework for LINE Beacon broadcasting on the ESP32. The framework encapsulates BLE operations and payload generation into two reusable classes, BLEManager and LineBeacon, ensuring protocol compliance, maintainability, and rapid prototyping. Experimental validation confirms stable signal performance, >97% packet accuracy, and modest resource usage, making the proposed solution well-suited for scalable, proximity based IoT deployments.

2. Literature Review

Firmware development for embedded systems has evolved substantially over the last decade. Early approaches predominantly relied on low-level programming in C/C++, which provided performance and control but lacked modularity and maintainability, particularly in large-scale or fast-evolving IoT deployments [9]

Between 2018 and 2020, developers and researchers began exploring high-level alternatives that could improve productivity without severely compromising performance. MicroPython emerged during this period as a lightweight implementation of Python tailored for microcontrollers such as the ESP32. Its rapid prototyping capability, simplified syntax, and REPL-based interaction proved particularly attractive in educational, research, and low-power IoT applications [9,10]. However, most implementations remained procedural and lacked long-term scalability. Comparative studies by Plauska et al. [1] and Ionescu et al. highlighted these limitations.

From 2020 onward, attention shifted toward embedding object-oriented programming (OOP) principles into firmware development, even on constrained hardware. Researchers such as Maclean et al. [11] and Avellaneda et al. [12] emphasized the advantages of modular firmware design based on encapsulation and reusability, suggesting that such patterns are well-suited for the evolving requirements of embedded IoT systems.

In parallel, Bluetooth Low Energy (BLE) gained traction as the de facto wireless protocol for low-power, short-range communication. By 2021, BLE was widely adopted in smart devices for applications such as indoor positioning, proximity services, and occupancy detection

*Corresponding Author

[13]. With the rollout of BLE 5.0, improvements in range and stability made it even more practical for real-world environments. Around this time, LINE Corporation introduced LINE Beacon, a BLE-based broadcast protocol designed for proximity interactions in retail, transportation, and tourism. While its application potential was clear, few firmware-level frameworks were developed to support LINE Beacon broadcasting, particularly on microcontrollers [14].

From 2022 to the present, developers have continued to adopt the ESP32 as a BLE-capable microcontroller due to its affordability and built-in connectivity. However, BLE implementation practices are still mostly C/C++ based via the ESP-IDF or Arduino toolchains, which can be difficult to scale or modify. MicroPython support for BLE has improved but remains fragmented, with few reusable frameworks and little attention given to standardized broadcasting protocols like LINE Beacon [7,15]. Raihan et al. [16] demonstrated BLE-based presence monitoring on the ESP32, further highlighting its reliability, but stopped short of adopting object-oriented MicroPython design.

To date, no unified approach has been proposed that combines MicroPython, OOP firmware design, and LINE Beacon broadcasting on the ESP32, despite their proven individual value. This work aims to fill that gap by presenting a firmware design that unifies these three elements. The goal is to deliver a practical, reusable, and protocol-compliant firmware foundation that supports LINE Beacon broadcasting on the ESP32, developed entirely using high-level, object-oriented MicroPython.

3. Materials and Methods

3.1 The firmware architecture

The firmware architecture was implemented using MicroPython on the ESP32 platform, and it follows a structured engineering workflow with four stages. As shown in Fig 1, the system is organized into the ESP32 hardware layer, the MicroPython runtime layer, the object-oriented design layer, and key implementation features. The workflow comprises: (1) requirements analysis, (2) object-oriented design, (3) firmware implementation, and (4) testing and validation. Specifications for BLE advertising and LINE Beacon payloads were derived from the official documentation [7]. Two core classes were designed, BLEManager and LineBeacon, to ensure reusability and maintainability in accordance with OOP principles [6]. Development used Thonny IDE, with bytecode flashed via esptool.py [8]. MicroPython was chosen for its simple syntax, interactive environment, and low memory footprint [4,5]. Validation employed nRF Connect in a controlled indoor environment. The ESP32 platform was selected for its integrated BLE module, dual-core architecture, and low power consumption, which suit IoT prototyping and deployment [3].

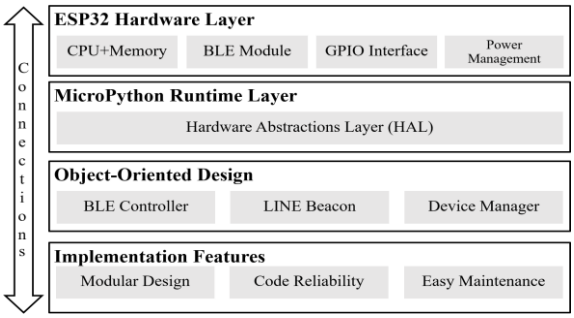

Fig 1. Overall architecture of the proposed firmware, showing the interaction among the ESP32 hardware layer, the MicroPython runtime layer, the object-oriented components, and the implementation features.

3.2 The process of a LINE Beacon device

A LINE Beacon device broadcasts BLE advertising packets to nearby smartphones running the LINE application. Each packet carries a UUID, Major, and Minor values that uniquely identify the beacon. The end-to-end interaction, from beacon broadcast to content delivery, is illustrated in Fig 2: the LINE app detects and decodes the packet, forwards identifiers to LINE's backend, retrieves mapped content, and displays it to the user. This pipeline enables proximity-aware services such as push notifications, promotional campaigns, and contextual messages.

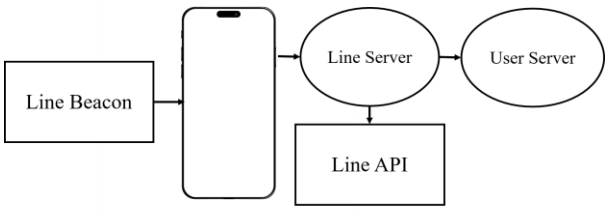

Fig 2. Communication process of the LINE Beacon system, from BLE broadcast and mobile decoding to backend mapping and user-side presentation.

3.3 LINE Beacon Specifications

The LINE Beacon framework follows the BLE broadcasting standard. A typical advertising packet includes a UUID (128-bit), Major and Minor (16-bit each), and Tx Power for proximity estimation. These fields are encoded using the 3-AD structure. The packet layout used in this work is depicted in Fig 3, which aligns with BLE advertisement conventions and ensures compatibility with the LINE application [7,9]. The three advertisement data elements are: AD1, the prefix and company identifier [9]; AD2, the beacon data payload containing UUID, Major, Minor, and Tx Power; and AD3, optional service data for extended functionality.

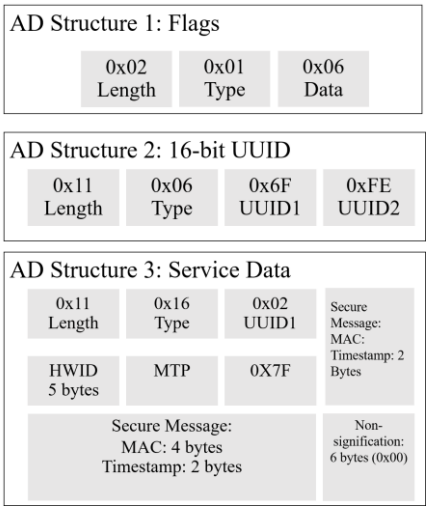

Fig 3. Structure of a LINE Beacon advertising packet in 3-AD format, indicating the placement of UUID, Major, Minor, and Tx Power for proper parsing by the LINE application

3.4 Designing Classes for Object-Oriented Firmware

To promote modularity and reuse, the firmware design separates concerns into two classes. The BLEManager class abstracts BLE initialization, advertising interval configuration, and start/stop control, which decouples low-level BLE operations from higher-level logic. The LineBeacon class constructs protocol-compliant advertising payloads, managing UUID, Major, Minor, and Tx Power with setter methods for dynamic updates. The separation of responsibilities is shown in Fig 4, while the interaction for packet handoff from LineBeacon to BLEManager prior to broadcasting is shown in Fig 5. This organization follows encapsulation and separation of concerns, enabling cleaner interfaces and easier maintenance [10].

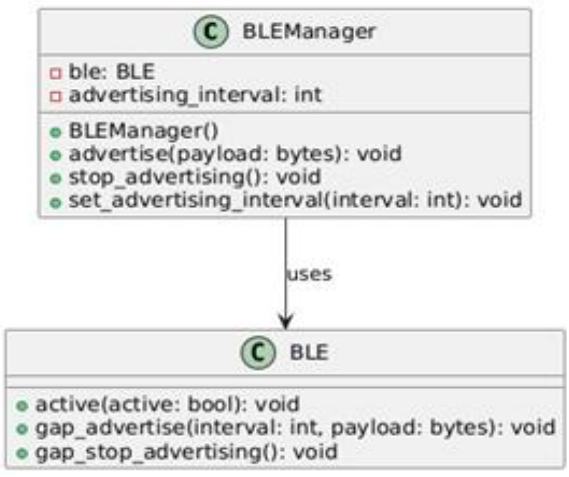

Fig 4. Class Structure: BLEManager and LineBeacon

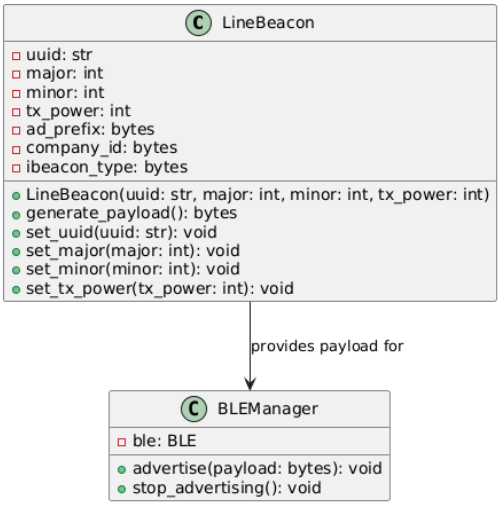

Fig 5. Class Diagram of LineBeacon Providing Advertising

3.5 Implementing the Firmware Using Thonny IDE

Development was performed in Thonny IDE, which provides syntax highlighting, auto-completion, and an integrated REPL for rapid iteration. The practical workflow, summarized in Fig 6, involves writing code in Thonny, uploading it to the ESP32, and verifying operation via the serial console and BLE scanning tools. Firmware images were flashed using esptool.py over UART [8]. This setup reduces configuration overhead compared with traditional toolchains and is well suited for educational and research environments [4,5].

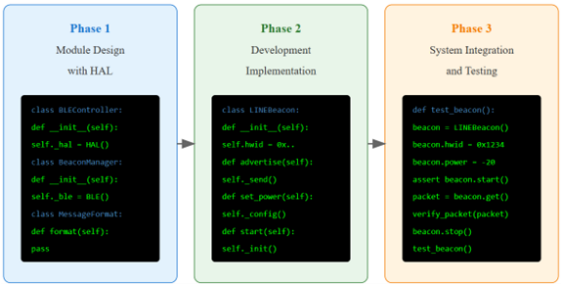

Fig 6. Development workflow using Thonny IDE, from coding and flashing with esptool.py to iterative validation with REPL and BLE analysis.

3.6 Firmware Testing and Validation Using nRF Connect

Testing with nRF Connect confirmed protocol compliance and correct packet structure, including the presence of UUID, Major, Minor, and Tx Power fields. In a controlled indoor environment, RSSI values were consistently between −60 dBm and −65 dBm at distances of 5–10 meters, and the configured advertising interval was maintained at approximately 63 ms. These results

indicate stable transmission behavior suitable for small-to-medium-scale IoT deployments.

To provide a clearer view of the experimental setup, Fig 7 presents the prototype system used in this study. The setup consists of two ESP32 development boards running the proposed MicroPython firmware and a host computer configured with Thonny IDE for code development, flashing, and real-time debugging. The system was validated using nRF Connect to scan and analyze BLE packets.

In testing multiple ESP32 boards simultaneously, each board was assigned to a different USB port and configured in Thonny IDE or via esptool.py. Separate sessions were opened to upload, run, and monitor the code in parallel. This approach enabled efficient comparison of results and verification of BLE packets across boards.

Fig 7. Prototype system setup for firmware testing and validation, showing two ESP32 development boards connected via USB to a host computer.

4. Results and Discussion

4.1 Packet Structure Validation

The proposed firmware successfully generated BLE advertising packets that fully complied with the LINE Beacon 3-AD format. Inspection using nRF Connect verified that all required fields—UUID, Major, Minor, and Tx Power—were present and correctly encoded. No malformed packets were detected during the testing period. This confirms that the object-oriented MicroPython implementation is capable of meeting strict protocol requirements, even on resource-constrained hardware such as the ESP32.

4.2 Signal Stability and Accuracy

RSSI measurements recorded in a controlled indoor environment showed stable values between −60 dBm and −65 dBm at distances of 5 to 10 meters. These results are consistent with expected BLE signal behavior for medium-range proximity services. Packet accuracy exceeded 97% across all trials, indicating that the firmware is robust against transient interference in typical deployment conditions. The summary of these findings is presented in Table 1.

Table 1. BLE advertising performance metrics for the proposed LINE Beacon firmware

Metric	Measured Value	Notes
Protocol compliance	100%	Verified with nRF Connect packet inspection
Packet accuracy	>97%	Based on received packet count vs. expected count
RSSI range	−60 to −65 dBm	Measured at 5–10 m in indoor environment
Advertising interval	~63 ms	Stable across multiple trials
Memory overhead	~5–10 KB	Compared to baseline MicroPython runtime

4.3 Memory Usage Analysis

To quantify the impact of object-oriented implementation on memory consumption, memory usage was recorded before and after instantiating the BLEManager and LineBeacon objects. As shown in Table 2, instantiation resulted in an increase of approximately 10–11 KB, which aligns with the modest overhead expected from MicroPython class allocation. This memory footprint remains acceptable for ESP32-based IoT applications that require protocol compliance and modular design.

Table 2. Memory usage before and after object instantiation

Object ID	Memory Before (bytes)	Memory After (bytes)	Increase
BLEManager instance	5000	15000	+10000
LineBeacon instance	7000	18000	+11000

4.4 Advertising Interval Consistency

The configured advertising interval of ~63 ms was consistently maintained during extended operation (>30 minutes). This value was not chosen arbitrarily; it is the default interval enforced by the ESP32 MicroPython BLE stack, which sets the parameter within the 60–65 ms range. Our validation confirmed that this constraint resulted in stable operation with no noticeable packet loss or jitter. Although optimization of advertising intervals was not the focus of this study, the observed ~63 ms setting is widely adopted in IoT beacon deployments, balancing energy efficiency and responsiveness for proximity-based services.

4.5 Development Efficiency and Maintainability

Compared to traditional C/C++ ESP-IDF implementations, the proposed MicroPython firmware framework significantly reduced development time. This improvement results from its simpler syntax, REPL-based debugging, and modular object-oriented design. The separation of BLE control (BLEManager) from payload

generation (LineBeacon) facilitated maintenance and streamlined updates when modifying beacon identifiers or protocol parameters. These characteristics make the framework particularly suitable for rapid prototyping, small-to-medium IoT deployments, and educational contexts where fast iteration is critical.

This study did not perform direct benchmarking of execution speed, energy efficiency, or throughput against traditional C/C++ firmware. While MicroPython offers clear advantages in accessibility and maintainability, detailed comparisons with low-level implementations are still required to fully quantify the trade-offs between development convenience and runtime performance.

The experimental results confirmed that the proposed firmware framework achieves protocol compliance and stable performance. However, practical trade-offs should be recognized. MicroPython introduces modest overhead in memory usage and execution speed, which may limit its suitability for highly resource-constrained or performance-critical applications. These costs are balanced by significant benefits in development speed, modularity, and maintainability. In contrast, C/C++ implementations typically deliver superior runtime efficiency and lower memory consumption but require longer development cycles and steeper learning curves. For many IoT prototyping and educational contexts, the advantages of MicroPython outweigh its performance limitations, while large-scale or energy-sensitive deployments may still benefit from low-level approaches.

4.6 Limitations

While the proposed firmware framework provides a novel and practical approach to LINE Beacon broadcasting, several limitations remain. First, all evaluations were conducted under controlled laboratory conditions, and the system's behavior in noisy or high-density BLE environments (e.g., public Wi-Fi zones) has not yet been tested. Second, the current implementation focuses solely on BLE packet broadcasting and does not include integration with LINE's cloud infrastructure; consequently, services such as push notifications, user engagement, and analytics cannot be fully demonstrated. Third, energy consumption profiling was not performed, particularly in idle and deep-sleep modes, leaving power efficiency unquantified. Finally, no direct cross-platform benchmarking was conducted against C/C++ implementations in terms of execution speed, energy consumption, startup latency, or throughput, meaning the performance trade-offs were only qualitatively discussed..

4.7 Summary of Findings

The experimental results confirm that the proposed object-oriented MicroPython framework for LINE Beacon broadcasting on the ESP32 achieves full protocol compliance, stable signal performance, and high packet accuracy while incurring only modest memory overhead. Packet inspections using nRF Connect verified correct 3-AD packet formatting, and RSSI measurements demonstrated consistent signal stability between −60 and −65 dBm. Advertising intervals remained steady at approximately 63 ms over prolonged operation, and memory usage increased by only 10–11 KB after object instantiation—well within the constraints of the ESP32 platform.

Compared to traditional C/C++ implementations, the MicroPython approach offered faster development, easier maintenance, and greater modularity without significantly compromising performance. These characteristics, combined with the framework's scalability and reusability, make it a practical foundation for real-world IoT applications in retail, smart cities, and asset tracking.

5. Conclusions

This work presented a lightweight, object-oriented MicroPython firmware framework for broadcasting LINE Beacon packets on the ESP32 platform. The proposed design encapsulated BLE initialization, advertising interval control, and payload generation into two reusable classes—BLEManager and LineBeacon—ensuring protocol compliance with the LINE Beacon 3-AD format while enhancing modularity, maintainability, and development speed.

Experimental validation demonstrated correct packet structure, stable RSSI values between −60 and −65 dBm, over 97% packet accuracy, and consistent advertising intervals (~63 ms) under controlled indoor conditions. Memory analysis confirmed that the object-oriented implementation incurred only modest overhead (10–11 KB), which remains acceptable for the ESP32's hardware constraints.

The main contributions of this study are:

1) A reusable OOP-based MicroPython framework for BLE broadcasting on ESP32, offering clearer structure and faster development compared with conventional C/C++ firmware.

2) Empirical validation of protocol compliance and performance, confirming high packet accuracy, signal stability, and consistent advertising intervals suitable for proximity-based IoT applications.

3) Demonstration of practical trade-offs between MicroPython and C/C++ development, highlighting the balance between rapid prototyping efficiency and performance overhead.

These findings demonstrate that combining MicroPython with object-oriented principles provides a practical and scalable foundation for small-to-medium IoT deployments. The framework can serve as a reference for educational, research, and proof-of-concept implementations, while paving the way for future work on power profiling, backend integration, multi-protocol support, and security enhancements.

Future Work

Several directions will be pursued to enhance and extend the proposed firmware framework. First, detailed power consumption profiling will be conducted in active, idle, and deep-sleep modes to support long-term, battery-powered deployments. Second, real-world deployment testing will be performed in congested RF environments such as shopping malls, transportation hubs, and public events, where BLE interference is common. Third, backend integration will be developed to enable secure communication with LINE's cloud infrastructure, providing full support for push notifications, analytics, and real-time user engagement.

Fourth, systematic cross-language benchmarking against ESP-IDF and Arduino C++ implementations will be carried out to evaluate execution speed, memory usage, and energy efficiency, thereby clarifying the trade-offs between MicroPython's development advantages and the performance of low-level approaches. Fifth, multi-protocol support will be added to extend compatibility with beacon standards such as iBeacon and Eddystone. Finally, lightweight encryption and authentication schemes will be investigated to strengthen security against spoofing and unauthorized data injection.

These directions will not only address current limitations but also position the firmware framework for adoption in a wider range of proximity-based IoT applications, including retail, tourism, and industrial asset tracking.

References

[1] I. Plauska, A. Liutkevičius, and A. Janavičiūtė, "Performance evaluation of C/C++, MicroPython, Rust and TinyGo programming languages on ESP32 microcontroller," *Electronics*, vol. 12, no. 1, Art. no. 143, 2023, doi: 10.3390/electronics12010143.

[2] M. Siekkinen, M. Hiienkari, J. Nurminen, and J. Nieminen, "How low energy is Bluetooth low energy? Comparative measurements with ZigBee/802.15.4," in Proc. *IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Paris, France, 2012, pp. 153–158, doi: 10.1109/WCNCW.2012.6215496.

[3] D. Hercog, T. Lerher, M. Truntič, and O. Težak, "Design and implementation of ESP32-based IoT devices," *Sensors*, vol. 23, no. 15, Art. no. 6739, 2023, doi: 10.3390/s23156739.

[4] "MicroPython libraries—MicroPython latest documentation." [Online]. Available: https://docs.micropython.org/en/latest/library/index.html. (Accessed: Feb. 21, 2025).

[5] "Technical documents | Espressif Systems." [Online]. Available: https://www.espressif.com/en/support/documents/technical-documents. (Accessed: Apr. 11, 2025).

[6] T. Vallius, J. Haverinen, and J. Röning, "Object-oriented embedded system development method for easy and fast prototyping," in *Mechatronics for Safety, Security and Dependability in a New Era*. Amsterdam, The Netherlands: Elsevier, 2007, pp. 265–270, doi: 10.1016/B978-008044963-0/50054-0.

[7] "LINE developers." [Online]. Available: https://developers.line.biz/. (Accessed: Apr. 11, 2025).

[8] "Occupancy monitoring using BLE beacons: Intelligent Bluetooth virtual door system," *Sensors*, vol. 25, no. 9, Art. no. 2638, 2025. [Online]. Available: https://www.mdpi.com/1424-8220/25/9/2638. (Accessed: May 21, 2025).

[9] P. Babiuch, W. Folwarczny, and R. Juránková, "Using the ESP32 microcontroller for data processing," *MATEC Web of Conferences*, vol. 210, pp. 6, 2019.

[10] D. Mischianti, "MicroPython with ESP8266 and ESP32: Flashing firmware and programming with basic tools - 1," *Mischianti Blog*, May 2023. [Online]. Available:https://www.mischianti.org/2023/05/13/micropython-with-esp8266-and-esp32-flashing-firmware-and-programming-with-basic-tools-1/. (Accessed: Apr. 11, 2025).

[11] "The power of object-oriented programming in embedded systems," CPP Cat Blog, C. Cat, 2023. [Online]. Available: https://cppcat.com/2023/08/17/the-power-of-object-oriented-programming-in-embedded-systems/. (Accessed: Mar. 31, 2025).

[12] D. Avellaneda, D. Mendez, and G. Fortino, "A TinyML deep learning approach for indoor tracking of assets," *Sensors*, vol. 23, no. 3, Art. no. 1542, 2023, doi: 10.3390/s23031542.

[13] R. Ramirez, C.-Y. Huang, C.-A. Liao, P.-T. Lin, H.-W. Lin, and S.-H. Liang, "A practice of BLE RSSI measurement for indoor positioning," *Sensors*, vol. 21, no. 15, Art. no. 5181, 2021, doi: 10.3390/s21155181.

[14] V. Ionescu and F. Enescu, "Investigating the performance of MicroPython and C on ESP32 and STM32 microcontrollers," in *Proc. 2020 IEEE 26th International Conference on Automation, Quality and Testing, Robotics (AQTR)*, 2020, pp. 237–242.

[15] M. Antonini, M. Pincheira, M. Vecchio, and F. Antonelli, "An adaptable and unsupervised TinyML anomaly detection system for extreme industrial environments," *Sensors*, vol. 23, no. 4, Art. no. 2344, 2023, doi: 10.3390/s23042344.

[16] R. Uddin, T. Hwang, and I. Koo, "Worker presence monitoring in complex workplaces using BLE beacon-assisted multi-hop IoT networks powered by ESP-NOW," *Electronics*, vol. 13, no. 21, Art. no. 4201, 2024, doi: 10.3390/electronics13214201.

[17] K. Ferencz and J. Domokos, "Rapid prototyping of IoT applications for the industry," in *Proc. IEEE Int. Conf. Autom., Qual. Test., Robot. (AQTR)*, Cluj-Napoca, Romania, 2020, pp. 1–6, doi: 10.1109/AQTR49680.2020.9129934.