TELKOMNIKA Telecommunication Computing Electronics and Control
Vol. 24, No. 4, August 2026, pp. 1278~1286
ISSN: 1693-6930, DOI: 10.12928/TELKOMNIKA.v24i4.27707

GenAI as an IoT programming assistant: a case study on automated debugging for air quality monitoring systems

Steven Imanuel Bawole, Handri Santoso
Information Technology, Pradita University, Banten, Indonesia

Article Info

Article history:

Received Dec 11, 2025
Revised Apr 3, 2026
Accepted May 25, 2026

Keywords:

Automated debugging
Gemini artificial intelligence
Generative artificial intelligence
Internet of things
Programming assistant

ABSTRACT

The rapid expansion of internet of things (IoT) technology has necessitated the development of user-friendly programming solutions for non–experts. While generative artificial intelligence (GenAI) offers the potential to democratize code development, its ability to assist in the intricate task of automated debugging, particularly regarding hardware integration remains a critical area of research. A design research approach was employed, employing a structured four – phase workflow: error analysis, diagnostic execution through prompting, iterative solution analysis, and functional verification. The methodology was applied to an experimental case study involving an air quality (AQ) monitoring system. The study tested the artificial intelligence (AI)'s capacity to debug C++ code intended for the Arduino integrated development environment (IDE). Gemini AI successfully identified and resolved three critical logic errors arising from mismanaged MQ135 calibration variables, incorrect loop sequencing, and data desynchronization between the organic light emitting diode (OLED) display and the internal status logic. GenAI proved effective as a programming assistant for resolving bugs in IoT applications. However, effective debugging still depends on well-structured prompts and a basic understanding of the underlying IoT hardware.

Corresponding Author:

Steven Imanuel Bawole
Information Technology, Pradita University
Scientia Business Park, Gading Serpong Boulevard Street, No. 1 Tower 1, Curug Sangereng
Kelapa Dua District, Tangerang Regency, Banten 15810, Indonesia
Email: steven.imanuel@student.pradita.ac.id

1. INTRODUCTION

Maintaining the reliability and speed of software demands a meticulous focus on identifying and rectifying defects. Despite advancements in development methodologies, persistent bugs continue to cause system crashes, compromise security, and frustrate users with an unsatisfactory user experience [1]. Manual debugging and static rules are insufficiently efficient and susceptible to errors to effectively manage the intricate nature of contemporary software systems [2]. They encounter a scalability limitation due to human resource constraints and the potential for oversight. Conversely, static detection methods are increasingly inadequate in identifying profound logical vulnerabilities within extensive systems [3].

The rapid expansion of the internet of things (IoT) has democratized access to sophisticated hardware, yet the process of firmware debugging remains one of the most formidable barriers for non-experts. Unlike traditional software development, where errors are often detected in controlled virtual environments, IoT debugging necessitates a profound understanding of the interplay between physical signals and digital logic. For novices, the transition from writing basic code to diagnosing a race condition or a memory leak on a remote microcontroller can be overwhelming. This technical complexity often hinders

innovation, as the inability to effectively troubleshoot "silent" hardware failures effectively prevents many aspiring creators from progressing beyond the prototype stage.

The IoT has profoundly transformed the global landscape [4], with applications spanning from industrial monitoring to smart agriculture [5]. As IoT hardware becomes more affordable and energy-efficient, it contributes to a sustainable digital revolution and advancement of Moore's law [6], [7]. Nevertheless, despite the reduction in the hardware acquisition barrier, a substantial gap persists in software development. Developing firmware is inherently complex, necessitating profound expertise in hardware limitation sensor calibration, and connectivity protocols. Recent data underscore this challenge: 74% of developers still rely on manual device access for debugging, and 62% encounter difficulties in managing system failures without compromising availability [8].

To address these challenges, generative artificial intelligence (GenAI) has emerged as a transformative technology, facilitating the automation of intricate tasks that typically require human intelligence [9], [10]. The integration of GenAI offers substantial potential for the future of programming by enhancing efficiency and accessibility [11]. By bridging the gap for individuals lacking technical proficiency, GenAI empowers novice users to harness IoT in their daily lives. This synergy between GenAI, blockchain, and IoT fundamentally reshapes the digital realm [12]. Provided that supporting technology and regulations develop as anticipated [13], [14], these integrated systems will continue to exert a lasting social and economic impact [10] by facilitating the independent interaction of interconnected devices within existing infrastructure [15], [16].

2. LITERATURE REVIEW

Sain *et al*. [17] examine the role of chat generative pre-trained transformer (ChatGPT) in software debugging, highlighting its potential while identifying critical dependencies and areas for future improvement. Tamkhade *et al*. [18] explores how GenAI is revolutionizing software engineering by making automated debugging and code synthesis indispensable tools for contemporary development. Amburle *et al*. [19] found that the GenAI-powered code error explainer addresses critical deficiencies in programming education by serving as a versatile instrument for multiple programming languages. Hai *et al*. [20] investigate strategies for optimizing cloud-based bug tracking, balancing operational cost reduction and timeline management with the preservation of stringent quality standards across both general and embedded software domains. Further research [21]–[24] delves into the automation of bug detection, defect forecasting, and feature selection within the software environments by employing machine learning techniques, as shown in Table 1.

Table 1. Summarisation of the literature review

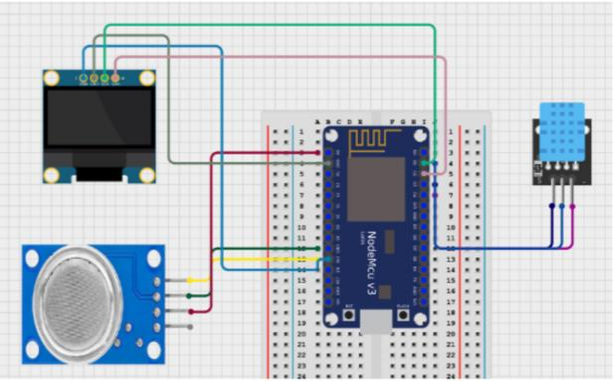
Study	Approach	Strength	Limitation
Sain *et al.* [17]	This research delves into the natural language processing capabilities, knowledge, and patern recognition of ChatGPT	Identify recurring code patterns frequently link to specific bug	The need to validate the conventional method is imperative
Tamkhade *et ali*. [18]	Employ a comprehensive review and case-study methodology to assess the integration GenAI into software development lifecycle	Facilitates expedited bug identification and minimalizes operational downtime	Challenges in achieving high precision, particularly in minimizing false positive
Amburle *et al*. [19]	The project centers on automated pedagogical intervention	Minimize the "wait time" for human tutoring	It lacks support for contemporary programming languages such as JavaScript, Rust, or Swift
Hai *et al.* [20]	This study examines the application of deep learning, particularly multi-layer perceptron neural network	High accuracy for defect predictiom, optimised model selection	Accuracy improvement is still a challenge
Al-Shameri [21]	This study delves into the capabilities of GenAI in identifying software defect with precision and proposing remedial measures or enhancements to the code	GenAI solutions can identify potential software vulneralbilities with greater accuracy than previous methods	Lack of explicitly stated limitation
Sharma *et al*. [22]	Conducts a comprehensive analysis of ensamble-based machine learning methodologies employed in the prediction of software defects	The paper emphasizes the paramount economic significance of early defect identification	Hyperparameter optimalisation remains challengging
Arasteh *et al*. [23]	Binary cahos-based olympiad optimization algorithm for defect prediction	Enhances predictive performance, thereby improving accuracy, precision, recall, and F1-score.	Computationally intensive due to intricate feature selection
Wang *et al.* [24]	Binary grey wolf optimser: a comprehensive approach to defect classification	Ehnaces classification precision and accuracy, demonstrating effectiveness across diverse real-world datasets	Performance may deteriorate when datasets are highly imbalanced

3. RESEARCH GAP

Traditional debugging necessitates the expertise of seasoned professionals, particularly considering the evolving nature of programming languages [25]. Despite the widespread adoption of GenAI-driven debugging tools in general software development, a critical gap persists in the specialized domain of IoT and firmware development. Current GenAI models are predominantly optimized for high-level languages and virtual environments, rendering them inadequate for addressing the "silent failure" and real-time physical-digital independencies inherent in hardware-integrated systems. Consequently, a staggering 74% of developers continue to rely on manual debugging methods, while non-experts are frequently discouraged from IoT innovation due to the substantial technical barrier associated with firmware troubleshooting. Consequently, there is an immediate need for a GenAI-assisted framework that supports contemporary language and provides diagnostics for hardware = specific errors.

4. METHOD

The experiments were conducted utilizing the Gemini 1.5 Pro model. The hardware environment comprises a NodeMCU ESP8266 microcontroller, an MQ135 gas sensor, a DHT11 temperature and humidity sensor, and an SSD1306 display. The code was compiled and verified using the Arduino integrated development environment (IDE), as seen in Figure 1. To convert the raw analog-to-digital converter (ADC) reading into a gas concentration, calibrate the sensor under known conditions. The MQ135 sensor requires a warm-up period as specified by the manufacturer, which can persist for up to ten hours to ensure stable operation [26]. To guarantee the reproducibility of the method, the study employs a structured diagnostic workflow. In contrast to fully automated tools, this framework utilizes specific prompt templates to guide the artificial intelligence (AI) through the debugging process, as seen in Table 2.



Arduino IDE Configuration

Board Manager URL:
http://arduino.esp8266.com/stable/package_esp8266com_index.json
Board Selection: NodeMCU 1.0 (ESP-12E Module).
Upload Speed: 115200 baud.

Required Libraries:
MQ135 by Georg K.
DHT sensor library by Adafruit (v1.4.4).
Adafruit SSD1306 (v2.5.7) & Adafruit GFX.
ThingSpeak by MathWorks.

Figure 1. Wiring diagram and Arduino IDE configuration

Table 2. Iterative prompting framework

Phase	Objective	Exact prompt template used
Phase 1	Problem contextualization	"I am using an ESP8266 with DHT11 and MQ135 sensor for air quality (AQ) monitoring. My code is not working as expected, can you help me identify the issues ?"
Phase 2	Diagnostic prompt execution	"Analyze the following C++ code for Arduino IDE: [paste code]. Identify specific logic errors, variable mismatches, or timing issues that could affect sensor accuracy."
Phase 3	Iterative solution analysis	"Provide the corrected code for the identified logic errors. Specifically, address how the MQ135 uses DHT11 data and ensure the loop sequence is correct for fresh data."
Phase 4	Functional verification	"Review the fixed code for duplicate Serial outputs and verify if the timing (delays) respects ThinkSpeak rate limits without hindering sensor responsiveness."

The proposed workflow "four phase methods" involves analyzing errors in programming. The initial phase involves identifying issues with the bugs or errors, whether they are in the code, library, or wiring. The second phase includes a diagnostic prompt that aims to identify existing obstacles. The third phase analyses iterative solutions. If the solution is functional, it is considered successful, and the process ends. If not functional, the process returns to the diagnostic phase, indicating a new diagnostic prompt; the solution is not functional. The fourth phase involves verifying the function's functionality. If the function is successful, the bug error is considered fixed. If it fails, the process returns to the second phase, see Figure 2.

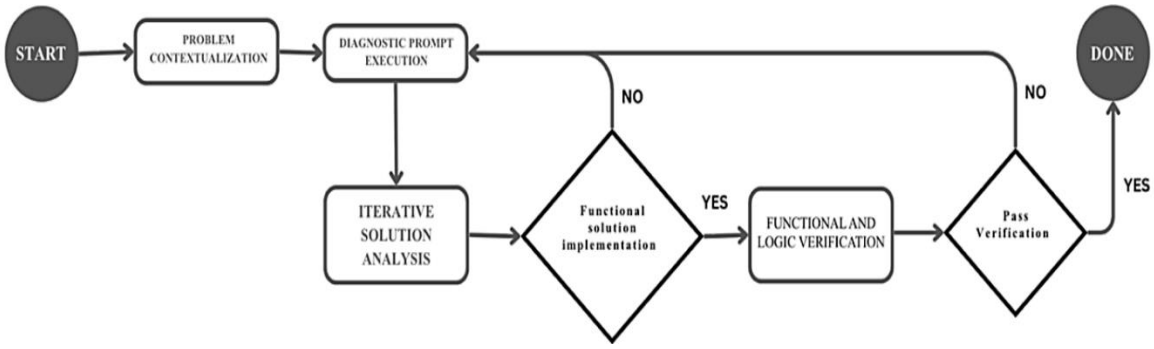


Figure 2. Workflow GenAI assistant

```
PSEUDOCODE
START
 CALL Problem_Contextualization()

 WHILE (Solution is not verified)
  CALL Diagnostic_Prompt_Execution()
  CALL Iterative_Solution_Analysis()

  IF (Pass_Verfication == SUCCESS) THEN
   CALL Functional_and_Logic_Verification()

   IF (Pass_Verification == TRUE) THEN
    GOTO DONE
   ELSE
    // Loop back to Diagnostic Prompt Execution
    Continue
   END IF
  ELSE
    // Implementation failed, loop back
    Continue
  END IF
 END WHILE
DONE
```

5. RESULTS AND DISCUSSION

Table 3 presents the capabilities of GenAI, particularly Gemini AI, in addressing coding challenges and bugs encountered by users and IoT users lacking programming expertise. In this context, when issuing commands to GenAI, such as Gemini AI, a prompt is required to furnish the requisite information. This information enables AI to execute commands in a manner consistent with the desired outcomes of the user.

Table 3. Prompt effectiveness analysis

Iteration phase	Type of prompt	Prompt content	AI response analysis	Effectiveness
Phase 1	Generic/vague	My IoT code for AQ is not working. Please Fix it.	AI response with the specific answer, the code: past the script you are currently using the hardware: which microcontroller (e.g., Arduino UNO, ESP8266< ESP32, Raspberry Pi) and which sensor (e.g., MQ135, BME680, DHT11) are you using? the error: is it a compilation error, or the sensor reading just wrong.	AI provides solutions to the tasks that require it.
Phase 2	Code only	Here is my full code (paste code).	AI find the main issues in the code: code with incomplete, variable mismatch, and ThingSpeak timing.	AI can pinpoint the specific area that needs to be fixed.
Phase 3	Structured diagnostic prompt	Analyze the code and identify the issue and provide the code to fix it.	AI analyzed the code.	AI can solve the problem and provide the reason behind the code's malfunction.

Based on the provided workflow, an experiment was conducted by integrating the IoT program code into the Arduino IDE. Initially, it's crucial to understand the code in the AQ IoT. This phase involves identifying the purpose and functionality of the code. The primary objective is to monitor ambient AQ, while its key functions include detecting air temperature and humidity using the DHT11 sensor, reading the AQ sensor to obtain the revolutions per minute (RPM) value from the MQ135 sensor, making corrections to the MQ135 parts per million (PPM) reading based on temperature and humidity data, displaying the data (PPM, temperature, and humidity) on the SSD1306 organic light emitting diode (OLED), and transmitting the data (PPM, temperature, and humidity) to the ThingSpeak IoT platform. However, potential issues include inaccurate sensor readings, erroneous PPM logic, wireless fidelity (Wi-Fi) connection failures that prevent data transmission to ThingSpeak, and OLED displays that may overlap or not be updated correctly.

The second and third phases of diagnostic execution and iterative analysis, as identified by Gemini AI, revealed a logic error in the MQ135 logic. Specifically, there's an issue with the readMQ135() function, which relies on global variables temperature and humidity to make corrections (`genCorrectedRZero` and `getCorrectedPPM`). The problem fact that the DHT11 sensor (`readdht11()`) feeds its data to the temperature and humidity variables, while `readMQ135()` never utilizes these values. Instead, it defaults to hardcoded values of 29.0 °C and 25.0%RH. As a result, the PPM correction fails to function correctly. The `correctedPPM` value is always calculated based on the temperature of 29°C and the humidity of 25%, regardless of the actual temperature and humidity in the room.

Gemini AI's solution involves removing unused global float temperature and float humidity variables. Additionally, the function `readMQ135()` needs to be modified to use the correct global variable. Gemini AI also provides information about the incorrect order of function execution. Additionally, the order of invocation in a `loop()` creates a new problem. The code calls `readMQ135()` before `readDHT11()` inside the `loop()`, which causes a problem. In the first iteration of the `readMQ135` loop, the code uses the temperature and humidity values from the previous iteration because the new values from DHT11 have not been read yet. To fix this issue, you need to change the order of the calls.

The next issue is the inconsistency in the PPM data for calls. The program code contains two different PPM variables, which is confusing. Gemini AI discovered a problem with `correctedPPM` when sending it to ThingSpeak. The `correctedPPM` was displayed on the "AQ" line on the OLED, but for the status logic ("fresh air", "poor air", and "danger"), the PPM variable (which was not corrected) was used. The user will see the corrected PPM number in the layer, but the text state ('fresh air') is based on a different PPM number, which is inconsistent. To resolve this issue, use the corrected PPM for all PPM-related logic, see Table 4.

Table 4. Comparative analysis of logical errors and AI-generated fixes

Bug ID	Problem description	Original code (buggy)	GenAI solution (fixed)	Technical note

Bug 1	Static variable usage: the function used hardcoded global variables instead of live sensor data	`float temperature = 29.0;`
		`//....`
		`correctedRZero = mq135_sensor.getCorrectedRZero(temperature, humidity);`
		`...`
		`correctedPPM = mq135_sensor.getCorrectedPPM(temperature, humidity);`
			`void readMQ135()`
			`{`
			`rzero = mq135_sensor.getRZero();`
			`correctedRZero = mq135_sensor.getCorrectedRZero(temp, humi);`
			`resistance = mq135_sensor.getResistance();`
			`ppm = mq135_sensor.getPPM();`
			`correctedPPM = mq135_sensor.getCorrectedPPM(temp, humi);`
			`// ...`
				GenAI recognized the intent to use dynamic data from DHT11

Bug 2	Execution sequence readMQ135 was called before readDHT1, causing it use old/null data	`void loop() {`
		`readMQ135();`
		`readDHT11();`
		`...`
			`void loop() {`
			`readDHT11(); // Data Source`
			`readMQ135(); // Data Consumer`
				GenAI successfully modelled the temporal dependency of the data flow

Bug 3	Data inconsistency: display logic used raw ppm while OLED showed correctedPPM	`if (ppm<=1000)`
		`{`
		`display.printin("Fresh"); }`
			`if (correctedPPM<=1000)`
			`{`
			`display.printin("Fresh");`
			`}`
				Unfiled data visualization and logic decision variables

Optimization issue: duplicate code. There's an issue with the `Serial.print` code block for MQ135 data appearing twice. This happens once inside the `readMQ135()` function and once inside the main `loop()` after ThingSpeak delivery. Although this doesn't cause any bugs, it's inefficient and causes the serial monitor to spam. To fix this, remove the `Serial.print` block that's inside the `loop()` (after the ThingSpeak block) and leave the one inside the `readMQ135()` function.

The fourth phase is functional verification. Once the bug is fixed, we can verify the functionality and suggest best practices for fixes. There's a problem with the code that relies heavily on `delay()` there's a delay of 5000 milliseconds inside the WiFi connection loop and a delay of 2000 milliseconds or 2 seconds at the end of the main `loop()`. For 2 seconds, the device is essentially "dead" because it doesn't read the sensor and doesn't respond to anything. If the quality of the air suddenly becomes dangerous, the device won't know about it until 2 seconds later.

This method successfully identifies three logical bugs and one iteration anomaly that the compiler will overlook. The primary issues are variable handling errors (temperature versus temp) and incorrect execution sequences, which completely negate the purpose of connecting the MQ135 sensor. By implementing the proposed fixes, particularly the first three bugs, this code will function as intended.

5.1. Comparative analysis

Following the execution of the same action with ChatGPT 4.0, a comparative analysis was conducted between the outcomes generated by Gemini 1.5 Pro and ChatGPT 4.0. Notably, Gemini provides a more comprehensive "iterative analysis" of the impact of the bug in the system over time. For instance, while GPT identified an incorrect sequence, Gemini elucidated that the initial iteration utilized data from the previous loop due to the sensor's unread state. Furthermore, Gemini specifically pinpointed that the OLED display displayed the `correctedPPM`, while the status logic erroneously employed the raw PPM.

Gemini identified a non-fatal "Optimization Issue" pertaining to duplicate code. It was observed that the `serial.print` function was invoked both within the sensor function and in the main loop, resulting in superfluous output in the serial monitor. GPT's solution is primarily focused on rectifying the mathematical inputs and resolving the blocking WIFI code to prevent the device from appearing "unresponsive." Gemini solution, suggested more streamlined architectural modification by eliminating redundant global variables and ensuring that all logins consistently utilized the `correctedPPM` variable.

In comparison to traditional debugging techniques, such as using a compiler or standard manual testing, GPT and Gemini demonstrate distinct approaches to logical and temporal error management. Although traditional methods excel in syntax validation, the AI models excel in identifying "silent" logic failures that a compiler would overlook.

5.2. Bug detection rate and hardware context fidelity (HCF)

The results demonstrate that AI has successfully identified and rectified three critical logic errors. Bug 1 (static variable): replace the hardcoded temperatures/humidity value with actual data retrieved from the DHT11 sensor. Bug 2 (execution order): after the execution order to ensure that `readDHT11()` is executed prior to `readMQ135()`. Bug 3 (data inconsistency): ensure uniform utilization of correctedPPM variables across OLED display and status logic.

Gemini AI proposed to eliminate the `delay2000` feature, which was intended to achieve a specific purpose. However, Gemini AI misinterpreted the existing hardware and, while it was able to identify bugs in the code, it failed to provide any information about the physical tools required to implement the delay, as seen in Table 5. In this case, human intervention is still necessary to ensure the accuracy of the system. Therefore, the accuracy HCF matrix is calculated as:

$$HCF = \left(\frac{Valid\ Hardware\ Sugestion}{Total\ Hardware\ Intervention}\right) \times 100\% \quad (1)$$

Gemini AI hallucinates, so only two valid suggestions can be used from the three available ones. This leaves 66.7% of the HCF.

IoT debugging not only necessitates code devoid of syntax errors (compilation errors) but also adheres to physical laws and the electrical limitations of hardware. The 66.7% score indicates a "semantic gap" where GenAI excels in logical reasoning (data flow) while lacking physical awareness (chemical characteristics of sensors). These metrics formally demonstrate the continued significance of human involvement as a crucial intermediary between the ideal logic of GenAI and physical reality in the IoT domain.

Table 5. HCF scorecard

Hardware requirement	Critical constraint	GenAI suggestion	Fidelity status	Score
Pin mapping (MQ135)	ESP8266 only has one analog pin (AO)	Suggest utilizing AO as a replacement for the erroneous pin definition	Valid	1
Data depedency	MQ135 necessitates temperature and humidity data (DHT11) for PPM compensation	Enhance the execution sequence to ensure that MQ135 utilizes the most recent data retrieved from DHT11	Valid	1
Sensor stabilization	MQ135 necessitates a prehating period to ensure the stability of the readings	Propose eliminating the `delay()` function for code optimization, disregarding the potential impact on sensor stability	Halucination	0
Total score				2/3
HCF percentage				66.7%

The findings of this study emphasize the transformative potential of GenAI, particularly Gemini AI, in bridging the technical gap for IoT practitioners who lack advanced programming expertise. The results from the prompt effectiveness analysis are seen in Table 3, indicating that the quality of GenAI-driven debugging is significantly influenced by the structure of the input. While a generic prompt yields only preliminary diagnostic questions, the transition to structured diagnostic prompts enables the AI to progress beyond syntax verification and delve into profound logical analysis. This suggests that "prompt engineering" is emerging as a crucial skill in the IoT development lifecycle, serving as a mediator between human intent and machine execution.

A notable contribution of this research is the identification of Gemini AI's ability to pinpoint "semantic gap" in IoT firmware-errors where the code is syntactically valid but exhibits logical flaws within a physical context. For instance, the GenAI accurately identified that the `readMQ135()` function was erroneously relying on hardcoded temperature and humidity values (29.0 ℃ and 25.0%RH) instead of utilizing the live data stream from the DHT11 sensor. Such errors are notoriously challenging for novice developers to debug because the code compiles without errors, yet produces inaccurate environmental data. Furthermore, the AI's detection of execution sequence errors, specifically the invocation of `readMQ135()` prior to `readDHT11()`-demonstrates a sophisticated understanding of data dependencies and temporal logic in microcontrollers.

However, the HCF score of 66,7% highlights a critical limitation of current GenAI models: their absence of "physical awareness" regarding hardware-specific constraints. While the GenAI excelled in logical reasoning (data flow and variable consistency), it encountered difficulties in comprehending the chemical and electrical nuances of the hardware, such as the MQ135 sensor's preheating requirements. This hallucination suggests the removal of the `delay()` function at the expense of sensor stability-underscores that AI is not yet a complete replacement for human expertise.

6. CONCLUSION

This study set out to evaluate the effectiveness of GenAI as a debugging assistant for IoT applications. RQ1: can GenAI assist in the programming of embedded systems? yes, research definitively confirms that GenAI, particularly Gemini, can aid in developing complex IoT firmware. It seamlessly bridges the gap between high-level logic and low-level C++ implementation, enabling users to concentrate on system design rather than getting bogged down by syntactical details. RQ2: is GenAI effective in resolving logical bugs? yes, it is. In this study, the GenAI successfully identified and resolved three distinct logical errors. Notably, these errors were ones that standard compilers failed to detect, demonstrating that GenAI offers a layer of semantic debugging that was previously inaccessible to non-experts. RQ3: what are the limitations of AI-assisted debugging? the GenAI lacked consideration for the chemical and electrical requirements of sensors, such as the preheating period required for the MQ135 to achieve stability. Given that the GenAI does not fully comprehend the physical implications of its code suggestions, it cannot currently serve as a complete substitute for human oversight in a hardware-intensive environment. RQ4: is specialized expertise required to use GenAI as a programming assistant? yes, a modified form of expertise is necessary. While users don't need deep proficiency in C++ syntax or library definitions, they do require an understanding of the specific hardware function.

To maximize the utility of GenAI in professional embedded engineering, future research should transition from general code generation to addressing intricate system-level complexities. Key priorities include investigating the efficacy of GenAI in debugging RTOS-specific bottlenecks, such as resolving priority inversion or identifying deadlock in high-concurrency environments. Furthermore, studies should assess GenAI's performance in managing hard real-time constraints, particularly its ability to minimize jitter

and optimize interrupt service routine overhead. Lastly, there is a critical need for research in AI-driven sensor calibration and robust error handling. The research should focus on automating fail-safes for secure communication protocols and preventing memory corruption on resource-constrained microcontrollers.

ACKNOWLEDGMENTS

This research is designed for beginners in the IoT to simplify the integration of IoT into daily life. The authors express their gratitude to Pradita University for providing a supportive academic environment, laboratory facilities, and administrative assistance that enabled the successful completion of this research. They also extend their appreciation to the faculty members and colleagues at Pradita University for their invaluable guidance, technical discussions, and thoughtful suggestions, which significantly enhanced the quality of this work.

FUNDING INFORMATION

There is no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Steven Imanuel Bawole	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓
Handri Santoso	✓	✓			✓	✓			✓	✓		✓		

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

Data availability is not relevant to this paper, because no new data were analyzed during this study.

REFERENCES

[1] R.-W. Bello and S. J. Tobi, "Software Bugs: Detection, Analysis and Fixing," *SSRN Electronic Journal*, 2024, doi: 10.2139/ssrn.4662187.
[2] S. Hore, A. Shah, and N. D. Bastian, "Deep VULMAN: A deep reinforcement learning-enabled cyber vulnerability management framework," *Expert Systems with Applications*, vol. 221, p. 119734, Jul. 2023, doi: 10.1016/j.eswa.2023.119734.
[3] M. Samir, N. Sherief, and W. Abdelmoez, "Improving Bug Assignment and Developer Allocation in Software Engineering through Interpretable Machine Learning Models," *Computers*, vol. 12, no. 7, p. 128, Jun. 2023, doi: 10.3390/computers12070128.
[4] S. Tiwari, "A Review of Internet of Things Application in Malaysia," *Borneo Journal of Sciences & Technology*, vol. 4, no. 1, Jan. 2022, doi: 10.35370/bjost.2022.4.1-11.
[5] D. Lynda, F. Brahim, S. Hamid, and C. Hamadoun, "Towards a semantic structure for classifying IoT agriculture sensor datasets : An approach based on machine learning and web semantic technologies," *Journal of King Saud University - Computer and Information Sciences*, vol. 35, no. 8, p. 101700, Sep. 2023, doi: 10.1016/j.jksuci.2023.101700.
[6] S. Dhal *et al.*, "Internet of Things (IoT) in digital agriculture: An overview," *Agronomy Journal*, vol. 116, no. 3, pp. 1144–1163, May 2024, doi: 10.1002/agj2.21385.
[7] P. Wang *et al.*, "Device Technology in Post-Moore Era: Research Progress and Future Trends," *Science and Technology Foresight*, vol. 1, no. 3, pp. 130–145, 2022, doi: 10.3981/j.issn.2097-0781.2022.03.011.
[8] A. Makhshari and A. Mesbah, "IoT bugs and development challenges," in *Proceedings - International Conference on Software Engineering*, IEEE, May 2021, pp. 460–472. doi: 10.1109/ICSE43902.2021.00051.
[9] C. A. G. da Silva, F. N. Ramos, R. V. de Moraes, and E. L. dos Santos, "ChatGPT: Challenges and Benefits in Software Programming for Higher Education," *Sustainability (Switzerland)*, vol. 16, no. 3, p. 1245, Feb. 2024, doi: 10.3390/su16031245.
[10] R. Vinuesa *et al.*, "The role of artificial intelligence in achieving the Sustainable Development Goals," *Nature Communications*, vol. 11, no. 1, p. 233, Jan. 2020, doi: 10.1038/s41467-019-14108-y.

[11] V. Saravanan, S. Kavitha, S. Ravi, A. Seetha, Ch Rambabu, and Tatiraju V. Rajani Kanth, "Generative AI in Software Engineering: Revolutionizing Code Generation and Debugging," *International Journal of Computational and Experimental Science and Engineering*, vol. 11, no. 2, May 2025, doi: 10.22399/ijcesen.1718.
[12] P. Sandner, J. Gross, and R. Richter, "Convergence of Blockchain, IoT, and AI," *Frontiers in Blockchain*, vol. 3, Sep. 2020, doi: 10.3389/fbloc.2020.522600.
[13] L. O. Ofusori and R. Hendradi, "ChatGPT and Its Impact on Students Assessment Practices in the Higher Educational Sector: A Systematic Review," *Journal of Information Systems Engineering and Business Intelligence*, vol. 11, no. 1, pp. 65–78, Mar. 2025, doi: 10.20473/jisebi.11.1.65-78.
[14] S. Villamil, C. Hernández, and G. Tarazona, "An overview of internet of things," *Telkomnika (Telecommunication Computing Electronics and Control)*, vol. 18, no. 5, pp. 2320–2327, Oct. 2020, doi: 10.12928/TELKOMNIKA.v18i5.15911.
[15] D. Hercog, T. Lerher, M. Truntič, and O. Težak, "Design and Implementation of ESP32-Based IoT Devices," *Sensors*, vol. 23, no. 15, p. 6739, Jul. 2023, doi: 10.3390/s23156739.
[16] A. Gerodimos, L. Maglaras, M. A. Ferrag, N. Ayres, and I. Kantzavelou, "IoT: Communication protocols and security threats," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 1–13, 2023, doi: 10.1016/j.iotcps.2022.12.003.
[17] Z. H. Sain, R. Serban, M. A. Agoi, and S. H. Sain, "Leveraging ChatGPT to Enhance Debugging: Evaluating AI-Driven Solutions in Software Development," *Asian Journal of Computer Science and Technology*, vol. 13, no. 1, pp. 41–44, Apr. 2024, doi: 10.70112/ajcst-2024.13.1.4261.
[18] J. P. Tamkhade, A. M, and A. V Deshpande, "Generative AI and Code Synthesis Frameworks for Automated Software Engineering and Debugging," in *Multidisciplinary Engineering Applications of Artificial Intelligence in Design Control and Infrastructure Systems*, RADemics Research Institute, 2025, pp. 167–195. doi: 10.71443/9789349552609-06.
[19] A. Amburle, C. Almeida, N. Lopes, and O. Lopes, "AI based Code Error Explainer using Gemini Model," in *Proceedings of the 3rd International Conference on Applied Artificial Intelligence and Computing, ICAAIC 2024*, IEEE, Jun. 2024, pp. 274–278. doi: 10.1109/ICAAIC60222.2024.10574931.
[20] T. Hai, J. Zhou, N. Li, S. K. Jain, S. Agrawal, and I. Ben Dhaou, "Cloud-based bug tracking software defects analysis using deep learning," *Journal of Cloud Computing*, vol. 11, no. 1, p. 32, Aug. 2022, doi: 10.1186/s13677-022-00311-8.
[21] Y. N. H. Al-Shameri, "Applications of Artificial Intelligence for Enhanced Bug Detection in Software Development," in *Integrating Technology in Problem-Solving Educational Practices*, 2024, pp. 155–188. doi: 10.4018/979-8-3693-6745-2.ch008.
[22] T. Sharma, A. Jatain, S. Bhaskar, and K. Pabreja, "Ensemble Machine Learning Paradigms in Software Defect Prediction," *Procedia Computer Science*, vol. 218, pp. 199–209, 2022, doi: 10.1016/j.procs.2023.01.002.
[23] B. Arasteh, K. Arasteh, A. Ghaffari, and R. Ghanbarzadeh, "A new binary chaos-based metaheuristic algorithm for software defect prediction," *Cluster Computing*, vol. 27, no. 7, pp. 10093–10123, Oct. 2024, doi: 10.1007/s10586-024-04486-4.
[24] H. Wang, B. Arasteh, K. Arasteh, F. S. Gharehchopogh, and A. Rouhi, "A software defect prediction method using binary gray wolf optimizer and machine learning algorithms," *Computers and Electrical Engineering*, vol. 118, p. 109336, Aug. 2024, doi: 10.1016/j.compeleceng.2024.109336.
[25] I. Hoffman and N. Brooks, "Automated Bug Detection and Correction in Software Development using Machine Learning," *International Journal on Advanced Computer Theory and Engineering*, vol. 12, no. 1, pp. 11–16, Apr. 2025, doi: 10.65521/ijacte.v12i1.108.
[26] M. Kumar, G. Mishra, A. Sharma, A. Shaini, and S. Saxena, "Air Quality Monitoring Using MQ135 Gas Sensor and Arduino Uno," *International Journal of Latest Technology in Engineering Management & Applied Science*, vol. 14, no. 5, pp. 1097–1101, Jun. 2025, doi: 10.51583/ijltemas.2025.140500119.

BIOGRAPHIES OF AUTHORS

Steven Imanuel Bawole received a Master of Arts degree in Pastoral Counseling from STTB The Way, obtained in 2008. Currently, He is teaching in high school and am pursuing a Master of Informatics in Education program at Pradita University. He can be contacted at email: steven.imanuel@student.pradita.ac.id.

Handri Santoso received his Bachelor's degree in Physics from the University of Indonesia in 1996, followed by both Master's in 2005 and Doctoral degrees in Engineering from Nagaoka University of Technology, Japan, 2008. His academic and professional journey reflects a strong interdisciplinary foundation, bridging the domains of AI, ML, instrumentation and control engineering, cybersecurity, project management, and digital transformation frameworks. With extensive experience across both industrial and academic sectors, he has been instrumental in the development and deployment of innovative solutions in software engineering, control systems, and intelligent automation. He has led and contributed to numerous projects involving the practical application of ML and DL technologies, particularly within industrial and engineering environments. His research interests center around the convergence of AI, IoT-enabled systems, automation, and robotics, with an emphasis on integrating intelligent technologies to improve operational efficiency, system reliability, and data-driven decision-making. Actively engaged in interdisciplinary collaborations, he is committed to leveraging emerging technologies to drive impactful innovations in engineering, education, and industry. He can be contacted at email: handri.santoso@pradita.ac.id.