MANTECH Publications
International Journal of Programming Languages and Internet of Things
Volume 2, Issue 1, January-June 2026

Programming Languages and Software Platforms for Embedded and Internet-of-Things Systems

Dr. Karan Malhotra[1], Shreya Singh[2], Vikram Sethi[3], Neetu Jha[4]

ABSTRACT

The devices of embedded and Internet-of-Things systems operate under severe constraints of memory, processing, and energy, and the choice of programming language and software platform for them must respect these constraints; this paper reviews the languages and platforms used. It describes the constraints of embedded and Internet-of-Things devices and their consequences for software, and the programming languages used across the tiers of a system: the languages of the constrained device, principally the C language, which offers efficiency and control at some cost in safety, together with the Rust language, which offers memory safety without garbage collection, and interpreted languages such as MicroPython for prototyping; and the languages of the gateway and the cloud, which favour productivity. It describes the runtime environments, from bare-metal execution through real-time operating systems to embedded operating systems, and the platforms and frameworks that support development, including software development kits, cloud platforms for device management, and frameworks for machine learning on constrained devices, together with the provision of secure updates. The study finds that the choice of language and platform is governed by the constraints and the role of the device, lower tiers favouring control and efficiency and higher tiers favouring productivity, so that a system commonly combines several languages and platforms across its tiers. It notes that specific tools are representative of an evolving field. It concludes with the outlook for the field. Certain figures are illustrative.

KEYWORDS: *Embedded Systems, Internet of Things, Firmware, C Language, Rust, MicroPython, Real-Time Operating System, TinyML, Over-the-Air Updates*

INTRODUCTION

The devices at the heart of embedded and Internet-of-Things systems, the microcontrollers and small processors that sense, act, and communicate, operate under constraints far more severe than those of general-purpose computers: they have limited memory, often measured in kilobytes; limited processing, often a low-power microcontroller; and limited energy, often a battery that must last for years. The software that runs on these devices, and the tools with which it is built, must respect these constraints, and the choice of programming language and software platform is therefore a central concern in the engineering of embedded and Internet-of-Things systems. This paper reviews the languages and platforms used [1], [2].

The constraints of embedded devices shape the choice of language. The C language has long been dominant for firmware because it offers efficiency, a small footprint, and direct control of the hardware, though its manual management of memory is a source of errors; the Rust language is increasingly used because it offers memory safety without the overhead of garbage collection, preventing whole classes of errors; and interpreted languages such as MicroPython are used for prototyping and education, offering ease of development at some cost in efficiency. The appropriate language depends on the constraints of the device and the purpose of the software [1], [3].

An Internet-of-Things system, moreover, spans several tiers, from the constrained device through the gateway to the cloud, and the choice of language and platform differs across them, the constrained device favouring efficiency and control and the higher tiers favouring productivity. The central purpose of this paper is to review the programming languages, runtime environments, and software platforms used in embedded and Internet-of-Things systems, and to explain how the choice among them is governed by the constraints and the role of the device. The paper draws on the established and current literature, notes that specific tools are representative of an evolving field, and notes that certain figures are illustrative.

LITERATURE REVIEW

The constraints of embedded and Internet-of-Things devices and the languages used for them are described in the literature [1], [2], [4]. The devices are constrained in memory, processing, and energy, and often have real-time requirements and must operate unattended and reliably

for long periods, so the software must be efficient, predictable, and robust. The C language is the traditional choice for firmware because it compiles to efficient code, has a small footprint, and permits direct manipulation of the hardware, but its manual memory management makes it prone to errors, including the memory-safety errors that underlie many security vulnerabilities; the C++ language adds abstraction while retaining control and is used, with care, on more capable devices; and the Rust language provides memory safety at compile time, through an ownership system, without a garbage collector, and is increasingly adopted for embedded and safety-critical software. The literature describes these languages and the trade-off between control and efficiency, on the one hand, and safety and productivity, on the other [1], [3], [5].

Interpreted and higher-level languages and the runtime environments are described in the literature [6], [7], [8]. Interpreted languages, notably implementations of Python for microcontrollers such as MicroPython and CircuitPython, and, at the gateway, JavaScript, offer ease and speed of development and readability, at the cost of greater memory use and lower efficiency, and are used for prototyping, education, and less constrained devices. The software runs in a range of environments: on the smallest devices, bare-metal, without an operating system, giving the greatest control and smallest footprint; on many devices, a real-time operating system, which provides task scheduling and timing with a small footprint and deterministic behaviour; and on capable devices and gateways, an embedded operating system, which provides full networking and services at greater cost. The literature describes these environments and their suitability to different devices [6], [7].

The platforms, frameworks, and supporting services are described in the literature [9], [10]. Software development kits and toolchains, including accessible environments for prototyping and vendor and open frameworks for production, support the writing, cross-compilation, and deployment of software to devices. Cloud platforms provide for the management of devices, the ingestion of telemetry, and the delivery of updates, though reliance on a single provider's proprietary services raises the concern of lock-in. Frameworks for machine learning on constrained devices allow inference to be performed on the device itself. And the provision of secure over-the-air updates, together with secure communication and device identity, is essential for devices deployed in the field for long periods. The literature emphasises that these platforms and services, together with the language and runtime, make up the software

environment of an embedded or Internet-of-Things system. These considerations frame the present review [9], [10].

RESEARCH GAP

The languages, runtimes, and platforms for embedded and Internet-of-Things systems are treated in specialised literatures, but a concise, integrated account that presents them together across the tiers of a system, and explains how the choice among them is governed by the constraints and role of the device, is useful for orientation and for design. There is therefore value in a consolidated review that connects languages, runtimes, and platforms and situates them within the engineering of embedded and Internet-of-Things systems.

OBJECTIVES

The specific objectives of this study are:

- To describe the constraints of embedded and Internet-of-Things devices and their consequences.
- To describe the languages used, including C, Rust, and MicroPython, and their trade-offs.
- To describe the runtime environments, from bare-metal to embedded operating systems.
- To describe the platforms, frameworks, and services, including on-device machine learning.
- To explain how the choice of language and platform is governed by constraint and role.

METHODOLOGY

A review-based approach was adopted. The languages, runtimes, and platforms used in embedded and Internet-of-Things systems were consolidated from the established and current literature [1]-[10], and the material was organised into a framework comprising the languages across the tiers of a system, the development and deployment workflow, and the platforms and frameworks. Where figures present the tiers, the workflow, or the platforms, these are illustrative and schematic, and the tools named are representative examples of a rapidly evolving field rather than an exhaustive account.

Languages Across Tiers and Workflow

The programming languages used across the tiers of an Internet-of-Things system are represented in Figure 1, showing the languages of the constrained device, the gateway, and the cloud, and the shift from control and efficiency at the lower tiers to productivity at the higher. The development and deployment workflow is represented in Figure 2, showing the

writing, cross-compilation, flashing, running, monitoring, and updating of software. These representations organise the subject as set out in the literature [1], [9].

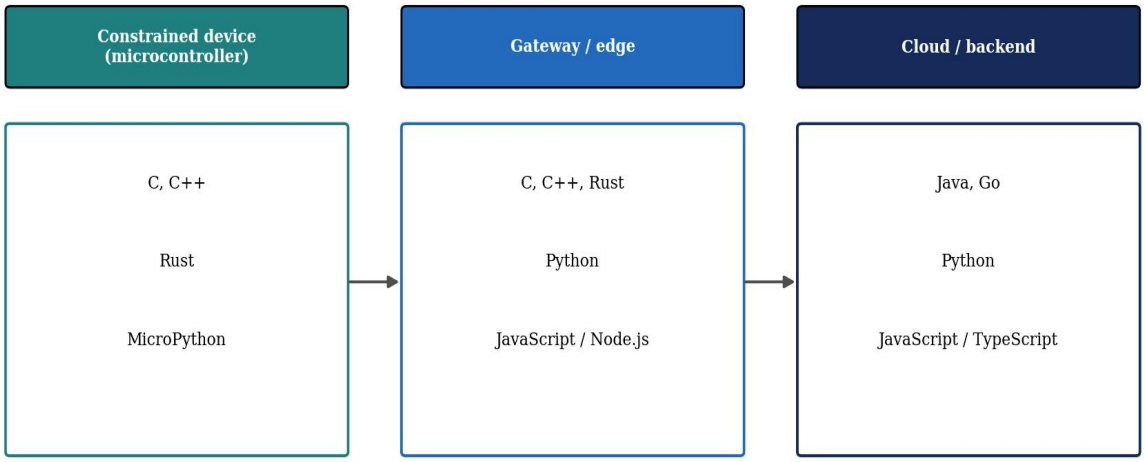

Figure 1: Programming languages across the IoT tiers

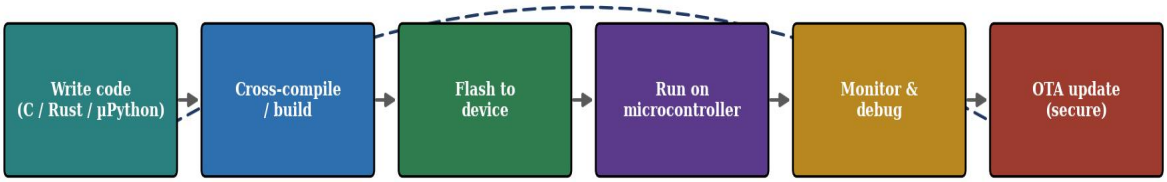

Figure 2: Embedded / IoT development and deployment workflow

Platforms and Frameworks

The software platforms and frameworks for embedded and Internet-of-Things systems are represented in Figure 3, showing the languages, the real-time operating systems and development kits, the cloud platforms, and the frameworks for machine learning on devices. This representation conveys the range of software resources on which development draws [9], [10].

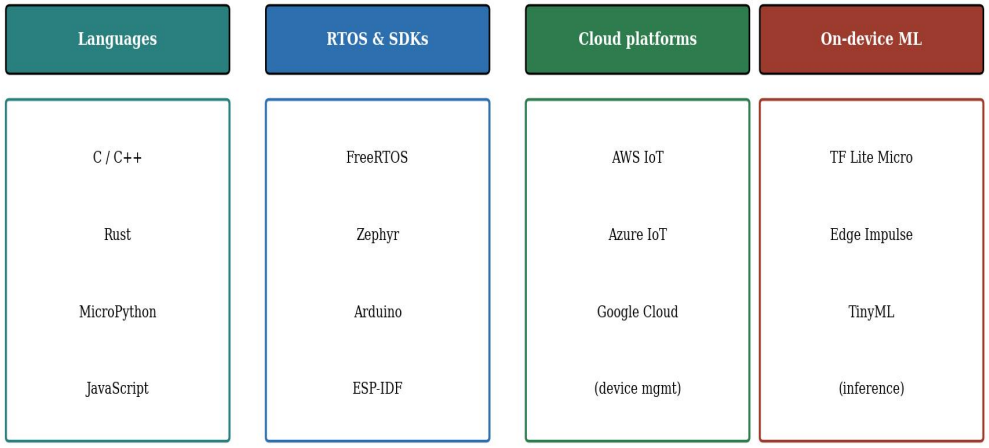

Figure 3: Software platforms and frameworks for IoT.

RESULTS AND FINDINGS

Languages

Table 1 sets out the principal languages and their uses, and Figure 1 depicts them across the tiers. The findings show that the C language offers efficiency and control for firmware, that the Rust language offers memory safety without garbage collection, that MicroPython offers ease of prototyping, and that JavaScript serves the gateway and higher tiers, so that the languages span a range from control and efficiency to productivity. The finding is that the choice of language is governed by the constraints and role of the device, the most constrained devices requiring the efficiency and control of C or the safety and efficiency of Rust, and prototyping and less constrained or higher tiers permitting the productivity of interpreted and higher-level languages [1], [3], [5].

Table 1: Languages for embedded and IoT systems.

Language	Strength	Typical use
C	Efficiency, control, small footprint	Firmware on constrained devices
C++ / Rust	Abstraction; Rust adds memory safety	Capable and safety-critical devices
MicroPython	Ease and speed of development	Prototyping, education
JavaScript / Node.js	Large ecosystem, event-driven	Gateways and higher tiers

Runtime Environments

Table 2 sets out the runtime environments, and Figure 2 depicts the workflow in which they run. The findings show that the smallest devices run bare-metal, without an operating system, for the greatest control and smallest footprint; that many devices run a real-time operating system, for task scheduling and deterministic timing with a small footprint; and that capable devices and gateways run an embedded operating system, for full networking and services. The finding is that the runtime is chosen according to the capability of the device and the demands of the application, bare-metal and real-time operating systems suiting constrained and time-critical devices and embedded operating systems suiting capable gateways, so that the runtime, like the language, is matched to the tier [6], [7].

Table 2: Runtime environments for embedded and IoT devices.

Runtime / OS	Description
Bare-metal	No operating system; greatest control, smallest footprint
Real-time operating system	Task scheduling and deterministic timing (e.g. FreeRTOS, Zephyr)
Embedded operating system	Full networking and services on capable devices (embedded Linux)

Platforms and Services

Table 3 sets out the platforms, frameworks, and services, and Figure 3 depicts them. The findings show that software development kits and toolchains support the writing and deployment of software, that cloud platforms provide device management and updates, that frameworks allow machine learning to run on constrained devices, and that secure updates and communication are essential for fielded devices. The finding is that these platforms and services complete the software environment, supporting development from prototype to production and the operation and maintenance of devices over their lifetime, and that their selection, like that of the language and runtime, involves trade-offs, including the ease of a particular platform against the risk of dependence on a single provider [9], [10].

Table 3: Platforms, frameworks, and services (representative).

Category	Examples
Development kits / toolchains	Arduino, ESP-IDF, Zephyr; cross-compilers

Category	Examples
Cloud IoT platforms	AWS IoT, Azure IoT Hub (device management, OTA)
On-device machine learning	TensorFlow Lite Micro, Edge Impulse (TinyML)
Security	Secure boot, signed OTA updates, TLS/DTLS, device identity

DISCUSSION

The principal point of this review is that the choice of programming language and software platform for an embedded or Internet-of-Things system is governed by the constraints and the role of the device, and that this choice differs across the tiers of a system. On the constrained device, where memory, processing, and energy are scarce and behaviour must be efficient and predictable, the efficiency and control of the C language, or the safety and efficiency of the Rust language, are appropriate; at the gateway and in the cloud, where resources are greater and productivity is valued, higher-level and interpreted languages are appropriate. The result is that a single Internet-of-Things system commonly combines several languages and platforms across its tiers, each matched to its role [1], [3].

The trade-off between control and efficiency, on the one hand, and safety and productivity, on the other, runs through these choices, as it does through programming generally, but it is sharpened by the constraints of embedded devices. The dominance of the C language reflects the premium on efficiency and control; the rise of the Rust language reflects the value of memory safety, especially for security, achieved without sacrificing efficiency; and the use of interpreted languages for prototyping reflects the value of rapid development where constraints permit. The runtime environment and the platform are chosen on the same basis, bare-metal and real-time operating systems for constrained and time-critical devices and richer environments for capable ones. These choices are the substance of the software engineering of embedded and Internet-of-Things systems [5], [6], [7].

The software environment of embedded and Internet-of-Things systems continues to evolve. The adoption of the Rust language for embedded software is growing, driven by the importance of memory safety and security; frameworks for machine learning on constrained devices are extending the capabilities of small devices; and the provision of secure over-the-air updates is becoming standard, as the difficulty of maintaining fielded devices and the

importance of their security are recognised. These developments are broadening and strengthening the software environment, while the underlying trade-offs, governed by constraint and role, remain. Presented together, the languages across the tiers, the development workflow, and the platforms and frameworks convey the software environment as a connected whole. Specific tools are representative of an evolving field, and certain figures are illustrative and convey structure rather than exact configurations.

LIMITATIONS

This study is a review of a rapidly evolving field and presents the languages, runtimes, and platforms for embedded and Internet-of-Things systems in summary form; the tools and frameworks named are representative examples rather than an exhaustive or final account, and the field will continue to change. The review addresses the subject at a high level and does not provide detailed technical comparison, benchmarks, or measured data. The tiers, workflow, and platforms shown in the figures are illustrative and schematic rather than exact configurations. Readers requiring authoritative and current detail are referred to the primary literature and to the documentation of particular languages, operating systems, and platforms.

FUTURE SCOPE

Further study building on this account can proceed in several directions:

- Advancement of memory-safe languages such as Rust for embedded and IoT firmware.
- Development of frameworks for machine learning on constrained devices (TinyML).
- Study of real-time operating systems and their scheduling and footprint.
- Improvement of secure over-the-air update and device-identity mechanisms.
- Investigation of portable runtimes, including WebAssembly, at the edge.
- Study of energy-efficient software and its effect on device lifetime.

Pursuing these directions would strengthen the software environment of embedded and Internet-of-Things systems.

CONCLUSION

This paper has reviewed the programming languages and software platforms used in embedded and Internet-of-Things systems. It describes the constraints of the devices and their consequences for software; the languages used across the tiers, the C language for efficiency

and control, the Rust language for memory safety without garbage collection, interpreted languages such as MicroPython for prototyping, and higher-level languages for the gateway and cloud; the runtime environments, from bare-metal through real-time operating systems to embedded operating systems; and the platforms, frameworks, and services, including development kits, cloud platforms, frameworks for machine learning on devices, and secure updates. The central conclusion is that the choice of language and platform is governed by the constraints and the role of the device, lower tiers favouring control and efficiency and higher tiers favouring productivity, so that a system commonly combines several languages and platforms across its tiers. These choices are central to the engineering of embedded and Internet-of-Things systems. Specific tools are representative of an evolving field, and certain figures presented are illustrative.

REFERENCES

1. D. E. Simon, *An Embedded Software Primer*. Boston, MA, USA: Addison-Wesley, 1999.
2. E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA, USA: MIT Press, 2017.
3. S. Klabnik and C. Nichols, *The Rust Programming Language*. San Francisco, CA, USA: No Starch Press, 2019.
4. M. Barr and A. Massa, *Programming Embedded Systems*, 2nd ed. Sebastopol, CA, USA: O'Reilly, 2006.
5. Balasubramanian *et al.*, "System programming in Rust: Beyond safety," in *Proc. HotOS*, 2017, pp. 156–161.
6. D. George *et al.*, "MicroPython: Python for microcontrollers," MicroPython Documentation, 2020.
7. R. Barry, *Mastering the FreeRTOS Real Time Kernel*. Real Time Engineers Ltd., 2016.
8. Zephyr Project, *Zephyr RTOS Documentation*. The Linux Foundation, 2023.
9. P. Warden and D. Situnayake, *TinyML: Machine Learning with TensorFlow Lite*. Sebastopol, CA, USA: O'Reilly, 2019.
10. R. Sanchez-Iborra and A. F. Skarmeta, "TinyML-enabled frugal smart objects," *IEEE Circuits Syst. Mag.*, vol. 20, no. 3, pp. 4–18, 2020.
11. K. Yaghmour *et al.*, *Building Embedded Linux Systems*, 2nd ed. Sebastopol, CA, USA: O'Reilly, 2008.

12. N. Kajwadkar and V. K. Jain, "A survey on operating systems for Internet of Things," in *Proc. Int. Conf. Comput. Commun. Netw. Technol.*, 2018, pp. 1–6.
13. O. Hahm *et al.*, "Operating systems for low-end devices in the Internet of Things: A survey," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 720–734, 2016.
14. B. Chandrasekaran *et al.*, "Secure firmware update: Challenges and solutions," *IEEE Internet Things J.*, vol. 8, no. 12, pp. 9527–9540, 2021.
15. Haas *et al.*, "Bringing the web up to speed with WebAssembly," in *Proc. ACM PLDI*, 2017, pp. 185–200.
16. Bormann, M. Ersue, and A. Keranen, *Terminology for Constrained-Node Networks, RFC 7228*. IETF, 2014.
17. J. Yiu, *The Definitive Guide to Arm Cortex-M0 and Cortex-M0+ Processors*, 2nd ed. Oxford, U.K.: Newnes, 2015.
18. S. Cass, "The top programming languages," *IEEE Spectrum*, 2023.

***Author for Correspondence**
Dr. Karan Malhotra
E-mail: karanmalhotra73783@yahoomail.com

[1]Professor, Dept. of Computer Science and Engineering, Baba Banda Singh Bahadur Engineering College
[2]Research Scholar, Dept. of Computer Science and Engineering, Baba Banda Singh Bahadur Engineering College
[3]Research Scholar, Dept. of CSE, I. K. Gujral Punjab Technical University, Jalandhar
[4]Assistant Professor, Dept. of CSE, I. K. Gujral Punjab Technical University, Jalandhar

Received Date: March 14, 2026
Accepted Date: March 29, 2026
Published Date: April 10, 2026

Citation: Dr. Karan Malhotra, Shreya Singh, Vikram Sethi, Neetu Jha. Programming Languages and Software Platforms for Embedded and Internet-of-Things Systems. International Journal of Programming Languages and Internet of Things. 2026; 2(1):