electronics

MDPI

Article

An Enhanced Generative AI-Based Assistant for IoT Application System Deployment on SEMAR Platform

Noprianto [1], Nobuo Funabiki [1,*], Htoo Htoo Sandi Kyaw [1], Komang Candra Brata [1,2], I Nyoman Darma Kotama [1] and Yan Watequlis Syaifudin [3]

[1] Department of Information and Communication Systems, Okayama University, Okayama 700-8530, Japan; noprianto@s.okayama-u.ac.jp (N.); htoohtoo@okayama-u.ac.jp (H.H.S.K.); k.candra.brata@ub.ac.id (K.C.B.); p9363bg2@s.okayama-u.ac.jp (I.N.D.K.)
[2] Department of Informatics Engineering, Universitas Brawijaya, Malang 65145, Indonesia
[3] Department of Information Technology, State Polytechnic of Malang, Malang 65141, Indonesia; qulis@polinema.ac.id
* Correspondence: funabiki@okayama-u.ac.jp

Abstract

Nowadays, *IoT application systems* are increasing in importance for improving efficiency, safety, and quality of operations, services, and products in various sectors of factories, shops, offices, and farms. However, the hurdle in deploying a new system is high for many possible users due to lack of expertise in their organizations. To solve this drawback, we have studied and developed a *SEMAR* IoT application platform that implements necessary functions to allow a new deployment by only setting the configuration file. To assist its setup, we have also offered a *generative AI*-based assistant to produce the file and sensor connection programs depending on user requirements automatically. Unfortunately, this assistant is limited to mainly sensor-level configurations. In this paper, we enhance this *generative AI*-based assistant to support the end-to-end deployment of an IoT application system on the *SEMAR* platform, involving *sensors*, *edge devices*, and the *server*. The proposal accepts inputs on hardware specifications and application-level requirements. Then, it analyses them using an *LLM*-based reasoning mechanism to determine suitable deployment strategies. After that, it automatically generates deployment artifacts, including the configuration file, the *Docker* configuration, the environment variables, and the firmware configurations tailored to the target environment. For evaluation, we conducted a within-subject user study involving 13 participants across three IoT application scenarios and two edge device platforms under controlled laboratory conditions and evaluated the usability through the *System Usability Scale (SUS)*. In addition, we measured the average time of the setup process as the deployment performance. The results showed that the proposed system achieved a high usability score (SUS = 85.00) and reduced the mean deployment time by 24.8% compared with the conventional manual process (21.80 min vs. 16.40 min).

Keywords: AI assistant; software deployment; requirement-driven deployment; IoT platform; configuration generation; deployment automation

Check for updates

Academic Editor: Hung-Yu Chien

Received: 11 July 2026
Revised: 4 September 2026
Accepted: 10 September 2026
Published: 14 September 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Currently, the number of connected *Internet of Things (IoT)* devices is expected to reach billions worldwide, with projections estimating approximately 39 billion devices by 2030 [1], driving the rapid adoption of *IoT application systems* in various sectors such as manufacturing, retail, office automation, and smart farming [2–6]. The IoT application

systems are widely utilized to improve operational efficiency, safety, and service quality through continuous environment monitoring, data collection, and intelligent decision-making. A typical *IoT application system* consists of three main components: *end devices* [7,8], *edge devices* [9,10], and *servers* [11,12]. *End devices* are directly connected to sensors and actuators for data acquisition and control, *edge devices* perform intermediate data processing and communication management, while *servers* provide data storage, analytics, and application services.

Despite enormous benefits, deploying an *IoT application system* remains a challenging task. The deployment process requires users to configure heterogeneous hardware resources, communication protocols, networking parameters, software dependencies, and runtime environments across multiple devices. Consequently, successful deployment often requires extensive technical expertise in embedded systems, computer networks, cloud computing, and *IoT middleware* technologies. Most of small organizations and potential IoT adopters cannot possess such expertise, creating a significant barrier to the adoption of *IoT application systems* to them.

To solve this limitation by simplifying the development of *IoT application systems*, we have previously developed the *Smart Environmental Monitoring and Analysis in Real Time (SEMAR)* platform [13]. The platform provides built-in services for sensor data collection, storage, visualization, and analysis, allowing users to establish *IoT application systems* without implementing core functionalities from scratch. However, although *SEMAR* significantly simplifies system development, deploying this platform with heterogeneous *end devices*, *edge devices*, and *servers* still requires substantial manual configuration, which is a big burden, particularly for users with limited technical expertise.

To reduce this deployment complexity, our previous studies introduced a *generative AI*-based assistant that automatically generated sensor connection programs and configuration files using *prompt engineering* techniques [14]. Here, we subsequently extended the assistant by incorporating knowledge extracted from *Portable Document Format (PDF)* documents [15], enabling support for newly introduced sensor devices and improving deployment guidance. Although these studies significantly simplified sensor integration, they remained limited to *end-device* configuration, requiring users to manually configure *edge devices*, application *servers*, and deployment environments.

Furthermore, deployment requirements vary considerably depending on hardware specifications, communication protocols, application scale, and operational requirements. Different deployment scenarios require different combinations of sensors, communication settings, runtime environments, and computing resources. Therefore, manually preparing the deployment artifacts for heterogeneous environments is time-consuming and can be prone to configuration errors. An intelligent deployment mechanism capable of interpreting user requirements and automatically generating deployment configurations is highly demanded.

In this paper, we propose a *generative AI*-based assistant for the end-to-end deployment of *IoT application systems* on the *SEMAR* platform. The proposed assistant employs an *LLM*-based reasoning mechanism to analyse deployment requirements, including hardware specifications, application scenarios, communication protocols, sensor scale, and transmission intervals. Based on the inferred deployment strategy, the assistant automatically generates deployment artifacts, including firmware configurations, *Docker Compose* files, environment variables, and platform configuration files required for deploying *end devices*, *edge devices*, and the *SEMAR* server. The proposed assistant is intended to support users after a general application scenario and edge device class have been chosen, focusing on simplifying the deployment of the resulting configuration rather than on defining the application's architecture or hardware requirements from first principles.

To evaluate the proposed approach, extensive deployment experiments were conducted using multiple *edge device* platforms and *IoT* application scenarios involving novice users. The deployment performance was evaluated based on the time required to complete the deployment process, while usability was assessed using the *System Usability Scale (SUS)*. The experimental results demonstrate that the proposed assistant significantly simplifies end-to-end deployment while providing a positive user experience.

The main contributions of this paper are summarized as follows:

- We propose an end-to-end *generative AI*-based deployment assistant for the *SEMAR* platform that automates deployment across *end devices*, *edge devices*, and application *servers*.
- We design an *LLM*-based deployment reasoning mechanism that transforms natural-language deployment requirements into executable deployment artifacts, including firmware configurations, environment variables, and *Docker Compose* files.
- We validate the proposed deployment assistant through evaluating deployment performance and usability, involving multiple heterogeneous *edge device* platforms, *IoT* application scenarios, and novice users.

The remainder of this paper is organized as follows: Section 2 reviews related work on *IoT application system deployment* and *AI*-assisted deployment approaches. Section 3 presents the *generative AI*-based deployment assistant. Section 4 describes the deployment methodology and experimental setup. Section 5 presents and discusses the experimental results. Finally, Section 6 concludes the paper and outlines future research directions.

2. Related Work

In this section, we review related work on *IoT application system deployment* and the use of *AI* to assist in deployment processes.

2.1. Software Deployment Automation

Various approaches have been proposed to automate the deployment of software systems, including *IoT application systems*. These approaches typically involve the use of configuration management tools, containerization technologies, and orchestration frameworks to streamline the deployment process.

In [16], Oliveira et al. proposed *IoTDeploy* as a *CI/CD*-based framework for deployment and service orchestration across *mist*, *fog*, *edge*, and *cloud* environments. This framework supports dynamic service migration and was evaluated using a *smart irrigation* application. Experimental results showed that the proposed approach enables reliable deployment and service migration without application downtime or data loss.

In [17], Bhise and Rangisetti proposed *Security Aware Serverless Application Partitioning (SASAP)* for deploying *serverless applications* across *edge* and *cloud* environments. The proposal considers computational requirements, security constraints, and resource utilization during service deployment. Experimental results demonstrated improvements in deployment security, computational requirement fulfilment, and resource utilization compared to existing approaches.

In [18], Rahman and Candra proposed *PERISAI* as a remote deployment system for *IoT* applications running on heterogeneous and resource-constrained devices. The system utilizes *Kubernetes* with the lightweight *K3s* distribution to enable platform-agnostic deployment and device management. Experimental results showed that *PERISAI* successfully deployed a *blink LED* application on two *Raspberry Pi* devices within 4.7 s through a single-click deployment mechanism.

In [19], Mafeni and Kim proposed an *edge-based framework* for automated *IoT device registration* and heterogeneous *IoT application deployment*. The framework supports automatic device registration and application deployment in large-scale *IoT* environments. Evaluation

using a *smart irrigation* use case demonstrated that the proposed approach is scalable and feasible for managing heterogeneous *IoT* devices and applications.

In [20], Hosseini Shirvani and Ramzanpoor formulated *IoT application deployment* on *fog computing* infrastructure as a multi-objective optimization problem considering power consumption, bandwidth utilization, reliability, and *Quality of Service (QoS)*. A *Multi-Objective Genetic Algorithm (MOGA)* was proposed to optimize module placement across fog nodes. Experimental results showed that the proposed approach outperformed several existing methods in reducing power consumption and link wastage rates.

In [21], Du Plessis and Correia presented a comparative study of *microservices* and *monolithic* architectures for *constrained-device IoT deployments*. The comparison evaluated power consumption, runtime performance, and memory usage using *Go*, *Python*, and *C++* on resource-constrained IoT devices. Experimental results showed that the *monolithic* architecture outperformed the *microservices* approach for small-scale deployments, highlighting the importance of selecting an appropriate software architecture to improve resource utilization in IoT deployment environments.

In [22], Mohamed et al. presented a deployment framework for *Intelligent Smart Cities (ISC)* that integrates *Artificial Intelligence (AI)* and *Software-Defined Networking (SDN)* technologies. The proposed framework employs AI-based machine learning to improve network management and strengthen protection against *Denial-of-Service (DoS)* attacks in IoT-enabled smart city environments. The study demonstrated the potential of AI and SDN integration to support secure and scalable smart city deployments while identifying several deployment challenges.

Overall, previous studies have shown that deployment automation can improve reliability, scalability, resource utilization, and deployment efficiency in heterogeneous *IoT* environments. However, deploying complete *IoT application systems* still requires substantial technical expertise to analyse deployment requirements and configure *end devices*, *edge devices*, and application *servers*.

2.2. AI in DevOps/AIOps

The use of *Artificial Intelligence (AI)* in software development and operations, often referred to as *AIOps*, has gained significant attention in recent years. *AIOps* aims to leverage AI techniques to automate and enhance various aspects of software development lifecycle, including deployment.

In [23], Sahoo et al. presented a systematic survey of *prompt engineering* techniques for *Large Language Models (LLMs)*, categorizing methods such as zero-shot prompting, few-shot prompting, and chain-of-thought reasoning according to their application domains and methodological foundations. The survey highlights how such techniques guide *LLM* behaviour and improve task alignment without modifying model parameters, providing a relevant foundation for the *LLM*-based reasoning mechanisms employed in *AIOps* and deployment-assistance systems.

In [24], Wang et al. proposed a collaborative deployment framework for *edge Large Artificial Intelligence Models (LAMs)* in heterogeneous *edge computing* environments. The framework combines adaptive model decomposition for collaborative training and a microservice-based inference architecture to reduce communication overhead, improve resource utilization, and lower inference latency. The proposed approach enables real-time intelligent services and facilitates diverse *IoT* applications at the network edge.

In [25], Akram et al. proposed *iGenEdge* as an intelligent resource management framework for deploying *Generative AI (GenAI)* applications on *IoT edge devices*. The framework utilizes collaborative edge computing to dynamically allocate resources and improve deployment efficiency under varying computational demands. Experimental results using

lightweight language and vision models on *Raspberry Pi* devices showed significant reductions in context-switching and memory overhead.

In [26], Veiga et al. analysed the suitability of existing *AI deployment platforms* for supporting *cognitive IoT* applications, where sensor devices adapt their behaviours based on machine learning outcomes. A person-counting application in a skiing area was used as a case study to evaluate the integration of *IoT* devices with AI deployment platforms. This study identified several limitations, including the need for better IoT integration, explicit data-flow management, reusable cognitive *IoT* templates, and improved support for distributed application components.

In [27], Shen et al. proposed an intelligent deployment framework for *Reconfigurable Intelligent Surfaces (RISs)* in next-generation wireless networks. The framework employs *federated multi-agent reinforcement learning* to optimize *RIS* deployment parameters, including position, height, and orientation, using *AGV*-assisted *RIS* platforms. Experimental results demonstrated that the proposed approach achieved transmission throughput of up to $980 Mbps$ while enabling rapid and efficient *RIS* deployment.

In [28], Rafique and Marsden proposed a cloud-native system for automated *Large Language Model (LLM)* deployment and evaluation in cloud environments. The system automates infrastructure provisioning and model deployment while providing a lightweight evaluation framework based on the *LLM-as-a-Judge* approach. The proposed framework aims to simplify *LLM* deployment, reduce operational complexity, and support systematic comparison of different models.

In [29], Soureya et al. proposed an *AI-based software development framework* integrating *Artificial Intelligence (AI)*, *Software Product Lines (SPL)*, and *software aging principles* to support adaptive software evolution throughout the *Software Development Life Cycle (SDLC)*. A connected home case study demonstrated that the proposed framework improves software adaptability and sustainability through intelligent automation.

In [30], Hanzelik et al. proposed an *edge-computing* and *machine-learning*-based framework for developing and managing the lifecycle of *software sensors* in Industrial IoT environments. The framework enables continuous model monitoring, validation, maintenance, and version control, improving the accuracy and reliability of software sensor models.

Overall, *AI*-based deployment approaches have shown promising capabilities in automating deployment processes, optimizing resource allocation, and improving system performance in cloud and *IoT* environments. However, most studies focus on infrastructure optimization and resource management rather than assisting non-expert users in deploying complete *IoT application systems*.

2.3. Resource-Aware System

Various studies have explored resource-aware system design and deployment to optimize performance, energy efficiency, and cost in heterogeneous computing environments.

In [31], Giuliano et al. proposed an application-aware gateway deployment strategy for large-scale *IoT* networks using graph theory and graph signal processing techniques. The approach integrates application requirements with the network topology to optimize the gateway placement in *IoT* infrastructures. Evaluation using a smart water management case study demonstrated improvements in the network coverage, data extraction performance, and energy efficiency, with energy consumption reduced by up to 70%.

In [32], Li et al. investigated the *Service Function Chain (SFC)* deployment problem in *NFV-enabled* cloud-edge environments. They formulated the deployment problem as an *Integer Linear Programming (ILP)* model and proposed a *Resource-Aware Deployment Algorithm (RADA)* to optimize the resource utilization and deployment cost. Experimental

results showed that *RADA* improved the resource availability and increased the average service acceptance rate by 15% compared to existing approaches.

In [33], Faraji-Mehmandar et al. proposed a resource allocation and service placement approach for *IoT* applications in *fog computing* environments. The approach employs a *Harris Hawks Optimization*-based meta-heuristic algorithm to optimize the application placement while considering resource availability and *Quality of Service (QoS)* requirements. Simulation results demonstrated improvements in resource utilization and service acceptance rates, while reducing service delay and energy consumption compared to existing methods.

In [34], Islam et al. proposed an automatic service and resource discovery mechanism for on-the-fly deployment of *nanoservices* in *IoT edge computing* environments. The proposed approach enables the efficient resource matching and service deployment on local *IoT* nodes using a decentralized microservice-based platform. Experimental results on *Raspberry Pi* devices demonstrated that containerized deployment provides the better resource efficiency, while the service discovery and deployment process can be completed within 6–17 s.

In [35], Atlam et al. proposed a resource-aware framework for deploying *deep learning (DL)* applications in *fog computing* environments. The framework combines a *Binary Search-Inspired Recursive (BSIR)* optimization algorithm, memory-aware deployment analysis, and *Retrieval-Augmented Generation (RAG)* to dynamically optimize the task placement between fog nodes and the cloud. Experimental results demonstrated significant improvements in the memory utilization, resource efficiency, and decision-making speed compared to conventional optimization approaches.

In [36], Abdelmoumen et al. proposed a *resource-aware service-oriented framework* for *IoT-based System-of-Systems (SoS)* environments. The framework supports adaptive service orchestration and efficient resource management to improve the scalability, reliability, and interoperability of heterogeneous IoT systems.

In [37], Ashwin et al. proposed the *Resource-Aware Energy-Efficient Detector (RAEED)*, an edge–fog framework for real-time object detection in *Industrial IoT (IIoT)* environments. Experimental results demonstrated that the proposed framework improves detection accuracy while reducing latency and energy consumption under resource-constrained deployment conditions.

Overall, resource-aware approaches have significantly improved deployment efficiency and resource utilization in heterogeneous *IoT* environments. However, most studies focus on optimizing resource allocation and service placement, while limited attention has been given to assisting non-expert users in determining deployment configurations based on application requirements and available hardware resources.

3. System Architecture

In this section, we present the architecture of the proposed *generative AI*-based assistant for requirement-driven system setup in the *SEMAR* IoT application platform. This architecture consists of several key components that work together to analyse user requirements and generate deployment configurations. We note that hardware and software recommendations are derived from the extracted deployment profile (e.g., use case, sensor count, transmission interval) using heuristic and *LLM*-based reasoning, rather than a quantitative resource-estimation model based on measured CPU, memory, network, or latency requirements.

3.1. Overview of Proposed System

The proposed system is designed to assist non-expert users in deploying complete *IoT application systems* through a web-based *AI deployment assistant*. This system consists of a

Web Application, an *AI Assistance Module*, a *Database*, and a *Docker Hub* repository, as illustrated in Figure 1. Unlike conventional deployment tools that rely solely on predefined deployment templates, the proposed system employs an *LLM*-based reasoning mechanism to dynamically analyse deployment requirements and generate tailored recommendations. The final deployment artifacts (e.g., *Docker Compose* files, firmware packages) are produced by populating predefined, manually verified templates with parameters derived from the *LLM*-based analysis, rather than being freely generated by the *LLM*.

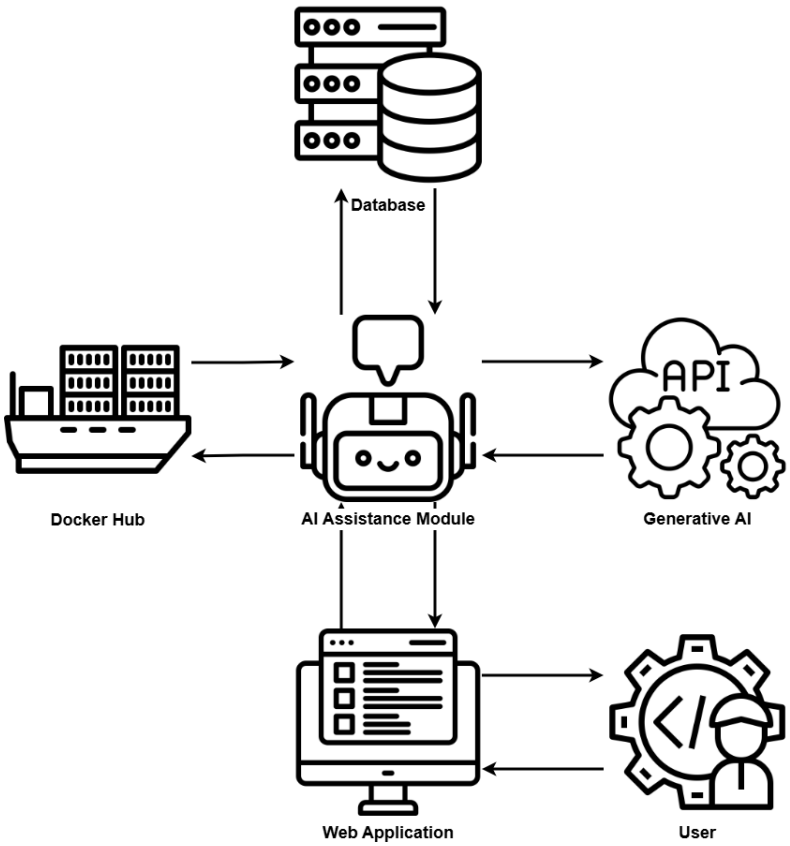


Figure 1. System architecture of proposed generative AI-based deployment assistant.

The deployment process begins when a user provides the deployment requirement through the *Web Application*. The requirement may include the target use case, the number of sensors, data transmission intervals, communication protocols, and other application-specific parameters. These user inputs are then forwarded to the *AI Assistance Module*, which acts as the core component of the proposed system.

The *AI Assistance Module* is responsible for analysing the deployment requirement, communicating with the *Generative AI* service, managing user sessions, and generating deployment artifacts. In addition, the module retrieves the required deployment packages and container images from *Docker Hub*. User interaction history, deployment configurations, and generated artifacts are stored in the *Database* to support deployment management and future reuse.

Based on the deployment requirement, the assistant generates the deployment recommendations and artifacts for three deployment layers: *end devices*, *edge devices*, and the *SEMAR IoT application server*. The generated artifacts include firmware configurations, deployment packages, *Docker Compose* files, environment variables, and deployment instructions. For server and edge deployment, containerized packages are utilized to provide platform-independent deployment and simplify installation procedures.

After receiving the deployment recommendations and generated artifacts, users can deploy the *SEMAR* IoT application system, configure edge devices, and provision end devices according to the generated instructions. As a result, the proposed system reduces the deployment complexity and minimizes the technical expertise required to build and operate an *IoT application system*.

3.2. Deployment Workflow

The detailed workflow of the proposed system is illustrated in Figure 2. This figure depicts the interactions among the user, the *Web Application*, and the *AI Assistance Module* within a single deployment cycle. The process begins when the user provides the *IoT* application requirements, followed by requirement analysis, architecture and hardware recommendations, and the automatic generation of deployment artifacts. The resulting artifacts are then used to support the end-to-end deployment across *end devices*, *edge devices*, and the *SEMAR IoT application server*. Each stage is executed sequentially, while the generated outputs from the previous stage are reused as inputs for the following deployment stage.

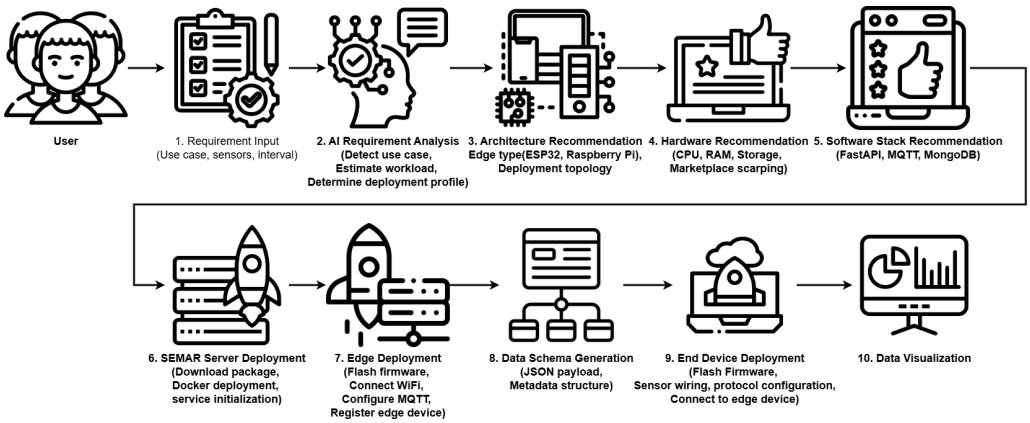


Figure 2. User flow for deployment.

3.3. LLM-Based Deployment Reasoning

The proposed *AI Assistant* employs a *Large Language Model (LLM)* as the reasoning engine throughout the deployment workflow, from the *Requirement Analysis* stage to the *End Device Deployment* stage, as illustrated in Figure 2. Rather than being used only for the requirement extraction, the *LLM* supports the multiple deployment stages by interpreting user requests, generating deployment recommendations, and producing deployment artifacts.

Figure 3 illustrates the reasoning process of the proposed system. At each deployment stage, users interact with the *Web Application* through a chat-based interface by providing the deployment requests in natural language. The requests are then forwarded to the *AI Assistance Module*, which constructs a stage-specific prompt template according to the current deployment task. The generated prompt is subsequently submitted to the *LLM*, which produces structured outputs in *JSON* format, including deployment profiles, hardware and software recommendations, deployment justifications, and deployment artifacts.

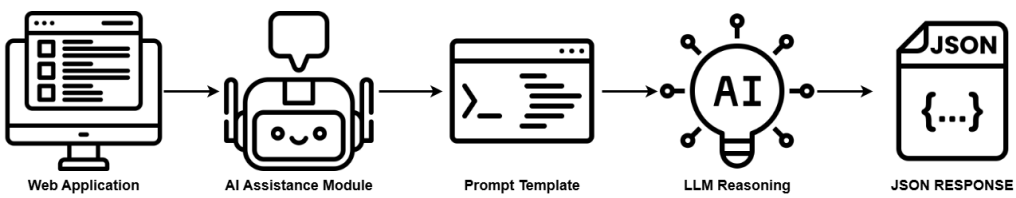


Figure 3. LLM-based deployment reasoning process.

3.4. Requirement Analysis Stage

The first stage of the deployment process is *Requirement Analysis*. To avoid ambiguity, this paper adopts the following terminology consistently across all tables, equations, prompts, and firmware descriptions. A *physical module* refers to a hardware component installed on the end device (e.g., one DHT11 module). A *measurement channel* (or *sensor channel*) refers to an individual data stream produced by a module (e.g., a DHT11 module produces two channels: temperature and humidity). A *transmitted message* refers to the data packet sent from the end device to the edge device per transmission interval; when `payload_mode = combined`, one message contains all channels from all modules per device per interval. The variable s in Section 3.5 denotes the number of *measurement channels* per device (i.e., `sensor_per_device` in the deployment profile), and $N = \max(d \cdot s, 1)$ denotes the total number of measurement channels across all devices. In this stage, users describe their deployment requirements through the *Web Application* using natural language. The inputs may include the target *IoT* application, such as *smart farming*, *smart home*, or *air quality monitoring*, together with deployment-related information, including the number of sensors, communication protocol, data transmission interval, payload size, and other application-specific requirements.

The *AI Assistance Module* first performs a lightweight analysis using pattern-based techniques, such as *Regular Expressions (Regex)*, to extract explicitly defined parameters. This preliminary step reduces unnecessary *LLM* inference for the deployment requests containing straightforward information. If the extracted information is incomplete, ambiguous, or inconsistent, the remaining input is forwarded to the *Generative AI* service for semantic interpretation.

The *LLM* analyses the user requirements using a predefined prompt template and transforms the natural-language description into a structured deployment profile represented as a *JSON* object. The generated deployment profile contains the deployment parameters required by subsequent stages, including the application use case, sensor types, communication protocol, transmission interval, payload configuration, and target deployment environment. This structured representation serves as the primary input for architecture recommendation, hardware and software recommendation, and deployment artifact generation.

Rather than relying solely on the prompt instructions, the *AI Assistance Module* applies a code-level post-processing step that parses, validates, and, where necessary, corrects the returned *JSON* object (e.g., re-parsing malformed responses, enforcing the required schema, and applying the sensor-count and sensor-type consistency rules) before the profile is passed to subsequent stages.

3.5. Architecture and Resource Recommendation

After the deployment requirements have been analysed, the *AI Assistance Module* generates the recommendations for the deployment architecture, hardware resources, and software stack based on the deployment profile obtained during the *Requirement Analysis* stage.

The first recommendation concerns the selection of the *edge device*, such as a *Raspberry Pi* or an *ESP32*. If a *Raspberry Pi* is selected, the system recommends either *HTTP* or *MQTT* as the communication protocol between the *end device* and the *edge device*. In contrast, deployments using an *ESP32* employ *HTTP* communication.

Next, the system estimates the expected workload from the deployment profile and recommends hardware accordingly. Let d denote the number of *devices*, s the number of measurement channels per device (*sensor_per_device*; e.g., a DHT11 module contributes $s = 2$ channels: temperature and humidity), τ the transmission interval in seconds, and p

the payload size in kilobytes. The total number of sensors N and the estimated throughput θ (KB/s) are computed as

$$N = \max(d \cdot s,\ 1) \tag{1}$$

$$\theta = \mu \cdot p \tag{2}$$

The throughput estimate θ is reported to the user as an informational summary only and does not enter the hardware classification logic; only μ and L are used for decision-making. The estimated message rate μ (messages per second) depends on the payload mode:

$$\mu = \begin{cases} d \cdot s\ /\ \tau, & \text{if } \texttt{payload_mode} = \texttt{individual} \\ d\ /\ \tau, & \text{if } \texttt{payload_mode} = \texttt{combined} \end{cases} \tag{3}$$

The estimated message rate μ is then mapped to a discrete workload level L using fixed thresholds:

$$L = \begin{cases} \text{low}, & \mu < 1 \\ \text{medium}, & 1 \leq \mu < 10 \\ \text{high}, & \mu \geq 10 \end{cases} \tag{4}$$

These thresholds are engineering heuristics informed by reported *MQTT* broker performance on low-cost single-board computers: empirical benchmarks indicate that platforms comparable to the *conservative* tier (e.g., *Raspberry Pi* running *Mosquitto*) can sustain hundreds of concurrent *MQTT* connections under typical *IoT* sensor payloads [38], suggesting that the capacity boundaries between tiers broadly correspond to the threshold values used ($\mu < 1$, $1 \leq \mu < 10$, $\mu \geq 10$ messages/s). However, the specific threshold values have not been validated through direct benchmarking of the catalogued devices under the *SEMAR* deployment stack; they are declared as design parameters of the current implementation, and systematic benchmarking linking threshold values to measured device throughput is identified as a direction for future work.

The workload level L, together with other extracted requirement parameters (e.g., use case, budget preference), is passed to the *LLM*, which selects a hardware *profile* $\pi \in \{\text{conservative}, \text{general}, \text{ai_edge}\}$. A deterministic fallback rule accompanies this selection: $L = \text{low} \rightarrow \pi = \text{conservative}$; $L = \text{medium} \rightarrow \pi = \text{general}$; $L = \text{high} \rightarrow \pi = \text{ai_edge}$. The *LLM* may override this default using contextual information (use case, budget preference). Within the selected profile, three hardware options (minimum, intermediate, premium) are retrieved from the fixed catalogue in Table 1, and the *LLM* selects one as the primary recommendation, using the prompt in Listing A4. This mechanism is a heuristic, context-driven classification rather than a quantitative capacity-versus-demand optimisation; users may override the primary recommendation.

Table 1. Fixed hardware catalogue used for recommendation (profile π and tier index i).

Profile	Tier	Device	CPU	RAM (GB)	Storage (GB)	Cost (JPY)	Power (W)
conservative	Minimum	Raspberry Pi 5 (Entry)	ARM A76 4-core	4	64	8000–12,000	5
conservative	Intermediate	Orange Pi 5+ (Balanced)	ARM A76 4-core RK3588	8	128	12,000–18,000	10
conservative	Premium	Mini PC Intel N100	Intel N100 4-core	8	128	18,000–25,000	10
general	Minimum	Intel NUC 12 (Entry Edge)	Intel Core i3-1215U	8	128	28,000–38,000	8
general	Intermediate	Intel NUC 13 (Balanced)	Intel Core i5-1335U	16	256	52,000–72,000	15
general	Premium	Intel NUC 14 (Premium)	Intel Core i7-1485H	32	512	85,000–120,000	28

Table 1. *Cont.*

Profile	Tier	Device	CPU	RAM (GB)	Storage (GB)	Cost (JPY)	Power (W)
ai_edge	Minimum	Jetson Orin Nano	ARM A78AE 6-core	8	128	35,000–48,000	15
ai_edge	Intermediate	Jetson Orin NX	ARM A78AE 8-core	16	256	65,000–85,000	25
ai_edge	Premium	Jetson Orin AGX	ARM A78AE 12-core	32	512	180,000–240,000	60

Finally, the system recommends the software components required for the selected *edge device*. These recommendations include the operating system, communication services, and supporting software needed for deployment. Since the recommendations are generated from the deployment requirements, different *IoT* applications may produce different deployment configurations while following the same deployment workflow.

3.6. Deployment Artifact Generation

The *Deployment Artifact Generation* stage is responsible for generating the artifacts required for deployment. During the *SEMAR Server Deployment* phase, the system generates installation packages, *Docker Compose* configurations, and deployment instructions that can be followed by users until the services become operational.

The system then generates the deployment artifacts for the *edge device*. For *Raspberry Pi*-based deployments, the generated artifacts include the installation packages and *Docker*-based deployment configurations. For *ESP32*-based deployments, the system generates firmware packages that can be directly flashed to the device through the *Web Application*.

After the *edge device* deployment stage, the system generates a *Data Schema* that defines the structure of the data transmitted from the *end device* to the *edge device*. This information ensures compatibility and consistency in data exchange among system components.

The next stage is *End Device Deployment*. In this stage, the system generates the firmware and sensor connection instructions according to the application requirements. Users can directly flash the generated firmware to the target device through the *Web Application* without requiring a programming editor or third-party flashing tools.

Figure 4 illustrates the firmware flashing mechanism provided by the proposed system. Users only need to select the target device type, specify the communication port, and establish a connection with the device. After the flashing process is completed, users can configure the device to communicate with the *edge device* and the *SEMAR* IoT application server. This approach simplifies the deployment process and reduces the need for advanced programming skills.

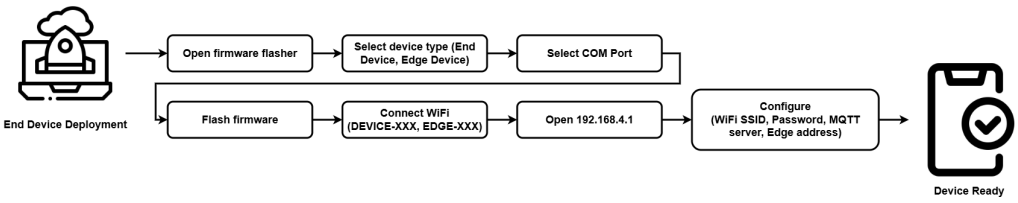


Figure 4. AI-assisted firmware flashing mechanism through Web Application.

Rather than being freely generated by the *LLM*, each artifact (e.g., *Docker Compose* configurations and firmware packages) is produced by populating a predefined, manually verified template with parameters derived from the deployment profile obtained during the *Requirement Analysis* and *Architecture and Resource Recommendation* stages (e.g., selected communication protocol, sensor types, and use case). This template-based approach confines the *LLM*'s role to determining the correct parameter values rather than generating the artifact structure itself, reducing the risk of structurally invalid or unexpected artifact content.

3.7. Worked Example: From User Request to Deployment Artifacts

To clarify the distinct roles of *rule-based processing*, *LLM-based reasoning*, *user selection*, and *template-based generation* within the deployment pipeline, Table 2 traces a single representative request through each stage of the workflow. The user submitted the following natural-language request for an *Air Quality Monitoring* deployment: *"Monitor air quality in my room with 4 sensors."*

Table 2. Worked example tracing a single deployment request through each stage of the pipeline, indicating the type of decision involved at each stage.

Stage	Output	Decision Type
Regex pre-processing	Detects "4 sensors"; use case and remaining fields incomplete	Rule-based
LLM requirement extraction	Deployment profile in JSON (usecase: air_quality; sensor_per_device: 4; sensor_types: [humidity, temperature, gas, co]; protocol: mqtt (default); payload_kb: 1; interval_seconds: 30; payload_mode: combined; storage: mongodb; dashboard: node-red; os: linux)	LLM-based
Code-level validation & correction	JSON parsed; schema and sensor-count/type rules verified and corrected if needed	Rule-based
Edge device selection	User selects *Raspberry Pi* (edge_type: raspberry_pi), guided by an LLM-generated explanatory message	User selection (LLM-assisted)
Protocol confirmation	User confirms *MQTT* for end device to edge device communication, guided by an LLM-generated explanatory message; edge-to-server communication fixed to *HTTP* (REST API)	User selection (LLM-assisted)/Rule-based
Hardware and software recommendation	Hardware tier and software stack recommended based on the deployment profile	LLM-based (with rule-based defaults)
Deployment artifact generation	*Docker Compose* configuration for the *SEMAR*-side deployment, populated with the confirmed protocol, sensor types, and use case	Template-based (parameters from prior stages)

This example illustrates that, while the *LLM* plays a central role in interpreting natural-language input and producing the structured deployment profile, several other decisions in the pipeline are rule-based (e.g., regex pre-processing, JSON validation and correction, fixed edge-to-server protocol), user-driven (e.g., edge device and protocol selection, albeit guided by LLM-generated explanations), or template-based (e.g., final artifact generation). The *LLM*'s contribution is therefore best characterized as interpreting ambiguous or incomplete natural-language requirements into a structured, schema-conforming representation, rather than autonomously generating the final deployment artifacts.

3.8. Integration with SEMAR Platform

The *SEMAR* IoT application server serves as the central component of the system, responsible for receiving, storing, processing, and visualizing the data transmitted from *end devices* through *edge devices*. *SEMAR* is an IoT application platform developed in our prior work [13], providing built-in functions for sensor data collection, storage, real-time visualization through dashboards, and basic data analysis. Therefore, the integration between the proposed deployment assistant and *SEMAR* is essential for supporting the end-to-end deployment of a new *IoT application system*.

The deployment of *SEMAR* is automatically performed using the deployment artifacts generated by the *AI Assistance Module*. The system produces the *Docker*-based deployment configuration, enabling platform-independent installation across different operating systems. Once a deployment is completed, *edge devices* communicate with the *SEMAR* IoT application server through *REST APIs* to transmit sensor data and receive the required configuration. As a result, all components can be integrated seamlessly, enabling the operation of a complete *IoT application system*. While the *SEMAR* functions used in this work build on our prior work [13], the *SEMAR* application server itself remains an active area of ongoing development, particularly toward improving its scalability, reliability, and multi-tenant support as an *IoT* application platform, as discussed in Section 6.

4. Experimental Design

In this section, we describe the methodology used to evaluate the proposed *generative AI*-based deployment assistant. The evaluation focuses on assessing both the deployment performance and usability of the system.

4.1. Deployment Scenario

To evaluate the proposed *AI Assistant*, several deployment scenarios were designed, as shown in Table 3. Each scenario represents a different *IoT* application with varying requirements in terms of the application use case, number of sensors, data transmission interval, and edge device type.

Table 3. Deployment scenarios for evaluation. Sensor count refers to the number of measurement channels (e.g., a DHT11 module contributes two channels: temperature and humidity).

Scenario	Use Case	Sensors	Edge Device
Scenario 1	Smart Home	4	Raspberry Pi/ESP32
Scenario 2	Smart Farming	5	Raspberry Pi/ESP32
Scenario 3	Air Quality Monitoring	4	Raspberry Pi/ESP32

Throughout this paper, the term *sensor* refers to a measurement channel rather than a physical module. For example, a *DHT11* module contributes two sensor channels (temperature and humidity) and is therefore counted as two sensors. This convention is consistent with the `sensor_types` field in the deployment profile (Section 3.4) and the firmware generation stage. A *use case* represents the target *IoT* application to be deployed, while the number of sensors and transmission interval determine the expected workload generated by the system. These parameters directly affect the volume of data transmitted through the network and influence metrics such as message rate and throughput. Therefore, they are used as the input for the *AI Assistant* to inform hardware and software recommendations through heuristic and *LLM*-based reasoning, rather than a quantitative resource-estimation model based on measured CPU, memory, network, or latency data.

Several types of *edge devices*, including *Raspberry Pi* and *ESP32*, were considered in the evaluation scenarios. The selected *edge device* is responsible for performing data aggregation and filtering before forwarding data to the *SEMAR* IoT application server. By varying the workload characteristics and edge device types, the proposed system is evaluated under diverse deployment conditions commonly found in real-world *IoT* environments.

4.2. User Study

A user study was conducted to evaluate the effectiveness of the proposed *AI Assistant* in supporting the deployment of *IoT* application systems. This study involved 13 participants (11 doctoral students and 2 master's students), all recruited from an IT-

related research laboratory covering diverse research areas including Technology-Enhanced Learning, Information Retrieval, Gamification, Networking, and Distributed Systems. Participants were classified as *IoT deployment novices*: none had prior experience completing an end-to-end IoT application deployment encompassing edge device configuration, firmware generation, and server setup. While some participants reported general technical familiarity (e.g., 5 out of 13 had CLI experience, 6 out of 13 had Docker experience), this does not constitute IoT deployment expertise; the majority had limited or no experience in the areas most directly relevant to the manual deployment workflow, specifically microcontroller wiring (4/13, 31%) and prior IoT component experience such as sensor or microcontroller use in isolation (e.g., Arduino or Raspberry Pi projects) (6/13, 46%). However, none of the 13 participants had completed an end-to-end IoT application deployment prior to this study. The study employed a within-subject, counterbalanced design: each participant completed both the manual and AI-assisted deployment conditions, with task order alternated across participants to control for potential learning effects. Each participant was assigned to one of the three deployment scenarios (Smart Home, Smart Farming, or Air Quality Monitoring) and completed both the manual and AI-assisted deployment conditions for that scenario. The scenario assignment resulted in five participants for Smart Home, five for Smart Farming, and three for Air Quality Monitoring. The complete participant profiles, including research area, prior technical experience, and assigned scenario, are summarized in Table 4.

Table 4. Participant profiles and scenario assignments.

ID	Status	Research Area	Scenario	CLI	Docker	IoT_comp *	MCU	Order
P01	Doctoral	Information Retrieval & Digital Library Systems	Smart Home	Yes	Yes	No	No	Manual first
P02	Doctoral	Database & Information Systems	Smart Home	No	No	No	No	Manual first
P03	Doctoral	Technology-Enhanced Learning/Ed. Tech.	Smart Home	No	Yes	Yes	No	Manual first
P04	Doctoral	Gamification & Exercise Games	Air Quality	No	No	Yes	Yes	Manual first
P05	Doctoral	Electronics & Thermal Infrared	Air Quality	No	No	Yes	Yes	AI first
P06	Doctoral	Technology-Enhanced Learning/Ed. Tech.	Air Quality	No	No	No	No	Manual first
P07	Doctoral	Technology-Enhanced Learning/Ed. Tech.	Smart Farming	No	No	No	No	Manual first
P08	Doctoral	Information Retrieval & Digital Library Systems	Smart Farming	No	No	No	No	AI first
P09	Doctoral	Networking & Communication	Smart Farming	Yes	Yes	Yes	Yes	AI first
P10	Doctoral	Gamification & Exercise Games	Smart Farming	Yes	No	No	No	Manual first
P11	Master	Distributed Systems & Cloud Computing	Smart Home	Yes	Yes	Yes	No	AI first
P12	Master	Technology-Enhanced Learning/Ed. Tech.	Smart Farming	No	Yes	No	No	AI first
P13	Doctoral	Networking & Communication	Smart Home	Yes	Yes	Yes	Yes	AI first

* IoT_comp = Prior IoT component experience (e.g., sensor or microcontroller use in isolation, such as Arduino or Raspberry Pi projects); this does not constitute end-to-end IoT application deployment experience. None of the 13 participants had previously completed a full end-to-end IoT deployment encompassing edge device configuration, firmware generation, and server setup simultaneously.

This focus on IoT deployment novices was motivated by the primary objective of the proposed system, which is to reduce the technical barriers associated with *IoT* system deployment and enable non-expert users to perform deployment tasks with minimal

assistance. During the evaluation, each participant completed two deployment tasks: one following the manual deployment workflow (Table 12) and one using the proposed *AI Assistant* (Table 9), with task order counterbalanced across participants.

The deployment process included the configuration of *end devices*, *edge devices*, and the *SEMAR IoT application server*. The deployment completion time was recorded for each participant, while the usability of the proposed system was assessed using the *System Usability Scale (SUS)* questionnaire after the deployment task was completed.

4.3. Experimental Setup

The experiments were conducted in a laboratory environment to provide a controlled setting for evaluating the proposed *AI Assistant*. The proposed deployment framework consists of a *Web Application*, an *AI Assistance Module*, a database system, and supporting deployment services. The server environment used to host these components is summarized in Table 5.

Table 5. Experimental environment used for AI Assistant and *SEMAR* server.

Component	Specification
CPU	AMD Ryzen 7 3800X (8 Cores, 16 Threads, 3.9 GHz)
Memory	64 GB RAM
GPU	NVIDIA GeForce RTX 2080 (8 GB)
Operating System	Ubuntu 24.04.1 LTS (Noble)
Kernel Version	Linux 6.8.0-124-generic
Python	3.12.7
LLM	Llama 3.1 8B (Ollama)
LLM Runtime	Ollama 0.12.5
Temperature	0.15
Top-*p*	0.9
Num-predict	2048
Docker	27.3.1
FastAPI	0.133.1
Uvicorn	0.40.0
MongoDB	8.0
Browser	Google Chrome 148.0.7778.168 (64-bit)

The proposed *AI Assistant* utilizes the *Llama 3.1 8B* large language model deployed locally on the experimental server through the *Ollama* framework. LLM generation was configured with a temperature of 0.15, top_p of 0.9, and num_predict of 2048. These parameters were kept fixed throughout the experiments, except for temperature, which was varied in the sensitivity analysis. The temperature was selected to favor reproducible outputs, while top_p = 0.9 and num_predict = 2048 were used as fixed generation settings to provide sufficient sampling flexibility and output capacity for the structured requirement-analysis task. This model is responsible for requirement analysis, resource estimation, architecture recommendation, hardware recommendation, and deployment artifact generation. Running the model locally enables the deployment assistance without relying on external cloud-based AI services. The *Web Application* was accessed using *Google Chrome* version 148.0.7778.168 (64-bit), which provides support for browser-based device flashing through the *Web Serial API*. This functionality allows users to upload firmware directly to *ESP32*-based devices without requiring external development tools or integrated development environments.

To evaluate the deployment process under different hardware configurations, several *edge devices* were utilized. The specifications of the *Raspberry Pi*-based devices are presented in Table 6, while the specifications of the *ESP32-WROOM-32E*-based devices are shown in

Table 7. In addition, the specifications of the *end device*, implemented using the *Wemos D1 Mini ESP32* platform, are provided in Table 8.

Table 6. *Raspberry Pi 4* specifications used as edge device.

Specification	Details
Board	Raspberry Pi 4 Model B
CPU	Quad-core ARM Cortex-A72 (64-bit)
Clock Frequency	1.5 GHz
Memory	4 GB LPDDR4 RAM
Storage	32 GB MicroSD Card
Network Connectivity	Gigabit Ethernet, 2.4/5 GHz Wi-Fi
Wireless Communication	Bluetooth 5.0 BLE
USB Ports	2 × USB 3.0, 2 × USB 2.0
Operating Voltage	5 V DC

Raspberry Pi 4 Model B was used as a container-capable edge device for deployment scenarios requiring higher computational resources. The device hosted the containerized services generated by the proposed *AI Assistant*, including communication and data processing components.

Table 7. *ESP32-WROOM-32E* specifications used as edge device.

Specification	Details
Module	ESP32-WROOM-32E
Architecture	Xtensa LX6
CPU	Dual-core 32-bit processor
Clock Frequency	Up to 240 MHz
SRAM	520 KB
Flash Memory	4 MB SPI Flash
Wireless Connectivity	2.4 GHz Wi-Fi (802.11 b/g/n), Bluetooth v4.2 BR/EDR + BLE
Interfaces	GPIO, UART, SPI, I2C, ADC
Operating Voltage	3.0 V–3.6 V

ESP32-WROOM-32E was used as a lightweight edge device in resource-constrained deployment scenarios. The device was responsible for receiving data from end devices, performing basic data aggregation and filtering, and forwarding the processed data to the *SEMAR* IoT application server.

Table 8. *Wemos D1 Mini ESP32* specifications used as end devices.

Specification	Details
Board	Wemos D1 Mini ESP32
Microcontroller	ESP32
Architecture	Xtensa LX6
CPU	Dual-core 32-bit processor
Clock Frequency	Up to 240 MHz
SRAM	520 KB
Flash Memory	4 MB SPI Flash
Wireless Connectivity	2.4 GHz Wi-Fi (802.11 b/g/n), Bluetooth v4.2 BR/EDR + BLE
Interfaces	GPIO, UART, SPI, I2C, ADC, PWM
USB Interface	Micro USB
Operating Voltage	3.0 V–3.6 V

Wemos D1 Mini ESP32 was used as the end device platform for sensor data acquisition. The device was responsible for collecting sensor measurements and transmitting them to the edge device through the communication protocol generated by the proposed *AI Assistant*. These hardware platforms were selected to represent heterogeneous *IoT* deployment environments with different computational capabilities and resource constraints. In particular, *ESP32-WROOM-32E* represents a lightweight edge platform, while *Raspberry Pi 4* represents a container-capable edge platform with higher processing capacity. This setup enables the evaluation of the proposed requirement-driven deployment framework under various deployment conditions.

4.4. Experimental Procedure

Before conducting the experiment, each participant was asked to read the experimental scenario presented in Table 9. This step ensured that all participants understood the deployment objectives, the tasks to be performed, the role of the proposed *AI Assistant*, and the expected outcomes of each deployment phase. To ensure a fair evaluation of the proposed approach, the experimental environment was prepared in advance. The operating systems and basic software dependencies had already been installed on the server and edge devices, allowing participants to focus solely on the deployment process assisted by the proposed *AI Assistant* rather than initial system setup.

The experiment consisted of deploying three main components of the *IoT* application system, namely the *SEMAR* IoT application server, an *edge device*, and an *end device*. For the deployment of the *SEMAR* IoT application server, participants remotely accessed the testing server using the *SSH* protocol, uploaded the deployment manifest generated by the proposed system, and executed the deployment commands to verify that the server could be successfully accessed.

The deployment procedure for the *Raspberry Pi*-based edge device followed a similar process, where participants remotely connected to the device and deployed the generated deployment manifest. In contrast, when the *ESP32*-based edge device was used, the deployment process was performed through a browser-based flashing interface automatically provided by the *Web Application*. After the edge device was successfully deployed, it was registered to the *SEMAR* IoT application server to enable communication between the edge and server components.

For the *end device*, participants performed firmware flashing using the same browser-based mechanism. After flashing, several configuration parameters were specified, including the wireless access point name, password, edge device endpoint address, and communication settings. Both the *edge device* and *end device* operated in *Access Point (AP)* mode during the configuration stage, allowing participants to complete the configuration process through a web-based interface.

After all deployment and configuration steps were completed, participants confirmed the successful operation of the system by observing sensor data transmission and data visualization through the *SEMAR* IoT application server. The total deployment time and user feedback were recorded for subsequent evaluation.

Table 9. User experiment scenario.

Step	Phase	User Action	AI Assistant Function	Expected Output
1	Requirement Analysis	Input use case and number of sensors	Requirement extraction	Deployment requirements generated
2	Architecture Recommendation	Select edge device (ESP32/Raspberry Pi)	Architecture recommendation	Edge architecture selected
3		Select protocol connection	Protocol configuration	Communication method selected
4	Hardware Recommendation	Review AI hardware recommendation	Hardware recommendation engine	Recommended hardware list
5		Review hardware availability	Marketplace integration	Available products displayed
6	Software Stack Recommendation	Review software stack recommendation	Software recommendation engine	Deployment stack selected
7	SEMAR Server Deployment	Download SEMAR package	Deployment package generation	Docker package downloaded
8		Access deployment server or PC	Deployment preparation	Deployment environment ready
9		Run SEMAR containers	Docker deployment	SEMAR services running
10		Register edge device	Device registration	Edge registered
11	Edge Deployment	Download edge package	Edge deployment generation	Edge package downloaded
12		Access edge device	Edge preparation	Edge environment ready
13		Run edge services	Edge deployment	Edge service active
14	Data Schema Generation	Review recommended data schema	AI schema generation	JSON schema prepared
15	End Device Deployment	Install USB driver	Firmware preparation	Flash environment ready
16		Connect sensors to ESP32	Hardware preparation	Sensor wiring completed
17		Flash firmware	Firmware flashing	Firmware installed
18		Connect Wi-Fi and configure device	Device configuration	End device operational
19		Verify sensor transmission	Data validation	Sensor data successfully received

4.5. Evaluation Metrics

To evaluate the proposed *AI Assistant*, two aspects were considered: usability and deployment efficiency. Usability was assessed using the *System Usability Scale (SUS)* [39,40], while deployment efficiency was evaluated by comparing the deployment time required by the proposed approach and a conventional manual deployment process.

4.5.1. System Usability Scale (SUS)

The *System Usability Scale (SUS)* was employed to evaluate the usability of the proposed system. *SUS* is a widely adopted usability assessment method used in both academic research and industry [41]. The questionnaire consists of ten statements rated using a five-point *Likert* scale ranging from *Strongly Disagree* to *Strongly Agree*. The objective of this evaluation is to assess the perceived ease of use of the proposed system, including learnability, complexity, and the amount of support or training required. The 10 *SUS* questionnaire items used in this study are presented in Table 10.

Table 10. System Usability Scale (SUS) questionnaire.

Item	Statement
Q1	I think that I would like to use this system frequently.
Q2	I found the system unnecessarily complex.
Q3	I thought the system was easy to use.
Q4	I think that I would need the support of a technical person to be able to use this system.
Q5	I found the various functions in this system were well integrated.
Q6	I thought there was too much inconsistency in this system.
Q7	I would imagine that most people would learn to use this system very quickly.
Q8	I found the system very cumbersome to use.
Q9	I felt very confident using the system.
Q10	I needed to learn a lot of things before I could get going with this system.

As shown in Table 10, the *SUS* questionnaire contains both positive and negative statements. Positive statements correspond to items 1, 3, 5, 7, and 9, while negative statements correspond to items 2, 4, 6, 8, and 10. To calculate the *SUS* score, the contribution score of each item is first determined. For positive items, the contribution score is calculated by subtracting one from the participant's response. For negative items, the contribution score is obtained by subtracting the response value from five, as defined in Equation (5). The contribution scores of all items are then summed and multiplied by 2.5 to obtain the final SUS score, as defined in Equation (6). The interpretation categories of *SUS* scores used in this study are presented in Table 11. Equation (5) is defined as follows:

$$S_i = \begin{cases} R_i - 1, & \text{if } i \in 1,3,5,7,9 \\ 5 - R_i, & \text{if } i \in 2,4,6,8,10 \end{cases} \tag{5}$$

where R_i represents the participant's response to item i, and S_i denotes the contribution score of the corresponding item. Equation (6) is defined as follows:

$$SUS = 2.5 \times \sum_{i=1}^{10} S_i \tag{6}$$

Table 11. SUS score adjective ratings [42].

SUS Score	Adjective Rating
<25	Worst Imaginable
$25 \leq x < 39$	Poor
$39 \leq x < 52$	OK
$52 \leq x < 73$	Good
$73 \leq x < 85$	Excellent
≥ 85	Best Imaginable

4.5.2. Deployment Efficiency

The deployment efficiency was evaluated by comparing the proposed *AI Assistant*-based deployment approach with a conventional manual deployment procedure. In the manual deployment scenario, participants performed all deployment tasks according to the predefined workflow presented in Table 12. In contrast, the proposed approach utilized the deployment guidance and deployment artifacts automatically generated by the *AI Assistant*.

Table 12. Manual deployment workflow used for deployment efficiency evaluation.

Step	Phase	Manual Deployment Task
1	SEMAR Server Deployment	Access the deployment server for the *SEMAR IoT application server*.
2		Install the *Docker* container runtime.
3		Clone the deployment manifest repository of the *SEMAR IoT application server*.
4		Execute the *Docker Compose* deployment.
5		Verify that the *SEMAR IoT application server* is running correctly.
6		Register the *edge device* by providing the required device information.
7	Raspberry Pi Edge Deployment	Access the *Raspberry Pi* edge device remotely.
8		Install the *Docker* container runtime.
9		Clone the deployment manifest repository for the *edge device*.
10		Verify that the *edge device* is operating correctly.
11		Configure the *edge device* according to the information registered in the *SEMAR IoT application server*.
12	ESP32 Edge Deployment	Install the required USB driver before connecting the device to the computer.
13		Download and install the development environment used for firmware compilation and flashing.
14		Flash the firmware to the *ESP32*-based edge device.
15	End Device Deployment	Clone the repository corresponding to the selected *IoT* use case.
16		Open the project using the development environment.
17		Build and upload the firmware to the *end device*.
18		Configure the *end device* by specifying the edge device address, network connectivity parameters, and communication protocol (e.g., MQTT or HTTP).

The deployment time was measured from the initial requirement input stage until the successful completion of the deployment process, including the deployment of the *SEMAR* IoT application server, *edge device*, and *end device*. The total duration required to complete the end-to-end deployment process was recorded and compared between the manual and AI-assisted deployment approaches.

5. Results and Discussion

In this section, we present the evaluation results of the proposed *generative AI*-based deployment assistant. They are organized with three main sections: deployment workflow execution, deployment time evaluation, and system usability evaluation. Finally, we provide a discussion of the findings and their implications for future research and development in this area.

5.1. AI-Assisted Deployment Workflow

The proposed *AI Assistant* was evaluated by deploying an *IoT* application system consisting of the *SEMAR* IoT application server, an *edge device*, and an *end device*. The deployment process was guided by the proposed deployment workflow, which included requirement analysis, architecture recommendation, hardware recommendation, software stack recommendation, and deployment execution. The results of the evaluation are presented in the following subsections.

5.1.1. Requirement Analysis

The *Requirement Analysis* stage is the initial step in the proposed deployment workflow. Its objective is to identify the application use case and the hardware requirements, including the sensors involved in the deployment scenario. Based on the user's input, the proposed *AI Assistant* extracts the deployment requirements and presents them in a structured format. As shown in Figure 5, the generated response includes the current deployment stage, a summary of the identified use case, the extracted deployment requirements, practical recommendations, and the next deployment step. This structured presentation enables users, particularly novices, to verify the extracted requirements before proceeding to the subsequent deployment stages.

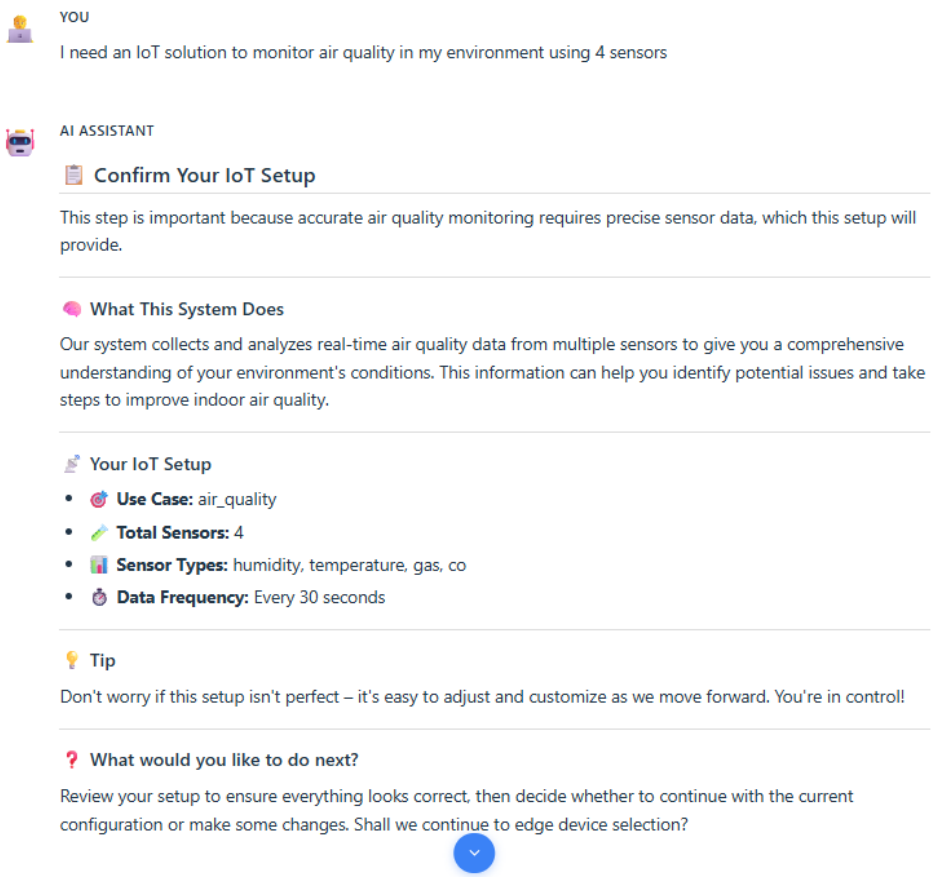
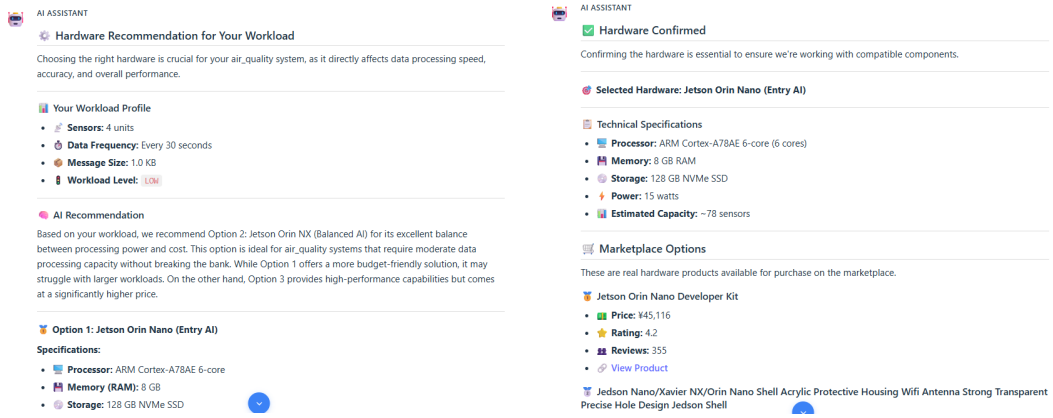


Figure 5. Requirement analysis interface.

5.1.2. Hardware Recommendation

Based on the extracted deployment requirements, the *AI Assistant* recommends suitable hardware platforms for the deployment. As shown in Figure 6, the system presents three alternative hardware configurations together with the rationale for each recommendation and identifies the most appropriate platform for the specified deployment scenario. Users may also select an alternative platform according to their preferences or resource

constraints. After a hardware platform is selected, the *AI Assistant* automatically retrieves product information from the *Amazon* marketplace, including product availability, pricing, and user ratings. This information enables users to compare the recommended hardware and make informed purchasing decisions before proceeding to the next deployment stage.



(**a**) Hardware recommendation interface. (**b**) Automated product information interface.

Figure 6. Hardware recommendation generated by AI-assisted deployment framework. (**a**) AI-generated hardware recommendations with supporting rationale. (**b**) Automatically retrieved product information, including pricing, availability, and user ratings, to assist hardware selection.

5.1.3. Software Stack Recommendation

Following hardware selection, the proposed *AI Assistant* recommends the required software components for both the *SEMAR* server and the *edge device*. The recommendation includes the operating system, container runtime, communication protocol, and supporting software packages required for deployment. As shown in Figure 7, the generated recommendation is accompanied by a brief explanation to help users understand the purpose of each software component before deployment.

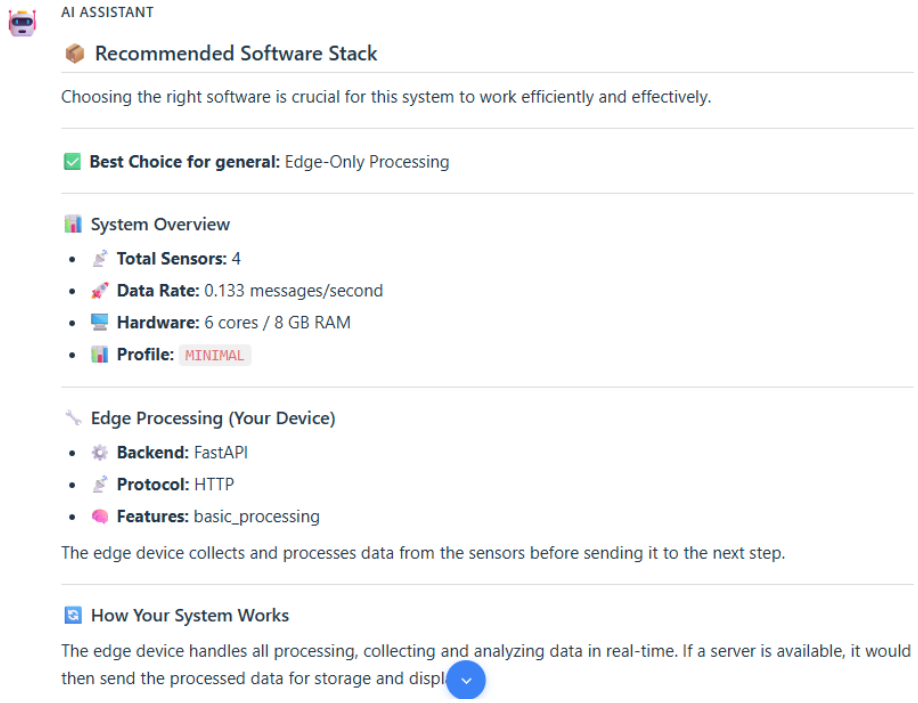


Figure 7. Software stack of recommendation interface.

5.2. Deployment Workflow Execution

The proposed deployment workflow consists of several phases, as previously presented in Table 9, covering the deployment of the *SEMAR* IoT application server, *edge devices*, and *end devices*. The objective of this workflow is to simplify the deployment process for novice users by automatically generating the required deployment artifacts and configuration parameters through the proposed *AI Assistant*.

5.2.1. SEMAR Server Deployment

The deployment of the *SEMAR* IoT application server is supported by the automatic generation of deployment artifacts, including *Docker Compose* and `.env` configuration files. These artifacts contain the necessary information for deploying and configuring the server components, such as container images, network settings, exposed ports, and database connection parameters.

The generated *Docker Compose* file deploys two main containers, namely *semar-fast-api* and *semar-web*. The *semar-fast-api* container provides the core *REST API* services of the *SEMAR* platform and is implemented using *Python* with the *FastAPI* framework. Meanwhile, *semar-web* serves as the web-based client application developed using *PHP* and the *Laravel* framework. Since both components are deployed as containers, the deployment process is independent of the underlying operating system. The required container images are stored and distributed through *Docker Hub*.

After the server deployment process is completed, the user registers the edge device within the *SEMAR* platform. During this stage, the user defines the sensor types, filtering parameters, and data aggregation rules to be applied. The registration process generates a unique device identifier and an authentication token, which are subsequently used during the edge device configuration process. Figure 8 illustrates the edge device registration interface within the *SEMAR* platform.

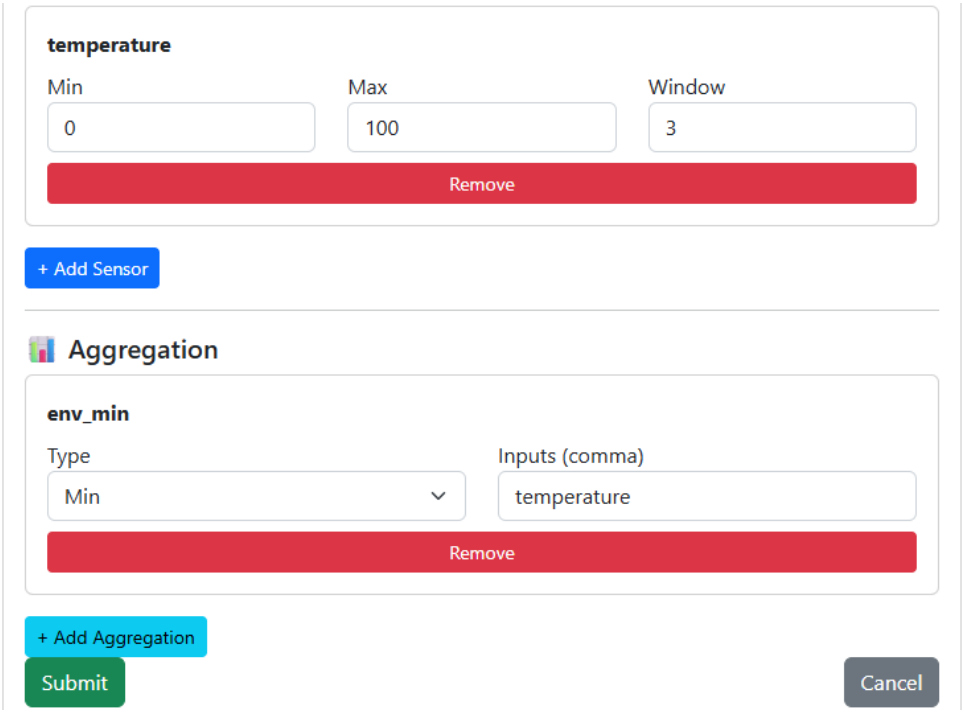


Figure 8. Edge device registration interface.

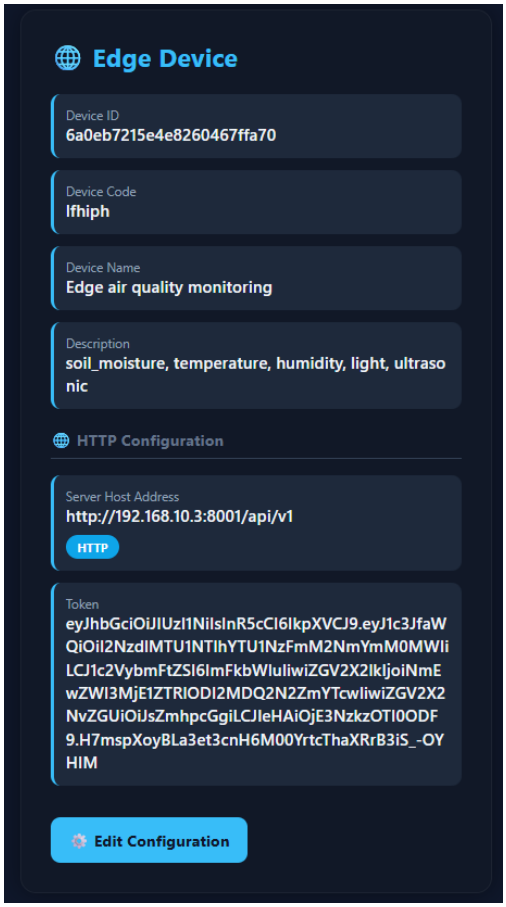
5.2.2. Edge Device Deployment

To evaluate the adaptability of the proposed *AI Assistant*, two different edge device platforms were considered, namely *Raspberry Pi* and *ESP32*. These platforms represent edge devices with different computational capabilities and deployment requirements.

For the *Raspberry Pi*-based deployment, the generated deployment package consists of *Docker Compose* and configuration files similar to those used in the server deployment process. The deployed services include data filtering, aggregation, and communication modules. In addition, an *MQTT* broker container is included to support communication between end devices and the edge device using either *MQTT* or *HTTP* protocols.

In contrast, the *ESP32*-based edge device requires firmware deployment instead of container deployment. In this case, the proposed *AI Assistant* automatically generates and compiles firmware based on deployment parameters such as the selected use case, sensor configuration, and communication protocol. The generated firmware can then be flashed directly to the target device.

After deployment, the *ESP32* operates in *Access Point (AP)* mode, allowing users to configure network and communication parameters through a web-based interface. Figure 9 presents the configuration interfaces generated for the two supported edge device platforms. *Raspberry Pi* provides a browser-based configuration interface, while *ESP32* exposes a lightweight web interface through *AP* mode, enabling users to configure deployment parameters without requiring additional software installation.



(**a**) *Raspberry Pi* edge device configuration interface. (**b**) *ESP32* edge device configuration interface.

Figure 9. Edge device configuration interfaces generated by the proposed AI-assisted deployment framework. (**a**) *Raspberry Pi* edge device. (**b**) *ESP32* edge device operating in *Access Point (AP)* mode for network configuration.

5.2.3. End Device Deployment

The end device deployment process utilizes the *Wemos D1 Mini ESP32* platform described in Table 8. Based on the deployment requirements provided by the user, the proposed *AI Assistant* automatically generates firmware, communication parameters, and sensor configurations tailored to the selected *IoT* application scenario.

The generated firmware can be flashed directly through the integrated web-based interface using the *Web Serial API*, eliminating the need for external development environments and manual source code modification. Figure 10 shows the firmware flashing interface, while Figure 11 presents the end device configuration interface.

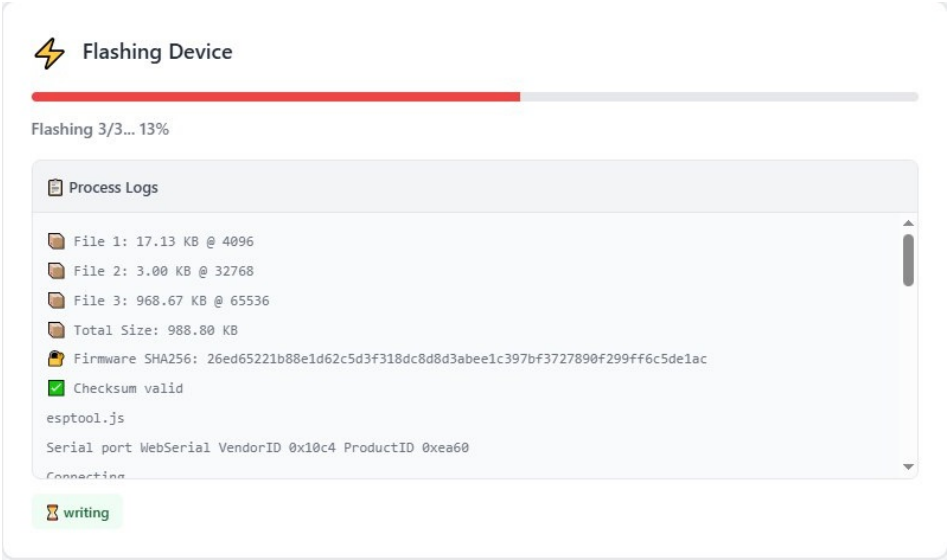


Figure 10. Firmware flashing interface.

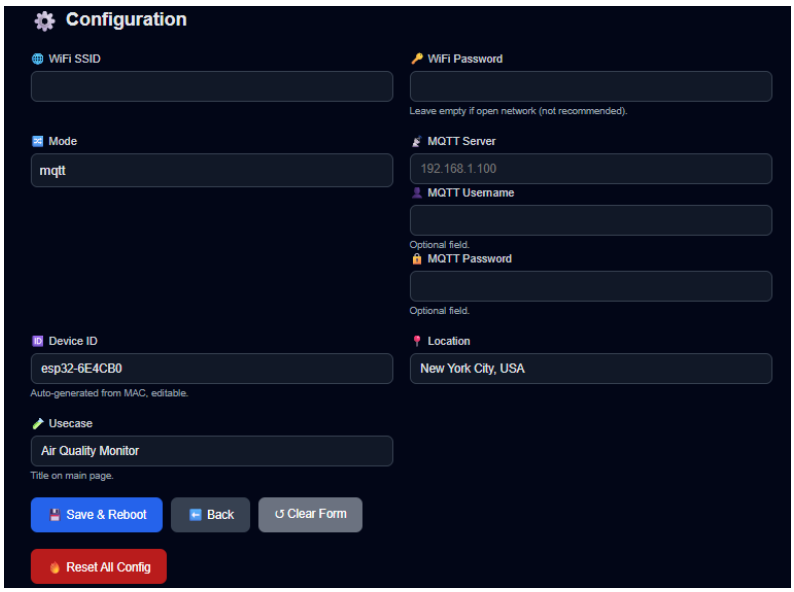
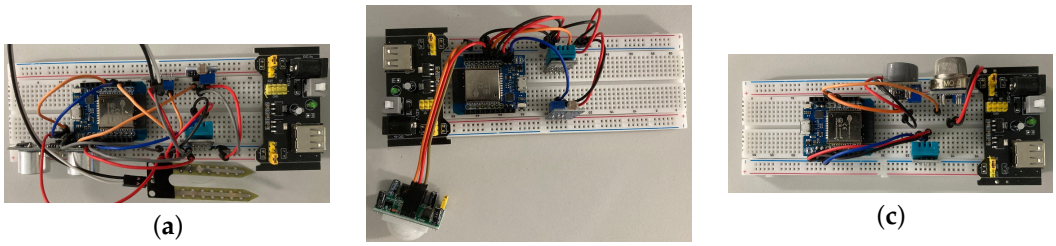


Figure 11. *ESP32* end device configuration interface.

To evaluate the flexibility of the proposed approach, three representative *IoT* application scenarios were considered, namely *smart farming*, *smart home*, and *air quality monitoring*. The sensor configurations used in each scenario are summarized in Table 13. The deployment results are presented in Figure 12. The results demonstrate that the proposed *AI Assistant* successfully generates and deploys firmware for different application requirements while maintaining a consistent deployment workflow.



(**a**) (**b**) (**c**)

Figure 12. Deployment results for three representative IoT application scenarios: (**a**) Smart Farming, (**b**) Smart Home, and (**c**) Air Quality Monitoring. The *AI Assistant* successfully generated deployment artifacts for all scenarios while maintaining a consistent deployment workflow.

Table 13. IoT use cases, physical sensor modules, and measurement channels.

Use Case	Physical Modules	Channels
Smart Farming	Soil Moisture, LDR, DHT11, Ultrasonic	5
Smart Home	PIR, LDR, DHT11	4
Air Quality	DHT11, MQ-2, MQ-7	4

DHT11 contributes 2 channels (temperature and humidity); all other modules contribute 1 channel each.

5.3. Analysis of AI Recommendation Across Different Use Cases

To evaluate the capability of the proposed *AI Assistant* in supporting the deployment process, three representative IoT use cases were considered, namely *Smart Farming*, *Smart Home*, and *Air Quality Monitoring*, as summarized in Table 14.

Although the deployment workflow remains identical for all scenarios, the recommendations generated by the *AI Assistant* differ according to the extracted application requirements. For the *Smart Farming* scenario, the *AI Assistant* recommends environmental sensors, including soil moisture, *LDR*, *DHT11*, and an ultrasonic sensor to monitor water levels. In contrast, the *Smart Home* scenario prioritizes occupancy detection by recommending a PIR motion sensor together with supporting environmental sensors. Meanwhile, the *Air Quality Monitoring* scenario emphasizes gas concentration measurement by recommending *MQ-2* and *MQ-7* gas sensors.

These variations demonstrate that the proposed *AI Assistant* dynamically adapts hardware and software recommendations according to the information extracted during the requirement analysis stage, while deployment artifacts are generated by populating predefined templates with the recommended parameters. Consequently, each deployment configuration is tailored to the functional requirements of the target IoT application while maintaining a consistent deployment workflow.

Furthermore, the comparison indicates that differences in application requirements directly influence the generated deployment artifacts, including sensor selection, communication protocols, firmware configuration, and hardware specifications. However, the deployment pipeline itself remains unchanged across all evaluated scenarios. This result suggests that the proposed framework successfully separates application-specific configurations from the deployment workflow, enabling the same deployment process to support multiple IoT applications with minimal user intervention.

Table 14. Comparison of AI-generated deployment configurations for different IoT use cases.

Component	Smart Farming	Smart Home	Air Quality
Sensor Configuration	Soil Moisture, LDR, DHT11, Ultrasonic	PIR, LDR, DHT11	MQ-2, MQ-7, DHT11
Edge Device	ESP32	ESP32	Raspberry Pi
Communication Protocol	HTTP	HTTP	HTTP/MQTT
Generated Deployment Artifact	ESP32 Firmware	ESP32 Firmware	Python Service + Docker

5.4. Deployment Artifact Evaluation

The functional correctness of the deployment artefacts generated by the proposed *AI Assistant* was evaluated through end-to-end verification across 13 participant sessions. The evaluation covered Docker Compose deployment, firmware compilation and flashing, data-schema consistency, and end-to-end data transmission. The results of the functional evaluation are summarized in Table 15.

All 13 generated Docker Compose configurations were successfully deployed on the first attempt without any modification, achieving a 100% success rate (13/13). Similarly, all generated firmware packages were successfully compiled. Firmware flashing succeeded on the first attempt for 10 participants (76.9%), while the remaining three required a second attempt due to USB connectivity issues. No firmware modification was required. The generated data schemas were also consistent with the corresponding sensor configurations in all 13 sessions, and all deployed devices successfully transmitted the expected sensor data to the application server.

Overall, all 13 participants successfully completed the AI-assisted deployment process, resulting in a 100% end-to-end completion rate. No researcher intervention was required to correct the generated deployment artefacts; assistance was only needed for minor hardware connectivity issues during firmware flashing.

Table 15. Functional evaluation of generated deployment artefacts.

Artifact/Evaluation	Success	Rate
Docker Compose execution	13/13	100%
Application operational state	13/13	100%
Firmware compilation	13/13	100%
First-attempt firmware flashing	10/13	76.9%
Final firmware deployment	13/13	100%
Data-schema consistency	13/13	100%
Sensor data transmission	13/13	100%
End-to-end completion	13/13	100%

5.5. Deployment Time Evaluation

To evaluate the effectiveness of the proposed *AI Assistant*, the deployment time was compared with the conventional manual deployment procedure. Participants performed the deployment following the manual workflow presented in Table 12 and the *AI-assisted* workflow presented in Table 9. The deployment time was measured from the beginning of the deployment process until sensor data were successfully visualized in the *SEMAR* Web Application.

The deployment time was measured for each participant under both conditions. Table 16 presents the individual deployment times recorded for all 13 participants. The mean deployment time for the manual condition was 21.80 min (SD = 8.03), compared with 16.40 min (SD = 2.93) for the AI-assisted condition, representing a reduction of 24.8%. Prior to the paired t-test, normality of the paired differences was assessed using the Shapiro–Wilk test ($W = 0.972$, $p = 0.913$), confirming that the normality assumption was satisfied. A paired-samples t-test indicated that this difference was statistically significant ($t(12) = 2.55$, $p = 0.025$, Cohen's $d = 0.71$), with a 95% confidence interval for the mean difference of [0.79, 10.01] min, suggesting a medium-to-large practical effect. As a robustness check, the Wilcoxon signed-rank test yielded a consistent result ($W = 16$, $p = 0.040$). To control for potential learning effects arising from the within-subject design, task order was counterbalanced across participants: approximately half completed the manual condition first and the other half completed the AI-assisted condition first.

The largest difference occurred during the end device deployment stage. In the manual workflow, participants were required to download the firmware source code, configure the development environment, compile the firmware, and upload it to the microcontroller. In addition, users had to manually configure the target board, communication port, baud rate, and required libraries before deployment.

Table 16. Individual deployment times (MM:SS) for manual and AI-assisted conditions (n = 13).

Participant	Manual	AI-Assisted	Difference (Manual − AI)
P01	29:09	18:11	+10:58
P02	15:00	14:18	+0:42
P03	29:11	21:28	+7:43
P04	37:28	19:09	+18:19
P05	24:58	10:26	+14:32
P06	19:44	15:54	+3:50
P07	22:38	18:59	+3:39
P08	14:59	13:19	+1:40
P09	12:03	18:26	−6:23
P10	28:41	15:57	+12:44
P11	11:35	15:33	−3:58
P12	23:53	14:21	+9:32
P13	14:03	17:11	−3:08
Mean (min)	**21.80**	**16.40**	**+5.40**
SD (min)	**8.03**	**2.93**	7.63

Mean and SD in decimal minutes (exact conversion: MM:SS → min + s/60). Paired t-test: $t(12) = 2.55$, $p = 0.025$, Cohen's $d = 0.71$; 95% CI for mean difference [0.79, 10.01] min. Shapiro–Wilk test on paired differences: $W = 0.972$, $p = 0.913$ (normality assumption satisfied). Negative difference indicates AI-assisted was slower.

In contrast, the proposed *AI Assistant* automatically generated the deployment artifacts and provided a browser-based firmware flashing mechanism. Consequently, participants no longer needed to configure the development environment or manually compile the firmware, thereby reducing technical errors and shortening the overall deployment process.

It is noted that three participants (P09, P11, and P13) recorded longer deployment times under the AI-assisted condition than under the manual condition. Post-task observation indicated two contributing factors: P09 had extensive prior CLI, Docker, and IoT component experience and completed the manual workflow more efficiently than less experienced participants, reducing the relative advantage of the *AI Assistant* for that participant; P11 and P13 experienced hardware connectivity issues during the firmware flashing stage (loose USB connections requiring repeated reconnection attempts), which added variable overhead to the AI-assisted condition independent of the system's performance. These cases are retained in the analysis as they reflect realistic deployment variability.

To assess whether task order influenced the results, deployment times were compared between the seven participants who completed the manual condition first and the six participants who completed the AI-assisted condition first (Table 4). No systematic difference was observed between the two groups, suggesting that the counterbalanced design successfully mitigated potential learning effects. Each paired comparison used the same deployment scenario and the same edge device type for both conditions.

5.6. System Usability Evaluation

The usability evaluation was conducted to assess the ease of use of the proposed *AI Assistant* in supporting the deployment process. A total of 13 participants were involved in the study, as described in Section 4. The *SUS* was administered only for the AI-assisted

condition; usability of the manual deployment process was not separately evaluated using *SUS*.

As shown in Table 17, the proposed AI Assistant achieved a *SUS* score of 85.00 (SD = 11.41, n =13, 95% CI [78.11, 91.89]), which falls within the *Best Imaginable* category according to the adjective rating scale by Bangor et al. [42]. This result indicates that the proposed system is well accepted by participants for performing IoT deployment tasks. These findings are presented as indicative of perceived usability within the study sample rather than as a generalisable claim, given the modest sample size (n = 13).

Table 17. Summary of *SUS* evaluation results.

Evaluation Metric	Result
Number of Participants	13
SUS Score (Mean ± SD)	85.00 ± 11.41
95% Confidence Interval	[78.11, 91.89]
Adjective Rating	Best Imaginable

The distribution of responses for each SUS statement is illustrated in Figure 13. For the positive statements (Q1, Q3, Q5, Q7, and Q9), the responses were dominated by *Strongly Agree* (58.46%) and *Agree* (33.85%), while 7.69% of responses were *Neutral* and none were negative. For the negative statements (Q2, Q4, Q6, Q8, and Q10), the majority of responses were *Strongly Disagree* (47.69%) and *Disagree* (36.92%), followed by *Neutral* (13.85%) and *Strongly Agree* (1.54%). These results indicate a consistently positive perception of the proposed system across all evaluated usability dimensions.

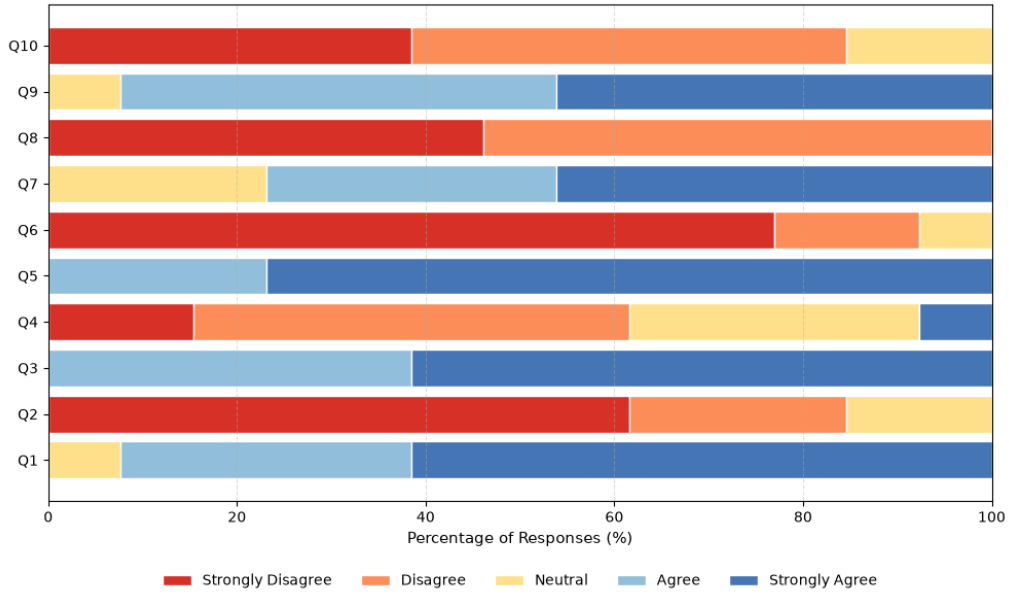


Figure 13. Distribution of *SUS* responses.

Although the *SUS* results indicate a high level of usability, qualitative feedback from participants revealed recurring themes regarding areas for improvement. Several participants noted that the step-by-step guidance was helpful for beginners, with comments such as "the written and guided steps are helpful for beginners in the IoT field" and "it is very helpful in the learning process as it provides step-by-step guidance in detail." However, some participants identified specific usability challenges: one noted that configuration values shown early in the conversation were difficult to retrieve later ("sometimes need to scroll back to the beginning of the chat to view some values and configuration"),

and another suggested the addition of video or visual guidance to complement the text-based instructions. These observations suggest that while the overall usability is high, improvements to information persistence and visual guidance could further enhance the user experience.

These results indicate that participants generally perceived the proposed *AI Assistant* as easy to use and helpful for supporting IoT deployment activities. The dominance of positive responses in favourable statements and negative responses in unfavourable statements further confirms the high usability and user acceptance of the proposed system.

5.7. LLM Temperature Sensitivity Analysis

The *LLM* generation parameters used in this study were *temperature* = 0.15, *top_p = 0.9*, and *num_predict* = 2048. Among these parameters, only the temperature was varied to examine its effect on the stability of the requirement-analysis output, while *top_p* and *num_predict* were kept fixed throughout the experiments.

The analysis was conducted using the requirement-analysis prompt (Listing A1) with eight representative user requests, each repeated 15 times for every temperature setting (0.15, 0.5, and 0.8), resulting in 120 runs per setting. The evaluation measured *JSON validity*, *schema conformance*, *business-rule conformance*, and *output consistency*. All metrics were calculated from the raw *LLM* outputs before the code-level correction step. Output consistency was defined as the proportion of repeated runs for the same input that produced an identical extracted requirement profile.

As shown in Table 18, JSON validity and rule conformance were comparable across the three temperature settings, with rule-conformance rates ranging from 85.0% to 89.2%. In contrast, output consistency decreased as the temperature increased, from 96.7% at temperature = 0.15 to 81.7% at temperature = 0.8. The variation was mainly observed in the *sensor_per_device* and *sensor_types* fields. Therefore, temperature = 0.15 was selected primarily to favor output reproducibility rather than to improve JSON validity or rule conformance.

The evaluation focuses on the stability and conformance of the requirement-extraction output and does not constitute a field-level accuracy evaluation against an annotated reference dataset. Therefore, the reported rule-conformance results should not be interpreted as absolute extraction accuracy. Furthermore, because the analysis was based on eight representative requests, each repeated 15 times, the 120 runs per temperature setting should not be interpreted as 120 independent content cases. Rather, this experiment should be interpreted as an exploratory stability analysis of repeated *LLM* outputs.

Table 18. Sensitivity of extraction quality and output consistency to temperature.

Temperature	JSON Valid (%)	Schema Valid (%)	Rule Conformance (%)	Output Consistency (%)
0.15	100.0	100.0	87.5	96.7
0.5	100.0	100.0	89.2	90.8
0.8	100.0	99.2	85.0	81.7

5.8. Discussion

This study proposes an *AI-assisted* deployment framework that supports end-to-end IoT application deployment, covering end devices, edge devices, and server applications. Unlike conventional deployment approaches that require users to manually configure hardware, software, firmware, and deployment manifests, the proposed framework automatically generates deployment configurations based on user requirements. Consequently, novice users with limited knowledge of IoT deployment can complete the deployment process with minimal manual intervention. The experimental results, including the comparison across different use cases, deployment time evaluation, and usability assessment,

demonstrate that integrating an *AI Assistant* into the deployment workflow reduces deployment time and achieves a high level of perceived usability, while maintaining deployment flexibility. The evaluation using three representative IoT use cases, namely *Smart Farming*, *Smart Home*, and *Air Quality Monitoring*, demonstrates that the proposed *AI Assistant* dynamically adapts deployment recommendations according to the extracted application requirements. Different application scenarios produce different hardware recommendations, sensor configurations, communication protocols, and deployment artifacts. These findings indicate that the proposed framework performs context-aware deployment planning, while deployment artifacts are generated by populating predefined templates with the recommended parameters. Although the current implementation is limited to three IoT scenarios, the results demonstrate the capability of the proposed framework to generate deployment configurations that are tailored to different application domains.

From the deployment efficiency perspective, the proposed framework reduces the deployment time by approximately 24.8% compared with the conventional deployment process. This improvement is primarily attributed to the automation of repetitive configuration tasks, including hardware and software recommendation, firmware generation, deployment manifest creation, and deployment execution. In conventional deployment, users are required to manually configure end devices, edge devices, and server applications, making the deployment process highly dependent on user expertise. The most challenging stage is the deployment of end devices, where users must correctly connect sensors to the microcontroller, configure firmware parameters, and upload firmware to the device. These tasks require both hardware and software knowledge and are prone to human error. By automating these activities, the proposed *AI Assistant* significantly reduces deployment complexity and minimizes manual configuration errors.

The usability evaluation further confirms the practicality of the proposed framework. The obtained *System Usability Scale (SUS)* score of 85.00 (SD = 11.41, n = 13, 95% CI [78.11, 91.89]) indicates that the proposed system is well accepted by participants in the study sample. Nevertheless, several respondents provided positive responses to the negative *SUS* statements (Q2, Q4, and Q10), indicating that some deployment stages remain difficult for first-time users. These responses suggest that textual deployment instructions alone may not be sufficient for users without prior IoT experience. Additional deployment guidance, particularly for hardware installation and sensor wiring, should therefore be presented using visual instructions, such as wiring diagrams, hardware illustrations, or interactive deployment guidance, to further improve user understanding and reduce dependency on external assistance. We emphasize that the *SUS* score reflects participants' perceived ease of use and acceptance of the proposed system, and, given that participants had no prior IoT deployment experience, it also partly reflects their ability to effectively follow the system's generative guidance; it should not be interpreted as a validation of the correctness of the underlying AI-generated recommendations. The latter is assessed separately, based on objective, schema- and rule-based criteria independent of user perception, in the *LLM Temperature Sensitivity Analysis* (Section 5.7).

Compared with previous studies, as summarized in Table 19, existing *AI-assisted* deployment approaches mainly focus on specific deployment stages, such as firmware generation for end devices [43], deployment of server-side IoT platforms [13], or requirement analysis and recommendation support [14]. In contrast, the proposed framework integrates these capabilities into a unified end-to-end AI-assisted deployment workflow, covering requirement analysis, hardware and software recommendation, firmware generation, deployment manifest generation, and application deployment. This comprehensive workflow reduces manual intervention and enables novice users to complete IoT deployment through a single *AI-assisted* pipeline.

Despite these advantages, several limitations remain. First, the proposed framework has been evaluated using only three representative IoT application scenarios and currently supports the implementation of two edge computing platforms, namely *ESP32* and *Raspberry Pi*. Although the *AI Assistant* can recommend broader hardware options, the current evaluation may not fully represent the complexity of diverse or large-scale IoT deployments.

Table 19. Comparison of AI-assisted IoT deployment approaches.

Feature	Noprianto et al. [43]	Panduman et al. [13]	Kotama et al. [14]	Proposed
Requirement analysis	–	–	✓	✓
Hardware recommendation	–	–	✓	✓
Software recommendation	–	–	–	✓
Firmware generation	✓	–	–	✓
Deployment manifest generation	–	–	–	✓
Application deployment	–	✓	–	✓
End-to-end deployment workflow	–	–	–	✓
Reported outcome [1]	Reliable and efficient OTA firmware update (qualitative)	Higher flexibility and interoperability vs. existing IoT platforms (qualitative)	Effective sensor setup guidance (qualitative)	24.8% deployment time reduction; SUS = 85.00

[1] As the compared studies evaluate different deployment stages using different experimental designs and metrics (e.g., OTA update reliability, platform interoperability, sensor-guidance effectiveness), their reported outcomes are not directly quantitatively comparable; they are listed here as reported in the original studies rather than converted to a common metric.

Second, the current implementation relies on a small set of high-level application use case categories (e.g., *smart farming*, *smart home*, and *air quality monitoring*), each mapped to a predefined sensor configuration. Although this design simplifies the deployment process for novice users, it may not capture finer-grained, domain-specific requirements. Furthermore, the present study evaluates the stability and rule conformance of the generated requirement profiles but does not include a semantic accuracy evaluation against an annotated reference standard. Furthermore, the present study evaluates the stability and rule conformance of the generated requirement profiles but does not include a semantic accuracy evaluation against an annotated reference standard. Whether any participant noticed a discrepancy between their intended deployment requirements and the profile extracted by the *LLM* was not systematically recorded during the study sessions; this is acknowledged as a limitation of the current evaluation design.

Third, the generated deployment artefacts were functionally evaluated through Docker Compose execution, firmware compilation and deployment, data-schema consistency, and sensor data transmission. However, this evaluation is limited to the scenarios and configurations examined in the present study and does not constitute a broader systematic validation across diverse application requirements, hardware platforms, and deployment environments. In addition, a systematic analysis of cases requiring corrective intervention after the automated processing steps has not yet been conducted.

Fourth, the user study involved 13 participants, all of whom were postgraduate students with IT-related backgrounds and were recruited from a single research laboratory. Although potential order and learning effects were mitigated through a counterbalanced within-subject design and the same deployment scenario and edge device type were used for both conditions, the small and technically homogeneous participant group limits the generalisability of the findings to broader and non-technical user populations. Moreover, SUS was collected only for the AI-assisted condition; therefore, the study does not support a direct comparative claim of improved usability relative to the manual workflow.

Fifth, while Table 19 provides a feature-level comparison with related work, no empirical head-to-head evaluation was conducted against existing IoT deployment tools under identical conditions. Such a comparison would provide stronger evidence of the relative advantages of the proposed approach.

Additionally, the workload estimation model assumes that each device uses a distinct set of sensor channels with no repeated physical module types. The case in which a user deploys multiple identical physical modules on the same device (e.g., three DHT11 modules providing six temperature and humidity channels) is not explicitly handled by the current prompt schema or estimation logic, and this is identified as an edge case for future work.

Finally, although the resource estimation and hardware recommendation processes have been analytically formalised, the current fit heuristic has not been systematically validated against measured hardware capacities or benchmark data. Its ability to discriminate among multiple candidate platforms and to relate estimated requirements to actual device capacities therefore requires further empirical investigation.

Future work will focus on extending the framework by supporting additional hardware platforms and evaluating more diverse IoT application scenarios. Furthermore, multimodal AI capabilities will be integrated to generate visual deployment instructions, including wiring diagrams, deployment architecture, hardware configuration illustrations, and interactive guidance. Such enhancements are expected to further reduce the learning curve for novice users while improving the usability and scalability of *AI-assisted IoT deployment*. We also plan to extend the requirement analysis stage toward an iterative, requirements-first workflow, in which high-level application requirements and constraints are progressively refined through one or more clarification rounds before an architecture and hardware specification are derived, rather than assuming that the application scenario and edge device class have already been decided. Such enhancements are expected to further reduce the learning curve for novice users while improving the usability and scalability of *AI-assisted IoT deployment*.

Overall, the experimental results demonstrate that integrating an *AI Assistant* into the deployment workflow reduces deployment time significantly and achieves a high level of perceived usability, while maintaining deployment flexibility across different IoT application scenarios. These findings suggest that the proposed framework provides a practical solution for lowering the entry barrier of IoT deployment and supporting end-to-end deployment for users with limited technical expertise.

6. Conclusions

This paper proposed a *generative AI*-based assistant to simplify the end-to-end deployment of *IoT application systems*, covering *end devices*, *edge devices*, and the *SEMAR* IoT application server. By automatically generating deployment recommendations and artifacts according to user requirements using a hybrid pipeline of rule-based extraction, *LLM*-based reasoning, and template-based artifact generation, the proposed system automates the generation of deployment artifacts (Docker Compose files, firmware packages, and configuration files) that would otherwise require users to perform 18 manual technical steps independently (Table 12), thereby reducing the technical expertise required from non-expert users. A within-subject evaluation involving 13 participants across three *IoT* application scenarios and two edge device platforms, conducted under controlled laboratory conditions, demonstrated a statistically significant reduction in mean deployment time under the AI-assisted condition compared with the manual condition (21.80 $\pm$ 8.03 min vs. 16.40 $\pm$ 2.93 min; t(12) = 2.55, p = 0.025, Cohen's d = 0.71), representing a reduction of 24.8% within the described study sample. The AI-assisted condition also achieved a high subjective usability rating (SUS = 85.00, SD = 11.41, n = 13, 95% CI [78.11, 91.89]). These

results suggest that the proposed assistant lowers the technical barrier to *IoT* deployment for users without prior end-to-end deployment experience, within the scope and sample of this study. Future work will extend the proposed assistant to support cloud-native deployment and scalability management for *SEMAR* application server clusters and incorporate richer visual guidance to further assist novice users during the deployment process.

Author Contributions: Conceptualization, N. and N.F.; Methodology, N. and N.F.; Software, N.; Writing—Original Draft Preparation, N., K.C.B. and I.N.D.K.; Writing—Review and Editing, N.F., K.C.B., I.N.D.K. and H.H.S.K.; Validation, Y.W.S.; supervision, N.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: This study involved human participants in a low-risk usability evaluation of a software deployment tool. No sensitive personal, medical, or psychological data were collected. Formal ethical review by an Institutional Review Board was not required under the applicable institutional guidelines for this category of study.

Informed Consent Statement: Informed consent was obtained from all participants prior to the study. Participants were informed of the study objectives, of the nature of the tasks, and that their anonymized timing and usability data would be used for academic research and publication. Participation was voluntary.

Data Availability Statement: The individual participant deployment-time measurements and SUS responses supporting the findings of this study are included in Appendix B. The source code of the proposed system is not publicly available at this time as the platform remains under active development, but it is available from the corresponding author upon reasonable request.

Acknowledgments: The authors thank the reviewers for their thorough reading and helpful comments.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Prompt Examples for AI Assistant Deployment

Listing A1. Prompt template for requirement analysis.

```
You are an AI Assistant that extracts IoT deployment configuration from
    user input.
Your task is to convert the user's message into a structured JSON object
    that strictly follows this schema:

{{
  "usecase": string,
  "devices": integer,
  "sensor_per_device": integer,
  "sensor_types": string[],
  "protocol": string,
  "payload_kb": integer,
  "interval_seconds": integer,
  "payload_mode": string,
  "storage": string,
  "dashboard": string,
  "os": string
}}

IMPORTANT RULES:
- Output MUST be valid JSON only
- No explanation, no markdown, no extra text
- All values must be lowercase
- Do NOT add fields outside~schema
```

Listing A1. *Cont.*

```
DEFAULT VALUES:
- usecase: "smart_home"
- devices: 1
- sensor_per_device: 1
- sensor_types: ["temperature"]
- protocol: "mqtt"
- payload_kb: 1
- interval_seconds: 30
- payload_mode: "combined"
- storage: "mongodb"
- dashboard: "node-red"
- os: "linux"

ALLOWED SENSOR TYPES (STRICT):
["temperature","humidity","light","gas","motion","soil_moisture","
    ultrasonic","co"]


NOTE: sensor_types lists measurement channels, not physical modules.
- DHT11 (1 physical module) -> 2 channels: "temperature", "humidity"
- PIR, LDR, MQ-2, MQ-7, soil_moisture, ultrasonic, co -> 1 channel each
sensor_per_device MUST equal len(sensor_types) (number of channels).


USECASE SENSOR PRIORITY:
- air_quality -> ["humidity","temperature","gas","co"]
- smart_farming -> ["temperature","humidity","light","soil_moisture","
    ultrasonic"]
- smart_home -> ["temperature","humidity","light","motion"]
- default fallback -> ["temperature","humidity","light","gas","motion","
    soil_moisture","ultrasonic","co"]

CRITICAL RULES:

1. If~user mentions number of sensors (e.g. "4 sensors"):
   -> sensor_per_device MUST be exactly that number.
2. Determine sensor_per_device FIRST before anything else.
3. sensor_types MUST match sensor_per_device EXACTLY.
4. sensor_types MUST:
   - include sensors explicitly mentioned by user
   - use only allowed sensors
   - contain NO duplicates unless fallback list is fully exhausted
5. If~sensor_types count < sensor_per_device:
   -> fill missing slots using USECASE SENSOR PRIORITY
     (skip sensors already used)
6. If~sensor_types count > sensor_per_device:
   -> truncate from the end.
7. NEVER change sensor_per_device to match sensor_types.
   Always adjust sensor_types~instead.

FINAL VALIDATION (MANDATORY):
Before output:
1. Remove duplicates from sensor_types.
2. If~length < sensor_per_device:
     refill using USECASE SENSOR PRIORITY (skip used sensors).
3. If~length > sensor_per_device:
     truncate from the end.
4. Ensure final length(sensor_types) == sensor_per_device.

USECASE MAPPING:
- smart home -> "smart_home"
- farming / greenhouse -> "smart_farming"
- air quality -> "air_quality"

Now extract from:

USER INPUT:
{message}
```

Listing A2. Prompt template for edge device selection.

```
You are an IoT deployment assistant.
Generate a short, friendly markdown message to help the user choose which
    edge device they want to~use.

IMPORTANT:
- Keep it natural and conversational (not like a form)
- Do NOT sound like instructions or commands
- Explain Raspberry Pi vs ESP32 in simple, beginner-friendly language
- Use light, friendly emojis
- Keep it short (4-6 lines total)

OUTPUT FORMAT:

### Choose Your Edge Device
(Write 1 short sentence inviting the user to pick the device they want to
    use)
(Write 1 short sentence explaining Raspberry Pi in simple terms - a small
    computer that runs Linux and handles heavier tasks)
(Write 1 short sentence explaining ESP32 in simple terms - a tiny, low-
    power microcontroller great for sensors)

Summary:
- Raspberry Pi -> more powerful, runs full apps
- ESP32 -> lightweight, cheaper, perfect for sensors
(Write 1 short sentence asking the user which one they prefer)
Write ONLY the markdown output.
```

Listing A3. Prompt template for Protocol Selection.

```
You are an IoT deployment assistant.
Generate a short, friendly markdown message to help the user choose which
    protocol their edge device should use: MQTT or~HTTP.

IMPORTANT:
- Keep it natural and conversational (not like a form)
- Do NOT sound like instructions or commands
- Explain MQTT vs HTTP in simple, beginner-friendly language
- Use light, friendly emojis
- Keep it short (4-6 lines total)

OUTPUT FORMAT:

### Choose Your Communication~Protocol

(Write 1 short sentence inviting the user to pick the protocol they want to
     use)
(Write 1 short sentence explaining MQTT in simple terms - lightweight, fast
    , ideal for frequent sensor updates)
(Write 1 short sentence explaining HTTP in simple terms - simple request/
    response, good for periodic data sending)

Summary:
- MQTT -> fast, efficient, great for real-time sensor data
- HTTP -> simple, reliable, good for occasional~uploads

(Write 1 short sentence asking the user which protocol they prefer)

Write ONLY the markdown output.
```

Listing A4. Prompt template for hardware recommendation.

```
You are an IoT deployment assistant helping non-technical users choose~
    hardware.

Your job:
- Explain ALL available options clearly
- Recommend ONE best option
- Use simple, friendly English
- Be consistent with the use case: "{use_case}"

IMPORTANT RULES:
- Use "-" for ALL bullet lists (valid Markdown)
- Keep explanations short (2-3 sentences)
- Do NOT use technical jargon
- Do NOT repeat instructions in the~output

CRITICAL RULES:
- You MUST generate ALL 3 options (Option 1, Option 2, Option 3)
- Do NOT skip any option
- Each option MUST be fully explained
- You MUST recommend ONE best option, but~still explain ALL options
- The explanation MUST reflect the use case "{use_case}"

USER WORKLOAD:
- Sensors: {workload['total_sensors']}
- Interval: {workload['interval_seconds']} seconds
- Payload: {workload['payload_kb']} KB
- Level: {workload['workload_level']}
- Use case: {use_case}

AVAILABLE OPTIONS:
{hardware_list_text}

[Output format omitted for brevity: structured Markdown sections for
    workload profile, per-option specifications, one recommended option
    with justification, and~a closing prompt inviting the user to confirm
    or select an alternative option.]
```

Appendix B. Participant Data

Table A1 presents the individual deployment times recorded for each participant under both conditions, and Table A2 presents the corresponding SUS item responses for the AI-assisted condition.

Table A1. Individual deployment times (MM:SS) for manual and AI-assisted conditions. Positive difference indicates AI-assisted was faster than manual; negative difference indicates AI-assisted was slower.

Participant	Manual	AI-Assisted	Difference (Manual − AI)
P01	29:09	18:11	+10:58
P02	15:00	14:18	+0:42
P03	29:11	21:28	+7:43
P04	37:28	19:09	+18:19
P05	24:58	10:26	+14:32
P06	19:44	15:54	+3:50
P07	22:38	18:59	+3:39
P08	14:59	13:19	+1:40
P09	12:03	18:26	−6:23
P10	28:41	15:57	+12:44
P11	11:35	15:33	−3:58
P12	23:53	14:21	+9:32
P13	14:03	17:11	−3:08

Statistical summary (decimal minutes): Manual = 21.80 ± 8.03 min; AI-Assisted = 16.40 ± 2.93 min; paired *t*-test: t(12) = 2.55, $p = 0.025$, Cohen's d = 0.71.

Table A2. Individual SUS item responses (1–5) for the AI-assisted condition (n = 13). Q1, 3, 5, 7, and 9 are positive items; Q2, 4, 6, 8, and 10 are negative items.

P	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	SUS
P01	3	2	4	3	4	2	3	2	3	2	65.00
P02	5	1	5	1	5	1	5	1	5	1	100.00
P03	5	1	5	3	5	1	4	2	4	2	85.00
P04	4	3	4	3	5	1	3	2	4	2	72.50
P05	5	3	5	5	5	1	5	1	4	3	77.50
P06	5	1	5	1	5	1	5	1	5	1	100.00
P07	5	1	4	2	5	1	5	2	5	2	90.00
P08	4	2	5	2	5	1	5	1	5	1	92.50
P09	4	1	5	2	4	2	4	2	5	2	82.50
P10	5	1	5	2	5	1	5	1	5	1	97.50
P11	4	2	4	3	4	3	4	2	4	2	70.00
P12	5	1	4	2	5	1	3	1	4	3	82.50
P13	5	1	5	2	5	1	4	2	4	1	90.00
Mean											**85.00**
SD											**11.41**

References

1. Sinha, S. State of IoT 2025: Number of Connected IoT Devices Growing 14% to 21.1 Billion Globally. IoT Analytics, 2025. Available online: https://iot-analytics.com/number-connected-iot-devices/ (accessed on 7 June 2026).
2. Vyas, S.; Ghanghorkar, Y.; Kumar, C.; Ghosal, I. Edge Computing for Smart Retail: Enhancing Customer Experience, Efficiency, and Sustainability. *Stud. Comput. Intell.* **2026**, *1267*, 325–358. [CrossRef]
3. Morchid, A.; Ismail, A.; Khalid, H.M.; Qjidaa, H.; Alami, R.E. Blockchain and IoT technologies in smart farming to enhance the efficiency of the agri-food supply chain: A review of applications, benefits, and challenges. *Internet Things* **2025**, *33*, 101733. [CrossRef]
4. Moleme, L.O.; Omoruyi, O.; Quayson, M. Supply chain visibility and integration in the age of the Internet of Things: A retail perspective. *Mod. Supply Chain. Res. Appl.* **2024**, *6*, 330–350. [CrossRef]
5. Kim, J.; Kim, G.; Bang, J.I.; Choi, A.; Sung, M. CO2 concentration prediction in office spaces using physics-informed neural network based on number of occupants and IoT sensor data. *Build. Environ.* **2026**, *288*, 114035. [CrossRef]
6. Mohammed, M.J.; Qasim, M.A. A Comprehensive Analysis of the Techniques and Methods Used in the Manufacture of Toxic Gas Leak Detection and Monitoring Devices—A Review. In *Innovations for Climate-Resilient Sustainable Development. ICSDT 2024*; Springer Proceedings in Earth and Environmental Sciences; Springer: Cham, Switzerland, 2026; Part F1927, pp. 914–927. [CrossRef]
7. Ghasemi, M.; Fu, Y.; Ouyang, X.; Wang, P.; Turkcan, M.K.; Tavori, J.; Kleisarchaki, S.; Calmant, T.; Gürgen, L.; Kostic, Z.; et al. Real-Time Video Analytics for Urban Safety: Deployment over Edge and End Devices. In *SEC 2025—Proceedings of the 2025 10th ACM/IEEE Symposium on Edge Computing*; Association for Computing Machinery: New York, NY, USA, 2025. [CrossRef]
8. Corradi, R.F.; Cappiello, B.; Bonizzi, F. Non-Conventional IoT Applications Through Low-End Devices Hybrid Network Approach. In *Proceedings of the International Conference on Artificial Intelligence, Computer, Data Sciences, and Applications, ACDSA 2026*; IEEE: Piscataway, NJ, USA, 2026. [CrossRef]
9. Liu, S.; Shen, H.; Che, S.; Ghandi, M.; Li, M. AIMS: Cost-Efficient LLM-based Agent Deployment in Hybrid Cloud-Edge Environments. In *EUROSYS 2026—Proceedings of the 2026 European Conference on Computer Systems*; Association for Computing Machinery: New York, NY, USA, 2026; pp. 1862–1878. [CrossRef]
10. Fu, K.; Zhang, W.; Chen, Q.; Zeng, D.; Peng, X.; Zheng, W.; Guo, M. QoS-aware and resource efficient microservice deployment in cloud-edge continuum. In *Proceedings of the 2021 IEEE 35th International Parallel and Distributed Processing Symposium, IPDPS 2021*; IEEE: Piscataway, NJ, USA, 2021; pp. 932–941. [CrossRef]
11. Li, Z.; Du, Q.; Zhang, S.; Qi, Y.; Wei, X.; Yan, Z.; Hu, Y. Design and Implementation of Customizable Cloud–Edge–End Cooperated IoT System. In *2025 IEEE Cyber Science and Technology Congress (CyberSciTech)*; IEEE: Piscataway, NJ, USA, 2026; pp. 825–830. [CrossRef]
12. Meegammana, N.W.; Fernando, H. Detection of Network Attacks on Application Servers Using Deep Learning in IoT Environments. In *Proceedings of the ICAC 2023—5th International Conference on Advancements in Computing: Technological Innovation for a Sustainable Economy, Proceedings*; IEEE: Piscataway, NJ, USA, 2023; pp. 645–650. [CrossRef]

13. Panduman, Y.Y.F.; Funabiki, N.; Puspitaningayu, P.; Kuribayashi, M.; Sukaridhoto, S.; Kao, W.C. Design and Implementation of SEMAR IoT Server Platform with Applications. *Sensors* **2022**, *22*, 6436. [CrossRef] [PubMed]
14. Kotama, I.N.D.; Funabiki, N.; Panduman, Y.Y.F.; Brata, K.C.; Pradhana, A.A.S.; Noprianto; Desnanjaya, I.G.M.N. Implementation of Sensor Input Setup Assistance Service Using Generative AI for SEMAR IoT Application Server Platform. *Information* **2025**, *16*, 108. [CrossRef]
15. Kong, D.; Funabiki, N.; Kyaw, H.H.S.; Kotama, I.N.D.; Zhu, Z.; Rahmadani, A.A. A Generative AI–Based Technical Data Extraction Tool for IoT Application Systems. *Sensors* **2026**, *26*, 1081. [CrossRef] [PubMed]
16. Oliveira, F.B.; Felice, M.D.; Kamienski, C. IoTDeploy: Deployment of IoT Smart Applications over the Computing Continuum. *Internet Things* **2024**, *28*, 101348. [CrossRef]
17. Bhise, C.A.; Rangisetti, A.K. Edge computational resources and security aware serverless application deployment. *Clust. Comput.* **2025**, *29*, 58. [CrossRef]
18. Rahman, M.G.E.; Candra, M.Z.C. PERISAI: Design and Implementation of a Remote Deployment System for IoT Applications. In *Proceedings of the 2024 IEEE International Conference on Data and Software Engineering: Data-Driven Innovation: Transforming Industries and Societies, ICoDSE 2024*; IEEE: Piscataway, NJ, USA, 2024; pp. 194–198. [CrossRef]
19. Mafeni, V.; Kim, Y. An Automated Edge Computing Approach for IoT Device Registration and Application Deployment. *IEEE Syst. J.* **2024**, *18*, 1447–1458. [CrossRef]
20. Shirvani, M.H.; Ramzanpoor, Y. Multi-objective QoS-aware optimization for deployment of IoT applications on cloud and fog computing infrastructure. *Neural Comput. Appl.* **2023**, *35*, 19581–19626. [CrossRef]
21. Plessis, S.D.; Correia, N. A Comparative Study of Software Architectures in Constrained Device IoT Deployments. In *Proceedings of the 2021 IEEE International Conference on Internet of Things and Intelligence Systems, IoTaIS 2021*; IEEE: Piscataway, NJ, USA, 2021; pp. 35–41. [CrossRef]
22. Mohamed, N.; Upadhyay, R.; Jakka, G.; Rambabu, P.V.; Alfurhood, B.S.; Singh, D.P. Framework for the Deployment of Intelligent Smart Cities (ISC) using Artificial Intelligence and Software Networking Technologies. In *Proceedings of the 2023 3rd International Conference on Advance Computing and Innovative Technologies in Engineering, ICACITE 2023*; IEEE: Piscataway, NJ, USA, 2023; pp. 667–671. [CrossRef]
23. Sahoo, P.; Singh, A.K.; Saha, S.; Jain, V.; Mondal, S.; Chadha, A. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. *arXiv* **2025**, arXiv:2402.07927.
24. Wang, Z.; Shi, Y.; Letaief, K.B. Edge Large AI Models: Collaborative Deployment and IoT Applications. *IEEE Internet Things Mag.* **2025**, *8*, 42–49. [CrossRef]
25. Akram, F.; Malik, A.W.; Khan, S.U. iGenEdge: Intelligent Generative AI Service Deployment for Edge-Connected IoT Devices. *IEEE Internet Comput.* **2025**, *29*, 16–24. [CrossRef]
26. Veiga, T.; Asad, H.A.; Kraemer, F.A.; Bach, K. Towards containerized, reuse-oriented AI deployment platforms for cognitive IoT applications. *Future Gener. Comput. Syst.* **2023**, *142*, 4–13. [CrossRef]
27. Shen, L.H.; Feng, K.T.; Lee, T.S.; Lin, Y.C.; Lin, S.C.; Chang, C.C.; Chang, S.F. AI-Enabled Unmanned Vehicle-Assisted Reconfigurable Intelligent Surfaces: Deployment, Prototyping, Experiments, and Opportunities. *IEEE Netw.* **2024**, *38*, 289–299. [CrossRef]
28. Rafique, A.; Marsden, B.D. Automated LLM Deployment and Evaluation: A Cloud-Native Approach Using LLM-as-a-Judge. In *IEEE International Conference on Cloud Computing, CLOUD*; IEEE: Piscataway, NJ, USA, 2025; pp. 448–450. [CrossRef]
29. Soureya, Y.G.; Amougou, N.; Ngossaha, J.M.; Tsakou, S.B.; Ndjodo, M.F. Adaptive Software Development: A Comprehensive Framework Integrating Artificial Intelligence for Sustainable Evolution. *Int. Arab. J. Inf. Technol.* **2025**, *22*, 248–262. [CrossRef]
30. Hanzelik, P.P.; Kummer, A.; Abonyi, J. Edge-Computing and Machine-Learning-Based Framework for Software Sensor Development. *Sensors* **2022**, *22*, 4268. [CrossRef] [PubMed]
31. Giuliano, F.; Pagano, A.; Garlisi, D.; Cattai, T.; Cuomo, F. Sustainable application-aware gateway deployment for massive IoT networks. *Internet Things* **2026**, *37*, 101940. [CrossRef]
32. Li, H.; Li, X.; Qian, Z.; Qin, X. Resource-aware service function chain deployment in cloud-edge environment. In *IEEE INFOCOM 2021—IEEE Conference on Computer Communications Workshops, INFOCOM WKSHPS 2021*; IEEE: Piscataway, NJ, USA, 2021. [CrossRef]
33. Faraji-Mehmandar, M.; Ghobaei-Arani, M.; Shakarami, A. A cost-aware IoT application deployment approach in fog computing. *Clust. Comput.* **2025**, *28*, 199. [CrossRef]
34. Islam, J.; Kumar, T.; Kovacevic, I.; Harjula, E. Resource-Aware Dynamic Service Deployment for Local IoT Edge Computing: Healthcare Use Case. *IEEE Access* **2021**, *9*, 115868–115884. [CrossRef]
35. Atlam, M.; Attiya, G.; Elrashidy, M. Resource-Aware Deep Learning Deployment for IoT–Fog Environments: A Novel BSIR and RAG-Enhanced Approach. *AI* **2026**, *7*, 44. [CrossRef]

36. Abdelmoumen, A.; Benzadri, Z.; Rodriguez, I.B. A Service-Oriented Framework for Resource-Aware System-of-Systems Modeling in IoT Environments. In *Service-Oriented Computing—ICSOC 2024 Workshops*; Lecture Notes in Computer Science; Springer: Singapore, 2026; Volume 15834, pp. 163–169. [CrossRef]
37. Ashwin, M.; Solleti, P.K.; Kodati, S.; Ravi, T.; Parasa, G.; Vamsikrishna, M.; Vetrithangam, D. Lightweight Convolutional Neural Network based Resource-Aware Energy-Efficient Detector within Edge–Fog-enabled Industrial IoT systems. *Int. J. Glob. Acad. Sci. Res.* **2026**, *5*, 85–115. [CrossRef]
38. Ford, T.N.; Gamess, E.; Ogden, C.; Gamess, E.; Ford, C.O.T.N.; Ford, T.N.; Gamess, E.; Ogden, C. Performance Evaluation of Different Raspberry Pi Models as MQTT Servers and Clients. *Int. J. Comput. Netw. Commun.* **2022**, *14*, 1–18. [CrossRef]
39. Jaladara, H.S.; Pahlevi, R.R.; Nuha, H.H. System Usability Scale Analysis of Infusion Fluid Level Monitoring and Notification System Using IoT. In *Proceedings of the IEEE International Conference on Communication, Networks and Satellite, COMNETSAT 2022*; IEEE: Piscataway, NJ, USA, 2022; pp. 112–117. [CrossRef]
40. Maulana, S.M.R.; Huda, H.; Jibran, M.T.S. Design and Analysis of the Use of Heart Rate and SpO2 Monitoring System based on an RFID-integrated Pulse Oximeter PPG Reflector using System Usability Scale. In *Proceedings of the 3rd International Conference on Software Engineering and Information Technology, ICoSEIT 2025*; IEEE: Piscataway, NJ, USA, 2025. [CrossRef]
41. Brooke, J. SUS: A 'Quick and Dirty' Usability Scale. In *Usability Evaluation In Industry*; CRC Press: Boca Raton, FL, USA, 1996; pp. 207–212. [CrossRef]
42. Bangor, A.; Kortum, P.; Miller, J. Determining what individual SUS scores mean: Adding an Adjective Rating Scale. *J. Usability Stud.* **2009**, *4*, 114–123.
43. Noprianto; Funabiki, N.; Kyaw, H.H.S.; Brata, K.C.; Kotama, I.N.D. A Proposal of Secure and Automated Over-the-Air Firmware Update Mechanism for IoT Devices Using Continuous Integration and Continuous Delivery. *Sensors* **2026**, *26*, 1535. [CrossRef] [PubMed]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.