automation

MDPI

Article

Industrial Intelligence at the Edge: Dynamic Fog Orchestration on Low-Cost Computing Nodes

Santiago Urbina, Dennis Proaño and William Oñate *

Carrera de Electrónica y Automatización, Universidad Politécnica Salesiana (UPS), Quito 170146, Ecuador; surbinay@est.ups.edu.ec (S.U.); maproanoa4@est.ups.edu.ec (D.P.)
* Correspondence: wonate@ups.edu.ec; Tel.: +593-989-325-276

Abstract

This study presents the implementation and experimental evaluation of a resource-aware Fog Computing architecture for executing heterogeneous services at the storage station of a Festo MPS-500 manufacturing system. The architecture integrates one Fog Master Node and two Fog Slave Nodes, with cloud-based service provisioning and resource monitoring using Node Exporter, Prometheus, and Grafana. The evaluation included facial recognition for access control, color-based token recognition for industrial identification, and a multiprocessing-based mathematical workload designed to increase computational demand. The Fog Master Node orchestrates incoming requests according to CPU utilization, operating temperature, and service execution status, using admission thresholds of 60% CPU utilization and 70 °C. The recognition services achieved image processing times of approximately 10 ms and 100% accuracy under the evaluated experimental conditions. During periods of high computational demand, the orchestration mechanism restricted new task admission according to the predefined criteria, while node temperatures remained below the thermal threshold without requiring forced-air cooling. The results demonstrate the feasibility of integrating containerized service provisioning, resource monitoring, and resource-aware orchestration on low-cost computing nodes in an industrial manufacturing environment.

Keywords: Fog Computing; resource-aware orchestration; industrial automation; Industry 4.0; Edge Computing

1. Introduction

The digitalization of information in Internet of Things (IoT) applications, such as smart city services, healthcare, housing, transportation, and agriculture, is gaining increasing importance as the cloud is progressively adopted as a unified platform to support these services. However, the flexibility demanded by today's market requires a digitalized factory supported by several key elements, including an interoperable architectural model across the entire production chain, immediate response capabilities, decision-making mechanisms, cybersecurity, quality of service (QoS), standardized communication channels, multilayer storage, and energy efficiency [1]. These requirements are characteristic of critical systems in which end devices must be capable of acquiring and monitoring production activities in real time [2]. This challenge can be addressed by leveraging the resources and services provided by different cloud platforms. Nevertheless, smart production lines must maintain flexibility and adaptability in response to sudden changes in production batch types. Consequently, modern industry is considering different computing paradigms, including Fog Computing

Check for updates

Academic Editor: Quanyan Zhu

Received: 29 September 2026
Revised: 19 November 2026
Accepted: 20 November 2026
Published: 24 November 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.

(FC), which plays a crucial role in Industry 4.0 and the Industrial Internet of Things (IIoT) by enabling data processing closer to the edge of the network infrastructure. This approach reduces latency and bandwidth consumption, increases the autonomy of cyber-physical systems, and enhances security and privacy. Furthermore, as an emerging technology, FC allows flexible architectural designs tailored to specific manufacturing requirements through the integration of edge devices, agents, Edge Computing (EC), Manufacturing Execution Systems (MES), and Cloud Computing [3]. This provides a promising near-term framework for developing digital transformation and evolution solutions for companies requiring greater scalability or optimization of their production processes [4].

Recent research has identified resource management and task scheduling as central challenges in Fog Computing environments, particularly because of the limited computational capacity and heterogeneous characteristics of distributed nodes. Costa et al. [5] reviewed orchestration mechanisms for coordinating applications and services across Fog infrastructures, while Hosseinzadeh et al. [6] systematically classified task scheduling strategies according to their underlying mechanisms and performance criteria. Similarly, Vergara et al. [7] examined resource allocation approaches in Fog–Cloud environments, emphasizing the challenges associated with dynamic resource availability and heterogeneous infrastructure. These studies establish the importance of resource-aware service management; however, the practical evaluation of orchestration policies under specific industrial hardware and workload constraints requires further investigation.

This study presents the design, implementation, and experimental evaluation of a resource-aware Fog Computing architecture deployed at the storage station of a Festo MPS-500 manufacturing system. The proposed architecture comprises a centralized Fog Master Node and two Raspberry Pi 4 Fog Slave Nodes, with service admission decisions based on CPU utilization, operating temperature, and process execution status. Containerized services are provisioned and managed through a distributed execution infrastructure, enabling the evaluation of heterogeneous workloads involving facial recognition, color-based token identification, and mathematical computation. The main contribution lies in the experimental integration and characterization of resource-aware service admission, container lifecycle management, and distributed execution on low-cost computing nodes within an industrial environment. The study assesses the feasibility and operational behavior of this implementation under the evaluated workload conditions, without claiming the development of a new general-purpose scheduling algorithm.

2. Related Work and Research Gap

2.1. Resource-Aware Task Scheduling in Fog Computing

Task scheduling in Fog Computing has been investigated through strategies that consider communication requirements, computational resources, and application-specific constraints. Caminero and Muñoz-Mansilla [8] developed a network-aware scheduling mechanism by extending Kubernetes to incorporate network conditions and application deadlines into container placement decisions. Their approach addresses the feasibility of executing services under communication and timing constraints, illustrating how task allocation can depend on factors beyond the computational capacity of individual nodes.

In an industrial application context, Teoh et al. [9] proposed a genetic algorithm-based resource management approach integrated with a machine learning model for predictive maintenance in Industry 4.0. The scheduling strategy was evaluated using FogWorkflowsim and compared with MinMin, MaxMin, first-come-first-served, and RoundRobin policies in terms of execution time, cost, and energy consumption. Although the study addresses industrial resource management, its scheduling evaluation was simulation-based and did not investigate temperature-dependent admission on physical Fog nodes [9].

Optimization-based approaches have also been proposed to improve resource allocation. Liu et al. [10] investigated a scheduling strategy combining particle swarm optimization and an artificial bee colony-based mechanism to address workload distribution and task scheduling across Fog resources. In a different approach, Krivić et al. [11] developed a dynamic scheduling mechanism for contextually categorized IoT services, considering deployment restrictions, communication latency, and quality-of-service requirements. These contributions demonstrate how scheduling policies can incorporate optimization objectives or service characteristics when selecting execution resources.

More recent investigations have introduced predictive and adaptive decision-making mechanisms. Nazeri et al. [12] proposed an energy-aware workflow scheduling framework incorporating an adaptive neuro-fuzzy inference system (ANFIS) to predict future resource loads and an I-ALO-GWO optimization algorithm for workflow scheduling. Their approach integrates workload prediction with resource management to address energy consumption. Jin and Rezaeipanah [13] developed DTA-FLE, a hierarchical task allocation method that uses fuzzy logic to classify tasks across Fog–Cloud layers and dynamically schedules their execution according to resource requirements and timing constraints. The method was evaluated using iFogSim simulations against alternative scheduling strategies.

Distributed decision-making represents another direction in resource-aware orchestration. Liutkevičius et al. [14] proposed an agent-based model incorporating computational resources, battery conditions, and security-related constraints into service placement decisions. Unlike centralized scheduling architectures, their approach distributes orchestration responsibilities among cooperating agents and addresses service mobility and resilience.

These investigations reveal distinct scheduling objectives and control structures. Network-aware policies emphasize communication and timing constraints, optimization-based methods seek improved allocation according to predefined objectives, and predictive or fuzzy approaches adapt decisions to changing resource conditions. In contrast, threshold-based admission policies provide a simpler mechanism for determining whether a node is eligible to receive a service request. Their suitability must therefore be examined in relation to the operational requirements and experimental conditions of the intended deployment.

2.2. Fog Orchestration in Industrial and Resource-Constrained Environments

The practical implementation of Fog orchestration involves additional challenges associated with service deployment, coordination, and execution on heterogeneous computing devices. Basak and Srirama [15] developed FogDEFT, a framework that uses the Topology and Orchestration Specification for Cloud Applications (TOSCA) and Docker Swarm to support the dynamic deployment of containerized services. Their work emphasizes application portability and interoperability, addressing how services can be provisioned across distributed Fog resources.

Davoli et al. [16] proposed a Fog orchestrator architecture incorporating service model awareness into resource selection and deployment decisions. The architecture was experimentally evaluated using real-world equipment, while a complementary simulation-based analysis examined the resource selection process. Their results provide evidence of the feasibility of deploying services dynamically in a physical Fog infrastructure. This work is particularly relevant because it demonstrates that experimental orchestration on real hardware is already established in the literature, although its primary emphasis is service model-aware deployment rather than admission decisions based on processor utilization and operating temperature.

Other implementations have investigated orchestration using single-board computers. Liutkevičius et al. [14] included Raspberry Pi 4 devices in the evaluation of their distributed agent-based architecture and complemented physical experiments with emulation to in-

vestigate larger configurations. Similarly, Barros et al. [17] presented a Fog Computing model for energy management in microgrids, employing a Raspberry Pi 3 and first-in, first-out task scheduling alongside cloud-based optimization and simulated electrical-system components. These studies demonstrate different ways of combining physical computing platforms, orchestration mechanisms, and application-specific processing.

The reviewed implementations establish that containerized deployment, orchestration on Raspberry Pi devices, and experimental Fog infrastructures are not novel in isolation. Instead, the relevant distinctions concern how services are admitted and managed, which operational variables influence execution decisions, and how the resulting behavior is evaluated. These considerations are especially relevant when Fog nodes must execute heterogeneous workloads under limited computational and thermal operating conditions.

2.3. Comparative Analysis and Research Gap

Table 1 compares representative Fog scheduling and orchestration studies according to their decision criteria, strategies, evaluation platforms, and principal scope or limitations. The comparison includes optimization-based and adaptive scheduling methods, distributed orchestration architectures, deployment frameworks, and implementations using resource-constrained computing devices.

Table 1. Comparative Analysis of Fog Scheduling and Orchestration Approaches.

Study	Scheduling/Orchestration Criteria	Strategy	Evaluation Platform/Hardware	Scope and Limitations
Caminero and Muñoz-Mansilla (2021) [8]	Network conditions, deadlines	Network-aware Kubernetes scheduling	Kind-based Kubernetes cluster (1 master, 4 workers; single physical server)	Does not evaluate thermal admittance
Teoh et al. (2023) [9]	Execution time, cost, energy consumption	Genetic algorithm; comparison with MinMin, MaxMin, FCFS and RoundRobin	FogWorkflowsim simulation	Industrial predictive maintenance; no physical thermal-admission evaluation
Liu et al. (2023) [10]	Execution delay, energy consumption, workload balancing	PSO-based load balancing and PGABC task scheduling	Computational evaluation of a multi-cluster Fog model; physical hardware not confirmed	Optimization-based evaluation; no industrial thermal-admission test reported
Krivić et al. (2022) [11]	Service context, network latency, QoS and deployment restrictions	Context-aware dynamic scheduling and rescheduling	Docker-container-based simulated Fog nodes and network conditions	Simulated network scenarios; no industrial thermal-admission evaluation
Nazeri et al. (2024) [12]	Predicted resource load, energy consumption	ANFIS-based prediction and I-ALO-GWO scheduling	iFogSim simulation (scientific workflows)	Simulation-based energy-aware workflow scheduling; no physical Fog testbed reported
Jin and Rezaeipanah (2025) [13]	Task criticality, execution requirements, response time	DTA-FLE fuzzy task classification and scheduling	iFogSim simulation	Focus on response time, cost and energy; no industrial thermal evaluation
Liutkevičius et al. (2022) [14]	CPU, memory, battery, security	Distributed agent-based orchestration	2 Raspberry Pi 4 Model B (4 GB); Intel Core i5 PC; ESP8266 devices; agent-based emulation	Does not evaluate the specific thermal policy of this study

Table 1. *Cont.*

Study	Scheduling/Orchestration Criteria	Strategy	Evaluation Platform/Hardware	Scope and Limitations
Basak and Srirama (2024) [15]	Service dependencies, hardware heterogeneity and deployment requirements	TOSCA-based orchestration with Docker Swarm (FogDEFT)	Raspberry Pi 3 (1 GB), Raspberry Pi 4 (4/8 GB), PCs and virtual machines	Focus on deployment and interoperability; no CPU–temperature admission policy
Davoli et al. (2022) [16]	Service model compatibility, resource availability	Service model-aware orchestration	Physical Fog testbed and simulation	Focus on service deployment; no temperature-based admission
Barros et al. (2019) [17]	Task ordering and energy-management requirements	FIFO scheduling at Fog; genetic algorithm at Cloud	Raspberry Pi 3 (1.2 GHz, 1 GB RAM), Google Cloud VM and GridLab-D simulation	Hybrid physical–simulated microgrid; no temperature-based admission
This study	CPU, temperature, execution status	Centralized admission and task allocation	Raspberry Pi 4; Festo MPS-500	Two execution nodes; fixed thresholds; no comparative scheduler benchmark

The comparison indicates that existing research has addressed resource-aware task scheduling through network constraints, optimization algorithms, predictive models, and fuzzy decision-making [8–13]. Other studies have investigated distributed coordination, service deployment, and orchestration on physical Fog devices [14–16]. These contributions provide established foundations for Fog resource management, but their evaluation objectives and operational settings differ. Consequently, differences in scheduling criteria or implementation platforms should not be interpreted as evidence of superior performance without equivalent experimental comparisons.

Within this context, the present study focuses on the experimental implementation and characterization of a centralized Fog orchestration mechanism for a Festo MPS-500 industrial storage station (Festo Didactic, Denkendorf, Germany). The architecture employs two Raspberry Pi 4 execution nodes and uses CPU utilization, operating temperature, and process execution status to determine whether incoming service requests can be admitted. Its execution environment combines containerized service management with heterogeneous workloads involving facial recognition, color-based token identification, and mathematical computation.

This experimental implementation provides a basis for characterizing resource-aware service admission and execution behavior on low-cost computing nodes within a physical industrial environment. This experimental scope enables the examination of service execution and node operating conditions using low-cost hardware. Nevertheless, the two-node configuration, fixed admission thresholds, and lack of a direct scheduling benchmark limit conclusions regarding scalability, optimality, and comparative scheduling performance. These aspects define the boundaries of the present evaluation and motivate future investigations involving alternative admission policies and larger deployments.

3. Materials and Methods

In distributed industrial environments, the execution of computational tasks on resource-constrained devices can lead to uneven resource utilization and affect processing continuity. In this context, the proposed architecture provides a distributed processing mechanism designed to leverage the available resources of nodes located close to the production process and dynamically adapt task allocation according to their computational status.

The proposed architecture was implemented using a Fog Computing infrastructure comprising one Fog Master Node and two Fog Slave Nodes, responsible for the orchestration and execution of computational services, respectively. The master node receives processing requests, monitors the computational status of the Fog Slave nodes, and determines the most suitable node for executing each task. Once a node is selected, the corresponding service is transferred for execution. Upon completion, the result is returned to the master node, and the allocated resources are released. Additionally, a thermal criterion was incorporated into the orchestration mechanism to prevent new tasks from being assigned to nodes operating under unsuitable temperature conditions [14,15]. Figure 1 presents the overall architecture of the implemented system.

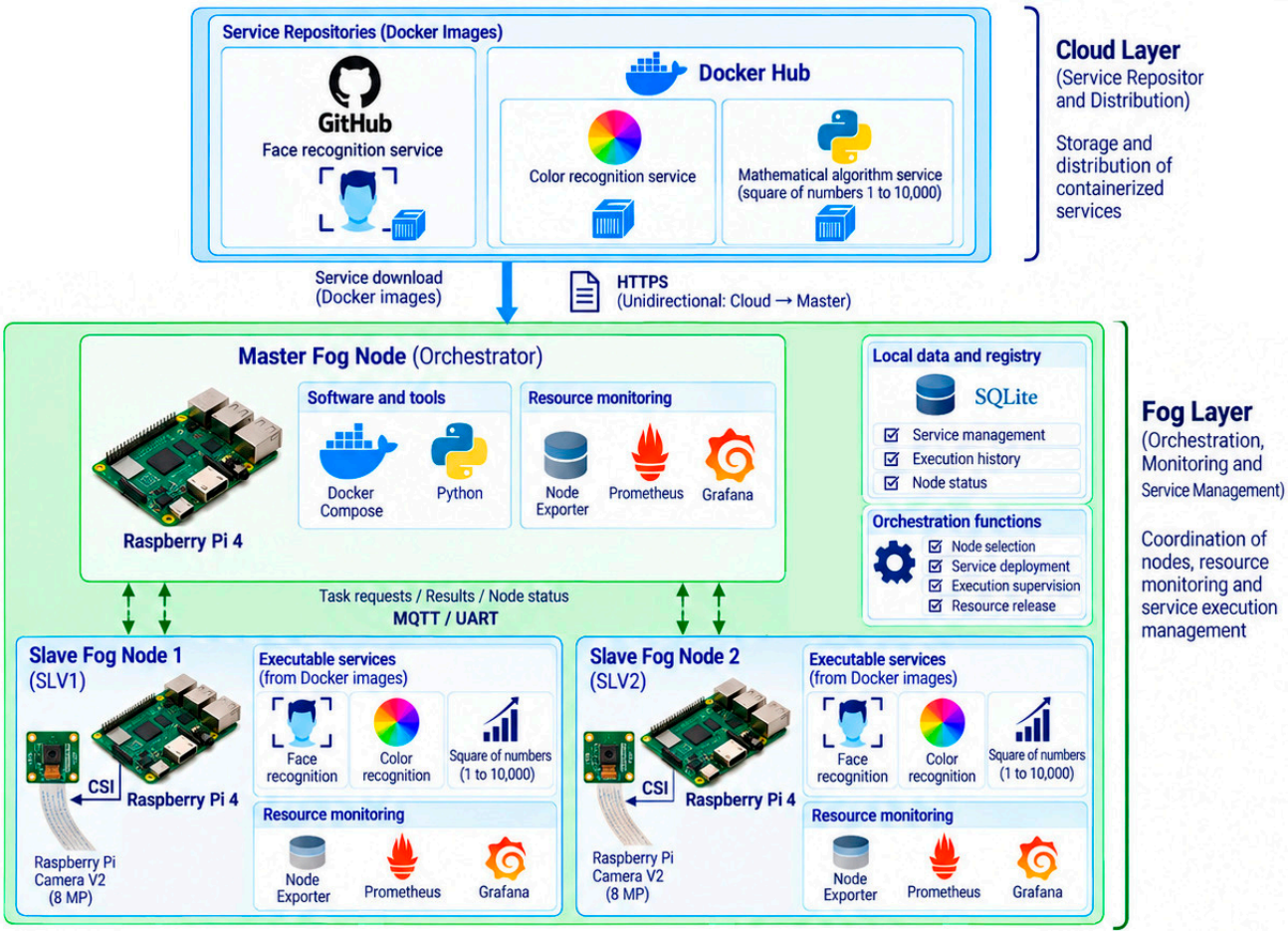


Figure 1. Fog Computing Architecture for Distributed Service Orchestration.

This section is organized into several subsections that progressively describe the proposed Fog architecture and the technologies supporting its implementation. The components of the infrastructure are presented from its architectural organization and the interaction mechanisms among its components to the services used for experimental validation. First, the Fog layer structure and the communication and interoperability mechanisms are described. Subsequently, the management of service images and their deployment within the processing infrastructure are addressed. Finally, the recognition and processing services implemented as application cases of the proposed architecture are presented, providing the basis for the experimental analysis of its behavior.

3.1. Fog Computing Architecture and Layered Organization

The proposed architecture is organized into three main layers: Cloud, Fog, and Edge. The Cloud Layer comprises GitHub and Docker Hub repositories, from which service images are provisioned to the Fog Master Node. The Fog Layer consists of one Fog Master Node and two Fog Slave Nodes, implemented on Raspberry Pi 4 Model B devices (Raspberry Pi Ltd., Cambridge, UK) running Raspberry Pi OS Bullseye 11 (64-bit), and is responsible for monitoring, orchestration, and distributed service execution. Finally, the Edge Layer integrates devices located close to the production process, including Raspberry Pi Camera v2 modules (Raspberry Pi Ltd., Cambridge, UK) connected to the Fog Slave nodes, as well as the elements associated with the recognition applications.

The Cloud Layer includes GitHub and Docker Hub repositories used to store and provide the Docker images and scripts required by the architecture services. The use of Docker containers enables each service to be packaged together with its dependencies and runtime environment, providing an isolated and reproducible environment for deployment on the Fog nodes. This approach allows specific versions of the services and their dependencies to be preserved, preventing subsequent modifications to the repositories or underlying libraries from directly affecting previously prepared versions [17]. Thus, the services remain available for on-demand retrieval and can be consistently deployed across the Fog nodes.

The Fog Layer consists of one master node and two Fog Slave nodes, implemented on Raspberry Pi 4 Model B boards, each equipped with a 1.5 GHz quad-core ARMv8 processor, 4 GB of RAM, a 16 GB SD card, 2.4 GHz Wi-Fi connectivity, and a static IPv4 address [16]. The Fog Master Node serves as the central coordination and monitoring component of the infrastructure, whereas the Fog Slave nodes provide the computational resources required for service execution. Docker Compose v2.20 [5] is used to manage the application lifecycle, enabling the required services to be deployed in a controlled manner on the execution nodes. The infrastructure also incorporates Prometheus v2.47 and Grafana v10.1 for collecting and visualizing system metrics. These metrics provide the master node with information about the operating conditions of the Fog Slave nodes and are used as input to the decision-making mechanism for assigning new tasks.

Additionally, the Fog Master Node incorporates a local SQLite v3.34 database to store the results obtained from the facial recognition and color-based token identification services. This storage mechanism preserves the information generated by both processes for subsequent retrieval, while task coordination and distribution are performed through the orchestration mechanisms implemented on the master node.

Communication mechanisms are selected according to the function of each link. Services stored in GitHub and Docker Hub are retrieved through HTTPS-based connections, which are used to transfer Docker images and the required files to the Fog Master Node. Once the services are available within the Fog infrastructure, the MQTT communication protocol is used to exchange information among the devices that constitute the intermediate layer [13].

The Edge Layer includes two Raspberry Pi cameras connected to the Fog Slave nodes through CSI interfaces and positioned in front of the MPS-500 storage station, where the processed tokens are deposited. These cameras acquire the images required for facial and color recognition tasks. Additionally, three push buttons are incorporated to deliberately trigger each of the evaluated services: facial recognition, color recognition, and mathematical processing.

Figure 2 presents the processing flow of the implemented Fog architecture, illustrating the interaction between the Fog Master Node, a slave node, and the services deployed through Docker containers. For clarity, only one slave node is represented, as the second

node follows the same functional structure and processing logic. The diagram integrates the main system components and operations, from request management and service acquisition to orchestration, execution, monitoring, and completion.

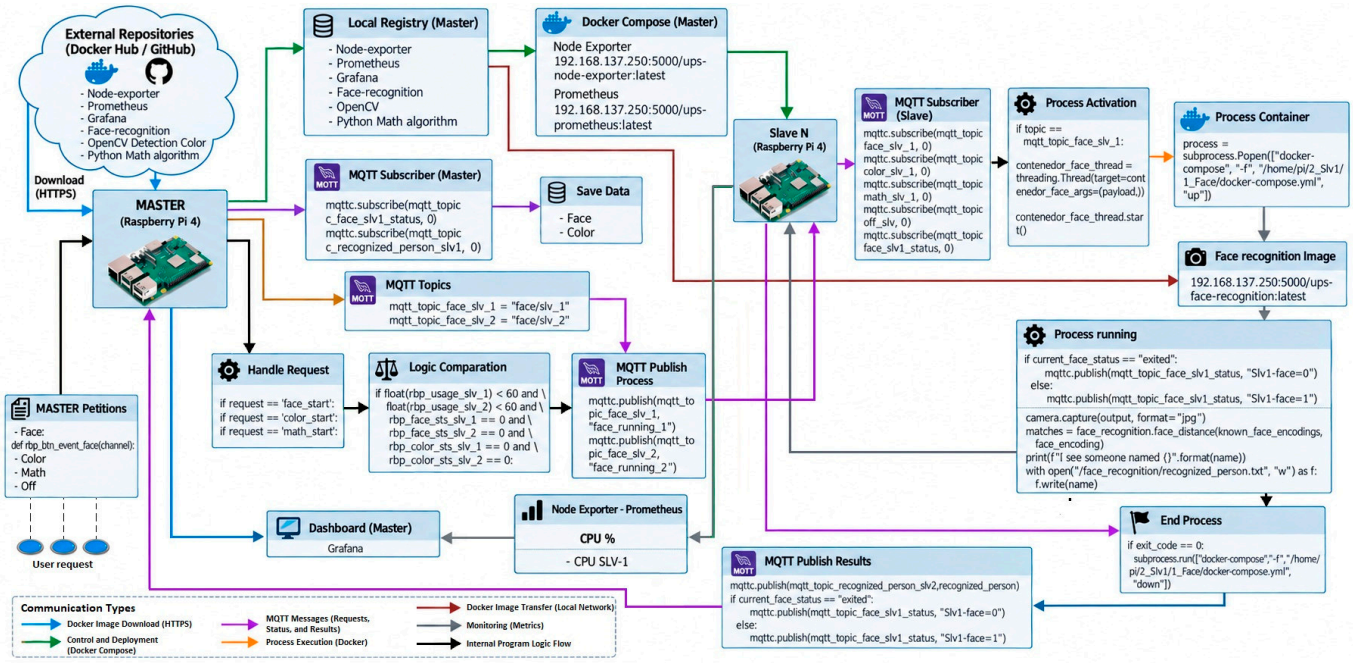


Figure 2. Fog Service Orchestration and Execution Workflow.

3.1.1. Communication and Interoperability Mechanisms

The implemented architectural model establishes two communication mechanisms according to the function of the system components. The first corresponds to the connection between the Fog Master Node and the external GitHub and Docker Hub repositories, using HTTPS requests to retrieve software resources [9]. Subsequently, Docker images are distributed through a local registry hosted on the master node, which the Fog Slave Nodes access through the Master Node IP address and port 5000. This structure distinguishes communication with external repositories from resource exchange within the Fog Layer. Communication performance was examined for two operations within the implemented architecture. The first involved local request–response exchanges between the Fog Master Node and the Fog Slave Nodes, including resource-status queries and service execution commands over the local network. The second involved remote requests from the Fog Master Node to the GitHub repository through HTTPS for software provisioning. The Master and Slave devices were installed within the same experimental workstation, with an approximate physical separation of 0.5 m. These operations were considered separately because local control messaging and remote repository access involve different communication paths and purposes. The corresponding observations were monitored through the infrastructure's Prometheus and Grafana environment.

The second mechanism establishes bidirectional communication between the Fog Master Node and the Fog Slave Nodes using the MQTT protocol and its publish–subscribe model [18]. Specific topics were defined according to the service and execution model, with separate channels for transmitting requests, status information, and results. Through the mqtt.subscribe() and mqtt.publish() operations, the nodes receive service-related instructions and transmit the information generated during execution. This organization maintains separate message flows while establishing a correspondence between each request, its destination node, and the corresponding response channel.

Additionally, the system incorporates a monitoring infrastructure designed to collect and visualize the operational metrics of the Fog nodes. The acquired data provide information on CPU utilization and device operating conditions, supplying the parameters required to assess their computational status. The monitoring information is integrated into the coordination mechanism of the Fog Master Node and serves as an input to the dynamic service allocation process.

3.1.2. Docker Image Acquisition and Deployment

The Docker image acquisition process is initiated by the Fog Master Node, which accesses the Docker Hub and GitHub repositories to retrieve the resources required by the implemented services. Node Exporter v1.6, Prometheus, and Grafana images are acquired for infrastructure monitoring, together with the denverdino/face_recognition resource for facial recognition, sixsq/opencv-python for color-based token identification, and python:3.9-slim for execution of the mathematical algorithm. The facial recognition resources are retrieved from GitHub, whereas the OpenCV v4.8 and Python images are obtained from Docker Hub. The acquired Docker images are then incorporated into a local registry hosted on the master node, which centralizes their storage and enables their subsequent distribution to the Fog Slave nodes.

Service configuration is defined through YAML files, which specify the Docker images to be used during deployment. From the Fog Master Node, the local registry is accessible through localhost:5000; for example, the Node Exporter image is referenced as localhost:5000/ups-node-exporter:latest. Docker Compose uses the definitions contained in the YAML files to manage the creation and initialization of the corresponding containers. In this manner, images retrieved from external repositories remain available within the local environment for subsequent use throughout the Fog architecture.

Once the Fog Master Node determines which services must be provisioned to a slave node, the selected node accesses the local registry through the master node's IP address and port 5000. For this purpose, the YAML files of the Fog Slave Nodes contain references in the form master-ip:5000/image-name:version; for example, master-ip:5000/ups-node-exporter:latest is used to retrieve the Node Exporter image. Based on this configuration, Docker Compose retrieves the specified images from the Fog Master Node registry and manages container deployment on the destination node. This procedure enables previously downloaded images to be reused and eliminates the need for each slave node to access external repositories individually.

3.1.3. Dynamic and Resource-Aware Service Orchestration

To experimentally evaluate the orchestration mechanism, two operational thresholds were established for the Fog Slave Nodes: CPU utilization below 60% and a maximum allowable temperature of 70 °C. The CPU threshold was selected through preliminary trial-and-error tests with the implemented workloads to establish a conservative admission criterion, whereas the temperature threshold was adopted as a preventive measure against excessive heating of the Raspberry Pi 4 devices. When a node reaches either threshold, the Fog Master Node restricts the assignment of new requests until its operating conditions return within the defined limits. The CPU threshold is applied only when admitting new requests; tasks already in execution are allowed to continue even if CPU utilization subsequently exceeds 60%. These thresholds were established for the hardware configuration and experimental conditions considered in this study and should not be interpreted as universally optimal values.

The orchestration mechanism was implemented on the Fog Master Node based on three parameters: service requests, CPU utilization and temperature metrics, and the execution status of processes running on the Fog Slave nodes. Service requests are generated through three push buttons associated with facial recognition, color-based token identification, and mathematical processing. Each activation is interpreted by the Handle Request module, which identifies the requested service and adds the request to a waiting queue. In parallel, the monitoring infrastructure collects CPU utilization and temperature metrics at 1 s intervals, while the Fog Slave Nodes report their execution status to the Master via MQTT every 2 s. These updates allow the Master to reassess pending requests and automatically assign them when an eligible slave node becomes available.

Task assignment is performed by the Logic Comparison module using a fixed-priority, resource-aware admission policy. Pending requests are processed in first-in, first-out (FIFO) order. Fog Slave Node 1 is always evaluated first and selected whenever it satisfies all admission requirements, regardless of the CPU utilization of Fog Slave Node 2. Only when Fog Slave Node 1 fails to meet at least one admission criterion does the Master evaluate Fog Slave Node 2. Node eligibility is determined by CPU utilization below 60%, operating temperature below 70 °C, and service execution status. In particular, a node cannot receive another instance of a service that is already running, whereas concurrent execution of different services is permitted when the corresponding admission requirements are satisfied. If neither node is eligible, the request remains in the queue and is automatically reconsidered as resources become available, without requiring resubmission by the user. Algorithm 1 summarizes the fixed-priority node selection and resource-aware admission procedure implemented by the Fog Master Node.

Algorithm 1. Fixed-Priority Resource-Aware Fog Orchestration

```
Input: FIFO request queue Q; Fog Slave Nodes S1 and S2; CPU threshold = 60%;
temperature threshold = 70 °C
while orchestration is active do
    Update available resource metrics and execution status
    if Q is not empty then
        request ← first pending request in Q
        assigned ← false
        for node in [S1, S2] do
            if CPU(node) < 60%
               and Temperature(node) < 70 °C
               and ServiceCompatible(node, request) then
                 Publish request to node via MQTT
                 Remove assigned request from Q
                 Update execution status
                 assigned ← true
                 break
            end if
        end for
        if not assigned then
            Keep request pending in Q
        end if
    end if
end while
```

Once a node has been selected, the Fog Master Node publishes the instruction through the MQTT topic corresponding to the requested service and destination Fog Slave Node. The selected node receives the request and updates its execution status, which is subsequently returned to the Master Node through the corresponding response channel. This information

is used to keep the variables evaluated by the Logic Comparison module up to date. Upon completion of the processing task, a new status update is transmitted to indicate task completion and allow the node to be considered again for subsequent assignments.

3.1.4. Dynamic Service Execution and Resource Management

Once the Fog Slave Node receives a processing instruction, the Process Activation module identifies the requested service and initiates its execution using a programming thread (threading.Thread), which invokes Docker Compose through subprocess.Popen(). This operation uses the corresponding YAML file and the docker-compose up command to create and start the container associated with the selected service. During processing, the node updates its execution status and generates the corresponding results. Upon task completion, the End Process module executes docker-compose down through subprocess.run(), stopping and removing the containers associated with the service and releasing the computational resources used during execution. The slave node then communicates the completion status and the obtained results to the Fog Master Node, allowing its availability status to be updated for subsequent requests. The Docker images remain stored in the local registry, enabling their reuse in subsequent executions without requiring them to be downloaded again from external repositories. In this manner, a dynamic service lifecycle is established based on on-demand service activation, execution, and resource release upon completion of each task.

3.2. Face Recognition System for Access Control

The Facial Recognition System (FRS) was implemented in Python using version 1.3 of the face_recognition library, which relies on dlib version 19.24. For enrollment, one reference photograph was obtained from each of the six participants and used to establish the facial patterns required for identification. Recognition tests were subsequently conducted on different days using new images captured by the Raspberry Pi Camera, ensuring that the evaluation images were distinct from those used for enrollment. Figure 3 presents the implemented facial recognition procedure.

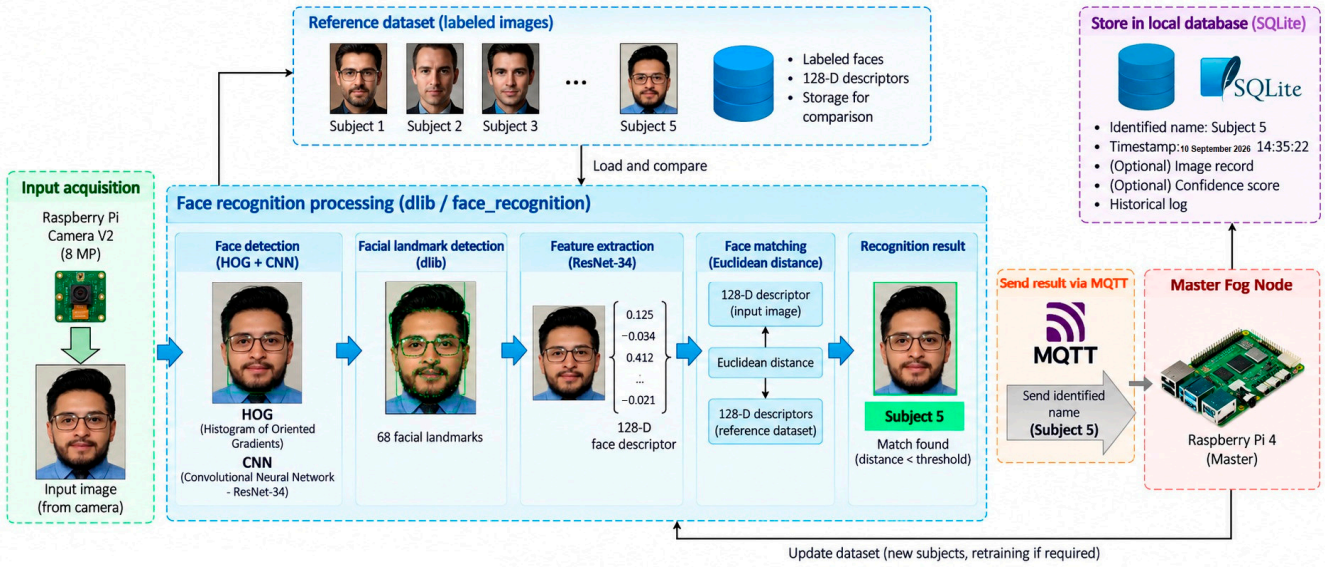


Figure 3. Face Recognition Workflow Based on HOG, CNN, ResNet-34, and Euclidean Distance for Identity Matching.

The algorithm begins with image acquisition and the detection of facial regions using techniques based on Histograms of Oriented Gradients (HOG) and Convolutional Neural Networks (CNNs). Subsequently, facial features are extracted using a deep learning model based on ResNet-34 [19], which generates a 128-dimensional descriptor to represent each face. These descriptors are compared with the corresponding reference descriptors by calculating the Euclidean distance in the feature space. Based on this distance and the established similarity criterion, the system determines whether the captured face matches a registered individual.

Under the operating conditions established for the implementation, the system can recognize faces located between 5 and 60 cm from the camera lens (see Figure 4). Once an individual is identified, the Fog Slave Node transmits the corresponding name to the Fog Master Node through the designated MQTT topic. Finally, the result is stored in the local SQLite database, preserving the information generated by the facial recognition process for subsequent retrieval.

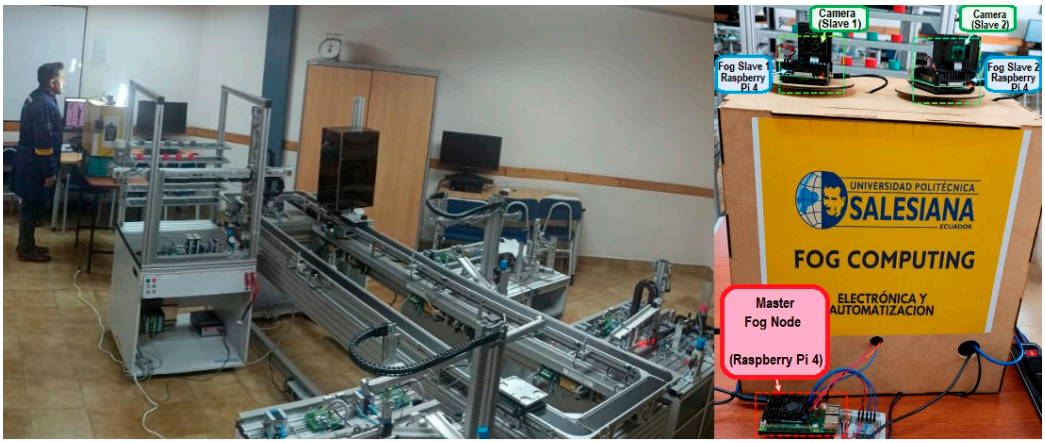


Figure 4. Experimental Setup for Face Recognition Evaluation in the MPS-500 System. The Spanish text on the equipment enclosure identifies Universidad Politécnica Salesiana (Salesian Polytechnic University) and the Electronics and Automation program.

3.3. Color-Based Token Recognition for Industrial Identification

The color-based token recognition system was implemented in Python using the OpenCV library and a Raspberry Pi Camera positioned in front of the storage station of the Festo MPS-500 system. Its function is to identify and count the red and green tokens present in the processing area using color segmentation and contour analysis techniques. Figure 5 presents the stages of the developed algorithm.

The processing begins with image acquisition and conversion to the Hue, Saturation, and Value (HSV) color space, enabling the chromatic information of the objects to be represented. Subsequently, the HSV ranges corresponding to the red and green colors are defined, and binary masks are generated to isolate the pixels that satisfy the specified color conditions. A contour detection procedure is then applied to the segmented regions, followed by area-based filtering to discard small regions associated with noise and prevent them from being counted as tokens [20]. The contours that satisfy the established criteria are used to determine the number of detected objects of each color.

As a result, the algorithm determines the number of red and green tokens identified in the image and records the information generated during processing.

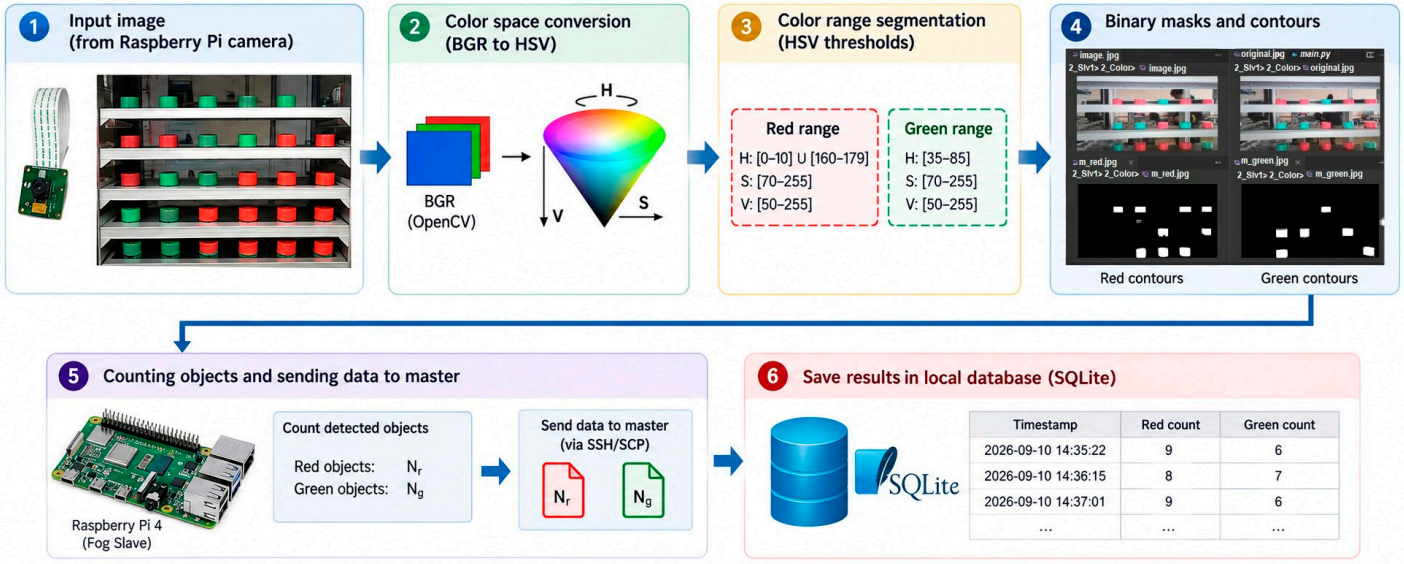


Figure 5. Color-Based token recognition workflow using HSV segmentation, contour detection, and object counting. Red and green indicate the corresponding token classes and their HSV segmentation ranges.

3.4. Mathematical Computation Service

The mathematical algorithm was incorporated as an additional computational workload to evaluate the behavior of the Fog Slave Nodes under increased CPU utilization and concurrent task execution. According to the official Raspberry Pi documentation, the Raspberry Pi 4 processor may reduce its operating frequency when temperatures approach 70 °C under high-load conditions [21]. For this reason, the admission thresholds defined for the architecture were experimentally set to CPU utilization below 60% and temperature below 70 °C. These criteria enable task allocation to be evaluated under preventive resource and thermal constraints.

As shown in Figure 6, the algorithm generates a computational workload by iteratively calculating the squares of integers from 1 to 10,000. To prevent this task from unnecessarily extending the experimental tests, a 40 s execution limit was implemented. The Python multiprocessing module is used to distribute the computation across independent processes and utilize approximately half of the Raspberry Pi 4 CPU cores, while preserving computational capacity for other tasks.

The mathematical workload can run concurrently with the recognition services when the node satisfies the defined CPU utilization and temperature thresholds. During the experimental evaluation, mathematical processing requests were deliberately generated under different operating conditions, including periods in which other services were already running, to introduce additional computational demand and observe the orchestration behavior. Once the termination condition is reached, the join() function is used to wait for the child processes to terminate before completing the main task. This procedure enabled the Fog system to be evaluated under simultaneous task execution and varying computational demands.

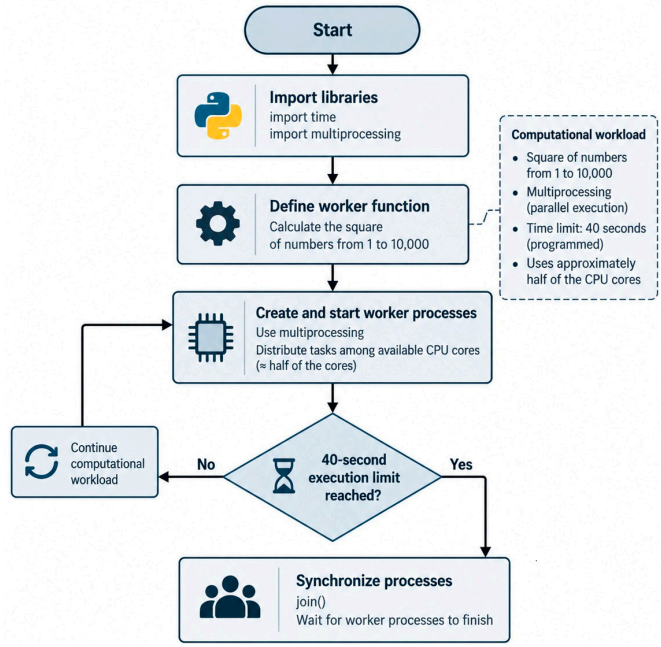


Figure 6. Time-Limited Mathematical Workload Using Multiprocessing.

4. Results

The experimental evaluation comprises three aspects: the computational performance of the implemented services, the validation of the recognition algorithms, and the behavior of the Fog architecture under varying workload conditions. CPU utilization and temperature metrics were collected using Node Exporter and Prometheus, while the recognition results were validated through tests conducted at the token storage station of the Festo MPS-500 system.

4.1. Computational Performance of the Implemented Services

The processing time and CPU utilization of the facial recognition and color-based identification services were evaluated. Additionally, the computational load generated by the mathematical algorithm was recorded, with its execution duration programmatically limited to control the duration of the experimental tests. Table 2 presents the results obtained.

Table 2. Processing performance and computational workload characteristics.

Service	Processing Time (ms)	CPU Utilization (%)	Programmed Time Limit (s)
Face recognition	10	52	—
Color recognition	10	32	—
Mathematical computation	—	90	40

Note: Processing time refers exclusively to the image processing operation performed by each recognition service and excludes Docker container startup and shutdown overheads. It does not represent the total service execution time.

The facial recognition and color-based token identification services achieved image processing times of 10 ms, with CPU utilization levels of 52% and 32%, respectively. These processing times exclude Docker container startup and shutdown overheads. In contrast, the mathematical algorithm reached 90% CPU utilization, exceeding the 60% admission threshold. A comparable processing time is not reported for this service because the 40 s value corresponds to the programmed execution limit rather than a measured processing

time. These differences enabled the architecture to be evaluated using tasks with varying computational demands.

In addition to the service processing measurements reported in Table 2, approximate communication times were observed for local and remote interactions. Request–response exchanges between the Fog Master Node and the Fog Slave Nodes were approximately 20 ms, whereas requests from the Fog Master Node to the remote GitHub repository exhibited an approximate latency of 700 ms. These observations refer to different operations: local coordination messaging and remote software repository access, respectively. Therefore, they do not constitute a direct comparison of end-to-end service execution latency between Fog and Cloud Computing. The 10 ms values reported in Table 2 correspond exclusively to the processing time of the recognition services and should not be interpreted as communication delays.

4.2. Recognition Accuracy

4.2.1. Face Recognition

The FRS was validated using a reference set of six subjects at distances ranging from 10 to 60 cm from the camera lens. Two lighting conditions were evaluated: white light at 470 lx and warm light at 210 lx, both applied across all considered distances. The 18 subject–distance combinations were evaluated under both lighting conditions, resulting in a total of 36 tests. Figure 7 presents the distribution of the evaluated combinations according to distance.

Face Recognition Results at Different Camera Distances and Lighting Conditions

Distance (cm)	Lighting condition	Subjects S1 	S2 	S3 	S4 	S5 	S6 	Combination tested
10	470 lx (white light)	—	✓	✓	—	✓	—	3
	210 lx (warm light)	—	✓	✓	—	✓	—	3
20	470 lx (white light)	✓	✓	—	—	—	✓	3
	210 lx (warm light)	✓	✓	—	—	—	✓	3
30	470 lx (white light)	✓	—	—	✓	✓	—	3
	210 lx (warm light)	✓	—	—	✓	✓	—	3
40	470 lx (white light)	✓	✓	—	—	✓	—	3
	210 lx (warm light)	✓	✓	—	—	✓	—	3
50	470 lx (white light)	—	—	✓	✓	—	✓	3
	210 lx (warm light)	—	—	✓	✓	—	✓	3
60	470 lx (white light)	—	—	✓	✓	—	✓	3
	210 lx (warm light)	—	—	✓	✓	—	✓	3
Total		6	6	6	6	6	6	36

✓ Correct recognition — Combination not documented

Figure 7. Face recognition results at different camera distances and lighting conditions.

Correct recognition was achieved in all 36 tests, corresponding to a 100% accuracy rate for the evaluated test set and confirming the operation of the algorithm at distances of up to 60 cm under the considered experimental conditions.

4.2.2. Color Recognition

Color-based token identification was evaluated at the storage station of the MPS-500 system under an illumination level of 1200 lx. The HSV range thresholds were adjusted to segment red and green objects, and six token configurations were evaluated, with ten repetitions performed for each configuration. The results are presented in Table 3.

Table 3. Color recognition results for different token configurations.

Fog Node	Token Configurations	Correct Tests	Incorrect Tests
Slave 1	6 red, 9 green	10	0
Slave 1	3 red, 12 green	10	0
Slave 1	10 red, 5 green	10	0
Slave 2	9 red, 6 green	10	0
Slave 2	3 red, 8 green	10	0
Slave 2	5 red, 8 green	10	0
Total	6 configurations	60	0

The algorithm correctly identified all six evaluated configurations, achieving 60 correct detections in 60 tests with no incorrect detections. The observed accuracy rate was 100% under the established lighting conditions and HSV parameters.

4.3. Resource-Aware Orchestration and Thermal Behavior

The mathematical algorithm was used to intentionally increase computational demand and evaluate the system response under the established admission constraints. During the experiments, the Fog Master Node monitored the CPU utilization and temperature of both Fog Slave nodes. When neither node satisfied the criteria for receiving a new task, requests remained in the waiting queue until one of the nodes returned to an admissible operating condition.

4.3.1. CPU Utilization

Figure 8 presents the CPU utilization profiles of Fog Slave Nodes 1 and 2 during the experimental evaluation, with measurements collected at one-second intervals. Both nodes exhibited repeated transitions between relatively low CPU utilization, approximately 25–30%, and periods of sustained computational demand, reaching maximum values of 88.10% in Slave Node 1 and 90.75% in Slave Node 2. These variations are consistent with the execution of computational workloads during the orchestration experiments. The annotations identify the startup and orchestration intervals considered in the evaluation.

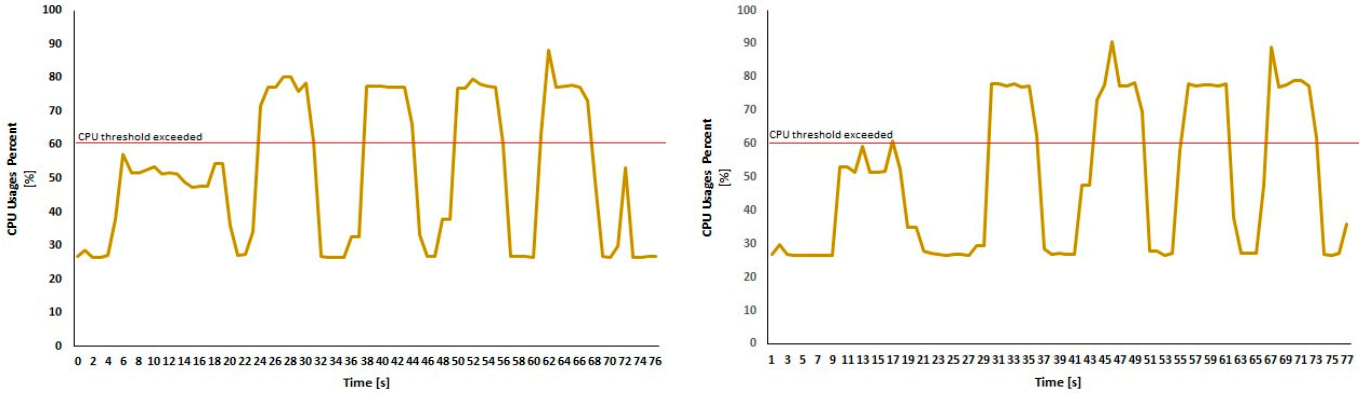


Figure 8. CPU utilization of the Fog Slave Nodes during service execution: (**a**) Fog Slave Node 1; (**b**) Fog Slave Node 2.

The mathematical processing service was deliberately used to generate sustained CPU demand and evaluate the behavior of the resource-aware admission mechanism. The 60% CPU threshold was applied when evaluating the admission of new requests rather than as an upper limit during task execution. Consequently, CPU utilization could exceed this threshold after a task had been accepted, as observed in both nodes. When neither node

satisfied the applicable admission criteria, pending requests remained in the waiting queue until an eligible node became available. The CPU utilization profiles also exhibit overlapping periods of elevated computational demand, indicating concurrent workload activity across both Fog Slave Nodes during portions of the experimental evaluation.

To complement the temporal CPU profiles, Table 4 presents descriptive statistics obtained from 77 monitoring samples collected for each Fog Slave Node at one-second intervals. The analysis characterizes the computational load experienced by both nodes during the evaluated workload sequence. CPU utilization exceeded the 60% admission threshold in 29 samples for Slave Node 1 and 28 samples for Slave Node 2. Furthermore, nine monitoring samples showed simultaneous utilization above this threshold in both nodes, indicating overlapping periods of high computational demand.

Table 4. CPU utilization statistics of the Fog Slave Nodes during the orchestration experiment.

Metric	Slave Node 1	Slave Node 2
Mean CPU utilization (%)	51.23	49.4
Maximum CPU utilization (%)	88.1	90.75
Minimum CPU utilization (%)	26.4	26.55
Samples above 60% CPU	29	28
Percentage of samples above 60% (%)	37.66	36.36

4.3.2. Thermal Behavior

Figure 9 jointly presents CPU utilization and temperature evolution for Fog Slave Nodes 1 and 2. Both variables were obtained from metrics collected during the experiments at one-second intervals. The maximum recorded temperatures were approximately 55 °C for Fog Slave Node 1 and 52 °C for Fog Slave Node 2. Therefore, the maximum temperature observed across both nodes was approximately 55 °C, corresponding to a thermal margin of 15 °C below the predefined admission threshold of 70 °C. This result confirms operation below the selected thermal threshold under the evaluated conditions but does not demonstrate a temperature reduction attributable to the orchestration strategy. Consequently, under the evaluated conditions, forced-air cooling was not required to maintain node operation.

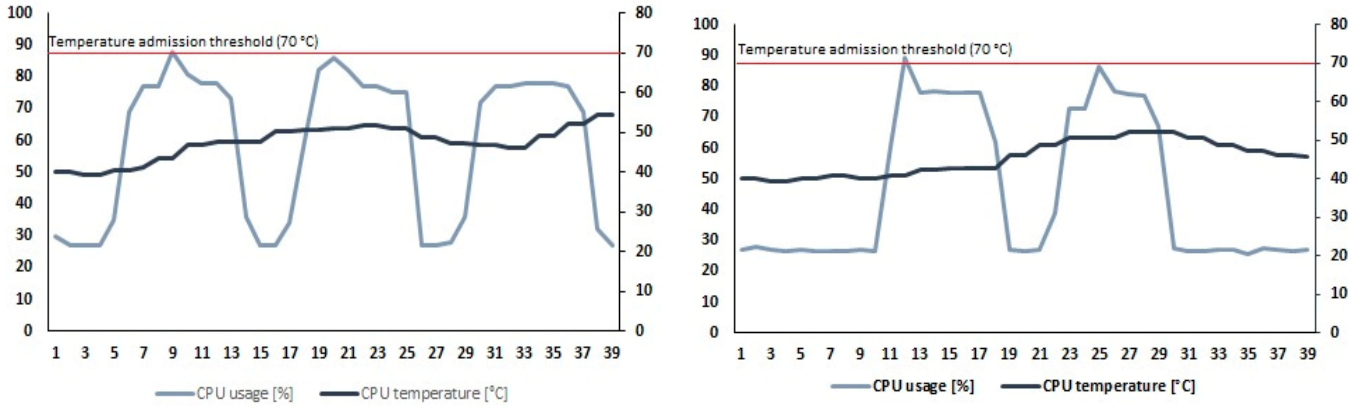


Figure 9. CPU utilization and temperature of the Fog Slave Nodes: (**a**) Fog Slave Node 1; (**b**) Fog Slave Node 2.

4.4. Industrial Process Timing Characterization

To characterize the temporal behavior of the industrial installation, transport times were measured for 25 workpieces consecutively traversing five segments of the Festo MPS-500 system. Table 5 presents the mean transport time for each segment and the cumulative mean

transport time per workpiece. These measurements characterize the physical transport process and exclude station processing times.

Table 5. Mean Workpiece Transport Times across the Festo MPS-500 Stations ($n = 25$).

Transport Segment	Mean Transport Time (s)	Workpieces
Distribution-Machining	7.60	25
Machining-Quality Control	7.34	25
Quality Control-Warehousing	12.00	25
Warehousing-Classification	18.20	25
Classification-Distribution	10.00	25
Cumulative transport time	55.14	25

As shown in Table 5, the shortest mean transport time was recorded between machining and quality control (7.34 s), whereas the longest occurred between warehousing and classification (18.20 s). The cumulative mean transport time was 55.14 s per workpiece. These results provide a temporal characterization of the industrial process, which can be considered alongside the image-processing measurements obtained from the Fog services.

5. Discussion

The experimental implementation demonstrates the feasibility of executing heterogeneous services within a resource-aware Fog Computing architecture deployed at the storage station of a Festo MPS-500 manufacturing system. The evaluated workloads included facial recognition, color recognition, and mathematical computation, representing different processing requirements. The orchestration mechanism coordinated service requests through a centralized Fog Master Node, while admission decisions considered CPU utilization, operating temperature, and process execution status. These results support the feasibility of integrating service management and distributed processing on low-cost computing nodes under the evaluated experimental conditions. The observed communication times provide an indicative characterization of local coordination and remote repository access under the experimental network conditions. However, because these operations differ in purpose and communication path, the reported values cannot establish a comparative latency advantage for executing the same service in Fog and Cloud environments. A controlled comparison using equivalent workloads and repeated timing measurements would be required to quantify such an advantage.

The findings should be interpreted in relation to existing Fog orchestration approaches. Previous studies have investigated network-aware scheduling, optimization-based resource allocation, predictive workload management, and fuzzy decision-making [8–13]. Furthermore, experimental implementations on physical Fog devices have been reported in the literature [14–16]. Unlike approaches that optimize task placement according to multiple performance objectives, the mechanism implemented in this study employs predefined resource-admission criteria to determine node eligibility. Its contribution therefore lies in the experimental integration of this policy with containerized service execution and monitoring in an industrial workstation.

The experimental results further characterize the operation of the resource-aware admission mechanism under varying computational workloads. The fixed-priority policy allows the Fog Master Node to consider an alternative execution node when the preferred node does not satisfy the admission criteria, while pending requests remain queued until suitable execution conditions are restored. The analysis of 77 CPU measurements per node revealed average utilization values of 51.23% and 49.40% for Fog Slave nodes 1 and 2, respectively. Moreover, both nodes simultaneously exceeded the 60% CPU admission

threshold in nine monitoring samples, indicating overlapping periods of high computational demand. These observations provide quantitative evidence of resource utilization across the distributed infrastructure and support the functional feasibility of the implemented admission mechanism under the evaluated conditions.

An important operational consideration was the configuration of the execution environment on resource-constrained devices. During implementation, the graphical desktop environment of Raspberry Pi OS was replaced with a console-based configuration to reduce the resources required by the operating system. This adjustment was adopted to support more stable operation of the containerized services. However, because no controlled comparison between graphical and console-based configurations was performed, its effect on CPU utilization, temperature, or execution performance cannot be quantified from the reported experiments.

The transport-time measurements presented in Table 5 provide additional context for the industrial application of the Fog architecture. The mean transport times ranged from 7.34 to 18.20 s, with a cumulative mean of 55.14 s per workpiece across the five segments. These values are considerably longer than the approximately 10 ms image-processing time observed for the recognition services. Nevertheless, this difference reflects the distinct timescales of mechanical transport and computational processing rather than a direct measure of orchestration efficiency. Since the complete service response time was not measured, the results do not establish whether the Fog architecture satisfies specific industrial timing requirements.

The use of Docker Compose supported service deployment and lifecycle management on the selected Raspberry Pi platform. Nevertheless, the suitability of container management technologies depends on the hardware architecture, available resources, and operational requirements. Consequently, the implementation choices reported here should be understood as specific to the evaluated infrastructure rather than as a general recommendation for all Fog Computing deployments.

Several limitations should be considered when interpreting the results. First, the experimental infrastructure comprised only two Fog Slave Nodes, limiting the assessment of scalability as the number of execution nodes, incoming requests, and industrial services increases. Although the architecture could conceptually accommodate additional Fog Slave nodes, this would require extending the monitoring, communication, and scheduling mechanisms. Furthermore, the centralized Fog Master Node represents a potential bottleneck for request coordination and service allocation as system demand increases, as well as a single point of failure. The fixed-priority node selection policy may also lead to uneven workload distribution because Fog Slave Node 1 is always prioritized when it satisfies the admission requirements. These limitations highlight the need for further evaluation under increasing request rates, larger Fog deployments, and fault-tolerant coordination mechanisms.

Second, the CPU and temperature admission thresholds were predefined rather than derived from an optimization procedure or systematic sensitivity analysis. The temperature criterion was incorporated into the admission logic, but the reported experiments do not establish whether the orchestration strategy reduced node temperatures compared with operation without thermal-aware admission, nor do they demonstrate its effectiveness under sustained thermal overload. Third, although the CPU monitoring data allowed the computational load of both Fog Slave Nodes to be characterized quantitatively, the experimental records did not provide the request-level timing and event information needed to evaluate waiting times, queue occupancy, end-to-end response times, throughput, and scheduling overhead. Furthermore, no direct comparison with alternative scheduling policies was conducted. Consequently, the results establish the observed operational behavior of the implemented mechanism rather than its comparative scheduling performance. Addition-

ally, the potential effects of delayed or unavailable CPU and temperature measurements on admission decisions were not systematically evaluated.

Finally, the recognition experiments were conducted under controlled conditions with a limited number of participants and token configurations. The facial recognition tests did not include unknown individuals, while the color identification experiments did not systematically evaluate variations in illumination, object placement, or partial occlusion. Consequently, the reported accuracy values should be interpreted within the evaluated conditions rather than generalized to more diverse industrial environments.

6. Conclusions

The communication observations indicated an approximate 20 ms request–response time between the Fog Master Node and the Fog Slave Nodes within the local network, whereas requests to the remote GitHub repository exhibited an approximate latency of 700 ms. Because these values represent different communication operations, they were not used to calculate a comparative latency reduction between Fog and Cloud Computing. Separately, the facial recognition and color identification services achieved processing times of approximately 10 ms and 100% accuracy under the evaluated experimental conditions. These findings support the feasibility of local service execution and coordination within the implemented Fog architecture, without establishing a direct end-to-end latency advantage over cloud-based service execution.

Effective workload orchestration was demonstrated using the mathematical algorithm designed to intentionally increase the computational demand of the Fog Slave Nodes. The 60% CPU admission threshold was established experimentally through preliminary trial-and-error tests using the processing algorithms considered in this study. When the mathematical workload was introduced, CPU utilization reached approximately 90%, intentionally creating a high-load condition. Although ongoing processes were allowed to exceed the 60% threshold, the orchestration mechanism prevented the assignment of new requests to a node once its CPU utilization exceeded this admission criterion. New task allocation resumed only after CPU utilization decreased and stabilized below the established threshold. This behavior demonstrates the role of the orchestration mechanism in preventing additional workload from being assigned to highly utilized nodes and in dynamically distributing requests according to the available computational resources. Such adaptability is particularly relevant in industrial applications where continuous processing and timely decision-making are required.

The thermal measurements showed that the Fog Slave Nodes operated within the predefined temperature limit under the evaluated conditions. During normal operation, node temperatures ranged from approximately 40 to 50 °C, while the maximum recorded temperatures under high computational load were approximately 55 °C for Fog Slave Node 1 and 52 °C for Fog Slave Node 2. The overall maximum of approximately 55 °C remained 15 °C below the 70 °C admission threshold. These observations indicate that the nodes maintained acceptable operating temperatures without requiring forced-air cooling during the experiments. However, since no comparative evaluation without temperature-based admission was conducted, the results do not establish a thermal reduction attributable to the orchestration strategy.

Author Contributions: Conceptualization, W.O.; methodology, S.U. and D.P.; software, S.U. and D.P.; validation, S.U., D.P. and W.O.; formal analysis, W.O.; investigation, W.O., S.U. and D.P.; data curation, S.U. and D.P.; writing—original draft preparation, W.O., S.U. and D.P.; writing—review and editing, W.O.; visualization, W.O.; supervision, W.O.; project administration, W.O. All authors have read and agreed to the published version of the manuscript.

Funding: The authors received financial support from Universidad Politécnica Salesiana for the research, use of laboratories, and publication of this article.

Data Availability Statement: The original contributions presented in this study are included in the article; further inquiries can be directed to the corresponding author.

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

Abbreviations

The following abbreviations are used in this manuscript:

IoT	Internet of Things
QoS	Quality of Service
FC	Fog Computing
EC	Edge Computing
MES	Manufacturing Execution System
ML	Machine Learning
FRS	Facial Recognition System
HOG	Histogram of Oriented Gradients
CNN	Convolutional Neural Network
HSV	Hue, Saturation, Value

References

1. Wójcicki, K.; Biegańska, M.; Paliwoda, B.; Górna, J. Internet of Things in Industry: Research Profiling, Application, Challenges and Opportunities—A Review. *Energies* **2022**, *15*, 1806. [CrossRef]
2. Korodi, A.; Crisan, R.; Nicolae, A.; Silea, I. Industrial Internet of Things and Fog Computing to Reduce Energy Consumption in Drinking Water Facilities. *Processes* **2020**, *8*, 282. [CrossRef]
3. Wang, Z.; Goudarzi, M.; Gong, M.; Buyya, R. Deep Reinforcement Learning-Based Scheduling for Optimizing System Load and Response Time in Edge and Fog Computing Environments. *Future Gener. Comput. Syst.* **2024**, *152*, 55–69. [CrossRef]
4. Kuchtíková, N.; Maryska, M. A Comparison of Cloud Computing vs. Fog vs. Edge for IIoT Sensors. In *Proceedings of the IDIMT 2023: New Challenges for ICT and Management—31st Interdisciplinary Information Management Talks*; Trauner Verlag: Linz, Austria, 2023; pp. 151–159. [CrossRef]
5. Costa, B.G.S.; Bachiega, J., Jr.; de Carvalho, L.R.; Araujo, A.P.F. Orchestration in Fog Computing: A Comprehensive Survey. *ACM Comput. Surv.* **2023**, *55*, 29. [CrossRef]
6. Hosseinzadeh, M.; Azhir, E.; Lansky, J.; Mildeová, S.; Ahmed, O.H.; Malik, M.H.; Khan, F. Task Scheduling Mechanisms for Fog Computing: A Systematic Survey. *IEEE Access* **2023**, *11*, 50994–51017. [CrossRef]
7. Vergara, J.; Botero, J.; Fletscher, L. A Comprehensive Survey on Resource Allocation Strategies in Fog/Cloud Environments. *Sensors* **2023**, *23*, 4413. [CrossRef] [PubMed]
8. Caminero, A.C.; Muñoz-Mansilla, R. Quality of Service Provision in Fog Computing: Network-Aware Scheduling of Containers. *Sensors* **2021**, *21*, 3978. [CrossRef] [PubMed]
9. Teoh, Y.K.; Gill, S.S.; Parlikad, A.K. IoT and Fog-Computing-Based Predictive Maintenance Model for Effective Asset Management in Industry 4.0 Using Machine Learning. *IEEE Internet Things J.* **2023**, *10*, 2087–2094. [CrossRef]
10. Liu, W.; Li, C.; Zheng, A.; Zheng, Z.; Zhang, Z.; Xiao, Y. Fog Computing Resource-Scheduling Strategy in IoT Based on Artificial Bee Colony Algorithm. *Electronics* **2023**, *12*, 1511. [CrossRef]
11. Krivic, P.; Kusek, M.; Cavrak, I.; Skocir, P. Dynamic Scheduling of Contextually Categorised Internet of Things Services in Fog Computing Environment. *Sensors* **2022**, *22*, 465. [CrossRef] [PubMed]
12. Nazeri, M.; Soltanaghaei, M.; Khorsand, R. A Predictive Energy-Aware Scheduling Strategy for Scientific Workflows in Fog Computing. *Expert Syst. Appl.* **2024**, *247*, 123192. [CrossRef]
13. Jin, W.; Rezaeipanah, A. Dynamic Task Allocation in Fog Computing Using Enhanced Fuzzy Logic Approaches. *Sci. Rep.* **2025**, *15*, 18513. [CrossRef] [PubMed]
14. Liutkevičius, A.; Morkevičius, N.; Venčkauskas, A.; Toldinas, J. Distributed Agent-Based Orchestrator Model for Fog Computing. *Sensors* **2022**, *22*, 5894. [CrossRef] [PubMed]
15. Basak, S.; Srirama, S.N. Fog Computing out of the Box: Dynamic Deployment of Fog Service Containers with TOSCA. *Int. J. Netw. Manag.* **2024**, *34*, e2246. [CrossRef]
16. Davoli, G.; Cerroni, W.; Borsatti, D.; Valieri, M.; Tarchi, D.; Raffaelli, C. A Fog Computing Orchestrator Architecture with Service Model Awareness. *IEEE Trans. Netw. Serv. Manag.* **2022**, *19*, 2131–2147. [CrossRef]

17. Barros, E.B.C.; Filho, D.M.L.; Batista, B.G.; Kuehne, B.T.; Peixoto, M.L.M. Fog Computing Model to Orchestrate the Consumption and Production of Energy in Microgrids. *Sensors* **2019**, *19*, 2642. [CrossRef] [PubMed]
18. Symeonides, M.; Georgiou, Z.; Trihinas, D.; Pallis, G.; Dikaiakos, M.D. Fogify: A Fog Computing Emulation Framework. In Proceedings of the 2020 IEEE/ACM Symposium on Edge Computing (SEC), San Jose, CA, USA, 11–13 November 2020; pp. 42–54. [CrossRef]
19. Zhang, J. Face Recognition Model Design Based on Convolutional Neural Networks. *Sci. Program.* **2022**, *2022*, 3922377. [CrossRef]
20. Song, P. Object Detection Based on HSV in OpenCV. *Appl. Comput. Eng.* **2025**, *121*, 116–122. [CrossRef]
21. Upton, E. New Product: Raspberry Pi 4 Case Fan. Raspberry Pi, 30 November 2020. Available online: https://www.raspberrypi.com/news/new-raspberry-pi-4-case-fan/ (accessed on 10 September 2026).

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.