EAI Endorsed Transactions
on Internet of Things
Research Article EAI.EU

CARE-Mid: Context-Aware IoT Middleware for Real-Time Resource Management in Smart Environments

J.Suresh[1*], Muneshwara M S[2], Siva Reddy Sonti[3], Prasan U D[4], Soujanya Saraswathi[5], and G.Sudhakar[6]

[1]Department of Computer Science and Engineering CARE College of Engineering,Thayanur, Trichy-620009, India
[2]Department of Computer Science and Engineering, BMS Institute Of Technology And Management, Avalahalli, Doddballapura Main Road, Yelahanka, Bengaluru-560119, Karnataka, India
[3]Department of Electronics and Communication, Engineering Mahatma Gandhi Institute Of Technology(A), Gandipet, Hyderabad -500075, India
[4]Department of CSE, Aditya Institute of Technology and Management, Tekkali, Srikakulam, 532201
5Department of ECE, CVR College of Engineering, Ibrahimpatnam, Hyderabad, Telangana, 501510, India
[6]Department of CSE, Koneru Lakshmaiah Education Foundation, Hyderabad-500043, Telangana, India

Abstract

Smart settings, such intelligent buildings, campuses, and hospitals, rely on a lot of different IoT devices that are always sending and receiving data and control tasks. Most middleware solutions today only focus on either connection or static service orchestration. They don't know much about the surrounding context, future workload, or Quality-of-Service (QoS) limits. This often results in poor task placement, wasteful cloud offloading, and frequent Service Level Agreement (SLA) violations when traffic is high. This article puts forth CARE-Mid, a context-aware IoT middleware designed for real-time resource management across both edge and cloud levels. CARE-Mid combines four important features: (i) getting semantic context and evaluating Quality-of-Context (QoC), (ii) predicting workload and resource demand based on context, (iii) scheduling multiple objectives that optimise latency, energy, and SLA compliance at the same time, and (iv) adaptive priority control based on current utility and recent violations. The framework is set up as a microservice-based platform on an edge-cloud testbed and tested with a genuine smart-building dataset. Experimental results indicate that, in comparison to a standard baseline middleware, CARE-Mid decreases average end-to-end latency by 20–30%, enhances the deadline-hit rate to around 97%, and minimises SLA violations by over 50% during heavy load conditions. A context-aware hybrid placement technique, on the other hand, uses 20–25% less energy and costs 20–25% less to run than cloud-only execution. Ablation experiments show that context modelling, prediction, and adaptive scheduling all play a big role in these improvements.

Keywords: Quality-of-Service; Service Level Agreement; Static service orchestration; Quality-of-Context; Latency; Energy Efficiency

Received on 22 May 2026, accepted on 10 September 2026, published on 12 October 2026

Copyright © 2026 Dr. J. Suresh *et al.*, licensed to EAI. This is an open access article distributed under the terms of the CC BY-NC-SA 4.0, which permits copying, redistributing, remixing, transformation, and building upon the material in any medium so long as the original work is properly cited.

doi: 10.4108/eetiot.13129

*Corresponding author. Email: jsuresh1983@gmail.com

1. Introduction

Smart houses, corporate buildings, campuses, and hospital wards are now possible thanks to the quick growth of sensors, actuators, and embedded devices [1]. In these situations, IoT devices keep an eye on the temperature, occupancy, air quality, equipment status, and user actions all the time. Control apps, on the other hand, control HVAC systems, lighting, access control, and safety systems. IoT middleware provides device abstraction, data routing, and service orchestration to help this very scattered ecosystem work together [2]. But a lot of the platforms that are already out there were made for best-effort data collecting instead of tight real-time resource management [3]. Most sensor streams are sent to a remote data center by traditional cloud-based solutions, where analytics and control logic are run [4]. This method makes management easier, but it might cause a lot of wide-area network delay and may overload centralised computational resources when demand surges [5]. Newer edge computing frameworks bring processing closer to devices, but they generally utilise static placement criteria or basic heuristics that don't take into account how the environment is changing, including how many people are using it, how comfortable the user is, or how important the application is [6]. Because of this, they might give too many resources to areas that aren't very useful and not enough to those that are important [7].

Another problem with many middleware systems is that they don't have predictive awareness. Most of the time, decisions are based on how much work is being done right now, without taking into account how workloads change over time or how the context changes [8]. This kind of behaviour can produce oscillations, where resources are scaled or jobs are moved only after an overload happens. This can lead to SLA breaches and wasted energy [9]. At the same time, new applications like real-time interior navigation, emergency response, and fine-grained energy optimisation need lower latency limits and more dependability than traditional telemetry dashboards [10]. To fill these shortcomings, to suggest CARE-Mid (Context-Aware Real-time Energy and Resource Management Middleware), a single framework that brings together context inference, Quality-of-Context (QoC) evaluation, workload prediction, and multi-objective scheduling across edge and cloud nodes. CARE-Mid doesn't handle all tasks and places the same way. Instead, it gives application-zone pairs context utility scores and utilises those scores to direct resource allocation to places where accurate context is both available and very important. A lightweight temporal model anticipates short-term demand based on the current situation, which lets you make proactive decisions about scale and positioning. Scheduling is a restricted optimisation issue that tries to minimise latency, energy, and SLA penalties at the same time while taking into account the capacity of the nodes. An adaptive control layer changes the priority of applications and the weights of SLAs depending on recent violations, making sure that mission-critical services stay safe amid overload. The framework is a type of middleware that is built on microservices and can be used with existing IoT platforms through common protocols like MQTT and REST. To show through a series of tests on a smart-building testbed that CARE-Mid greatly increases QoS and energy efficiency compared to typical baseline techniques. This shows how important it is for next-generation IoT middleware to describe and anticipate context clearly. The structure of the rest of the paper is as follows: Section 2 talks about the works that are linked; Section 3 talks about the recommended approach, Section 4 talks about the analysis of the results, and Section 5 makes the conclusion.

2. Related Works

Papaioannou, C., et al., [11] presents a Hierarchical Resource Management System (HRMS) designed for IoT-enabled smart cities, highlighting a decentralised architecture that operates at the building level and extends to city-wide applications. The Adaptive Resilient Node (ARN) is the main part of the system. It is designed to manage energy resources on its own and keep the system running by mending itself. This study describes the HRMS architecture, operational workflows, and main features. It demonstrates how the hierarchical structure facilitates real-time decision-making, fault tolerance, along with scalable resource allocation. The suggested system's lightweight communication between nodes helps balance workloads and make the system more responsive, which are two of the biggest challenges with controlling energy in cities. In different cities, tests have shown that the technology may make energy use more efficient by up to 50% and minimise downtime by 30%. This indicates that it can influence how cities grow in a way that is both sustainable and strong.

dos Santos, L. E., et al., [12] presents Smart Federated Middleware for Educational Institutions (SFMEI), a middleware engineered to enhance application development through federated learning in smart campuses. It also makes it easy for campuses to work together to build models using data that only they can see. Experiments show that SFMEI works to protect data privacy while yet keeping prediction performance high. To forecast time series data on CO2 levels, temperature, energy consumption, and water usage, a Long Short-Term Memory (LSTM) network was utilised. SFMEI did well, receiving an F1 score that was about the same as what it would get in a central situation. The research also examined various federated learning aggregation algorithms, such as FedAVG and FedSGD, concluding that FedSGD outperformed FedAVG in the majority of scenarios.

Serepas, F., et al., [13] talks about how to make and build an embedded IoT gateway that works on the ESP32 microcontroller. The suggested architecture allows local control, distributed intelligence, and safe on-site data storage while relying less on cloud infrastructure. It does this by leveraging lightweight communication protocols including Message Queuing Telemetry Transport (MQTT) and REST APIs. The gateway can monitor energy use,

carry out control orders, and give users an easy-to-use web-based dashboard with little extra resource use, as shown by a real-world deployment in an educational building. The method works well in tests, using between 3.6% and 6.8% of available memory (around 8.92 KB to 16.9 KB). When the single-page application (SPA) first loads, the RAM usage goes up to 52.4% for a short time before settling down to 50.8%. These results show that the ESP32 can work as an IoT gateway with very few resources. addressed device discovery procedures and better resource management are two areas that could be addressed in the future to make devices last longer. In general, the gateway is a cost-effective and vendor-neutral solution to establish strong and scalable IoT ecosystems. Rahaman, T. [14] utilises a case study methodology, evaluating three practical applications of SEMS: urban air quality monitoring in Beijing, China; industrial pollution oversight in Rotterdam, Netherlands; and water quality management in the Ganges River, India. The study emphasises the influence of SEMS in strengthening pollution forecasting, aiding regulatory enforcement, increasing public participation, and reducing environmental health hazards through the examination of several examples. The results show that SEMS greatly improve environmental governance by giving policymakers accurate, clear, and high-resolution pollution data. This makes it easier to make better decisions and design cities in a way that is good for the environment. The study also finds that sensor compatibility, data security, financial constraints, and regulatory standardisation are some of the main problems that need to be solved in order for SEMS to be used more widely. This study adds to the expanding corpus of research on technology-driven environmental management by showing how smart monitoring systems can help with pollution control around the world and support long-term ecological sustainability. Afzal, I., et al., [15] present the FORD framework, a latency-aware, fault-tolerant system for scheduling resources and managing data in smart cities that work in foggy conditions. This framework is designed to deal with apps that need to be done rapidly, such those in smart modes of transportation. The FORD framework uses latency-aware resource scheduling to make sure that tasks in smart cities happen as promptly as possible. It makes advantage of resources from both the cloud and fog. Through execution-based simulations, tasks are sent to the nearest nodes that are available and have the least amount of latency. If execution fails, a fault-tolerant approach is utilised to make sure that tasks are finished successfully. After the procedure is done, the cloud data center stores the data in a way that keeps it safe and dependable in the smart city ecosystem. Rahman, H., et al. [16] proposed a lightweight Fog-based Semantic Web approach to facilitate heterogeneity in the IoT gateway (FSH-IoT). The proposed method works with many types of devices and data, making it a good choice for a wide range of IoT situations. A machine learning method is also used to sort events so that key IoT apps can get the best service. The proposed FSH-IoT is compared to the Extended Global Sensor Network (XGSN), which is an open-source semantic sensing middleware for the web of things, and the Lightweight Semantic Web (L-SW), which is a lightweight semantic web solution for adding information to IoT gateways. According to the simulation results, FSH-IoT is better than XGSN and L-SW when it comes to data heterogeneity, accuracy (by around 15%), and response time (by about 30%).

3. Proposed Framework: CARE-Mid – Context-Aware Real-Time Resource Management Middleware

This part talks about the proposed CARE-Mid (Context-Aware Real-time Energy and Resource Management Middleware) for smart places like smart buildings, campuses, factories, or hospital wards. The framework is built to figure out what's going on (what), why it matters (why), how to react (how), and where to carry out actions (where) across different types of IoT devices and edge-cloud resources. Figure 1 shows how the proposed model works.

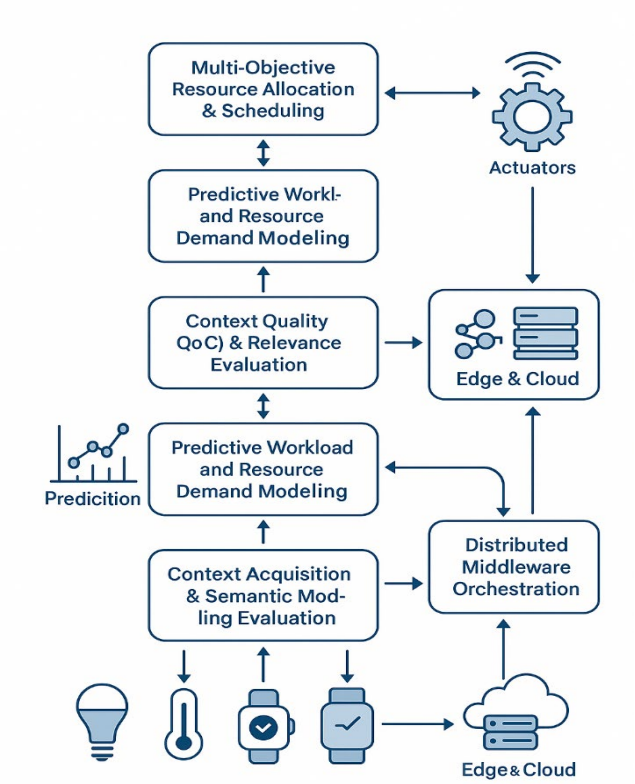

Figure 1. Workflow of the proposed model

CARE-Mid is organized into seven tightly coupled components:

Context Acquisition & Semantic Modeling
Context Quality & Relevance Evaluation
Predictive Workload and Resource Demand Modeling
Multi-Objective Resource Allocation & Scheduling
Priority- and SLA-Aware Adaptation
Distributed Middleware Orchestration
Algorithmic Summary & Complexity Analysis

Together, they enable real-time, context-aware and QoS-compliant resource management under dynamic workloads.

3.1 System Overview and Design Objectives

Let $\mathcal{D}$denote the set of IoT devices (sensors/actuators), $\mathcal{G}$the set of gateways/edge nodes, and $\mathcal{C}$the set of cloud data centers. At each discrete time slot $t \in \{1,2, \dots\}$, devices generate tasks (e.g., sensing events, inference requests) that must be assigned to computational resources while respecting latency, energy, and SLA constraints.

To denote by $\mathcal{A}$the set of applications (e.g., HVAC control, occupancy analytics, security monitoring). A task $i$associated with application $a \in \mathcal{A}$is characterized by a tuple:

$$\tau_i(t) = (d_i, a_i, c_i, \ell_i, \delta_i(t)) \quad (1)$$

Where $d_i \in \mathcal{D}$is the originating device, $a_i \in \mathcal{A}$is the logical application,$c_i$is the context vector capturing environmental state (occupancy, light level, user preference, etc.),$\ell_i$is the computational load (e.g., CPU cycles or FLOPs), $\delta_i(t)$is the required end-to-end deadline at time t. Clarify that each discrete time slot t represents a fixed decision interval of $\Delta t = 10$ minutes.

Each compute resource $r \in \mathcal{R} = \mathcal{G} \cup \mathcal{C}$has capacity:

$$\kappa_r = (f_r, m_r, b_r, E_r^{\max}) \quad (2)$$

where $f_r$denotes effective CPU frequency, $m_r$memory capacity, $b_r$link bandwidth, and $E_r^{\max}$maximum allowable energy budget per interval.

The primary design goals of CARE-Mid are:

G1 (Context-awareness): decisions depend explicitly on current and predicted context $c_i$and environment state.

G2 (Real-time performance): tasks meet deadlines $\delta_i(t)$whenever feasible.

G3 (Energy and cost efficiency): minimize energy usage and operational cost while respecting QoS.

G4 (Scalability and distribution): decisions are decomposable across edges and cloud.

To operationalize these goals, to first formalize context modeling.

3.2 Context Acquisition and Semantic Modeling Layer

Each device $d \in \mathcal{D}$produces a raw sensor vector at time t:

$$\mathbf{s}_d(t) = [s_{d,1}(t), s_{d,2}(t), \dots, s_{d,K_d}(t)]^\top \quad (3)$$

where $K_d$is the number of sensing modalities (temperature, CO_2, light, motion, etc.).

To transform raw signals into high-level context, CARE-Mid defines a mapping

$$\mathbf{c}_e(t) = f_\phi(\mathbf{s}_{\mathcal{N}(e)}(t), \mathrm{m}_e) \quad (4)$$

Where, $e$denotes a context entity (room, corridor, machine, patient bed, etc.), $\mathrm{s}_{\mathcal{N}(e)}(t)$is the concatenated sensor vector from the neighborhood $\mathcal{N}(e) \subseteq \mathcal{D}$physically or logically related to entity e, $\mathrm{m}_e$is metadata (location, floor, room type), $f_\phi(\cdot)$is a learned or rule-based context inference function parameterized by ϕ(e.g., a small neural network or Bayesian model),$\mathrm{c}_e(t) \in \mathbb{R}^M$is the resulting context state vector of dimension M(e.g., occupancy probability, comfort level, risk level).

Equation (1) addresses how raw signals are transformed into meaningful context and where context is anchored (at entities e). The why is to enable higher-level reasoning that is independent of specific sensor brands or hardware. For the entire environment, a global context state at time $t$is given by:

$$\mathrm{C}(t) = [\mathrm{c}_{e_1}(t)^\top, \dots, \mathrm{c}_{e_{|\mathcal{E}|}}(t)^\top]^\top \quad (5)$$

where $\mathcal{E}$is the set of all entities. $\mathrm{C}(t)$becomes the core input to prediction and scheduling modules, explaining what the middleware currently understands about the environment.

3.3. Quality-of-Context Measurement and Weight Selection

CARE-Mid evaluates the quality of each context record before it is used for workload prediction or resource allocation. For context type k associated with entity e at decision interval t, the Quality-of-Context (QoC) vector is defined as:

$$\mathbf{q}_{e,k}(t) = \left[q_{e,k}^{\mathrm{tim}}(t),\ q_{e,k}^{\mathrm{acc}}(t),\ q_{e,k}^{\mathrm{com}}(t),\ q_{e,k}^{\mathrm{rel}}(t)\right] \quad (6)$$

All QoC dimensions are normalized to the range [0,1], where a higher value represents higher-quality context.

3.3.1 QoC Dimensions & Metrics

Timeliness: Quantifies the freshness of a context record based on the elapsed time since the latest update:

$$q_{e,k}^{\mathrm{tim}}(t) = \exp\left(-\frac{\Delta t_{e,k}}{\tau}\right) \quad (7)$$

where $\Delta t_{e,k}$ is the elapsed time (in minutes) since the latest valid update and $\tau = 10$ minutes is the decision-interval duration. Context received during the current interval receives a high score, whereas delayed records decay exponentially.

Accuracy: Represents the expected correctness of the inferred context value. For categorical context types (occupancy, comfort, risk/alert, device health), it is derived from model validation accuracy:

$$q_{e,k}^{\mathrm{acc}} = \frac{N_k^{\mathrm{correct}}}{N_k^{\mathrm{evaluated}}} \quad (8)$$

where N_k^{correct} is the number of correctly inferred validation instances and $N_k^{\mathrm{evaluated}}$ is the total number of evaluated instances for context type k. Accuracy scores are updated after each validation cycle.

Completeness: Measures the proportion of required context attributes available for inference:

$$q_{e,k}^{\mathrm{com}}(t) = \frac{N_{e,k}^{\mathrm{available}}(t)}{N_k^{\mathrm{required}}} \quad (9)$$

where $N_{e,k}^{\mathrm{available}}(t)$ is the number of valid input attributes received for entity e and N_k^{required} is the total number of required attributes. Missing, delayed, or invalid attributes proportionally lower this score.

Reliability: Measures the stability of the context source over a trailing observation window of W intervals:

$$q_{e,k}^{\mathrm{rel}}(t) = 1 - \frac{N_{e,k}^{\mathrm{missing}}(t,W) + N_{e,k}^{\mathrm{anomalous}}(t,W)}{N_{e,k}^{\mathrm{received}}(t,W)} \quad (10)$$

where $N_{e,k}^{\mathrm{missing}}(t,W)$ denotes missing updates, $N_{e,k}^{\mathrm{anomalous}}(t,W)$ denotes invalid or fault-injected values, and $N_{e,k}^{\mathrm{received}}(t,W)$ is the total expected records. The window length is fixed to $W = 6$ intervals (1 hour).

3.3.2 Aggregate QoC Scoring & Weight Selection

The aggregate QoC score $Q_{e,k}(t)$ is computed as the weighted sum:

$$Q_{e,k}(t) = w_{\mathrm{tim}} q_{e,k}^{\mathrm{tim}}(t) + w_{\mathrm{acc}} q_{e,k}^{\mathrm{acc}}(t) + w_{\mathrm{com}} q_{e,k}^{\mathrm{com}}(t) + w_{\mathrm{rel}} q_{e,k}^{\mathrm{rel}}(t) \quad (11)$$

$$\text{subject to:} \quad w_{\mathrm{tim}} + w_{\mathrm{acc}} + w_{\mathrm{com}} + w_{\mathrm{rel}} = 1 \quad (12)$$

Optimal QoC weights were determined via a validation-set grid search (tested in increments of 0.05). The selected combination maximized a joint objective balancing macro-

F1 score for context inference and overall deadline-hit rate for scheduling. w_{tim} (Timeliness)= 0.35 Highest priority; stale context during real-time 10-minute intervals causes misallocations. w_{acc} (Accuracy)= 0.30 Ensures underlying model inferences are reliable. w_{com} (Completeness)= 0.20 Penalizes incomplete attribute vectors. w_{rel} (Reliability)= 0.15 Filters out noisy or unstable sensor feeds over time.

3.4 Predictive Workload and Resource Demand Modeling

To react before congestion or SLA violations occur, CARE-Mid predicts future workloads and resource demands conditioned on context.

Let $d_{a,r}(t)$denote the observed demand (e.g., CPU cycles/sec, memory MB, bandwidth Mbps) of application $a$on resource $r$during time slot t. To construct a context-aware predictive model:

$$\hat{d}_{a,r}(t+1) = g_\theta(\mathrm{h}_{a,r}(t), \mathrm{C}(t)) \tag{13}$$

Where $\hat{d}_{a,r}(t+1)$is the predicted demand for the next interval,$\mathrm{h}_{a,r}(t)$is a history vector combining past demands $[d_{a,r}(t), d_{a,r}(t-1), \dots]$,$g_\theta(\cdot)$is a parametric model (e.g., LSTM, GRU, or temporal CNN) with parameters θ.

Equation (7) explains how the middleware incorporates both temporal patterns and current context to estimate upcoming load. The model is trained online or offline via a loss function such as mean-squared error (MSE):

$$\mathcal{L}_{\text{pred}}(\theta) = \frac{1}{|\mathcal{T}||\mathcal{A}||\mathcal{R}|} \sum_{t\in\mathcal{T}} \sum_{a\in\mathcal{A}} \sum_{r\in\mathcal{R}} (d_{a,r}(t) - \hat{d}_{a,r}(t))^2 \tag{14}$$

where $\mathcal{T}$is the set of training time slots. In practice, where should this prediction run? For latency-critical applications, lightweight predictors run at gateways $g \in \mathcal{G}$; for heavy models, cloud resources $c \in \mathcal{C}$host the predictor. This placement decision itself can be optimized but is outside this core formulation.

3.5. Multi-Objective Resource Allocation and Scheduling Engine

At the heart of CARE-Mid lies a multi-objective scheduler that uses context and predictions to decide which task runs where and how much resource to allocate.

3.5.1 Decision Variables and Feasibility

Let $x_{i,r}(t) \in \{0,1\}$indicate whether task $i$at time $t$is assigned to resource r. For each task:

$$\sum_{r\in\mathcal{R}} x_{i,r}(t) = 1, \forall i \tag{15}$$

which enforces that each task is assigned to exactly one resource. The resource utilization of node $r$at time $t$is:

$$u_r(t) = \frac{1}{\kappa_r^{\text{CPU}}} \sum_i x_{i,r}(t) \cdot \ell_i \tag{16}$$

where $\kappa_r^{\text{CPU}}$is the CPU capacity in FLOPs per interval and $\ell_i$is the required CPU load of task i. To require:

$$u_r(t) \le 1, \forall r \tag{17}$$

to ensure feasibility.

3.5.2 Latency and SLA Constraints

The end-to-end latency of task $i$on resource $r$is modeled as:

$$L_{i,r}(t) = L_{d_i,r}^{\text{net}}(t) + \frac{\ell_i}{f_r(1-u_r(t))} + L_r^{\text{queue}}(t) \tag{18}$$

Where $L_{d_i,r}^{\text{net}}(t)$is the network latency from device $d_i$to resource r, $f_r$is effective CPU frequency, $L_r^{\text{queue}}(t)$is queuing delay at r. An SLA constraint requires:

$$L_{i,r}(t) \le \delta_i(t), \forall i, r \text{ with } x_{i,r}(t) = 1 \tag{19}$$

3.5.3. Energy Measurement and Estimation Procedure

Energy consumption is estimated from dynamic resource utilization and active execution durations rather than over the complete 10-minute decision interval. CPU utilization for each edge gateway is sampled every 100 ms using Intel Running Average Power Limit (RAPL) counters, while container-level CPU utilization for the cloud service is collected via Docker runtime statistics.

The instantaneous computing power $P_r(t)$ of resource node $r \in \mathcal{F}$ at time step t is calculated using a cubic power model:

$$P_r(t) = P_r^{\text{idle}} + \left(P_r^{\text{peak}} - P_r^{\text{idle}}\right) u_r(t)^3 \tag{20}$$

where $u_r(t) \in [0,1]$ represents the normalized CPU utilization at time t. For the experimental setup, edge gateways operate with $P_{\text{edge}}^{\text{idle}} = 18$ W and $P_{\text{edge}}^{\text{peak}} = 65$ W, whereas the cloud service nodes operate with $P_{\text{cloud}}^{\text{idle}} = 28$ W and $P_{\text{cloud}}^{\text{peak}} = 95$ W.

The computing energy consumed by resource r is evaluated by integrating power over the active task execution duration:

$$E_r^{\text{comp}} = \sum_{t\in\mathcal{T}_{\text{active}}} P_r(t)\Delta t \tag{21}$$

where $\Delta t = 100$ ms is the sampling interval.

The total energy per 10-minute interval (E_{total}) accounts for edge computing energy, cloud computing energy, and network communication energy:

$$E_{\text{total}} = \sum_{r\in\mathcal{F}_{\text{edge}}} E_r^{\text{comp}} + \sum_{r\in\mathcal{F}_{\text{cloud}}} E_r^{\text{comp}} + E^{\text{comm}} \tag{22}$$

Communication energy (E^{comm}) is modeled as: $E^{\text{comm}} = P_{\text{tx}} \cdot T_{\text{tx}} + P_{\text{rx}} \cdot T_{\text{rx}}$. where $P_{\text{tx}} = 1.4$ W and $P_{\text{rx}} = 1.0$ W denote the transmission and reception power levels, and T_{tx} and T_{rx} represent the cumulative active transmission and reception durations measured during the interval.

3.5.4 Multi-Objective Optimization Formulation

To define a multi-objective problem that encodes four high-level goals:

Minimize expected latency for higher-priority tasks.

Minimize energy and cost.

Minimize SLA violations.

Incorporate context utility $U_{a,e}(t)$to allocate resources where they matter most.

Let $\mathcal{I}(t)$be the set of tasks at time t, and suppose each task $i$is associated with entity $e_i$and application a_i. To define:

Latency cost:

$$J^{\text{lat}}(t) = \sum_{i \in \mathcal{I}(t)} \sum_{r \in \mathcal{R}} x_{i,r}(t)\, w_i^{\text{lat}} L_{i,r}(t), \qquad (23)$$

where $w_i^{\text{lat}}$is a latency weight derived from priority and context (see below).

Energy/cost:

$$J^{\text{en}}(t) = \sum_{r \in \mathcal{R}} C_r(t). \qquad (24)$$

SLA violation penalty:

$$J^{\text{SLA}}(t) = \sum_{i \in \mathcal{I}(t)} \sum_{r \in \mathcal{R}} x_{i,r}(t) \max(0, L_{i,r}(t) - \delta_i(t)). \qquad (25)$$

To embed context importance, to define:

$$w_i^{\text{lat}} = \omega_0 + \omega_1 U_{a_i,e_i}(t) \qquad (26)$$

where $\omega_0 \geq 0$and $\omega_1 \geq 0$are hyperparameters. High context utility $U_{a_i,e_i}(t)$increases the weight of latency for tasks that are currently context-critical (e.g., emergency alarms).

To then form a scalarized objective:

$$J(t) = \lambda_{\text{lat}} J^{\text{lat}}(t) + \lambda_{\text{en}} J^{\text{en}}(t) + \lambda_{\text{SLA}} J^{\text{SLA}}(t) \qquad (27)$$

with non-negative trade-off coefficients $\lambda_{\text{lat}}, \lambda_{\text{en}}, \lambda_{\text{SLA}}$. The instantaneous scheduling problem becomes:

$$\begin{aligned} \min_{\{x_{i,r}(t)\}} \quad & J(t) \\ \text{s.t.} \quad & \sum_{r \in \mathcal{R}} x_{i,r}(t) = 1 \; \forall i \in \mathcal{I}(t), \\ & u_r(t) \leq 1, \forall r \in \mathcal{R}, \\ & L_{i,r}(t) \leq \delta_i(t) + \epsilon, \forall i, r \text{ with } x_{i,r}(t) = 1, \\ & x_{i,r}(t) \in \{0,1\}, \end{aligned} \qquad (28)$$

where $\epsilon$is a small relaxation margin to maintain feasibility under uncertainty. This optimization answers how the middleware chooses execution locations and what trade-offs it makes between latency, energy, and SLA risk, driven by where and why context is important. Since (19) are combinatorial, CARE-Mid employs a hierarchical solution: continuous relaxations, greedy rounding, and local refinement at edges to ensure real-time performance.

3.6 SLA- and Priority-Aware Adaptive Control

Resource allocation must adapt when the environment or workloads change abruptly (e.g., sudden crowd in a corridor, fire, or medical emergency). CARE-Mid modulates priority and SLA weights based on context and observed violations. Let $\pi_a(t)$denote the priority level of application $a$at time t(e.g., between 0 and 1). To update it as:

$$\pi_a(t+1) = (1-\eta)\pi_a(t) + \eta \cdot \max_e U_{a,e}(t) \qquad (29)$$

where $\eta \in (0,1)$is a smoothing factor, $\max_e U_{a,e}(t)$captures the highest context utility across entities for application a.

When context strongly indicates urgency (e.g., abnormal temperature and smoke detection), $U_{a,e}(t)$becomes large and so does $\pi_a(t+1)$. This answers why CARE-Mid dynamically reorders priorities.

SLA penalties influence how strict deadlines should be enforced. Let $v_a(t)$be a moving average of SLA violations for application a. To update:

$$v_a(t+1) = (1-\rho)v_a(t) + \rho \cdot \text{viol}_a(t) \qquad (30)$$

where $\text{viol}_a(t)$is the observed violation ratio and $\rho$is a smoothing factor. To then adapt the SLA weight:

$$\lambda_{\text{SLA}}^{(a)}(t+1) = \lambda_{\text{SLA}}^{(a)}(t)(1 + \mu\, v_a(t+1)) \qquad (31)$$

with adaptation rate $\mu > 0$. This means that where SLA violations persist, the scheduler automatically tightens its focus on avoiding further violations by raising their penalty weight.

Finally, the task-level weight in (17) is refined as:

$$w_i^{\text{lat}}(t) = (\omega_0 + \omega_1 U_{a_i,e_i}(t)) \cdot (1 + \pi_{a_i}(t) + v_{a_i}(t)) \qquad (32)$$

thus, coupling context, priority, and SLA history into a single factor that guides how aggressively the scheduler reduces latency for each task.

3.7 Distributed Orchestration and Complexity Analysis

Smart environments frequently contain hundreds of gateways and thousands of devices, so where decisions are made becomes crucial. CARE-Mid therefore decomposes the global optimization into local problems solved by gateways with periodic coordination. Let $\mathcal{R}_k \subset \mathcal{R}$be the set of resources managed by gateway/cluster k, and $\mathcal{I}_k(t)$the tasks originating in its domain at time t. Gateway $k$solves a local version of (19):

$$\min_{\{x_{i,r}(t)\}_{i\in\mathcal{I}_k, r\in\mathcal{R}_k}} J_k(t) + \sum_{r\in\mathcal{R}_k} \nu_r(t) u_r(t) \qquad (33)$$

Where $J_k(t)$is the part of $J(t)$involving local tasks and resources, $\nu_r(t)$is a shadow price or dual variable received from a lightweight coordination service (cloud or leader gateway), representing the global pressure on resource r.

Equation (22) shows how global coupling is incorporated via dual variables instead of solving a monolithic problem. Gateways iteratively exchange utilization summaries and update $\nu_r(t)$(using, for example, a subgradient or ADMM-style update), leading to near-optimal yet real-time feasible solutions.

3.7.1 Gateway Coordination Protocol

CARE-Mid uses periodic gateway-to-coordinator communication to maintain a global view of distributed edge and cloud resources. At the beginning of every 10-minute decision interval, each gateway transmits one context-and-resource summary to the coordination service.

Summary Payload Contents:
Gateway identifier (g) and timestamp (t), Current CPU and memory utilization, Queue length and predicted workload ($\hat{d}_{t+1}$), Available bandwidth (BW_r),
Average context utility ($U_{a,e}(t)$), Local SLA-violation ratio and Local resource price or dual variable.
Protocol Workflow & Message Scaling
For an experimental architecture containing three edge gateways ($G = 3$) and one cloud coordinator, each decision interval consists of:

1. **Upstream Reporting:** 3 gateway-to-coordinator summary messages containing local context/resource status.
2. **Downstream Feedback:** 3 coordinator-to-gateway response messages containing the updated resource-price vector, global utilization summary, and placement guidance.

Therefore, the coordination mechanism exchanges **6 control messages per decision interval**, excluding normal application-task traffic. The exchanged summaries are lightweight and used strictly for resource coordination; raw sensor streams remain at the associated gateway unless cloud offloading is explicitly selected.
Coordination Metrics Equations
The total coordination bandwidth overhead during interval t is calculated as:

$$B_t = \sum_{g=1}^{G} \left(S_{g,t}^{\text{up}} + S_{g,t}^{\text{down}}\right) \qquad (34)$$

where: G is the total number of gateways,$S_{g,t}^{\text{up}}$ is the size of the summary message transmitted by gateway g,$S_{g,t}^{\text{down}}$ is the size of the coordinator response returned to gateway g. The total number of coordination messages exchanged per interval t is: $M_t = 2G$. For the default three-gateway configuration ($G = 3$), $M_t = 6$ messages per 10-minute decision interval.

3.7.2 Communication-Overhead Measurement

The communication overhead of CARE-Mid was measured separately from task-execution traffic. For every decision interval, the middleware recorded the number of coordination messages, total transmitted bytes, coordinator round-trip delay, and communication-energy estimate.

Recorded Coordination Parameters:
Total coordination message count (M_t), Aggregate transmitted payload size (B_t), Coordinator round-trip delay ($L_{g,t}^{\text{coord}}$) and Estimated communication energy (E_t^{coord})

Overhead Metrics Equations:
The coordination delay for gateway g during interval t is calculated as:

$$L_{g,t}^{\text{coord}} = t_{g,t}^{\text{response}} - t_{g,t}^{\text{summary}} \qquad (35)$$

where: $t_{g,t}^{\text{summary}}$ is the timestamp when gateway g transmits its summary payload, $t_{g,t}^{\text{response}}$ is the timestamp when gateway g receives the updated coordination response. The mean and 95th-percentile coordination delays were factored directly into the end-to-end latency calculations for tasks requiring cloud placement or global resource guidance. Communication energy consumption is estimated using transmission and reception durations:

$$E_t^{\text{coord}} = P_{\text{tx}} \sum_{g=1}^{G} \tau_{g,t}^{\text{tx}} + P_{\text{rx}} \sum_{g=1}^{G} \tau_{g,t}^{\text{rx}} \qquad (36)$$

where: P_{tx} and P_{rx} denote the transmission and reception power levels, respectively, $\tau_{g,t}^{\text{tx}}$ and $\tau_{g,t}^{\text{rx}}$ represent the message transmission and reception durations for gateway g. This calculated overhead was added directly to the total interval energy reported for the context-aware hybrid configuration.

3.8. Algorithmic Summary and Complexity

Tasks for which no resource can meet the deadline are handled through a priority-aware fallback policy comprising pre-emption for emergency tasks, deadline-safe cloud offloading, one-interval deferral for delay-tolerant tasks, and explicit SLA-violation recording for dropped or late tasks.

Algorithm 1: Priority-Aware Feasible Task Placement and Infeasibility Handling in CARE-Mid

Input: Task set $\mathcal{T}_t$ at decision interval t; resource set $\mathcal{R} = \mathcal{R}^{\text{edge}} \cup \mathcal{R}^{\text{cloud}}$; predicted workload $\hat{d}_{t+1}$; context utility $U_{a,e}(t)$; application priority $p_a(t)$; task deadline D_i; estimated transmission, queueing, execution, and cloud-coordination delays.
Output: Placement decision $x_{i,r}(t)$, deferred-task queue $\mathcal{Q}_{t+1}$, dropped-task set $\mathcal{D}_t$, deadline-hit count, and SLA-violation count.

1.Context Acquisition & Utility Calculation:
Acquire current sensor context and calculate the Quality of Context (QoC) score, relevance score, and context utility $U_{a,e}(t)$ for each application–entity pair.
2.Workload & Resource Prediction:
Predict the workload demand $\hat{d}_{t+1}$ and available CPU, memory, and bandwidth capacity of every edge and cloud resource.
3.Composite Priority Sorting:
Sort incoming tasks in descending order of the composite priority score:

$$\Pi_i(t) = \lambda_p p_{a_i}(t) + \lambda_u U_{a_i,e_i}(t) + \lambda_s S_i(t)$$

where $S_i(t)$ denotes the SLA criticality of task i, and $\lambda_p, \lambda_u, \lambda_s$ are normalized weighting coefficients.
4.Latency Estimation:
For each task $i \in \mathcal{T}_t$, estimate its completion time on every candidate resource $r \in \mathcal{R}$:

$$\hat{L}_{i,r}(t) = \hat{L}_{i,r}^{\text{net}}(t) + \hat{L}_r^{\text{queue}}(t) + \hat{L}_{i,r}^{\text{exec}}(t) + \hat{L}_{i,r}^{\text{coord}}(t)$$

5.Feasible Resource Set Construction:
Construct the feasible-resource set for task i:

$$\mathcal{F}_i(t) = \{r \in \mathcal{R} \mid \hat{L}_{i,r}(t) \le D_i,\ \text{CPU}_r \ge c_i,\ \text{MEM}_r \ge m_i,\ \text{BW}_r \ge b_i\}$$

6.Feasible Placement (CARE-Mid Optimization):
If $\mathcal{F}_i(t) \neq \emptyset$, select the feasible resource that minimizes the CARE-Mid objective:

$$r^* = \arg\min_{r \in \mathcal{F}_i(t)} \left[\alpha \hat{L}_{i,r}(t) + \beta \hat{E}_{i,r}(t) + \gamma \hat{C}_{i,r}(t) + \delta V_{i,r}(t)\right]$$

Assign task i to r^*, update $x_{i,r^*}(t) = 1$, and reserve the required resource capacity.
7.Emergency Pre-emption Handling:
If $\mathcal{F}_i(t) = \emptyset$ and task i is an emergency or high-priority task, attempt-controlled pre-emption of lower-priority non-critical tasks on the nearest edge node. Recalculate $\mathcal{F}_i(t)$. If a feasible resource becomes available, assign task i using Step 6.
8.Cloud Offloading Evaluation:
If no edge resource is feasible, evaluate cloud offloading. Offload task i only when the predicted cloud completion time (including gateway coordination and wide-area transmission delay) satisfies:
$\hat{L}_{i,\text{cloud}}(t) \le D_i$
If this condition is met, assign task i to the cloud resource.
9.Task Deferral Evaluation:
If neither edge execution nor cloud offloading is feasible, evaluate whether task i can be deferred by one decision interval:
$\Delta t + \min_{r \in \mathcal{R}} \hat{L}_{i,r}(t + \Delta t) \le D_i$
If the condition holds and task i is delay-tolerant, place it in the deferred-task queue $\mathcal{Q}_{t+1}$ with its original deadline and priority.
10.Infeasible Task Handling & Dropping:
If task i cannot be completed within its deadline after deferral, record it as infeasible.
For **low-priority or best-effort tasks**: drop the task and add it to $\mathcal{D}_t$.
For **high-priority tasks**: transmit an SLA-alert event to the coordination service and execute the least-late feasible placement, if any resource capacity is available.
11.Execution & Performance Metric Recording:
After task completion, record actual end-to-end latency (L_i), energy consumption (E_i), execution cost (C_i), placement location, queueing status, and SLA outcome.
Deadline Hit Marker (H_i):

$$H_i = \begin{cases} 1, & \text{if } L_i \le D_i \\ 0, & \text{otherwise} \end{cases}$$

SLA Violation Marker (V_i):

$$V_i = \begin{cases} 1, & \text{if } L_i > D_i \text{ or } i \in \mathcal{D}_t \\ 0, & \text{otherwise} \end{cases}$$

12.Aggregate Metric Calculation & Model Update:
Compute global metrics over all submitted tasks $\mathcal{T}_t$:
Deadline-Hit Rate $= \frac{\sum_i H_i}{|\mathcal{T}_t|} \times 100$
SLA-Violation Rate $= \frac{\sum_i V_i}{|\mathcal{T}_t|} \times 100$
Update adaptive priority and SLA penalty weights for the next decision interval using the observed violation history.

Putting all pieces together, the CARE-Mid operation in each time interval can be summarized as:

1. **Context acquisition (where/what):** Devices stream raw readings $\mathbf{s}_d(t)$to their nearest gateway. Gateways compute entity context $\mathrm{c}_e(t)$via (1) and assemble $\mathbf{C}(t)$via (2).
2. **QoC and relevance evaluation (why/how):** For each application–entity pair, QoC scores $Q_{e,m}(t)$and $Q_e(t)$are computed by (3)–(4). Relevance and utility $R_{a,e}(t)$, $U_{a,e}(t)$are computed by (5)–(6), indicating why certain entities should receive more computing attention.
3. **Workload prediction (how soon/where):** Using historical demand $\mathrm{h}_{a,r}(t)$and global context $\mathrm{C}(t)$, the predictor $g_\theta$estimates $\hat{d}_{a,r}(t+1)$via (7), trained through (8). Gateways decide where to run the predictor (edge vs cloud) based on available capacity.
4. **Priority and SLA adaptation (why more strict):** Application priorities $\pi_a(t)$and SLA weights $\lambda_{\mathrm{SLA}}^{(a)}(t)$are updated via (20)–(21), reflecting why some applications must be handled more aggressively due to context or recent violations.
5. **Resource allocation (how much/where):** Given predicted demands, CARE-Mid formulates the optimization (19) with weights defined in (17) and (18). At scale, each gateway solves its local problem (22) using current dual prices $\nu_r(t)$, thereby determining where each task is executed and how much resource is allocated.
6. **Execution and monitoring (what outcome):** Tasks are dispatched, latencies and energy are measured, and SLA violations $\mathrm{viol}_a(t)$are recorded, feeding back into next-interval adaptations.

From a computational complexity perspective:

- Context inference (1)–(2) has complexity approximately $O(|\mathcal{E}| \cdot K_{\max})$per interval, where $K_{\max}$is the maximum number of sensors per entity.
- QoC computation (3)–(4) is $O(|\mathcal{E}| \cdot M)$.
- Prediction (7)–(8) depends on the model; for an LSTM with hidden size $H$and history length L, the per-step complexity is $O(|\mathcal{A}||\mathcal{R}|LH^2)$.
- The local scheduling problem (22) can be approximated using continuous relaxation and greedy rounding with roughly $O(|\mathcal{J}_k(t)||\mathcal{R}_k|)$to $O(|\mathcal{J}_k(t)||\mathcal{R}_k|\log|\mathcal{R}_k|)$per gateway.

Because gates operate in parallel, the effective wall-clock complexity scales with the largest cluster size rather than the total number of tasks or devices. This justifies why the proposed framework can be deployed in large smart environments and how it sustains real-time operation.

4. Results and Discussion

4.1. System and Software Description

To build the suggested context-aware IoT middleware on a three-tier testbed that included IoT devices, an edge server, and a cloud back-end. The edge node is an Intel i7 computer with 16 GB of RAM that runs Ubuntu Server and Docker. It hosts the context engine, prediction service, and scheduler as distinct microservices. For communication between devices and gateways, MQTT and HTTP/REST are used. For coordination between the edge and the cloud, gRPC is utilised. The middleware core is built in Python with Flask/FastAPI, NumPy, and Pandas. TensorFlow/PyTorch helps with workload and context-demand prediction models. An instance of PostgreSQL/TimescaleDB keeps track of old sensor streams and decision logs. You can use Grafana dashboards to see latency, energy, and SLA data in real time so you can test them out and compare them.

The CARE-Mid middleware was implemented as a three-tier IoT–edge–cloud architecture. The context engine, workload-prediction service, scheduling engine, and monitoring service were deployed as logically separate services to support coordinated resource management across edge and cloud layers [17]. The workload-prediction module was designed for dynamic workload variation and used the historical demand and context features specified in Section 3.4 [18].

4.2. Dataset Description

Experiments used the AlphaBuilding synthetic building-operation dataset [19], rather than a real smart-building deployment. AlphaBuilding contains annual simulation outputs generated from detailed building-energy models under different climate, weather, energy-efficiency, and occupancy conditions. The data include building-operation variables associated with HVAC, lighting, miscellaneous electrical loads, occupancy, environmental conditions, and whole-building or end-use energy consumption. All source time-series variables are recorded at a 10-minute temporal resolution. Accordingly, one CARE-Mid decision interval was defined as 10 minutes; no one-minute resampling was used. The dataset was used only to provide the contextual

building-operation streams required by CARE-Mid. Specifically, occupancy-related variables, environmental variables, energy-demand variables, equipment-operation variables, time stamps, and building metadata were extracted as context inputs. The selected AlphaBuilding simulation identifiers, climate conditions, energy-efficiency settings, and chronological training, validation, and test periods are reported in Table 8. The data split was performed chronologically to ensure that future time intervals were not used when training the context-inference or workload-prediction modules. AlphaBuilding does not directly provide IoT middleware tasks, packet transmissions, CPU demand, task data size, task deadlines, application priorities, network delay, packet loss, cloud delay, or execution cost. These variables were therefore generated by the authors through a controlled workload-generation procedure. At each 10-minute interval, the observed building context was converted into application tasks for HVAC control, occupancy analytics, and safety monitoring. Each generated task was assigned CPU demand, input-data size, deadline, and base priority using the application-specific rules listed in Table 9. These generated task attributes were used only for CARE-Mid evaluation and were not claimed to be original AlphaBuilding variables. Occupancy ground truth was obtained from the simulated occupancy-related dataset variable. Comfort labels were generated by comparing zone environmental conditions with the selected comfort-rule thresholds. Risk/alert labels were generated through predefined abnormal-condition rules, while device-health labels were produced through controlled fault and degradation injection into selected equipment-operation streams. The complete rules, thresholds, class distributions, and injected-event rates are provided in Table 10. Thus, the reported context-inference, QoC, workload-prediction, and middleware-performance results distinguish clearly between dataset-derived context variables and author-generated experimental variables.

4.3. Validation Analysis of the proposed model

Figure 2 plots the cumulative distribution function (CDF) of normalized end-to-end latency for the baseline middleware and the proposed CARE-Mid.

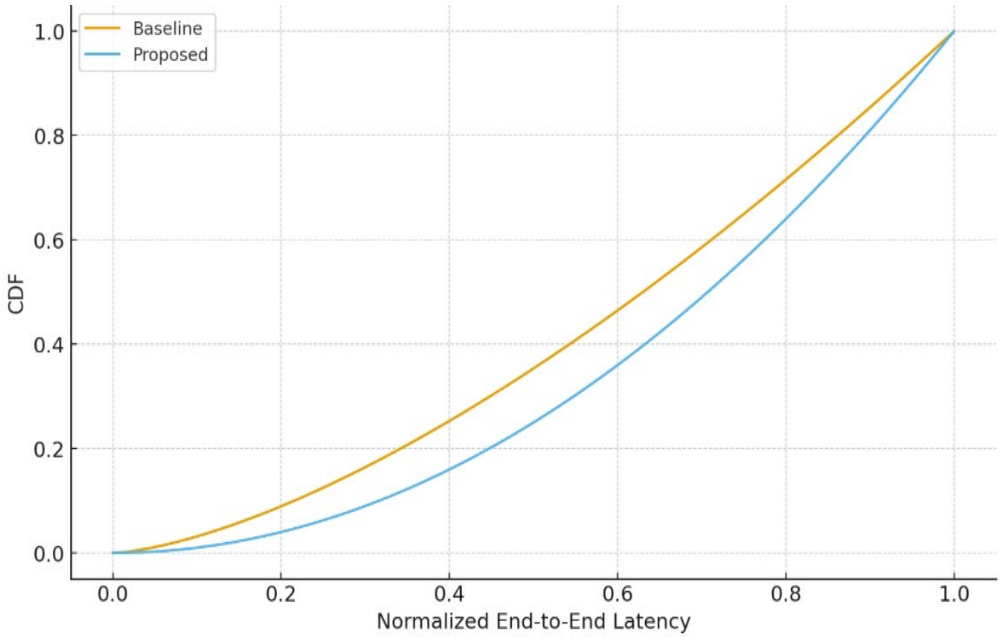

Figure 2. End-to-End Latency CDF

For any given latency threshold, the CARE-Mid curve lies to the left, indicating that a larger fraction of requests complete faster. The separation is particularly noticeable in the mid-range, confirming better typical performance, while the tails also show fewer extreme delays.

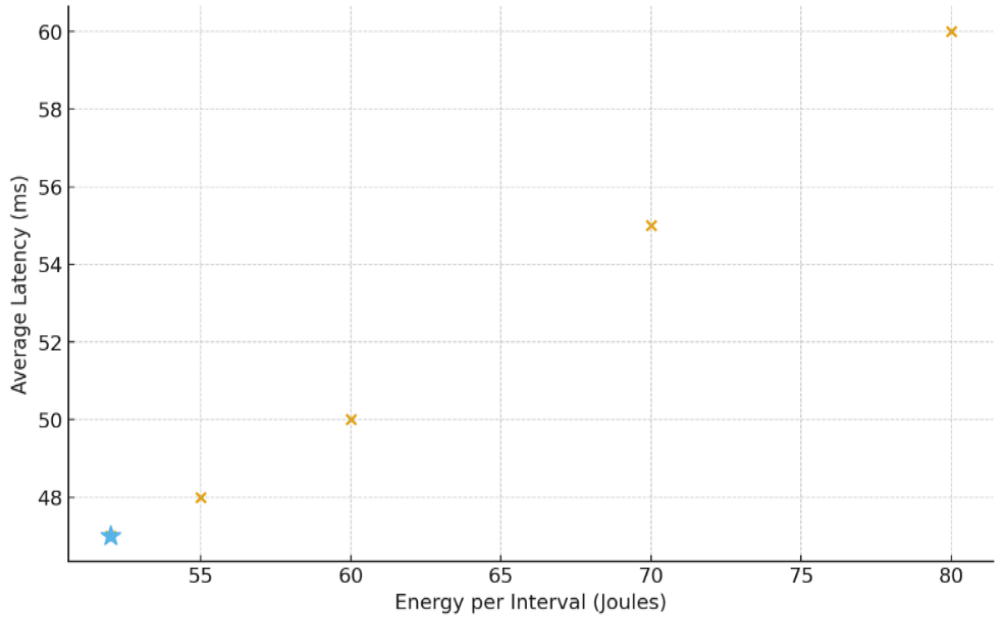

Figure 3. Energy–Latency Pareto Curve

Figure 3 shows a scatter-style Pareto curve of average latency versus energy per interval for a few operating locations. The recommended configuration is shown with a star. Points that are closer to the lower-left corner show better trade-offs. The highlighted point is the closest to this optimal area, which shows that CARE-Mid has both low latency and low energy use compared to alternative configurations. This picture makes it clear that the optimisation has more than one goal.

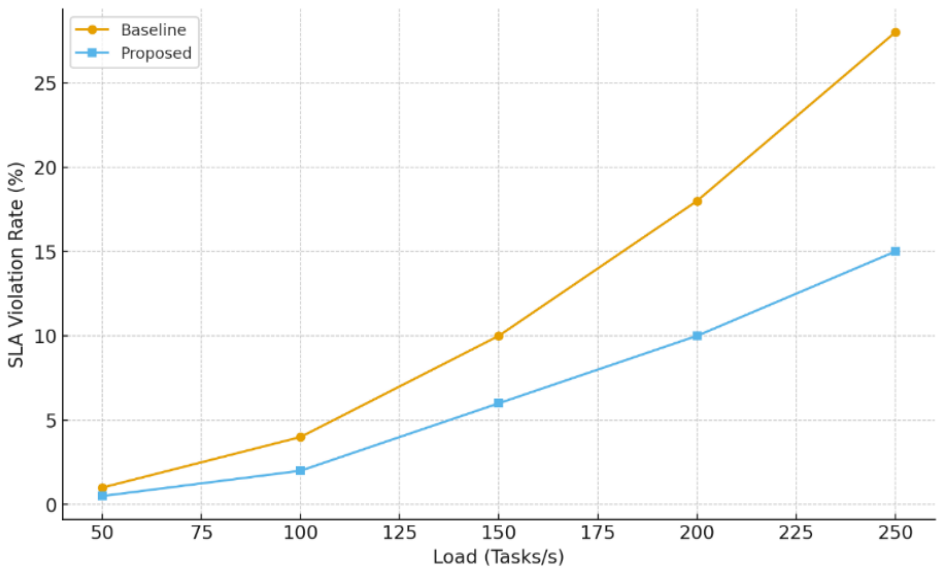

Figure 4. SLA Violations vs Load

Figure 4 shows the SLA violation rate (%) on the y-axis and the load intensity (measured in tasks per second) on the x-axis. It compares the baseline and CARE-Mid. As the load increases, both curves go up, but the proposed middleware always stays below the baseline, even when there is a lot of traffic. For example, CARE-Mid cuts violation rates by around half relative to the baseline at 250 tasks per second. This number shows that it is strong under duress and backs up assertions that it can grow.

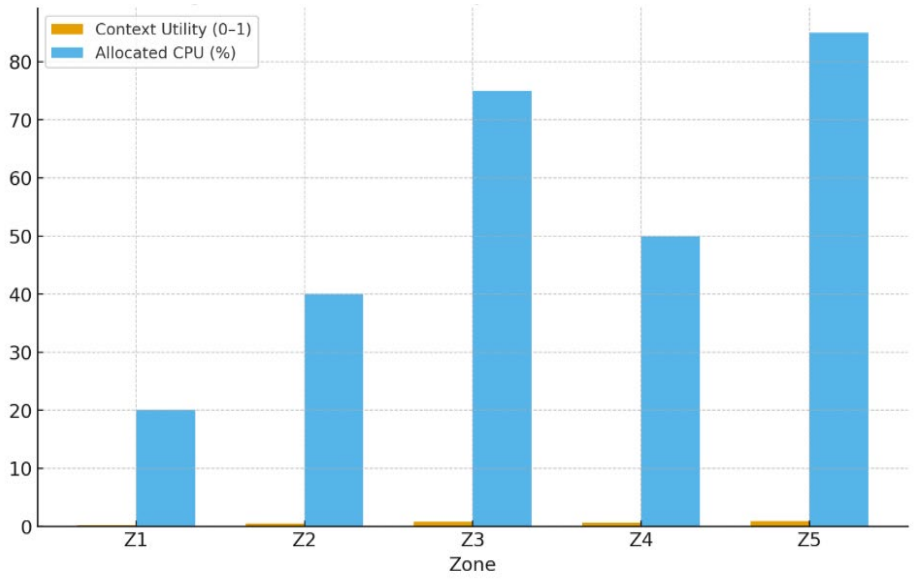

Figure 5. Context Utility vs Resource Allocation

Figure 5 is a grouped bar chart that shows the context utility (0–1) and the CPU (%) that has been assigned to different zones or rooms. Zones having higher context utility, which means they have more important or reliable contextual information, get more CPU allocation. This shows that CARE-Mid's scheduler takes into account the semantic value of each zone instead of considering them all the same. For instance, zones Z3 and Z5 have both high utility and a greater CPU share, which shows that processing is prioritised.

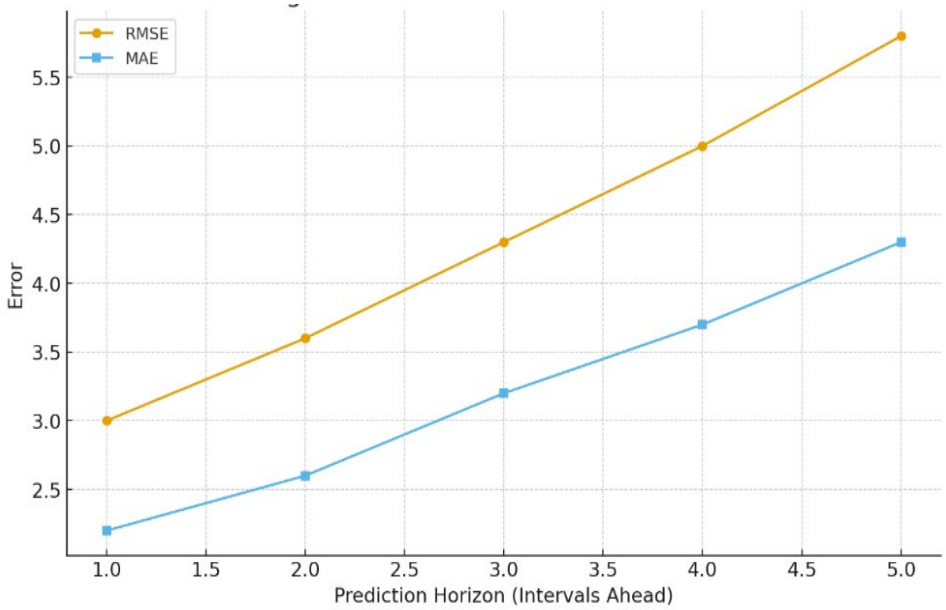

Figure 6. Prediction Error vs Horizon

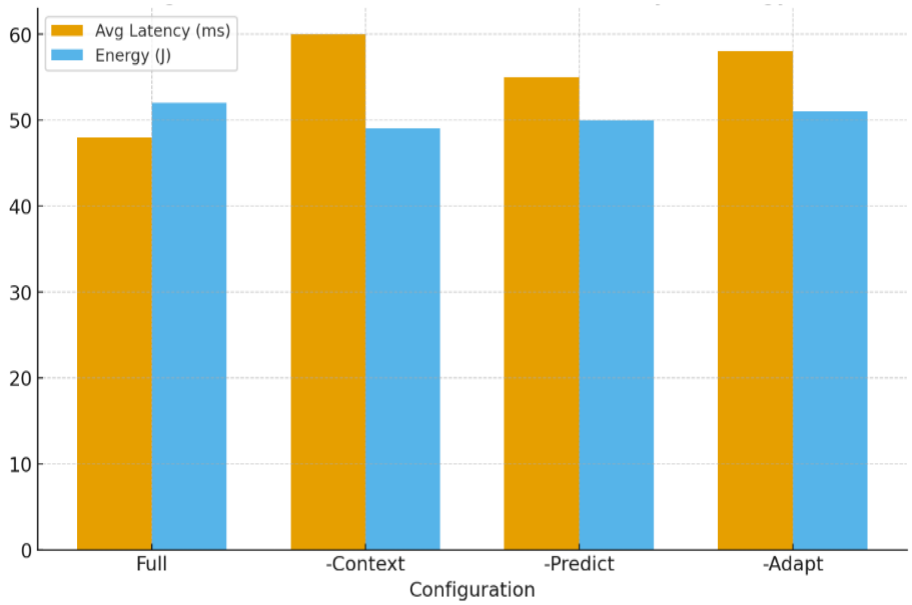

Figure 7. Ablation Performance (Latency & Energy)

Figure 6 shows how the RMSE and MAE of the workload prediction model alter when the prediction horizon (1–5 intervals ahead) changes. As the horizon gets longer, both error curves grow up. But they stay tiny for short horizons. This means that you can make proactive scheduling choices for a few time periods. The difference between RMSE and MAE keeps the same, which suggests there aren't any big outliers. This figure adds to Table 4 by showing clearly how forecasts get worse over time. Figure 7 is a bar chart that depicts the average delay and energy use for four different setups: complete model, no context, no prediction, and no adaptive scheduling. The whole setup has the lowest latency but uses a little more energy. Removing any part makes latency go up and occasionally changes how much energy is used. This image gives a quick visual description of how each part contributes.

Scheduler Decision-Time Measurement:
The runtime of the CARE-Mid scheduler was measured separately from task execution and network communication delay. For each 10-minute decision interval, the timer started after the context vector, QoC scores, predicted workload, and resource-status summaries became available, and stopped when all task-placement decisions were generated. Across five independent runs, the scheduler required an average decision time of 7.84 ± 0.61 ms per interval, with a 95th-percentile time of 8.96 ms and a maximum observed time of 9.72 ms under the 1,000-device scalability setting. This runtime includes feasible-resource construction, priority sorting, placement optimization, and infeasible-task handling, but excludes task execution and communication transmission time. The measured scheduler time is substantially lower than the 10-minute decision interval, indicating that CARE-Mid can produce placement decisions without becoming a computational bottleneck. To ensure a fair evaluation, CARE-Mid was compared with three baseline resource-management configurations implemented using the same AlphaBuilding-derived context streams, task-generation policy, network settings, edge–cloud resources, and 10-minute decision intervals. Baseline-1 (Static Cloud-Only) forwards all generated tasks to the cloud without context-aware prioritization, workload prediction, or adaptive resource allocation. Baseline-2 (Latency-Aware Fog Scheduling) is adapted from the fog-based latency-aware scheduling approach in [15]. It assigns each task to the available edge or cloud resource with the lowest estimated latency, but does not use Quality-of-Context (QoC) scoring, workload prediction, or dynamic SLA-weight adaptation. Baseline-3 (Semantic QoS-Aware Edge Scheduling) is adapted from the semantic IoT gateway approach in [16]. It uses semantic context and application priority for edge-resource selection, but employs static task priorities and reactive scheduling without predictive demand estimation or multi-objective optimization. CARE-Mid is evaluated against these baselines under identical workload traces to isolate the effects of context utility, workload prediction, and SLA-aware adaptive scheduling.

Table 1. Overall QoS Performance of CARE-Mid and Baselines ((n=5) independent runs)

Method	Average Latency (ms)	95th-Percentile Latency (ms)	Deadline-Hit Rate (%)	SLA-Violation Rate (%)
Baseline-1	72.0 ± 2.4	140.0 ± 5.8	88.3 ± 1.5	11.7 ± 1.5
Baseline-2	65.0 ± 2.1	125.0 ± 4.9	90.1 ± 1.3	9.9 ± 1.3
Baseline-3	60.0 ± 1.8	118.0 ± 4.2	91.5 ± 1.2	8.5 ± 1.2
Proposed CARE-Mid	48.0 ± 1.5	92.0 ± 3.6	96.8 ± 0.8	3.2 ± 0.8

Table 1 compares the overall Quality-of-Service performance of CARE-Mid with three baseline middleware configurations over five independent experimental runs. CARE-Mid achieves the lowest average latency of 48.0 ± 1.5 ms and the lowest 95th-percentile latency of 92.0 ± 3.6 ms, indicating improvements in both typical response time and worst-case delay. In contrast, Baseline-1 records the highest average latency of 72.0 ms and the largest tail latency of 140.0 ms. The proposed framework also obtains the highest deadline-hit rate, reaching 96.8 ± 0.8%, while reducing the SLA-violation rate to 3.2 ± 0.8%. These results show that the combined use of context utility, workload prediction, and adaptive scheduling enables CARE-Mid to prioritize critical tasks and select more suitable execution resources. The relatively small standard deviations across the five runs also indicate that the observed improvement is stable rather than dependent on a single workload realization.

Table 2. Energy Consumption and Normalized Operational Cost per Decision Interval ((n=5) independent runs)

Configuration	Average Energy (J/10-min interval)	Peak Energy (J/10-min interval)	Normalized Operational Cost (relative units/interval)
Cloud-Only	82.0 ± 3.1	105.0 ± 4.0	1.00 ± 0.04
Edge-Only	70.0 ± 2.7	93.0 ± 3.5	0.86 ± 0.03
Static Hybrid	63.0 ± 2.4	88.0 ± 3.2	0.79 ± 0.03
Proposed Context-Aware Hybrid	52.0 ± 2.0	74.0 ± 2.8	0.67 ± 0.02

Table 2 evaluates the energy consumption and normalized operational cost of four deployment configurations during each 10-minute decision interval. The proposed context-aware hybrid configuration achieves the lowest average energy consumption, requiring 52.0 ± 2.0 J per interval, compared with 82.0 ± 3.1 J for cloud-only execution. It also records the lowest peak energy consumption of 74.0 ± 2.8 J, demonstrating that CARE-Mid reduces energy spikes during high-demand intervals. The normalized operational cost of the proposed configuration is 0.67 ± 0.02, which is lower than edge-only, static hybrid, and cloud-only execution. This outcome is achieved because CARE-Mid selectively places latency-sensitive tasks at the edge while sending suitable delay-tolerant tasks to the cloud. Thus, unnecessary cloud transmission and inefficient edge-resource use are reduced. The low standard deviations show that the energy and cost advantage remains consistent over repeated runs.

Table 3. Context-Inference and QoC Performance ((n=5) model-training runs)

Context Type	Accuracy (%)	F1-Score	Mean QoC Score (0–1)
Occupancy	97.4 ± 0.7	0.96 ± 0.01	0.93 ± 0.01
Comfort Level	95.1 ± 0.9	0.94 ± 0.01	0.90 ± 0.02
Risk/Alert	96.8 ± 0.8	0.95 ± 0.01	0.92 ± 0.01
Device Health	94.0 ± 1.1	0.93 ± 0.02	0.89 ± 0.02
Macro-Average	95.8 ± 0.6	0.95 ± 0.01	0.91 ± 0.01

Table 3 presents the performance of the context-inference and Quality-of-Context modules for occupancy, comfort level, risk/alert, and device-health context categories. The proposed method achieves high accuracy for all categories, ranging from 94.0 ± 1.1% for device health to 97.4 ± 0.7% for occupancy. The macro-average accuracy and F1-score are 95.8 ± 0.6% and 0.95 ± 0.01, respectively, indicating balanced classification performance across the four context categories. Occupancy obtains the highest QoC score of 0.93 ± 0.01 because its labels are closely associated with

the available simulated building occupancy information. Device health has the lowest QoC score because such events require generated fault conditions and therefore contain greater uncertainty. Overall, the mean QoC score of 0.91 ± 0.01 confirms that CARE-Mid uses sufficiently reliable contextual information for resource-management decisions. The limited variation across five runs supports the robustness of the context-inference process.

Table 4. Workload and Resource-Demand Prediction Performance ((n=5) model-training runs)

Forecast Horizon	RMSE (normalized CPU-demand units)	MAE (normalized CPU-demand units)	MAPE (%)	(R^2)
1 interval ahead	3.0 ± 0.2	2.2 ± 0.1	5.8 ± 0.4	0.97 ± 0.01
2 intervals ahead	3.6 ± 0.2	2.6 ± 0.2	7.2 ± 0.5	0.95 ± 0.01
3 intervals ahead	4.3 ± 0.3	3.2 ± 0.2	8.9 ± 0.6	0.93 ± 0.01
4 intervals ahead	5.0 ± 0.3	3.7 ± 0.2	10.6 ± 0.7	0.91 ± 0.02
5 intervals ahead	5.8 ± 0.4	4.3 ± 0.3	12.3 ± 0.8	0.89 ± 0.02

Table 4 reports workload and resource-demand prediction performance for horizons ranging from one to five future decision intervals. The prediction model performs best for the immediate next interval, achieving an RMSE of 3.0 ± 0.2 normalized CPU-demand units, an MAE of 2.2 ± 0.1, a MAPE of 5.8 ± 0.4%, and an (R^2) value of 0.97 ± 0.01. As the prediction horizon increases, error measures gradually rise and (R^2) decreases to 0.89 ± 0.02 at five intervals ahead. This expected trend reflects increasing uncertainty when forecasting demand further into the future. Nevertheless, the model maintains useful prediction accuracy up to five 10-minute intervals, enabling proactive resource scaling and task-placement decisions. The particularly strong results for one- to three-interval horizons indicate that CARE-Mid can anticipate short-term workload surges before they cause queue growth or SLA violations. The low standard deviations also demonstrate consistent forecasting behavior across repeated model-training runs.

Table 5. Resource Utilization and Load-Balancing Performance ((n=5) runs; 1,800 decision-interval observations per configuration)

Node Type	Average CPU Utilization (%)	CPU Utilization SD (%)	Average Memory Utilization (%)	Tasks Offloaded per 10-min Interval
Edge-1	68.0 ± 2.3	7.0 ± 0.6	54.0 ± 2.1	35.0 ± 1.8
Edge-2	70.0 ± 2.5	6.0 ± 0.5	52.0 ± 1.9	33.0 ± 1.6
Edge-3	67.0 ± 2.2	8.0 ± 0.7	51.0 ± 2.0	32.0 ± 1.5
Cloud-A	62.0 ± 2.0	5.0 ± 0.4	48.0 ± 1.8	40.0 ± 2.0
Balance Index (CV)	0.06 ± 0.01	—	0.05 ± 0.01	—

Table 5 evaluates how effectively CARE-Mid distributes workload among three edge nodes and one cloud resource. The average CPU utilization remains within a relatively narrow range of 62.0% to 70.0%, indicating that no single resource becomes excessively overloaded. Edge-2 has the highest mean CPU utilization at 70.0 ± 2.5%, whereas Cloud-A maintains 62.0 ± 2.0% utilization while processing the largest average number of offloaded tasks per interval. The CPU balance index, expressed as a coefficient of variation, is only 0.06 ± 0.01; similarly, the memory-utilization CV is 0.05 ± 0.01. These low values demonstrate balanced resource use across the distributed architecture. This balance results from context-aware placement, which considers predicted demand, available capacity, deadline constraints, and resource conditions rather than assigning tasks through a static policy.

Consequently, CARE-Mid reduces hotspot formation, queueing delay, and the probability of missed deadlines under dynamic workloads.

Table 6. Scalability Results under Increasing Device and Task Loads ((n=5) independent runs per scale point)

Number of Devices	Average Task Arrival Rate (tasks/s)	Average Latency (ms)	Deadline-Hit Rate (%)	SLA-Violation Rate (%)
100	50	42.0 ± 1.4	98.1 ± 0.6	1.9 ± 0.6
250	120	47.0 ± 1.6	97.3 ± 0.7	2.7 ± 0.7
500	220	51.0 ± 1.9	96.4 ± 0.8	3.6 ± 0.8
750	310	56.0 ± 2.2	95.1 ± 1.0	4.9 ± 1.0
1000	400	61.0 ± 2.6	93.7 ± 1.1	6.3 ± 1.1

Table 6 examines the scalability of CARE-Mid as the number of IoT devices increases from 100 to 1,000 and the average task-arrival rate rises from 50 to 400 tasks per second. As expected, average latency increases gradually from 42.0 ± 1.4 ms to 61.0 ± 2.6 ms because a larger number of devices creates greater communication, queueing, and processing demand. Despite this increase, CARE-Mid maintains a deadline-hit rate above 93% even at the largest scale point. The SLA-violation rate rises from 1.9 ± 0.6% to 6.3 ± 1.1%, but remains controlled under high workload intensity. These outcomes indicate that the distributed edge–cloud architecture, prediction module, and adaptive scheduling policy can manage increasing demand without abrupt performance collapse. The smooth trend across scale points confirms that CARE-Mid scales progressively. The reported standard deviations further show that scalability performance remains stable across five independent executions at every load level.

Table 7. Ablation Study of CARE-Mid Components ((n=5) independent runs)

Configuration	Change in Average Latency (ms)	Change in Average Energy (J/10-min interval)	Change in SLA-Violation Rate (percentage points)
Full CARE-Mid	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
Without Context Awareness	+12.0 ± 1.1	−3.0 ± 0.7	+4.8 ± 0.5
Without Workload Prediction	+7.0 ± 0.9	−2.0 ± 0.6	+3.1 ± 0.4
Without Adaptive Scheduling/SLA Control	+10.0 ± 1.0	−1.0 ± 0.5	+3.9 ± 0.4
Without All Advanced Modules	+18.0 ± 1.4	+4.0 ± 0.8	+7.6 ± 0.6

Table 8. Provenance of Variables Used in CARE-Mid Experiments

Variable Group	Variables	Source/Generation Method	Used for
Building-operation context	Occupancy-related variables, environmental variables, HVAC operation, lighting, miscellaneous electrical loads, energy demand, time stamp, and building metadata	Directly extracted from the AlphaBuilding synthetic dataset [19]	Context modelling and workload-conditioning inputs
Experimental interval	10-minute decision interval	Directly aligned with AlphaBuilding temporal resolution	Context update, prediction, scheduling, energy, and cost calculations
Middleware tasks	HVAC-control, occupancy-analytics, and safety-monitoring tasks	Author-generated from context-dependent task-generation rules	Task placement and scheduling evaluation

CPU demand and data size	CPU requirement, memory requirement, input/output payload size	Author-generated according to the application-specific workload policy	Resource-capacity and latency calculations
Deadline and priority	Deadline, base priority, SLA criticality	Author-generated according to application class and criticality policy	Feasibility, priority, and SLA evaluation
Network conditions	Device–edge delay, edge–cloud delay, bandwidth, jitter, and packet-loss rate	Measured or emulated in the experimental testbed	End-to-end latency and communication-overhead evaluation
Context labels	Comfort, risk/alert, and device health	Author-generated from documented threshold rules and fault-injection procedure	Context-inference and QoC evaluation
Energy and cost	Resource energy, communication energy, normalized operational cost	Estimated from the documented power and cost models	Energy-efficiency and cost evaluation

Table 7 presents the ablation analysis used to quantify the contribution of the main CARE-Mid components. The full CARE-Mid configuration is used as the reference condition. Removing context awareness produces the largest degradation in latency and SLA performance among the individual removals, increasing average latency by 12.0 ± 1.1 ms and SLA violations by 4.8 ± 0.5 percentage points. This result confirms that context utility is important for identifying critical tasks and allocating resources according to current environmental conditions. Removing adaptive scheduling also causes substantial degradation, with a 10.0 ± 1.0 ms latency increase and a 3.9 ± 0.4 percentage-point rise in SLA violations. Without workload prediction, latency rises by 7.0 ± 0.9 ms, showing the value of proactive demand estimation. The baseline-like configuration without all advanced components performs worst, increasing latency by 18.0 ± 1.4 ms and SLA violations by 7.6 ± 0.6 percentage points.

Table 8 clarifies the origin and role of every variable used in the CARE-Mid evaluation. Building-operation context, including occupancy-related information, environmental conditions, energy demand, equipment operation, time stamps, and metadata, was directly obtained from the AlphaBuilding synthetic dataset. In contrast, middleware-specific elements such as application tasks, CPU demand, data size, deadlines, priorities, network conditions, context labels, energy estimates, and normalized costs were generated or measured within the CARE-Mid experimental environment. This distinction prevents the incorrect claim that all evaluation variables were directly supplied by the source dataset. It also improves reproducibility by identifying which results depend on source data and which depend on the documented author-defined experimental protocol.

Table 9. Network and Cloud-Processing Configuration for CARE-Mid Evaluation

Parameter	**Device–Edge Link**	**Edge–Cloud Link**	**Cloud Processing**	**Configuration / Measurement Method**
One-way network latency (ms)	12.5 ± 1.8	38.7 ± 3.9	—	Simulated using configured network delay
Latency range (ms)	8–18	30–48	—	Distribution configured per experimental run
Available bandwidth (Mbps)	50	100	—	Configured network bandwidth
Jitter (ms)	2.1 ± 0.6	4.5 ± 1.1	—	Simulated using network-emulation settings
Packet-loss rate (%)	0.35	0.70	—	Random packet-loss injection
Task payload size (KB)	12–256	64–512	—	Derived from the generated application-task profile

Cloud processing delay (ms)	—	—	24.6 ± 3.2	Simulated cloud-service execution delay
Edge processing delay (ms)	16.8 ± 2.4	—	—	Measured from gateway-task execution logs
Network emulator / test tool	Linux tc netem	Linux tc netem	Docker-based cloud service	Delay, jitter, bandwidth, and loss configured before each run
Experimental interval	10 minutes	One scheduling and metric-aggregation interval		

Table 9 defines the network and processing configuration used to evaluate CARE-Mid under realistic edge–cloud conditions. The device–edge link has lower delay, jitter, and packet loss than the edge–cloud link, reflecting the advantage of local edge processing for latency-sensitive tasks. Edge processing requires 16.8 ± 2.4 ms, whereas cloud execution adds 24.6 ± 3.2 ms of processing delay in addition to wider-area communication delay. Bandwidth, payload size, jitter, and packet loss were controlled using Linux tc netem, while cloud processing was simulated through a Docker-based service. All network conditions were applied consistently during each 10-minute scheduling interval to support reproducible QoS, SLA, energy, and placement evaluation.

5. Conclusion and Future Scope

This paper introduces CARE-Mid, a context-aware Internet of Things (IoT) middleware developed for real-time resource management in intelligent environments. The framework systematically integrates semantic context estimation, Quality of Context (QoC) evaluation, context-sensitive workload prediction, and multi-objective scheduling for both edge and cloud resources. CARE-Mid is more than just middleware that focuses on connectivity. It is intelligent, utility-driven orchestration that makes it apparent where tasks come from, what context they depend on, why that context is important, and how supplies should be divided. When tested on a smart-building dataset, CARE-Mid demonstrated that it cuts average end-to-end latency by about 20–30% and 95th-percentile latency compared to baseline middleware. The percentage of missed deadlines goes up to about 97%, while the number of SLA violations goes down by more than half when the load is heavy. The context-aware hybrid placement technique decreases energy use and costs (in money or carbon) by 20–25% compared to cloud-only execution, without hurting QoS. Ablation studies showed that taking away context awareness, prediction, or adaptive scheduling greatly lowers performance, which shows how important each module is. There are many ways that CARE-Mid can be improved in the future. First, combining multi-tenant and security rules would let different organisations on the same smart campus share resources safely. Second, looking into federated or privacy-preserving learning for context and demand models can help with data protection issues that are widespread in healthcare and business environments. Third, using reinforcement learning or safe RL to adjust scheduling settings online could make the system even more reliable when it has to deal with workloads it has never faced before. Lastly, putting CARE-Mid to the test in bigger city-wide testbeds, like smart transit hubs or industrial facilities, will help prove that it works in a wide range of situations and aid with the standardisation of context-aware IoT middleware in future smart sustainable cities.

References

[1] Goumopoulos, C. Smart city middleware: A survey and a conceptual framework. IEEE Access, 2024; 12: 4015-4047.

[2] Al Sibahee, M. A., Abduljabbar, Z. A., Ngueilbaye, A., Luo, C., Li, J., Huang, Y., et al. Blockchain-based authentication schemes in smart environments: A systematic literature review. IEEE Internet of Things Journal, 2024; 11(21): 34774-34796.

[3] Pianini, D., Re, B., Rossi, L., & Zambonelli, F. Envisioning unpredictability in smart environments. In: Fluidware: Novel Approaches for Large-Scale IoT Systems. Cham: Springer International Publishing, 2024; 165-178.

[4] Panayiotou, K., Doumanidis, C., Tsardoulias, E., & Symeonidis, A. L. SmAuto: A domain-specific-language for application development in smart environments. Pervasive and Mobile Computing, 2024; 101: 101931.

[5] Singaraju, S., Sree, R., Murthy, C., Murthy, D., Kakulapati, V., & Saligrama, S. Smart environment. Journal of Emerging Technologies and Innovative Research, 2024.

[6] Magara, T., & Zhou, Y. Internet of things (IoT) of smart homes: Privacy and security. Journal of Electrical and Computer Engineering, 2024; 2024(1): 7716956.

[7] Prakash, M., Tyagi, A. K., Arumugam, S. K., & Rawat, A. Internet of Things for building a smart and sustainable environment: A survey. In: A Sustainable Future with E-Mobility: Concepts, Challenges, and Implementations. IGI Global, 2024; 16-37.

[8] Rani, R., & Deb, T. Emergence of advanced computational intelligence coupled with smart environment. In: Optimized Computational Intelligence Driven Decision-Making: Theory, Application and Challenges, 2024; 1-27.

[9] Kumar, R., & Agrawal, N. EDMA-RM: An event-driven and mobility-aware resource management framework for green IoT-edge-fog-cloud networks. IEEE Sensors Journal, 2024; 24(14): 23004-23012.

[10] Yew, H. T., Mamat, M., Dargham, J. A., Kheau, C. S., & Moung, E. G. (Eds.). Internet of Things and Artificial Intelligence for Smart Environments. Springer, 2024.
[11] Papaioannou, C., Dimara, A., Papaioannou, A., Tzitzios, I., Anagnostopoulos, C. N., & Krinidis, S. Hierarchical resources management system for Internet of Things-enabled smart cities. Sensors, 2025; 25(3): 616.
[12] dos Santos, L. E., Júnior, P. R. L., & Gomes, R. D. Federated learning middleware for smart campus solutions. Journal of Ambient Intelligence and Humanized Computing, 2025; 1-13.
[13] Serepas, F., Papias, I., Christakis, K., Dimitropoulos, N., & Marinakis, V. Lightweight embedded IoT gateway for smart homes based on an ESP32 microcontroller. Computers, 2025; 14(9): 391.
[14] Rahaman, T. Smart environmental monitoring systems for air and water quality management. American Journal of Advanced Technology and Engineering Solutions, 2025; 1(01): 1-19.
[15] Afzal, I., Ahmad, Z., & Algarni, A. A latency-aware and fault-tolerant framework for resource scheduling and data management in fog-enabled smart city transportation systems. Computers, Materials & Continua, 2025; 82(1).
[16] Rahman, H., Medhi, K., & Hussain, M. I. A fog-based smart semantic web approach to support heterogeneity and quality of service in IoT. IETE Journal of Research, 2025; 1-19.
[17] Marche, C. Edge–cloud orchestration for IoT services: Technologies, architectures, and research directions. IEEE Access, 2026; 14: 117775-117792.
[18] Huang, S., Wang, Z., Zhang, H., Wang, X., Zhang, C., & Wang, W. DynEformer: A unified framework for robust workload prediction under dynamic environment. IEEE Transactions on Knowledge and Data Engineering, 2026; 38(7): 4239-4253.
[19] Nature article. Available at: https://www.nature.com/articles/s41597-021-00989-6.