National College of Ireland

Strategic Task Placement: A Comparative Analysis of Lightweight Scheduling in Edge–Fog–Cloud Systems

MSc Research Project
Cloud Computing

Raj Dhamdhere
Student ID: x23122498

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Raj Dhamdhere
Student ID:	x23122498
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	11/08/2025
Project Title:	Strategic Task Placement: A Comparative Analysis of Lightweight Scheduling in Edge–Fog–Cloud Systems
Word Count:	7462
Page Count:	23

Signature:	Raj Dhamdhere
Date:	13th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Strategic Task Placement: A Comparative Analysis of Lightweight Scheduling in Edge–Fog–Cloud Systems

Raj Dhamdhere
x23122498

Abstract

The surge in real-time, latency-sensitive applications and the growing complexity of job distribution across edge, fog, and cloud layer demands efficient task scheduling strategies. Modern IoT systems require intelligent decision-making near the data source rather than centralized computation. The current scheduling methods incorporate metaheuristic or AI-based techniques for optimal layer selection, but are resource-intensive, reducing speed and efficiency in edge-based environments. This study presents lightweight heuristic and priority-based scheduling policies that minimize latency overhead and optimally determine the best layer (edge, fog, or cloud) to delegate tasks. The proposed hybrid experimental framework was implemented with the Raspberry Pi device as the edge server and AWS EC2 instances as the fog and cloud servers. Synthetic IoT workloads were used to test the proposed scheduling strategies and concentrate on execution time, transfer latency, energy efficiency, and deadline adherence. The findings indicate that the heuristic strategy achieved a deadline adherence rate of 33.91%, while the priority algorithm achieved 24.63%, with low overhead. These lightweight strategies maintained competitive performance, making them suitable for latency-sensitive and resource-constrained IoT applications in opposition to metaheuristic solutions.

1 Introduction

The surge in growth of Internet of Things (IoT) devices, projected to hit 40 billion connected devices by the year 2030 [1], has led to the increasing need for real-time, low-latency applications, resulting in greatly stretched demands on centralized cloud infrastructure Elgazzar et al. (2025). The hierarchical solution proposed by edge-fog-cloud computing is that edge performs quick responses, fog compiles medium-scale processing, and the cloud nodes perform large-scale analytics. Although this architecture enhances the scalability and minimizes latency, but an important question still arises: **how to decide where tasks should execute in this hierarchy of layers**. Execution-layer selection, especially under resource-constrained and changing environments, remains an optimal task in reducing delays, energy efficiency and quality of service.

1.1 Motivation

The current task scheduling systems in edge-fog-cloud systems are commonly based on Artificial Intelligence (AI) and metaheuristic algorithms, some of which include Rein-

[1] `https://iot-analytics.com/number-connected-iot-devices/`

forcement Learning (RL), Genetic Algorithms (GA), and Particle Swarm Optimization (PSO) Saeik et al. (2021); Rahbari (2022); Wang et al. (2024). These approaches perform well theoretically but demand a lot of computational resources, long convergence processing time and large training sets. This aspect renders them unfit in real-time IoT applications where latency and resource-efficiency is pivotal. Such constraints bring up the importance of lightweight scheduling strategies that can provide rapid and responsive scheduling solutions without substantial overhead. Heuristic and priority-based methods handle this requirement by using properties of the system including execution time, energy, and importance of tasks to make responsive and resource-aware decisions related to allocation of jobs.

1.2 Research Problem & Contribution

There has been recent development of lightweight scheduling techniques like Heuristic Resource Allocation and Optimization algorithm in a MultiFog-Cloud (HeRAFC) Dehury et al. (2024) that uses a normalized cost-based approach of selecting execution layers, and Multi-Queue Priority-based task scheduling (MQP) Fahad et al. (2022) that improves deadline adherence by using multi-queue scheduling, none of these techniques have been evaluated under systematic unified framework. Current researches are also limited to the simulation-only platform or grounded in theoretical analysis, which contributes little to the actual picture of their performance in heterogeneous, resource-constrained scenarios. Moreover, strict real-time requirements at various layers are not enforced and flexible decision policy is not considered in the present implementations. This study addressed the described gaps by (a) created a hybrid evaluation system where physical testbeds were built using (Raspberry Pi, and AWS Elastic Compute Cloud (EC2) instances) and (b) proposed set of changes, including a **bottom-up execution-layer selection strategy** to enhance responsiveness and be energy-efficient and (c) produced a balanced synthetic IoT workload replicating realistic task diversity with regards to its size, priorities, and deadlines to maintain fairness and repeatable benchmarking. The proposed framework carries a systematic comparative study in latency, energy efficiency and achievement of deadlines and the information have been used to design real-time IoT scheduling systems.

1.3 Research Question

How do heuristic-based and priority-based scheduling strategies compare in optimizing execution-layer selection (edge, fog, or cloud) with respect to response time, energy efficiency, deadline adherence, and resource utilization in dynamic IoT environments?

1.4 Research Objectives

1. Development of a Hybrid test evaluation framework to promote scalability by leveraging real hardware (Raspberry Pi on the edge, AWS EC2 on the fog and AWS EC2 cloud) and simulation (iFogSim).

2. Generated synthetic IoT workloads that replicates the properties of real jobs (varying with different priorities)

3. Implementation of HeRAFC-inspired and MQP scheduling algorithms with bottom-up selection strategy, adapting to the real-time constrained environments.

4. Comparative analysis of these lightweight scheduling strategies for latency, energy efficiency, deadline compliance and task distribution across the layers.

1.5 Report Structure

The rest of this report has the following structure: Section 2 provides related work and gaps; Section 3 provides the research methods followed in this study; Section 4 describes framework design and the integration of scheduling algorithms; Section 5 provides a description of implementation; Section 6 analyzes implementation results and evaluation methodology; and the last section, Section 7, concludes with the contributions, limitations, and future work.

2 Related Work

Selecting the most appropriate layer between edge, fog and cloud has been a challenge because of the resource limitations both in terms of latency and energy efficiency in distributed computing. This research considers numerous contributions in this field and puts those methods into the following groups. (a) Learning and optimization-based scheduling methods, (b) Lightweight and priority-oriented scheduling algorithms, and (c) Unified evaluation and cross-layer frameworks. Each category is discussed in terms of weaknesses and strengths, and research gaps that laid the foundation for this research.

2.1 Learning and Optimization-based Scheduling Methods

In the recent years, several scheduling approaches like AI-based, metaheuristic have been generating stimulations in the edge-fog-cloud computing to be utilized as they achieve optimum distribution of workloads. Deep policy-based reinforcement methods (DRL), evolutionary algorithms (GA) and collaborative multi-agent models are applied when associated with the high level of computation in dealing with complex task allocation.

Zhang et al. (2019) proposed a GA-based approach in response to increasing the usability of resources in a cloud environment based on location of containers in an energy-efficient manner. The design was less efficient in latency-sensitive cases and flexibility of deployment. Intelligent-based offloading solutions were potentially addressed by Saeik et al. (2021), which highlighted the strong potential, but also described the need for such offloading techniques requiring vast training data that trigger an immense computing cost. Reddy et al. (2020) focuses on a GA-based approach for energy-saving fog resource management. The proposed work illustrates an improved model by enhancing energy savings through dynamic allocation strategies. However, it also highlights the lack of latency and responsiveness evaluation, making it less suitable for time-critical sensor data processing in edge environments.

A DRL-based task scheduling model proposed by Swarup et al. (2021) is specifically designed for energy-saving execution in edge-fog architectures. The methods used in the proposal significantly improved task offloading and minimized power usage across fog environments. However, like other DRL-based approaches, it exhibits a few limitations when used in real-time edge deployments, such as the use of massive training data and lack of hardware-level validation. Rahbari (2022) in their proposal offer a detailed explanation of metaheuristic-based scheduling algorithms, emphasizing task scheduling efficiency across cloud and edge environments. While the proposal provides insights into characteristics such as cost, execution time, and resource utilization, it falls short in addressing limitations like real-time deadlines or limited computational resources, which are often critical in latency-sensitive IoT applications. Gad (2022) provides a comprehensive review of PSO and its applications. In addition to PSO's strengths in optimized accuracy and convergence, it also covers limitations, stating the lack of validation in real-time edge environments, as the paper is simulation-focused. Due to the absence of real-time applicability, it becomes less useful for event-driven edge computing systems that deal with priority-based sensor event handling.

In addition, Tahmasebi-Pouya et al. (2023) proposed a reinforcement learning-based scheduler (RLIS), which aims at maximizing the performance of the system using an optimal distribution of workloads and low latency in fog computing networks. However, time-consuming training times and the absence of real-life hardware validation were still the concern. More recently, ReinFog, a modular framework leveraging DRL, was developed by Wang et al. (2024) with the purpose of minimizing latency and energy consumption. Although it has performed well in theory, it is still very computationally expensive, and has never been executed on real IoT devices. Such algorithms can be efficient in high-performance settings, but cannot be used in embedded systems that are resource-constrained in real-time applications. They are not desirable for dynamic edge-fog-cloud models, as the key problems are computational overhead, complexity, and the absence of deployment flexibility. Such drawbacks necessitate the issue of lightweight alternatives like heuristic and priority scheduling.

2.2 Lightweight and priority-oriented scheduling algorithms

The issues, such as high complexity and extensive computational overhead of AI-based and metaheuristic models in real-time application scenarios, have given rise to heuristic and priority-aware scheduling techniques as rule-based lightweight counterparts. Such methods are applicable in real-time, low-resource situations since they take advantage of the parameters available at a system level, such as CPU loading and memory usage, network latency, or task-specific overheads such as deadlines and urgency to effectively assign resources and schedule tasks in a resource-constrained dynamic environment in ways that incur the least cost of computation.

A general scheduling model for cloud-fog systems was proposed by Guevara and da Fonseca (2021), which integrates the dynamic priority levels for tasks based on the context of the system. This methodology showed significant improvement in minimizing execution time span and scheduling responsiveness across distributed resources, particularly for delay-sensitive services. Similarly, Chai et al. (2021) focused on offloading in mobile edge computing environments and interdependent task scheduling. The model sets dynamic priority levels and considers parallel transmission and execution mechanisms which gradually enhanced the execution time and resource allocation for structures with limited resources.

A pre-emptive scheduling algorithm using multi-priority queues was proposed by Fahad et al. (2022), which highlighted the improvement in latency with less computational overhead compared to meta-heuristic algorithms in the fog-edge computing context. The proposed scheduler improved fairness and reduced task starvation compared to the traditional approaches. Additionally, Hosseini et al. (2022) proposed a priority-based framework that integrates Fuzzy Analytical Hierarchy Process (FAHP) in mobile-edge computing, which decreased the latency issues. Similarly, in heuristic contexts Dehury et al.

(2024) designed a Heuristic strategy (HeRAFC) for a multifog-cloud environment, which enhanced the capabilities of latency, resource utilization, and execution time, decreasing the cloud workload depending on the results of simulations, but with limited verification in the real-world settings.

Lipsa et al. (2023) highlighted the utilization of the Fibonacci Heap and queuing analysis to optimize the preemptive task allocation in virtual machines. This priority-based approach demonstrated improved efficiency over traditional genetic algorithms, particularly in the context of managing cost and response-time-critical dynamic cloud environments. Additionally, a detailed research was provided by Alsadie (2024) regarding the efficiency of heuristic strategies and mentioned their ability to reduce overhead in dynamic workloads. Liberati et al. (2025) presented the integration of Model Predictive Control (MPC) with a hybrid scheduling approach. Its application in assembly-line systems has enabled rapid, near-optimal decision-making with significant execution time reduction and applicability in real-time industrial IoT contexts.

These works justify the heuristic and priority method for real-time time scenarios, yet two major limitations arise: (A) Overdependence on simulation-based appraisal and less implementation test of the edge-fog-cloud infrastructure (B) The supposed works do not have a framework for comparison and evaluation of the heuristics and the priority under identical conditions. These gaps, coupled with the need for scalability in hybrid models, highlight the need for an evaluation framework that can integrate simulation with physical hardware, enabling direct comparison of both approaches.

2.3 Unified evaluation and Hybrid frameworks

The current studies has shown that simulation-based evaluations were not capable of capturing complex and behavioral performance of real-world fog and edge environments, considering their resource-constrained and dynamic nature. The limitations discussed in the earlier section highlight that the lightweight scheduling strategies were tested majorly in simulated settings. Without actual device validation, algorithms which were well designed may fall short in real-time and critical IoT environments.

Several scheduling strategies were analyzed and their trade-offs were discussed in S and K (2024), but it relied entirely on simulation. Khaledian et al. (2024) assessed a hybrid AI and metaheuristic model and performed theoretical and conceptual analysis, and addressed the need of hybrid framework for performance benchmarking of algorithms in real-time conditions. To overcome these issues of latency and reliability in edge-fog systems, Panchal and Saxena (2025) performed a systematic review to identify key parameters for node selection in scheduling, but without practical deployment. Ju et al. (2024) proposed a low-latency architecture for fog-edge nodes, but did not provide any latency measurements under actual workloads. piFogBed, a Raspberry Pi-based fog computing testbed that supports basic validation, was introduced by Xu and Zhang (2019), but lacks in the comparative evaluation of lightweight algorithms.

In Addition, Lal et al. (2024) provided an in-depth comparison between fog and cloud models, including standard dimensions such as scalability, decentralization, and resource orchestration strategy in theory, but lacked real-hardware experimentation. Bamber et al. (2024) characterized the architectural paradigms of fog computing by describing the system in terms of service proximity, governance, and distribution of resources. The research highlighted that distributed resource placement can reduce end-to-end latency, but also stated that standardized benchmarking frameworks were less, and few architectures were tested under real-world dynamic workloads. A lightweight edge-computing network was tested on performance metrics like jitter and processing delay by Peng et al. (2018), but did not evaluate lightweight scheduling algorithms that were crucial to latency-sensitive applications.

Even though awareness has increased, major studies focus on isolated components or perform experiments in simulated settings. There were a few studies that implemented a unified physical testbed but lacked in comparing lightweight scheduling models under identical conditions. Such discrepancies of previous studies were an opportunity to address the gap by implementing a systematic real-device evaluation framework and comparing lightweight scheduling strategies.

2.4 Research Gap

The current research in task scheduling and distribution across edge-fog-cloud environments relied on simulation-based evaluation of AI and metaheuristic models. Although these models demonstrate high levels of optimization, but they often lack validation in real-world, resource-constrained IoT conditions. Also, major comparative studies focus on independent factors without direct comparison using a consistent workload. Some studies developed hybrid frameworks combining simulation and physical nodes, but in a limited scope, as they did not offer unified benchmarking across edge, fog, and cloud layers. In Addition, there was no systematic analysis for heuristic and priority-based models, as they were considered better for real-time conditions.

To overcome these limitations, this research creates a unified physical testbed that utilizes synthetic workloads to evaluate heuristic and priority-based scheduling algorithms in real-time conditions. The architecture comprises Raspberry PI edge nodes and AWS cloud instances for replicating fog and cloud layers. To achieve better resource placement, a bottom-up selection layer mechanism was applied that enhances the existing scheduling models, which enables the dynamic selection of the most suitable layer based on task characteristics and resource availability. The study fills the gap by providing a direct algorithmic comparison of priority and heuristic algorithms and an optimal resource placement strategy that was adaptive to the real-world distributed IoT environments.

Table 1: Comparison of Key Features in Related Works

Paper	**HW Valid.**	**RT Focus**	**Latency Opt.**	**Energy Eff.**	**Sim-Based**	**Technique Used**
Alsadie (2024)	-	✓	✓	✓	✓	Heuristic Review
S and K (2024)	-	-	✓	✓	✓	Comparative Study
Chai et al. (2021)	-	✓	✓	-	✓	Dynamic Priority
Dehury et al. (2024)	-	✓	✓	✓	✓	HeRAFC
Fahad et al. (2022)	-	✓	✓	✓	✓	MQP-Priority
Guevara and da Fonseca (2021)	-	✓	✓	-	✓	Priority-Based

Paper	**HW Valid.**	**RT Focus**	**Latency Opt.**	**Energy Eff.**	**Sim-Based**	**Technique Used**
Khaledian et al. (2024)	-	✓	✓	✓	✓	AI+Heuristic
Hosseini et al. (2022)	-	✓	✓	✓	✓	FAHP
Lipsa et al. (2023)	-	✓	✓	✓	✓	Fibonacci Heap
Peng et al. (2018)	✓	✓	-	✓	✓	Edge-evaluation framework
Reddy et al. (2020)	-	-	-	✓	✓	GA
Saeik et al. (2021)	-	-	✓	✓	✓	RL Survey
Swarup et al. (2021)	-	✓	✓	✓	✓	DRL
Tahmasebi-Pouya et al. (2023)	-	-	✓	✓	✓	RLIS
Wang et al. (2024)	-	✓	✓	✓	✓	ReinFog (DRL)
Xu and Zhang (2019)	✓	✓	✓	-	-	Pi-fogbed
Zhang et al. (2019)	-	-	-	✓	✓	GA
This research	✓	✓	✓	✓	-	Bottom-up Evaluation framework

3 Methodology

The research adopts an experimental approach to analyzing lightweight scheduling techniques to assess edge-fog-cloud architectures using a fully deployed physical testbed. The aim is to benchmark the architecture's performance on realistic loads in terms of latency, energy consumption, deadlines and load processing. The subsections below describe the workflow of the research, workload preparation and the algorithms assessed in this paper.

3.1 Research Framework

The study was carried out through several consecutive stages. A three-level edge-fog-cloud architecture was initially configured to simulate real-world, distributed computing scenarios where latency-sensitive applications execute closest to the data origin. The edge layer is composed of Raspberry PI devices that can be responsive in real-time, and the fog and cloud layers offer more computational capacity to offloaded tasks. Synthetic IoT workloads were created after setting up the infrastructure, which were then systematically deployed with the help of several scheduling strategies. Logs of every task were kept during the execution, with the information about the placement decisions, the processing time, and energy estimations. Lastly, performance metrics were calculated and measured to assess and make comparisons of the effectiveness of the applied algorithms.

3.2 Synthetic Workload Generation

The generation of realistic and unbiased workload was an important component in this experimental framework. Workloads were synthetically generated with a python generator, containing certain number of tasks, each of which was annotated with essential attributes: computational-complexity (in Million instructions), input data size (kb), deadline (milliseconds), and priority level (high, medium , or low). In the workload design stage heterogeneity was considered to include balanced distribution of light, medium, and heavy tasks, similar to those in applications involving IoT in the real world. To prevent bias towards any particular execution layer, task deadlines were proportionally allocated and the order of task submission was randomly designed to simulate changing conditions in the IoT systems.

3.3 Scheduling Algorithms

Three scheduling techniques were implemented and tested under same conditions to replicate real-time scenarios. The former is a heuristic scheduler based on Dehury et al. (2024) which calculates a normalized cost function using execution time, energy cost and priority of the tasks to calculate the most suitable execution layer. The second, an MQP strategy based on Fahad et al. (2022), which classifies and organizes tasks based on priority and burst lengths, with bottom-up starvation policy (edge $\rightarrow$ fog $\rightarrow$ cloud) to reduce violation of deadlines and starvation. Lastly, a baseline comparison was performed with GA-PSO-based metaheuristic approach. This algorithm is computationally intensive, which was used as a benchmark to evaluate and discuss the trade-off between accuracy of optimization and the real-time practicability. All algorithms were implemented in the physical testbed and executed with same workloads to ensure fairness and systematic comparison.

3.4 Measurement Approach

The performance metrics measurement that were considered essential for real-time IoT systems were carried out including end-to-end latency, the estimated energy consumption based on execution time and power models of the devices, deadline miss ratio (the ratio of tasks which missed their deadline), and resource utilization of three layers. The output logs of every algorithms were analyzed and visualized through python scripts and statistical analysis was also performed to find descriptive values like mean values and standard deviation in every cycle of experiment. This provided reliability and consistency of the reported results.

3.5 Analysis and Visualization

The processed data containing metrics was merged into organized CSV files, which were analyzed in the Google Colab platform for comparison. The data was processed and analyzed using the `pandas`[2] library, whereas the visual graphs were generated using `matplotlib`[3] and `seaborn`[4]. The generated plots and graphs included task distribution comparison, performance time, energy consumption, and deadline violations for all the scheduling strategies that were considered and evaluated. These visual results were used to decide the trade-offs between latency, energy efficiency, and computational intensity of each algorithm.

[2]https://pandas.pydata.org/
[3]https://matplotlib.org/
[4]https://seaborn.pydata.org/

Algorithm 1 HeRAFC-inspired Scheduler (Normalize $\rightarrow$ Check $\rightarrow$ Dispatch)

```
Require: Task $t = \{$cpu_cycles, input_size, deadline, priority$\}$;
    $[\mathrm{Edge}, \mathrm{Fog}, \mathrm{Cloud}]$ with parameters $(\mathrm{MIPS}, \mathrm{BW}, \mathrm{energy}, \mathrm{deadline\_thr})$;
    normalized global maxima $D_{\max}, P_{\max}, C_{\max}, E_{\max}$
0: for each layer $l$ in order do
0:   Energy and Time Estimation: $t^{\mathrm{exec}} \leftarrow \frac{\mathrm{cpu\_cycles}}{\mathrm{MIPS}(l)}$,   $t^{\mathrm{tx}} \leftarrow \frac{\mathrm{input\_size}}{\mathrm{BW}(l)}$
0:   $T \leftarrow 1000\,(t^{\mathrm{exec}} + t^{\mathrm{tx}})$,   $E \leftarrow \mathrm{cpu\_cycles} \cdot \mathrm{energy}(l)$
0:   Normalization: $\tilde{T} = T/D_{\max}$, $\tilde{E} = E/(C_{\max} \cdot E_{\max})$, $\tilde{P} = \mathrm{priority}/P_{\max}$
0:   heuristic calculation: $J = w_t\tilde{T} + w_e\tilde{E} - w_p\tilde{P}$
0:   Layer Feasibility (check): if $\tilde{T} \le \min\big(\mathrm{deadline}(t)/D_{\max},\ \mathrm{deadline\_thr}(l)/D_{\max}\big)$
      then dispatch $t$ to $l$ and return else continue
0: end for
0: Skip $t$ (no feasible layer) =0
```

Algorithm 2 MQP Priority Scheduler (Short/Long, Bottom-Up)

```
Require: Threshold $\theta$; layers $[\mathrm{Edge}, \mathrm{Fog}, \mathrm{Cloud}]$ with $(\mathrm{MIPS}, \mathrm{BW}, \mathrm{deadline\_thr})$
0: Enqueue: $cpu\_cycles \le \theta \Rightarrow$ Short queue; else Long
0: while queues not empty do
0:   $t \leftarrow$ pop from Short if notempty, else from Long
0:   for each layer $l$ do
0:     $T \leftarrow 1000 \cdot \big(\frac{cpu\_cycles(t)}{MIPS(l)} + \frac{input\_size(t)}{BW(l)}\big)$
0:     if $T \le deadline\_thr(l)$ then
0:       dispatch $t$ to $l$; break
0:     end if
0:   end for
0:   if no acceptance then mark skipped logged and offloaded to Cloud
0:
```

Table 2: Formulas and Evaluation Metrics

Metric	**Formula**
Execution Time (t^{exec})	$t^{exec} = \dfrac{cpu\_cycles}{MIPS(l)}$
Transfer Time (t^{tx})	$t^{tx} = \dfrac{input\_size}{BW(l)}$
Latency (T)	$T = 1000 \cdot (t^{exec} + t^{tx})$ (ms)
Energy (E)	$E = cpu\_cycles \times energy\_coeff(l)$
Average Latency	$\overline{T} = \dfrac{1}{N}\sum_{i=1}^{N} T_i$
Deadline Miss Ratio (DMR)	$DMR = \dfrac{\text{Tasks Missed}}{\text{Total Tasks}}$
Deadline Adherence Rate (DAR)	$DAR = \Big(1 - DMR\Big) \times 100$
Utilization (U)	$U_l = \dfrac{\sum_{i \in S_l} T_i}{W_l}$

4 Design Specification

The hybrid **edge-fog-cloud architecture** aims to optimize task scheduling in IoT environments that require low-latency execution. The framework focuses on bottom-up **execution layer selection policy**: the tasks were executed at the edge layer to achieve low latency depending on the capacity of the layer, or else it was offloaded to fog when resources are insufficient, and to the cloud when fog's capacity was overloaded. This top-down strategy was followed, which allows the optimal use of resources with strong limitations on the deadlines.

4.1 Workflow Overview

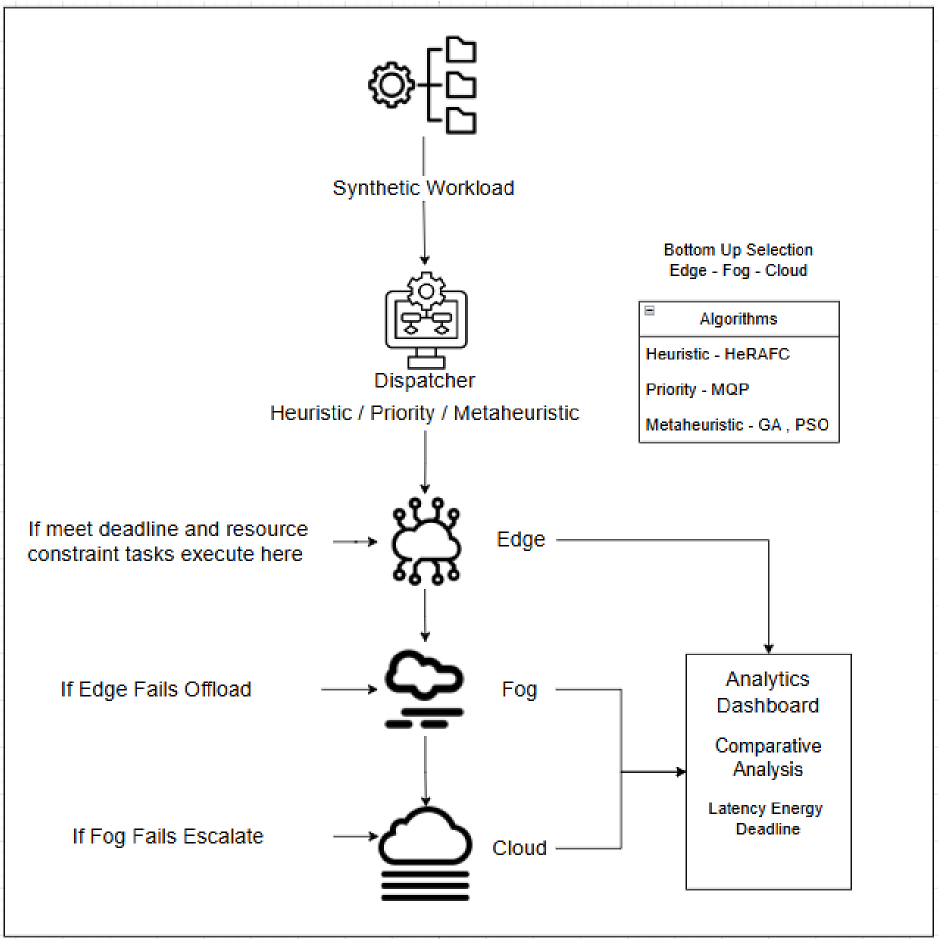


Figure 1: Proposed Edge–Fog–Cloud Architecture and Scheduling Workflow

The initial process of scheduling begins with synthetic workload generator, which generates heterogeneous IoT jobs with programmed requirements, priorities and deadlines. These tasks were sent to a centralized dispatcher that executes one of the following strategies (Heuristic, Priority-based or Metaheuristic) with bottom-up selection policy. The dispatcher was responsible for making decision regarding layer placement. In the execution phase, tasks were logged with metadata of layer placement, average execution time, transfer time and energy estimations. These logs were further used in comparative analysis of algorithms in terms of performance, such as latency, energy efficiency, and deadline adherence.

4.2 Scheduling Approach and Contributions

The combination in the framework included purpose-tailored optimizations for dynamic, resource-constrained environments along with the use of lightweight scheduling algorithms.

1. **Heuristic Scheduler (HeRAFC-Inspired):** This scheduler calculates a normalized cost function for the decision of execution layer placement considering execution time (T), energy usage (E), and task priority (P). The cost is calculated as follows:

$$C = w_t \frac{T}{T_{\max}} + w_e \frac{E}{E_{\max}} - w_p \frac{P}{P_{\max}}$$

   The weight of each factor, denoted by w_t, w_e, and w_p can vary and represent relative importance associated with that factor. The bottom-up policy (edge $\rightarrow$ fog $\rightarrow$ cloud) was followed, so that task allocation follows the view to execute latency-sensitive task at the lowest layer and considering resource restrictions. This approach extends the standard heuristic algorithm in two ways: It offers the capabilities of **layer-specific deadline testing** and **adjusting the weights dynamically depending on the real-time load on the system** to increase the adaptiveness and flexibility in changing workloads.

2. **Priority-Based Scheduler (MQP Adaptation):** This strategy makes use of urgency-based reasoning to sequence the tasks into several queues, considering the priority and burst lengths. The bottom-up decision approach was followed, where short-duration and priority-intensive tasks were executed close to the edges. If the deadline criteria were not met at the lowest level, the task was further offloaded to the fog layer and then to the cloud. The major improvements over the baseline MQP models were **deadline-driven escape logic**, which was used to reduce frequent deadline violations occurring, and **task starvation avoidance methods** using dynamic time slicing to ensure fairness within the low-priority queues without interfering with the higher-priority tasks.

$$Q = \begin{cases} Q_{\text{short}}, & \text{if } \text{CPU}_{\text{task}} \leq \theta \\ Q_{\text{long}}, & \text{otherwise} \end{cases}$$

$$\text{Execution order: Priority}_{\text{high}} \rightarrow \text{Edge} \rightarrow \text{Fog} \rightarrow \text{Cloud}$$

A hybrid **GA-PSO metaheuristic** algorithm was implemented on the synthetic workload, which worked as an optimization-related reference point to benchmark the tradeoffs between accuracy and computational overhead of operating in a real-time scenario.

4.3 System Specifications

Table 3 provides the summary of hardware capabilities and deadlines associated with each architectural layer.

Table 3: Layer Configuration and Capabilities

Layer	**Hardware**	**MIPS**	**Deadline (ms)**
Edge	Raspberry Pi 3B (1GB RAM)	500	1500
Fog	AWS EC2 t2.small (2vCPU)	1000	2500
Cloud	AWS EC2 t3.large (4vCPU)	2000	—

5 Implementation

The lightweight scheduling strategies were executed in the implementation step on a fully configured edge-fog-cloud testbed, which includes controlled tests that evaluated their efficiency. The implemented solution with a physical hardware architecture provides a comparison of the algorithms based on parameters such as latency, energy, and deadline achieved during the results.

5.1 Technologies and Tools used

The following resources were used for system implementation:

1. **Scripts and Programming:** Workload generation, replication of scheduling algorithms, and results logging were performed using Python Scripts
2. **Execution Layers:** Raspberry Pi 3B for the edge tier; AWS EC2 `t3.small` instance for fog; AWS EC2 `t3.large` instance for cloud.
3. **Data Analysis:** Libraries such as `pandas`, `matplotlib`, and `seaborn` were used for visualization and statistical analysis in Google Colab environment.

5.2 Synthetic Workload Preparation

A heterogeneous IoT task generator was based on Python scripts that replicated the diversity of applications based on real-world scenarios. The workload generated consists of four significant attributes, including the computation time (in Million Instructions), input size (in KB), deadline (in milliseconds), and priority (High, Medium, or Low). To ensure fairness or to give no layer an advantage, deadlines were spread evenly, and the order of submission would be randomized to simulate dynamic traffic. A summary of workload characteristics was illustrated in Table 4.

5.3 Scheduling Algorithm Integration

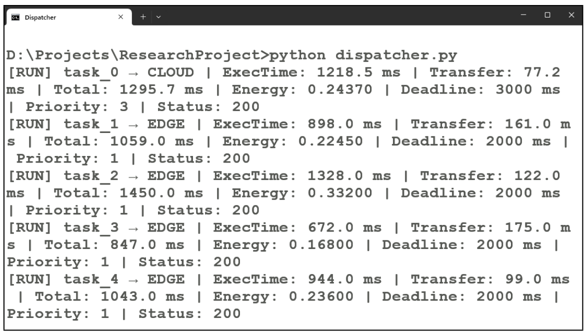
Three scheduling methods were implemented: (1) Heuristic scheduler based on HeR-AFC, which contains the adjustments of layer-specific deadlines check and adjustment of weights dynamically; (2) Priority-based scheduler (MQP extension), which decreased the starvation of processes and priority-based escalation; and (3) GA-PSO metaheuristic algorithm that was used to benchmark optimization accuracy. The algorithms were based on the bottom-up layer selection policy (edge $\rightarrow$ fog $\rightarrow$ cloud), providing minimum latency to critical requests and optimal resource management.

Table 4: Synthetic Workload Parameters

Parameter	Description
`task_id`	Unique identifier for each task
`type`	Category of (e.g., A = Light, B = Medium, C = Heavy)
`cpu_cycles_mi`	Computational complexity (Million Instructions)
`input_size_kb`	Input data size in kilobytes
`deadline_ms`	Task completion Deadline (milliseconds)
`priority`	Priority level (1 = Low, 2 = Medium, 3 = High)
`timestamp`	Scheduling sequence - Task creation time

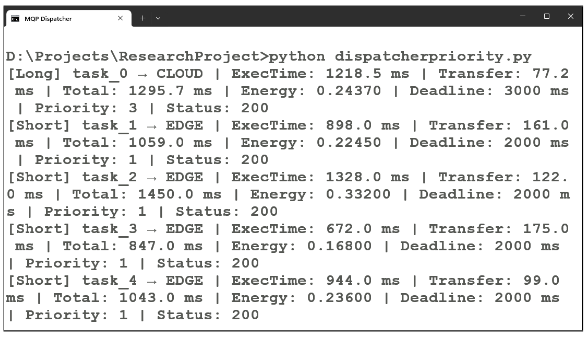
5.4 Execution Workflow

The experimental procedure was as follows: (1) Synthetic workloads generated dynamic traffic; (2) Tasks were scheduled to layers and dispatched using parameterized algorithms; (3) The data generated during execution was recorded in real-time with details like task ID, layer to which the task is allocated, start and completion time, estimation of energy consumption as well as meeting of the deadlines;(4) The logs were exported into structured CSV files and processed. The example of how tasks were distributed between edge, fog, and cloud layers was shown in Figure 2 according to the results obtained in the experiment.



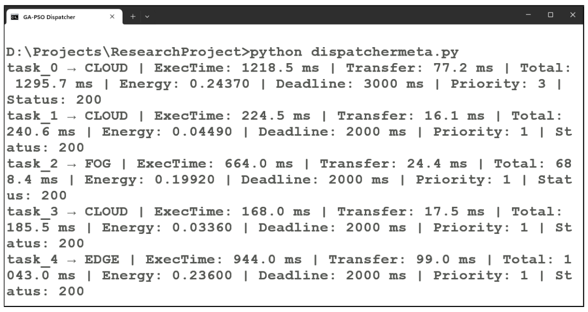
```
D:\Projects\ResearchProject>python dispatcher.py
[RUN] task_0 → CLOUD | ExecTime: 1218.5 ms | Transfer: 77.2
ms | Total: 1295.7 ms | Energy: 0.24370 | Deadline: 3000 ms
| Priority: 3 | Status: 200
[RUN] task_1 → EDGE | ExecTime: 898.0 ms | Transfer: 161.0 m
s | Total: 1059.0 ms | Energy: 0.22450 | Deadline: 2000 ms |
 Priority: 1 | Status: 200
[RUN] task_2 → EDGE | ExecTime: 1328.0 ms | Transfer: 122.0
ms | Total: 1450.0 ms | Energy: 0.33200 | Deadline: 2000 ms
| Priority: 1 | Status: 200
[RUN] task_3 → EDGE | ExecTime: 672.0 ms | Transfer: 175.0 m
s | Total: 847.0 ms | Energy: 0.16800 | Deadline: 2000 ms |
Priority: 1 | Status: 200
[RUN] task_4 → EDGE | ExecTime: 944.0 ms | Transfer: 99.0 ms
 | Total: 1043.0 ms | Energy: 0.23600 | Deadline: 2000 ms |
Priority: 1 | Status: 200
```

(a) HeRAFC-Based Heuristic Logs



```
D:\Projects\ResearchProject>python dispatcherpriority.py
[Long] task_0 → CLOUD | ExecTime: 1218.5 ms | Transfer: 77.2
 ms | Total: 1295.7 ms | Energy: 0.24370 | Deadline: 3000 ms
 | Priority: 3 | Status: 200
[Short] task_1 → EDGE | ExecTime: 898.0 ms | Transfer: 161.0
 ms | Total: 1059.0 ms | Energy: 0.22450 | Deadline: 2000 ms
 | Priority: 1 | Status: 200
[Short] task_2 → EDGE | ExecTime: 1328.0 ms | Transfer: 122.
0 ms | Total: 1450.0 ms | Energy: 0.33200 | Deadline: 2000 m
s | Priority: 1 | Status: 200
[Short] task_3 → EDGE | ExecTime: 672.0 ms | Transfer: 175.0
 ms | Total: 847.0 ms | Energy: 0.16800 | Deadline: 2000 ms
| Priority: 1 | Status: 200
[Short] task_4 → EDGE | ExecTime: 944.0 ms | Transfer: 99.0
ms | Total: 1043.0 ms | Energy: 0.23600 | Deadline: 2000 ms
| Priority: 1 | Status: 200
```

(b) Priority-Based Scheduling (MQP)



```
D:\Projects\ResearchProject>python dispatchermeta.py
task_0 → CLOUD | ExecTime: 1218.5 ms | Transfer: 77.2 ms | Total:
 1295.7 ms | Energy: 0.24370 | Deadline: 3000 ms | Priority: 3 |
Status: 200
task_1 → CLOUD | ExecTime: 224.5 ms | Transfer: 16.1 ms | Total:
240.6 ms | Energy: 0.04490 | Deadline: 2000 ms | Priority: 1 | St
atus: 200
task_2 → FOG | ExecTime: 664.0 ms | Transfer: 24.4 ms | Total: 68
8.4 ms | Energy: 0.19920 | Deadline: 2000 ms | Priority: 1 | Stat
us: 200
task_3 → CLOUD | ExecTime: 168.0 ms | Transfer: 17.5 ms | Total:
185.5 ms | Energy: 0.03360 | Deadline: 2000 ms | Priority: 1 | St
atus: 200
task_4 → EDGE | ExecTime: 944.0 ms | Transfer: 99.0 ms | Total: 1
043.0 ms | Energy: 0.23600 | Deadline: 2000 ms | Priority: 1 | St
atus: 200
```

(c) Metaheuristic Scheduling (GA–PSO)

Figure 2: Execution logs of task placement decisions across scheduling algorithms

5.5 Outputs Produced

The following outputs were logged in the implementation phase:

1. CSV files containing execution metrics per task(latency, energy estimates, layer placement, and deadline status).
2. Aggregation was applied to the performance data of each scheduling strategy.
3. To compare the results of algorithms based on task distribution, average execution and transfer time, energy consumption, and the ratio of missed deadlines, various plots and graphs were generated

5.6 Analysis Pipeline

The libraries of pandas, matplotlib, and seaborn were used on Google Colab platform to visualize and compare the details of the solution. The Python scripts were used to process the execution logs and get descriptive statistics like mean and standard deviation. The generated bar graphs highlighted a precise summary of distributed tasks, which denoted the trade-offs in latency, energy consumption and resource allocation in scheduling algorithms.

6 Evaluation

The performance details of the three scheduling strategies included in the proposed framework (heuristic (HeRAFC-inspired), priority-based (MQP adaptation), and metaheuristic (GA-PSO hybrid)) were analyzed. To determine the performance behaviour, three different workload environments of **500 tasks**, **1000 tasks**, and **1500 tasks** were processed in the hybrid edge-fog-cloud infrastructure. The analysis was carried out based on the following performance indicators which are as follows:

1. Average Execution Time (ms): Total time taken for task completion.
2. Average Transfer Time (ms): Data transmission latency between layers.
3. Average Energy Consumption: Total Energy consumption per task.
4. Standard Deviation of Execution Time (ms): Variance in task completion time.
5. Missed deadlines (absolute count): Tasks not meeting the deadline threshold during pre-dispatch decision.
6. Deadline adherence rate: % of tasks completing before deadline thresholds.

6.1 Case Study 1: 500 Tasks

Observation: The priority algorithm based on the MQP algorithm recorded the lowest average execution time (1064.87 ms) and transfer time (89.60 ms), which performed better than the heuristic and GA-PSO. The long and short priority queues ensured that high-priority tasks were executed near the edge. Also, the MQP had the lowest energy consumption (0.2670 units), as the short tasks were processed in the lower layers.

The heuristic algorithm highlighted balanced performance, while GA-PSO achieved high transfer latency(133.00 ms), energy consumption(0.3080), and most missed deadlines due to its optimized allocation, computational overhead, and greater reliance on offloading tasks to the cloud layer.

Table 5: Performance Metrics for 500 Tasks

Algorithm	Avg Exec (ms)	Avg Transfer (ms)	Avg Energy	Std Dev (ms)	Missed DL
Heuristic	1159.33	92.73	0.2910	247.37	322.67
Priority	1064.87	89.60	0.2670	220.37	370.33
GA–PSO	1197.93	133.00	0.3080	769.17	493.67

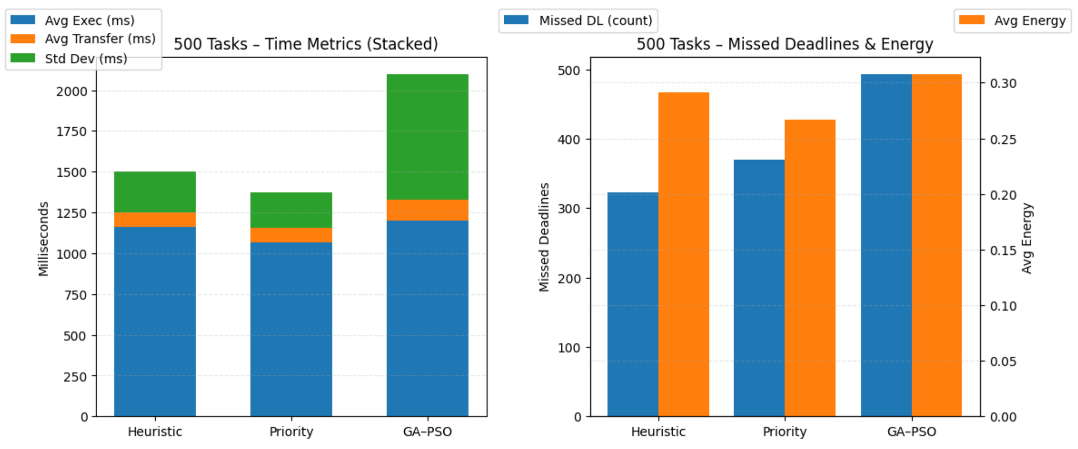


Figure 3: Summary of Performance Metrics (500 Tasks)

6.2 Case Study 2: 1000 Tasks

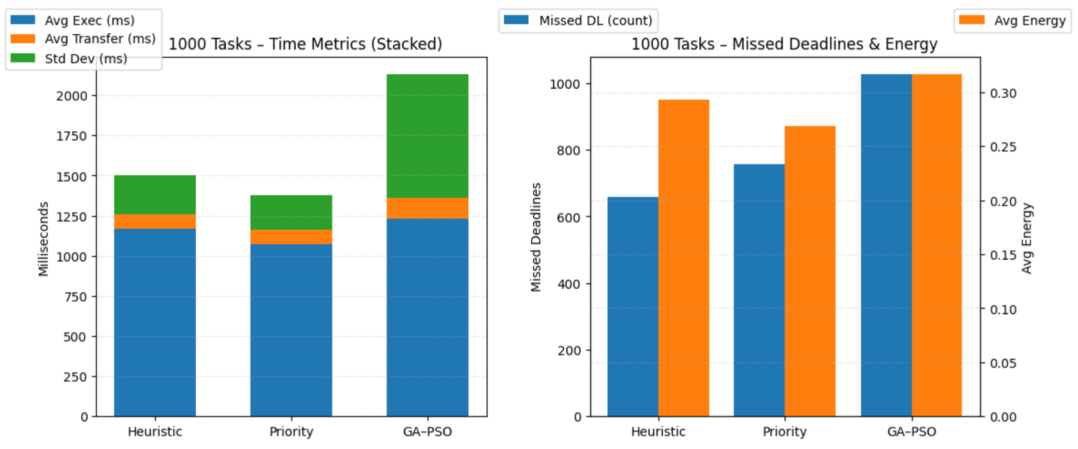


Figure 4: Summary of Performance Metrics (1000 Tasks)

Observation: Similar trends were observed in the 500 tasks case study, where the priority scheduler maintained the lowest execution time(1073.67 ms) and transfer (90.47 ms) time. As the task workload was doubled, deadline violations also increased gradually. GA-PSO was the most variable in terms of standard deviation (769.77 ms), which indicates that it is less preferable for real-time IoT workloads. Stability was observed in the heuristic approach, which indicates its better scalability in mixed latency environments.

Table 6: Performance Metrics for 1000 Tasks

Algorithm	Avg Exec (ms)	Avg Transfer (ms)	Avg Energy	Std Dev (ms)	Missed DL
Heuristic	1166.47	93.03	0.2930	243.27	659.33
Priority	1073.67	90.47	0.2690	214.97	756.67
GA–PSO	1228.40	135.50	0.3170	769.77	1028.00

6.3 Case Study 3: 1500 Tasks

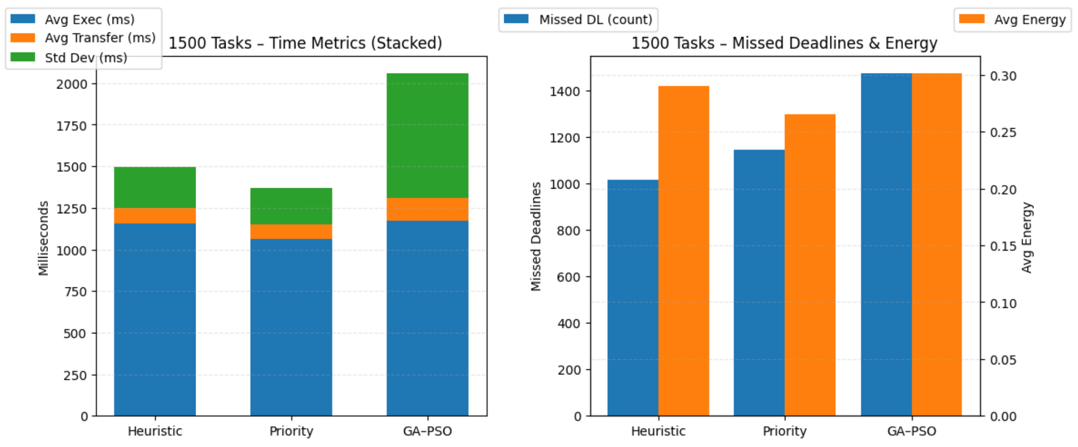


Figure 5: Summary of Performance Metrics (1500 Tasks)

Observation: An increase to 1500 tasks confirmed the previous observations. The priority algorithm recorded the lowest execution time (1061.00 ms) and consumed the lowest energy (0.2657 units), which demonstrated robustness under increased load. Still, the heuristic algorithm was consistent across all workloads, which makes it reliable at the mid-level between speed and efficiency. In contrast, GA-PSO was competitive in some metrics with other algorithms, but still recorded high latency and energy consumption due to its highly computationally intensive fitness calculation and metaheuristic approach. These consistent results demonstrate the scalability and reliability of the assessment framework, which supports its applicability to benchmark the scheduling strategy process in real-world demanding IoT settings.

Table 7: Performance Metrics for 1500 Tasks

Algorithm	Avg Exec (ms)	Avg Transfer (ms)	Avg Energy	Std Dev (ms)	Missed DL
Heuristic	1156.20	94.47	0.2902	246.80	1017.33
Priority	1061.00	91.83	0.2657	214.17	1145.67
GA–PSO	1174.80	133.00	0.3017	753.00	1475.00

6.4 Task Distribution

Observation: The Figure 6 highlights the distribution methods and their influence on performance results. A bottom-up allocation policy was followed by both heuristic and priority schedulers, which favoured the edge and fog layers for major tasks. Significant reduction in transfer delays and energy usage was observed with alignment of lower

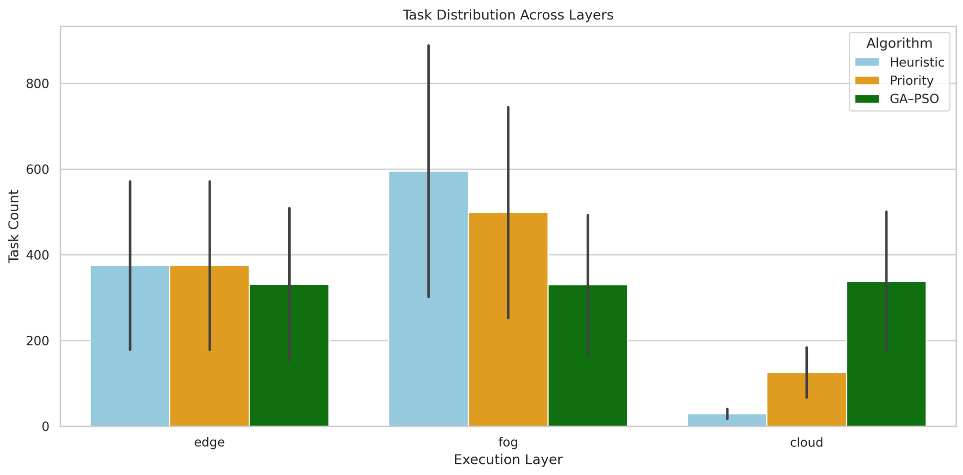


Figure 6: Task Distribution Across Edge, Fog, and Cloud Layers

latency value recorded in Tables 5–7. Additionally, GA-PSO offloads more tasks to the cloud layer, which increases data transfer time and energy consumption. This distribution pattern clearly explains the higher latency and incurred energy costs observed for the GA-PSO approach.

6.5 Discussion of Findings

The three algorithms were tested on the same synthetic workloads as outlined in Section 5 with uniform hardware requirements at the edge, fog, and cloud layers. The summary of important trends was stated in Figure 7 and performance measures can be found in Tables 5, 6, and 7.

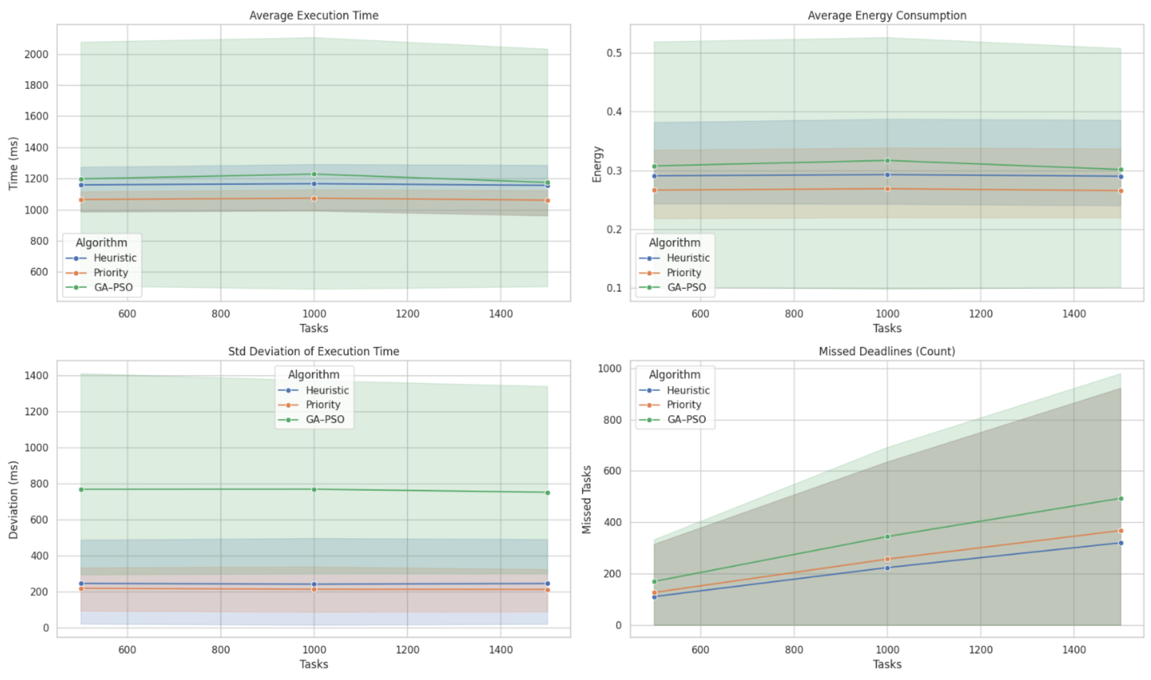


Figure 7: Summary trend of all metrics

The experiment results showed a consistent trend in all three cases (500, 1000, and 1500), which strengthens the validity of indicators during performance measurement and creates a strong platform for the comparative analysis that follows in the discussion:

1. **Priority-Based Scheduling:** This approach can be considered best in latency-based applications, including emergency warning systems or patient care monitoring, as its execution and transfer time is less compared to other algorithms. The MQP-based approach had a higher rate of deadline violation due to its hard priority queues compared to the heuristic approach.

2. **Heuristic Scheduling:** The strategy was balanced in terms of trade-off between execution time, energy consumption, and the satisfaction of deadline constraints. It was specifically designed for mixed IoT (with a critical need for latency and energy), e.g., smart grids or industrial IoT platforms.

3. **GA-PSO Metaheuristic:** As much as the GA-PSO is capable in theory of identifying globally optimized task allocations, its high computational load implies that it cannot be used to handle real-time computational loads. It is more appropriate when applied in an offline optimization case or in batch scheduling with a lot of flexibility in the deadlines, with the need for long-term optimization of resource usage.

The results highlighted that lightweight algorithms (Heuristic and Priority-based) are more efficient in comparison with metaheuristic methods in dynamic (real-time) IoT systems with low latency and energy consumption requirements.

7 Conclusion and Future Work

7.1 Conclusion

This research aimed to investigate and optimize task scheduling in distributed edge–fog–cloud architectures through lightweight and metaheuristic strategies under realistic IoT workloads. The initial objective was to compare the performance of scheduling algorithms, i.e., heuristic-based, priority-based, and GA-PSO based metaheuristics, in terms of latency, energy consumption, and the ability to meet deadlines in a fully deployed physical testbed.

A three-layer edge–fog–cloud system was implemented with identical hardware restrictions and various synthetic workloads that replicate an actual IoT environment. The heuristic scheduling approach inspired by the HeRAFC model shows a balanced trade-off on performance metrics. In latency-critical scenarios priority-based scheduling approach was effective, although it violated deadlines under high-load conditions occasionally. The metaheuristic algorithm was theoretically capable of optimal placements, but proved less efficient due to its high computational overhead and variance in deadline violations.

Lightweight scheduling algorithms outperformed metaheuristics in real-time responsiveness and energy efficiency were the key findings demonstrated in this research. The heuristic scheduler had the best deadline satisfaction compared to GA-PSO, while the priority strategy minimized execution time. These observation and results highlights the importance of customized scheduling strategies in resource-constrained IoT systems. The experimental framework confirmed the effectiveness of a physical testbed in capturing realistic system behaviours.

7.2 Limitations and Future Work

Although the study showed encouraging results there are still limitations and future studies that can be followed up. The architecture was tested with multiple synthetic task profiles to significantly replicate realistic IoT use-cases, but it still may lack in handling burstiness, unpredictability, and patterns with multi-modal data occurring in real IoT deployments. Adaptive scheduling algorithms, which dynamically respond to fluctuations in workload and network conditions, can be explored in the future to enhance the robustness and scalability of the proposed architecture. In the future, simulation-based environments can be incorporated with the physical testbed.

1. The simulation platforms like iFogSim and CloudSim can be utilized to increase the testing of IoT workloads on a broader range.
2. The future research can include dynamic scheduling methods using reinforcement methods and fuzzy logic integration to adapt towards changing contexts.
3. The proposed evaluation framework can be extended by integrating real sensor data from Industries and smart cities, which would validate its practicality and scalability for its deployment in commercial settings.

References

Alsadie, D. (2024). Advancements in heuristic task scheduling for IoT applications in fog-cloud computing: challenges and prospects, *PeerJ Computer Science* **10**: e2128.
URL: *https://peerj.com/articles/cs-2128*

Bamber, S. S., Dogra, N., Alhawat, S. K., Jaswal, J. K. and Sekhon, A. K. (2024). Navigating the Fog: A Comprehensive Survey and Comparative Analysis of Fog Computing Architectures, *2024 11th International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 1201–1206.
URL: *https://ieeexplore.ieee.org/document/10499030*

Chai, R., Li, M., Yang, T. and Chen, Q. (2021). Dynamic Priority-Based Computation Scheduling and Offloading for Interdependent Tasks: Leveraging Parallel Transmission and Execution, *IEEE Transactions on Vehicular Technology* **70**(10): 10970–10985.
URL: *https://ieeexplore.ieee.org/document/9531532/*

Dehury, C. K., Veeravalli, B. and Srirama, S. N. (2024). HeRAFC: Heuristic resource allocation and optimization in MultiFog-Cloud environment, *Journal of Parallel and Distributed Computing* **183**: 104760.
URL: *https://www.sciencedirect.com/science/article/pii/S0743731523001302*

Elgazzar, K., Khalil, H., Alghamdi, T., Badr, A., Abdelkader, G., Elewah, A. and Buyya, R. (2025). (PDF) Revisiting the internet of things: New trends, opportunities and grand challenges, *ResearchGate* .
URL: *https://www.researchgate.net/publication/365617050*

Fahad, M., Shojafar, M., Abbas, M., Ahmed, I. and Ijaz, H. (2022). A multi-queue priority-based task scheduling algorithm in fog computing environment, *Concurrency*

and Computation: Practice and Experience **34**(28): e7376.
URL: *https://onlinelibrary.wiley.com/doi/10.1002/cpe.7376*

Gad, A. G. (2022). Particle Swarm Optimization Algorithm and Its Applications: A Systematic Review, *Archives of Computational Methods in Engineering* **29**(5): 2531–2561.
URL: *https://link.springer.com/10.1007/s11831-021-09694-4*

Guevara, J. C. and da Fonseca, N. L. S. (2021). Task scheduling in cloud-fog computing systems, *Peer-to-Peer Networking and Applications* **14**(2): 962–977.
URL: *https://doi.org/10.1007/s12083-020-01051-9*

Hosseini, E., Nickray, M. and Ghanbari, S. (2022). Optimized task scheduling for cost-latency trade-off in mobile fog computing using fuzzy analytical hierarchy process, *Computer Networks* **206**: 108752.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S138912862100596X*

Ju, C., Ma, Y., Yin, Z. and Bi, Z. (2024). A Low Latency Processing System Architecture Strategy for Edge Service Nodes in Fog Computing, *2024 16th International Conference on Communication Software and Networks (ICCSN)*, pp. 107–112. ISSN: 2472-8489.
URL: *https://ieeexplore.ieee.org/document/10793367*

Khaledian, N., Voelp, M., Azizi, S. and Shirvani, M. H. (2024). AI-based & heuristic workflow scheduling in cloud and fog computing: a systematic review, *Cluster Computing* **27**(8): 10265–10298.
URL: *https://link.springer.com/10.1007/s10586-024-04442-2*

Lal, R., Singla, S. and Samal, N. (2024). Navigating the Cloud and Fog: An Extensive Comparative Analysis of Cloud Computing and Fog Computing Architectures, *2024 2nd International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pp. 1–6.
URL: *https://ieeexplore.ieee.org/document/10991096*

Liberati, F., Donsante, M., Cirino, C. M. F. and Tortorelli, A. (2025). Fast Task Scheduling With Model Predictive Control Integrating a Priority-Based Heuristic, *IEEE Access* **13**: 14499–14515.
URL: *https://ieeexplore.ieee.org/document/10843846/*

Lipsa, S., Dash, R. K., Ivković, N. and Cengiz, K. (2023). Task Scheduling in Cloud Computing: A Priority-Based Heuristic Approach, *IEEE Access* **11**: 27111–27126.
URL: *https://ieeexplore.ieee.org/document/10065487/*

Panchal, B. and Saxena, P. (2025). Fog Node Selection and Task Scheduling in Fog Computing Environment: A Review, *2025 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*, pp. 1–6. ISSN: 2688-0288.
URL: *https://ieeexplore.ieee.org/document/10940161*

Peng, F., Shou, G., Hu, Y. and Liu, Y. (2018). Lightweight Edge Computing Network Architecture and Network Performance Evaluation, *2018 Asia Communications and Photonics Conference (ACP)*, pp. 1–3.
URL: *https://ieeexplore.ieee.org/document/8596139*

Rahbari, D. (2022). Analyzing Meta-Heuristic Algorithms for Task Scheduling in a Fog-Based IoT Application, *Algorithms* **15**(11): 397.
URL: *https://www.mdpi.com/1999-4893/15/11/397*

Reddy, K. H. K., Luhach, A. K., Pradhan, B., Dash, J. K. and Roy, D. S. (2020). A genetic algorithm for energy efficient fog layer resource management in context-aware smart cities, *Sustainable Cities and Society* **63**: 102428.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S2210670720306491*

S, S. and K, V. (2024). A Comprehensive Analysis of optimization of task scheduling algorithms for IOT applications in a Combined Environment of Cloud, Fog and Edge, *2024 Third International Conference on Smart Technologies and Systems for Next Generation Computing (ICSTSN)*, pp. 1–6.
URL: *https://ieeexplore.ieee.org/document/10670804*

Saeik, F., Avgeris, M., Spatharakis, D., Santi, N., Dechouniotis, D., Violos, J., Leivadeas, A., Athanasopoulos, N., Mitton, N. and Papavassiliou, S. (2021). Task offloading in Edge and Cloud Computing: A survey on mathematical, artificial intelligence and control theory solutions, *Computer Networks* **195**: 108177.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S1389128621002322*

Swarup, S., Shakshuki, E. M. and Yasar, A. (2021). Energy Efficient Task Scheduling in Fog Environment using Deep Reinforcement Learning Approach, *Procedia Computer Science* **191**: 65–75.
URL: *https://linkinghub.elsevier.com/retrieve/pii/S1877050921014046*

Tahmasebi-Pouya, N., Sarram, M. A. and Mostafavi, S. (2023). A reinforcement learning-based load balancing algorithm for fog computing, *Telecommunication Systems* **84**(3): 321–339.
URL: *https://link.springer.com/10.1007/s11235-023-01049-7*

Wang, Z., Goudarzi, M. and Buyya, R. (2024). ReinFog: A DRL Empowered Framework for Resource Management in Edge and Cloud Computing Environments. Version Number: 1.
URL: *https://arxiv.org/abs/2411.13121*

Xu, Q. and Zhang, J. (2019). piFogBed: A Fog Computing Testbed Based on Raspberry Pi, *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*, pp. 1–8. ISSN: 2374-9628.
URL: *https://ieeexplore.ieee.org/document/8958741*

Zhang, R., Chen, Y., Dong, B., Tian, F. and Zheng, Q. (2019). A Genetic Algorithm-Based Energy-Efficient Container Placement Strategy in CaaS, *IEEE Access* **7**: 121360–121373.
URL: *https://ieeexplore.ieee.org/document/8813096/*