A New Deep Learning-Based Approach for IoT Task Offloading in Multi-access Edge Computing

Oussama Lagnfdi, Marouane Myyara, and Anouar Darif

***Abstract*—The exponential growth of Internet of Things (IoT) devices and the growing demand for resource-intensive applications have introduced significant challenges in computation, storage, and network efficiency. Although cloud computing provides partial relief, its centralized nature leads to unacceptable latency for delay-sensitive applications. Multi-access Edge Computing (MEC), especially with the advent of 5G, has emerged as a compelling solution by relocating computation closer to data sources, thereby reducing latency and improving responsiveness in applications such as smart agriculture, autonomous vehicles, augmented reality, and telemedicine. However, efficient workload offloading in MEC environments remains complex due to system heterogeneity, varying application requirements, and limited edge resources. This paper proposes a novel neural network-based approach to computation offloading in MEC, integrating workload allocation and resource management while accounting for application delay sensitivity, processing capacity, and communication constraints. The proposed model enables driving offloading decisions, adapting to fluctuating system states without relying on complex mathematical formulations. Simulation results demonstrate that the approach significantly reduces service time and enhances resource utilization, ensuring responsiveness for modern IoT applications. This research underscores MEC's potential to meet the rising computational and latency demands of next-generation IoT infrastructure.**

***Index Terms*—Multi-access Edge Computing Network, IoT, Task offloading, Deep Learning, Neural Network, Service time, Processing time**

I. INTRODUCTION

THE Internet of Things (IoT) is a rapidly developing ecosystem of distinct physical items connected through various wired and wireless networks [1]. It has enabled different industries to innovate new products, systems, and service offerings. This integration across sectors has led to a significant increase in computational load, especially in terminal devices that support AI-enabled functions [2]. Constraints such as manufacturing costs, limited battery capacity, and restricted processing capabilities prevent these devices from executing complex tasks efficiently.

With the advancement of high-speed internet and communication technologies, large volumes of data are generated by computation-heavy use cases such as augmented reality, online gaming, and video streaming [3]. This growth demands a platform that can effectively collect, manage, and process data from a growing number of IoT devices. An IoT device does not always have the computational resources to handle high-demand applications, both in terms of CPU power and memory size. Offloading these computations to more powerful devices becomes essential to meet communication and processing requirements [4].

The rise of Multi-Access Edge Computing (MEC), especially with 5G, has made it a key solution for offloading tasks from resource-limited IoT devices. By processing data closer to the source, MEC reduces latency and network congestion, which is crucial in low-bandwidth environments [5]. Unlike cloud computing, which introduces delays due to its centralized nature, MEC supports real-time, high-QoS applications more effectively [6]. However, MEC still faces challenges. Devices often depend on nearby edge servers due to limited local resources, and large-scale IoT deployments add complexity due to hardware and software heterogeneity [7]. Furthermore, the strict and varied resource needs of applications in fields like healthcare and smart agriculture make task migration more difficult.

Existing task offloading strategies using heuristic or rule-based methods often fail to adapt effectively to such complexity and real-time fluctuations in network and resource conditions. These approaches typically rely on static assumptions, which limit their scalability and responsiveness. In contrast, deep learning offers a data-driven solution that can model complex relationships between IoT task characteristics and available edge resources. However, there remains a lack of comprehensive approaches that apply deep learning models specifically to the joint optimization of task placement and execution decisions in MEC-enabled IoT systems. This paper proposes a deep learning-based offloading model that predicts the optimal execution point—local, edge, or cloud—for incoming IoT tasks based on system state and task attributes, including VM utilization, network delay, and workload. By continuously updating its decision policies according to the current environment, the model balances resource usage and improves performance under varying IoT loads.

The structure of this paper includes a review of related work (Section II), a detailed system model and problem formulation (Section III), the proposed deep learning-driven offloading strategy (Section IV), performance evaluation through simulation results (Section V), and a conclusion with future research perspectives (Section VI).

II. RELATED WORK

The Internet of Things (IoT) enables large-scale connectivity among heterogeneous devices [8], but limited computa-

The authors are with the LIMATI Laboratory, Department of Mathematics and Computer Science, Polydisciplinary Faculty, Sultan Moulay Slimane University, PO Box 592, Beni Mellal, 23000, Morocco. (E-mail: lagnfdi.o@gmail.com, marouane.myyara@usms.ac.ma, anouar.darif@gmail.com)

DOI: 10.36244/ICJ.2026.1.2

tional and energy resources require efficient task offloading. Mobile Edge Computing (MEC) brings computation closer to devices, reducing latency and improving Quality of Service (QoS). Surveys highlight the need for adaptive, scalable offloading strategies in edge and cloud environments [9].

Early task offloading solutions primarily relied on heuristic and rule-based approaches due to their low complexity. A representative example is the latency-classification-based deadline-aware (LCDA) task offloading algorithm proposed in [10], where tasks are classified as latency-sensitive or latency-tolerant based on deadlines. A weight-based formulation considering task size, execution time, and urgency is used to prioritize execution at edge servers. Similar heuristic-driven strategies focusing on reducing service time and improving QoS are presented in [11], [12]. Although efficient, these approaches depend on predefined rules and exhibit limited adaptability under dynamic IoT workloads. Several studies formulate task offloading as a mathematical optimization problem to achieve optimal decisions. In [13], a decentralized Lagrangian-based approach is proposed for online edge task scheduling, jointly optimizing latency, offloading cost, and resource utilization. Mixed-integer programming and greedy heuristics are adopted in [14] to maximize user service satisfaction in dense and delay-sensitive IoT environments. Joint optimization of computation offloading, software caching, and communication resources is investigated in [15], demonstrating notable latency and energy efficiency gains. Energy-aware optimization models are further explored in [16], [17], where power consumption and resource utilization are jointly optimized in cellular and NOMA-enabled IoT networks.

Game-theoretic approaches have been employed to model competition among IoT devices for limited edge resources. In [18], the task offloading problem is formulated as a non-cooperative game, where each device independently selects its offloading strategy to minimize latency. The existence of a Nash equilibrium enables decentralized decision-making without a central controller. However, such approaches may suffer from convergence delays and scalability issues as the number of devices increases. To address the NP-hard nature of task offloading, meta-heuristic and swarm intelligence methods have been widely adopted. Particle Swarm Optimization (PSO) is used in [19] to minimize execution delay and energy consumption in industrial IoT scenarios. Genetic algorithms are applied in [20] to optimize offloading decisions under bandwidth and computing constraints. Despite their effectiveness, these methods require iterative search processes, leading to high computational overhead and limited suitability for real-time deployment.

Control-theoretic frameworks have also been proposed to handle dynamic system behavior. Lyapunov-based optimization techniques are employed in [21], [22] to jointly manage task offloading and resource allocation while ensuring system stability and energy efficiency, particularly in UAV-enabled and energy-harvesting MEC systems. Although theoretically robust, these approaches often involve complex mathematical modeling, limiting practical implementation. Fuzzy-logic-based offloading mechanisms provide an alternative for handling uncertainty in MEC environments. In [23], a fuzzy workload orchestration framework is proposed to dynamically distribute workloads across edge nodes. A flexible fuzzy-based mobile edge orchestrator is further introduced in [24], enabling adaptive offloading decisions based on system context. Large-scale offloading challenges are also explored in [25], highlighting the need for scalable orchestration mechanisms.

Recent studies have explored hybrid and collaborative architectures to enhance MEC capabilities. Task offloading in cloud–edge collaboration environments is investigated in [6], while comprehensive discussions on challenges and future research directions are presented in [26]. These works emphasize the increasing complexity of modern MEC-enabled IoT systems.

Despite advances, most approaches rely on static heuristics or complex models, limiting scalability. Learning-based methods can predict offloading decisions from real-time system states. Motivated by this, we propose a neural network–based framework that uses task characteristics and VM utilization to enable adaptive, low-latency offloading and improved QoS.

III. SYSTEM MODEL AND PROBLEM FORMULATION

Figure 1 illustrates a multi-tier Mobile Edge Computing (MEC) architecture supporting a set of Internet of Things (IoT) devices. Let $\mathcal{I}$ denote the set of devices, where each device $i \in \mathcal{I}$ generates a set of tasks $\{\tau_{ij}\}$, with j indexing the tasks. Due to limited computational and energy resources, each task τ_{ij} must be offloaded to one of three execution tiers: (i) *local edge servers*, co-located with IoT devices and offering low latency but limited computing power; (ii) *remote edge servers*, distributed MEC nodes accessible via the network with moderate latency; or (iii) *cloud servers*, centralized data centers providing abundant computing resources at higher latency. Each task is executed entirely at a single tier, as determined by the offloading decision.

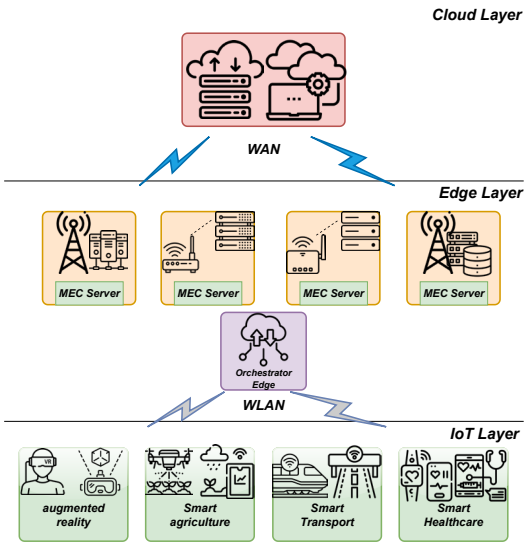

Fig. 1. An Overview of Multi-Access Edge Computing Systems

A. Base Notation

We consider a multi-tier Mobile Edge Computing (MEC) environment with multiple IoT devices, where each device generates tasks τ_{ij} characterized by computational workload L_{ij} (MI), input size D_{ij} (MB), maximum delay $T_{ij}^{\max}$ (ms), and memory requirement R_{ij}^{req} (MB). Tasks can be offloaded to three execution tiers: local edge servers ($k = 0$) close to devices with low delay but limited resources, remote edge servers ($k = 1$) with moderate latency, and cloud servers ($k = 2$) with high computational capacity but larger latency. Offloading decisions are denoted $x_{ij} \in \{0, 1, 2\}$. Each tier k is described by processing capacity f_k, bandwidth BW_k, maximum CPU $CPU_k^{\max}$, available memory R_k^{avail}, and current utilization U_k^{current}, with available capacity $f_k^{\text{avail}} = f_k(1 - U_k^{\text{current}})$.

B. Task and Offloading Model

The total execution time of a task τ_{ij} generated by device i depends on the selected offloading tier $k \in \{0, 1, 2\}$ (local edge, remote edge, or cloud) and comprises the service time and queuing delay:

$$T_{ij}^{\text{serv}} = T_{ij}^{\text{exec}} + T_{ij}^{\text{queue}}.$$

The service time T_{ij}^{serv} captures both computation and communication delays, influenced by processing capacity, bandwidth, and current utilization. For local execution ($k = 0$), no network transfer is needed, and the service time reduces to

$$T_{ij}^{\text{exec}} = \frac{L_{ij}}{f_0(1 - U_0^{\text{current}})}.$$

For remote edge ($k = 1$) and cloud ($k = 2$) execution, it includes both processing and network transmission delays:

$$T_{ij}^{\text{exec}} = \frac{L_{ij}}{f_k(1 - U_k^{\text{current}})} + \frac{D_{ij}^{\text{up}} + D_{ij}^{\text{down}}}{BW_k^{\text{avail}}},$$

where L_{ij} is the computational workload, f_k the processing speed, U_k^{current} the CPU utilization, D_{ij}^{up} and D_{ij}^{down} the input/output data sizes, and BW_k^{avail} the available bandwidth. While T_{ij}^{total} accounts for queuing delays.

C. Problem Formulation

The objective of the optimization is to **minimize the average service time** for all tasks generated by the IoT devices. Each task τ_{ij} can be processed locally, at a remote edge, or in the cloud, depending on the offloading decision $x_{ij} \in \{0, 1, 2\}$. The total service time includes both computation and transmission delays:

$$\min \frac{1}{N} \sum_{i,j} T_{ij}^{\text{serv}},$$

where N is the total number of tasks.

Subject to:

$$x_{ij} \in \{0, 1, 2\} \quad \forall \tau_{ij} \tag{C1}$$

$$T_{ij}^{\text{serv}} = \frac{L_{ij}}{f_{x_{ij}}(1 - U_{x_{ij}}^{\text{current}})} + \delta_{x_{ij}} \leq T_{ij}^{\max} \quad \forall \tau_{ij} \tag{C2}$$

$$U_{x_{ij}}^{\text{current}} + \frac{L_{ij}}{f_{x_{ij}} T_{ij}^{\max}} \leq CPU_{x_{ij}}^{\max} \quad \forall \tau_{ij} \tag{C3}$$

$$R_{x_{ij}}^{\text{avail}} \geq R_{ij}^{\text{req}} \quad \forall \tau_{ij} \tag{C4}$$

Here, $\delta_{x_{ij}} = 0$ for local execution ($x_{ij} = 0$) and $\delta_{x_{ij}} = \frac{D_{ij}^{\text{up}} + D_{ij}^{\text{down}}}{BW_{x_{ij}}^{\text{avail}}}$ for edge/cloud ($x_{ij} \in 1, 2$). Tasks are assigned sequentially in batches, updating system state ($U_k^{\text{current}}, R_k^{\text{avail}}$) after each task. Only offloading choices x_{ij} are decision variables. Constraints (C1)–(C4) enforce valid offloading, respect deadlines, and ensure CPU and memory capacities are not exceeded, allowing online allocation decisions to reflect the most recent resource availability. This design enforces sequential, state-aware allocation and aligns with the Problem Formulation. Overall, it minimizes the average service time across all tasks.

IV. Proposed Deep Learning-Driven Offloading Strategy

Figure 2 illustrates a Deep Learning (DL)-based offloading framework for a three-tier IoT–MEC–Cloud architecture. The system employs Neural Networks (NN) to optimize task allocation using real-time parameters such as link quality, task complexity, and latency requirements. The DL model dynamically selects the optimal execution tier—local edge, remote edge, or cloud—to **minimize service time**. By processing tasks in batches, the neural network efficiently handles resource contention and multi-user scheduling, ensuring high responsiveness and scalability for time-sensitive IoT applications, including autonomous systems and smart healthcare, while maintaining balanced load distribution across the infrastructure.

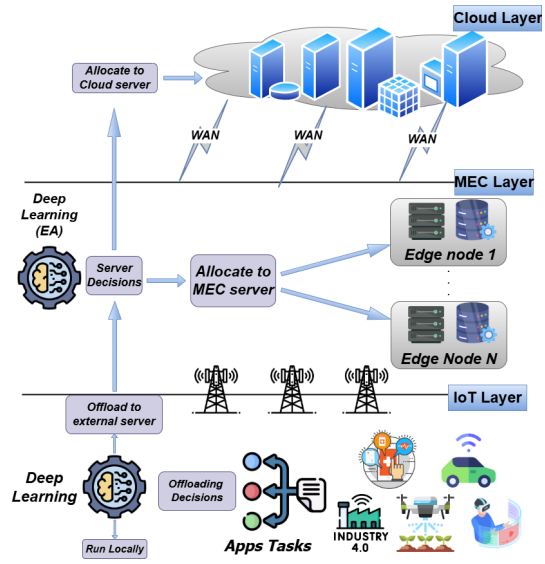

Fig. 2. An Illustration of Computation Offloading type in MEC networks

A. Neural Network Training Methodology

The task offloading neural network (NN) is trained in a supervised manner using datasets generated from diverse MEC simulation scenarios, varying the number of IoT devices, task arrival rates, CPU utilization, and network bandwidth. Each instance includes task-specific features (workload L_{ij}, input/output sizes $D_{ij}^{\text{up}}, D_{ij}^{\text{down}}$, delay sensitivity T_s) and system-level features (current VM utilization U_k^{current}, available memory R_k^{avail}, bandwidth BW_k^{avail}).

Training labels are generated using expert-defined rules:

- **Local Edge Assignment:** Tasks with high delay sensitivity ($T_s > 0.7$) and small workload ($L_{ij} < 10$ GI) are assigned locally if $U_0^{\text{current}} < 0.6$.
- **Remote Edge Assignment:** Tasks with moderate delay sensitivity ($0.3 \leq T_s \leq 0.7$) and medium workload ($10 \leq L_{ij} \leq 30$ GI) are assigned to the remote edge when $BW_1^{\text{avail}} > 50$ Mbps.
- **Cloud Assignment:** Tasks with low delay sensitivity ($T_s < 0.3$) or large workload ($L_{ij} > 30$ GI) are assigned to the cloud.

These rules ensure feasible and diverse training examples for the NN, capturing both task characteristics and dynamic system states. This process enables the NN to learn realistic offloading decisions for heterogeneous MEC scenarios.

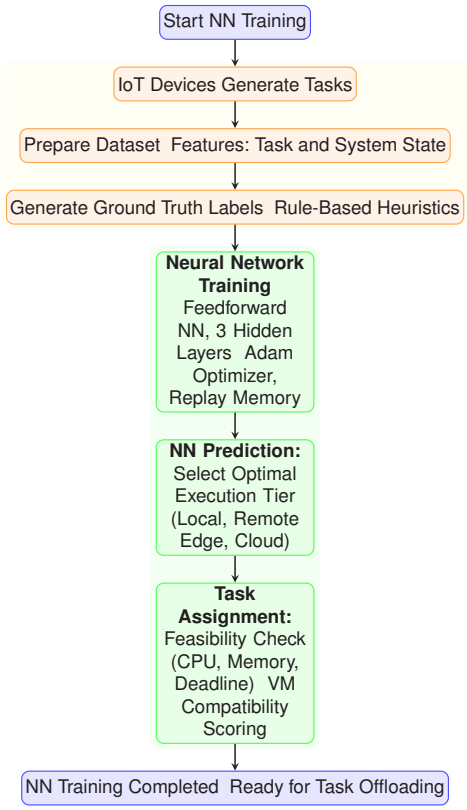

Fig. 3. Neural network-based IoT task offloading process in MEC, from dataset preparation to tier selection and task assignment.

The NN-based tier selection and VM assignment uses a feedforward network with three hidden layers (64, 32, 16 neurons, ReLU) and a softmax output for tier prediction. It is trained with Adam (learning rate 0.001), batch size 32, 10,000 samples, 100 episodes, 3,000 steps, discount 0.9, and soft replacement 0.01.

Algorithm 1 presents a hierarchical deep learning framework for tier selection and task-to-VM assignment in multi-tier MEC systems. Tasks are processed sequentially by delay sensitivity. For each task, the TierSelectionNN predicts the most suitable execution tier using task-specific features (workload, delay constraints, input/output sizes) and system-level parameters (available CPU, memory, bandwidth, and utilization). Within the selected tier, candidate VMs are scored by the VMCompatibilityNN to assess task–VM compatibility. The task is assigned to the highest-ranked feasible VM that meets CPU, memory, and delay constraints; otherwise, it is deferred to an overflow queue. The system state is updated after each assignment, ensuring efficient, feasible task placement and optimal resource utilization across heterogeneous MEC tiers.

Algorithm 1 DL-Based Tier selection and VM assignment

Require: Task batch τ_{batch}, system state S, TierSelectionNN, VMCompatibilityNN
Ensure: Task-to-(tier, VM) assignments and overflow queue

```
Sort τ_batch by delay sensitivity (ascending)
Initialize temporary state S_temp ← S
Initialize assignments ← ∅, overflow ← ∅
for each task τ in τ_batch do
    Extract task features F_τ and system features F_S
    Predict tier scores p ← TierSelectionNN(F_τ, F_S)
    Rank tiers by descending p
    assigned ← false
    for each tier k in ranked tiers do
        if tier k is not feasible then
            continue
        end if
        Extract VM set V_k in tier k
        Compute compatibility scores for all v ∈ V_k using VMCompatibilityNN
        Rank V_k by descending compatibility score
        for each VM v in ranked V_k do
            if v satisfies resource and deadline constraints then
                Assign τ → (k, v)
                Update S_temp
                assigned ← true
                break
            end if
        end for
        if assigned then
            break
        end if
    end for
    if not assigned then
        Add τ to overflow queue
    end if
end for
return assignments, overflow
```

V. PERFORMANCE EVALUATION

This section evaluates the performance of the proposed deep learning offloading framework in a MEC environment, modeling IoT devices, edge servers, and cloud infrastructure to assess scalability, responsiveness, and robustness under varying workloads.

A. Simulation Setup

The offloading strategy was evaluated using a Python 3.7 simulation framework with TensorFlow 2.6 and Tkinter 8.6 on Windows 10. Simulations ran on a PC with a 1.9 GHz Intel i5-8365U CPU and 16 GB RAM, emulating MEC-based task offloading with variable network, computation, and task conditions.

The simulated MEC system comprises 200–2000 IoT devices, each generating tasks at random intervals of 0.3–0.7 s. Task types—*Heavy*, *Infotainment*, *AR/VR*, and *Health*—represent diverse real-world applications, including video analytics, media streaming, augmented reality, and remote healthcare. Task parameters, summarized in Table I, are adapted from EdgeCloudSim to ensure realistic workload sizes, latency sensitivity, and data volumes.

TABLE I
APPLICATION CHARACTERISTICS

Property	Heavy	Infotainment	AR/VR	Health
Task Length (GI)	45	15	9	3
Delay Sensitivity (T_s)	0.1	0.3	0.9	0.7
Upload Data (KB)	2500	25	1500	20
Download Data (KB)	200	1000	25	1250

The infrastructure includes three MEC hosts (8 VMs each) and one cloud host (4 VMs), with VMs provisioned with fixed CPU cores and processing capacities (Table II). Network bandwidth varies between 10–100 Mbps to emulate real-time fluctuations. Each task is processed sequentially: the neural networks predict the optimal execution tier and score candidate VMs, and resource constraints (CPU, memory, and bandwidth) are updated dynamically and provided as input to the neural offloading model.

TABLE II
INFRASTRUCTURE PARAMETERS

Parameter	MEC	Cloud
Number of Hosts	3	1
VMs per Host	8	4
Cores per VM	2	4
VM CPU Speed (MIPS)	10,000	20,000
VM Storage (MB)	50,000	100,000

B. Simulation Results

This section evaluates and compares the performance of five resource management strategies—Neural Network, Fuzzy Logic, Utilization-Based, Sonmez, and Flores—across key quality of service (QoS) metrics under varying IoT device loads. In the Neural Network-based model, task execution is selected among edge (MEC) and cloud resources based on system state, task requirements, and network conditions.

- **Utilization-Based Approach** [24]: Tasks are offloaded to the least-loaded server to balance edge resources and prevent overutilization, but application-specific delays and communication demands are not considered.
- **Flores Approach** [25]: Uses fuzzy logic for offloading without considering resource consumption, which can overload VMs and increase latency under peak loads.
- **Sonmez Approach** [23]: Fuzzy logic-based offloading evaluates application features and resource use but assumes homogeneous resources, limiting adaptability in heterogeneous environments.
- **Fuzzy Logic Approach** [12]: Assigns tasks based on multiple criteria, including VM utilization, task length, network demand, and delay sensitivity, adapting to resource heterogeneity to reduce failures and improve service time.

The Neural Network-based algorithm considers task delay sensitivity, workload, and system resource utilization to optimize offloading and VM assignment. Compared to existing methods, it provides more efficient resource allocation and improved QoS for IoT applications.

C. Modeling of Service Time

Figure 4 illustrates the variation in service time as the number of IoT devices increases. At 200 devices, the Neural Network achieves the lowest service time of 0.738 s, outperforming Fuzzy Logic (1.5 s) and the Utilization-Based method (1.8 s). At 1000 and 2000 devices, it maintains 0.848 s and 0.963 s, respectively, while Flores degrades to 7.5 s and 12 s. Sonmez and Utilization-Based methods also increase to 4.0 s at 2000 devices. The Neural Network's consistent performance arises from its ability to learn from historical system data and dynamically predict optimal execution sites. By prioritizing low-latency tasks for local or MEC nodes, it minimizes wait times and queue lengths. Unlike Flores and Utilization-Based methods, it adapts in real-time to avoid overloaded nodes, ensuring lower and more stable service times across varying workloads.

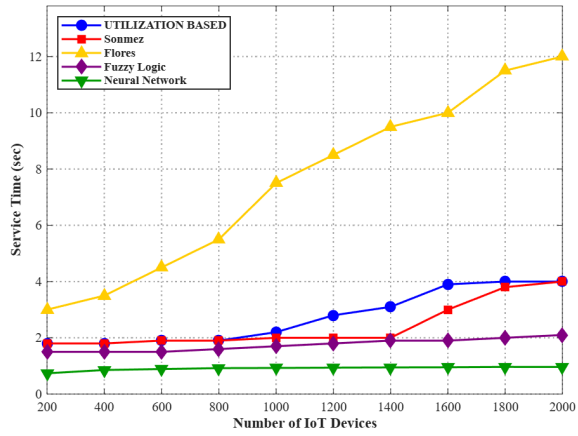

Fig. 4. The service time of all approaches VS the number of IoT devices

1) Modeling of Network Delay: Figure 5 illustrates the effect of increasing IoT devices on average network delay. For the Neural Network approach, the reported delay includes both transmission and queuing time, reflecting end-to-end latency. At 200 devices, all methods maintain delays below 0.21 ms. As device density increases, performance gaps widen: the Neural Network achieves 0.262 ms at 2000 devices, outperforming Fuzzy Logic (0.32 ms), Sonmez (0.35 ms), and Flores (0.35 ms), while the Utilization-Based approach remains stable at 0.20 ms. The Neural Network's advantage stems from its ability to learn real-time network conditions and predict optimal task placement. By recognizing congestion patterns, it dynamically offloads tasks closer to data sources, reducing transmission and queuing delays. Unlike static or rule-based methods, it adapts to system variations, ensuring consistently low latency and scalability across varying workloads.

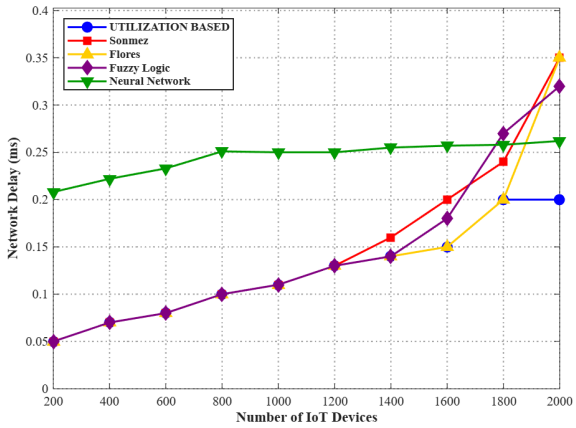

Fig. 5. The network delay of all approaches VS the number of IoT devices

2) Modeling of Processing Time: As illustrated in Figure 6, the Neural Network consistently achieves the shortest processing durations across all IoT device densities. At 200 devices, it records 0.530 s, outperforming Fuzzy Logic (0.6 s) and the Utilization-Based method (0.8 s).

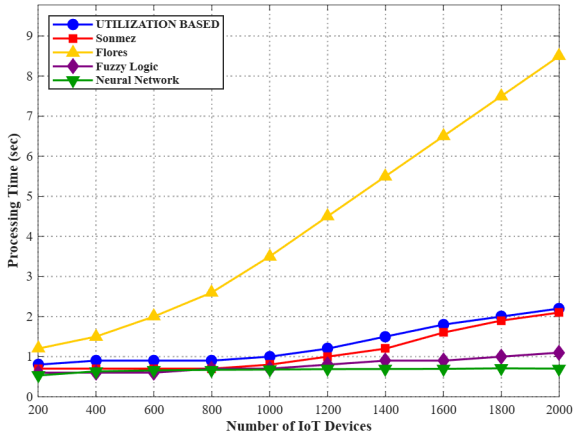

Fig. 6. The processing time of all approaches VS the number of IoT devices

The advantage becomes more pronounced at higher scales: at 2000 devices, the Neural Network maintains 0.701 s, while Flores escalates dramatically to 8.5 s, and Sonmez and the Utilization-Based method reach 2.1 s and 2.2 s, respectively. This performance stems from dynamic task allocation based on both node capacity and task complexity. By assigning tasks to nodes with optimal resources and processing capabilities, the system prevents CPU bottlenecks and reduces execution delays. Predictive analytics anticipate resource contention, allowing proactive redistribution of workloads before performance degradation occurs. Unlike static approaches such as Flores and Sonmez, which lack context-aware distribution, the Neural Network adapts in real time, balancing load effectively and ensuring efficient resource utilization even under high-demand scenarios, resulting in consistently low and stable processing times.

3) Modeling of Task Failure Rate: Figure 7 illustrates task failure rates across varying IoT device counts. The Neural Network consistently achieves the lowest rates, with 0.3% at 200–800 devices, 1.0% at 1200–1400, and 2.0% at 2000 devices. By comparison, Sonmez, Utilization-Based, and Flores reach 27%, 24%, and 22%, while Fuzzy Logic records 19%. This highlights the Neural Network's robust load-handling and fault-tolerance. It predicts potential overloads and avoids assigning tasks to resource-constrained nodes. Dynamic task reallocation using real-time feedback further reduces failures, preventing node saturation. Fixed-rule approaches lack such adaptability, often overcommitting resources and triggering cascading failures. The Neural Network maintains high reliability even as device density and system stress increase, making it particularly suitable for mission-critical IoT deployments that require consistent task completion and low failure rates.

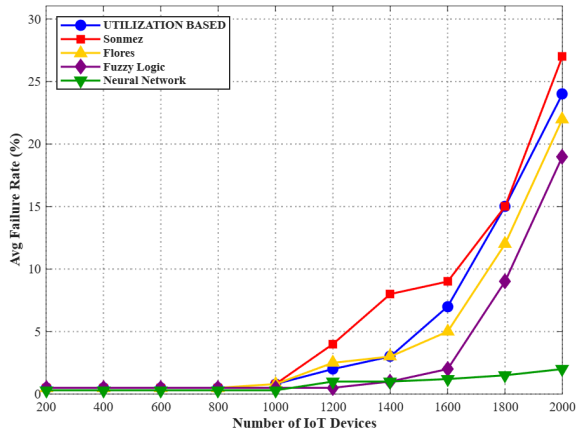

Fig. 7. Task Failure Rate of All Approaches vs. Number of IoT Devices

4) Modeling of VM Resource Utilization: Figure 8 shows VM CPU utilization patterns as IoT devices increase. At 200 devices, all methods exhibit low utilization (¡2.5%). At 2000 devices, the Neural Network remains efficient at 8.5%, whereas Sonmez reaches 97%, Flores stabilizes at 42%, Fuzzy Logic at 22%, and the Utilization-Based approach at 60%.

The Neural Network achieves this efficiency by learning from historical VM performance and using intelligent task allocation to balance loads. Reinforcement learning optimizes

resource use while avoiding VM overload. In contrast, static methods like Sonmez and Flores do not account for holistic VM usage or long-term resource trends, causing uneven distribution and bottlenecks. The Neural Network maintains balanced utilization across servers, supporting higher workloads and demonstrating scalability and efficiency for large-scale IoT deployments.

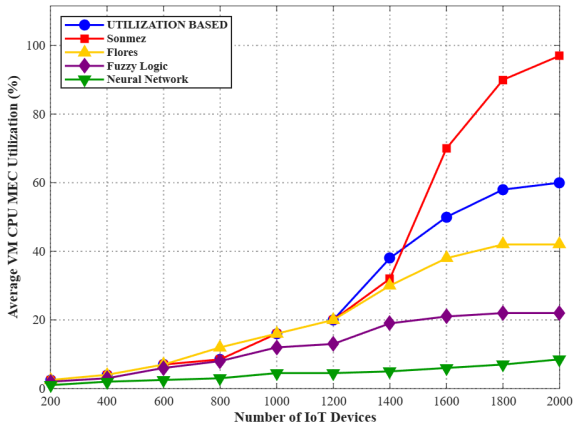

Fig. 8. Average CPU Utilization of MEC VMs of all approaches VS the number of IoT devices

Discussion

The experimental results demonstrate that the proposed Neural Network-based approach markedly outperforms traditional task offloading methods in Multi-access Edge Computing (MEC) environments for IoT applications. Across all evaluated metrics—including service time, network delay, processing time, task failure rate, and VM resource utilization—the Neural Network consistently achieves superior performance compared to Fuzzy Logic, Sonmez, Utilization-Based, and Flores approaches. Leveraging a learning-based architecture, the framework dynamically identifies the optimal execution location—local, edge, or cloud—based on real-time system conditions. This adaptability translates into reduced service and processing times, minimal network delays, and lower task failure rates under diverse workloads and device densities. In contrast to Flores, which exhibits limited scalability under high IoT loads, the Neural Network maintains robustness through intelligent VM management and real-time load balancing, achieving efficient resource utilization while sustaining high task completion rates.

VI. CONCLUSION

This study introduces a comprehensive neural network-based task offloading framework for Multi-access Edge Computing (MEC), addressing computational, networking, and scheduling challenges in large-scale IoT deployments. The proposed approach consistently outperforms traditional methods, including Fuzzy Logic and Utilization-Based strategies, by improving service times, task completion rates, and VM resource utilization. Its adaptive architecture dynamically manages fluctuating IoT workloads and heterogeneous application requirements, effectively balancing CPU, network, and latency demands while maintaining strong scalability. Future research will focus on predictive resource management, integrating workload forecasting and demand-aware scheduling for resource-intensive applications such as augmented reality, real-time gaming, and industrial IoT. Emphasis will also be placed on energy optimization for battery-powered devices and on integrating the framework into real-world IoT deployments. To validate simulation fidelity, we plan to reproduce baseline scenarios in Simu5G and EdgeCloudsim for per-task service time, network delay, and VM utilization comparisons, and to run small-scale Akraino IoT deployments to measure end-to-end latency and task completion. By combining intelligent learning mechanisms, predictive orchestration, and real-world validation, this framework aims to establish a foundation for efficient, reliable, scalable, and context-aware next-generation MEC-based IoT systems.

REFERENCES

[1] G. Paolone, D. Iachetti, R. Paesani, F. Pilotti, M. Marinelli, and P. Di Felice, "A holistic overview of the internet of things ecosystem," *IoT*, vol. 3, no. 4, pp. 398–434, 2022, DOI: 10.3390/iot3040022.

[2] B. Gupta and M. Quamara, "An overview of internet of things (IoT): Architectural aspects, challenges, and protocols," *Concurrency Computat.: Pract. Exper.*, vol. 32, no. 21, p. e4946, 2020, DOI: 10.1002/cpe.4946.

[3] Q. Chen, Z. Guo, W. Meng, S. Han, C. Li, and T. Q. Quek, "A survey on resource management in joint communication and computing-embedded SAGIN," *IEEE Commun. Surv. Tutor.*, 2024, DOI: 10.1109/COMST.2024.3421523.

[4] K. Bakar, F. Zuhra, B. Isyaku, and S. Sulaiman, "A review on the immediate advancement of the internet of things in wireless telecommunications," *IEEE Access*, vol. 11, pp. 21 020–21 048, 2023.

[5] X. Jin, W. Hua, Z. Wang, and Y. Chen, "A survey of research on computation offloading in mobile cloud computing," *Wirel. Netw.*, vol. 28, no. 4, pp. 1563–1585, 2022, DOI: 10.1007/s11276-021-02702-6.

[6] C. Wang, R. Guo, H. Yu, Y. Hu, C. Liu, and C. Deng, "Task offloading in cloud-edge collaboration-based cyber physical machine tool," *Robot. Comput.-Integr. Manuf.*, vol. 79, p. 102 439, 2023, DOI: 10.1016/j.rcim.2022.102439.

[7] S. Azizi, M. Othman, and H. Khamfroush, "Deco: A deadline-aware and energy-efficient algorithm for task offloading in mobile edge computing," *IEEE Syst. J.*, vol. 17, no. 1, pp. 952–963, 2023, DOI: 10.1109/JSYST.2022.3215789.

[8] A. Khanna and S. Kaur, "Internet of things (IoT), applications and challenges: A comprehensive review," *Wirel. Pers. Commun.*, vol. 114, no. 2, pp. 1687–1762, 2020, DOI: 10.1007/s11277-020-07363-y.

[9] K. Lone and S. A. Sofi, "A review on offloading in fog-based Internet of Things: Architecture, machine learning approaches, and open issues," *High-Confidence Comput.*, vol. 3, no. 2, p. 100 124, 2023, DOI: 10.1016/j.hcc.2023.100124.

[10] H. Choi, H. Yu, and E. Lee, "Latency-classification-based deadline-aware task offloading algorithm in mobile edge computing environments," *Appl. Sci.*, vol. 9, no. 21, p. 4696, 2019, DOI: 10.3390/app9214696.

[11] M. Myyara, O. Lagnfdi, A. Darif, and A. Farchane, "Enhancing QoS for IoT devices through heuristics-based computation offloading in multi-access edge computing," *Infocommun. J.*, vol. 16, no. 4, 2024, DOI: 10.36244/ICJ.2024.4.2.

[12] J. Almutairi and M. Aldossary, "A novel approach for IoT tasks offloading in edge-cloud environments," *J. Cloud Comput.*, vol. 10, p. 28, 2021, DOI: 10.1186/s13677-021-00241-3.

[13] Q. Peng, C. Wu, Y. Xia, Y. Ma, X. Wang, and N. Jiang, "Dosra: A decentralized approach to online edge task scheduling and resource allocation," *IEEE Internet Things J*., vol. 9, no. 6, pp. 4677–4692, 2022, DOI: 10.1109/JIOT.2022.3152310.

[14] J. Li, "Maximizing user service satisfaction for delay-sensitive IoT applications in edge computing," *IEEE Trans. Parallel Distrib. Syst*., vol. 33, no. 5, pp. 1199–1212, 2022, DOI: 10.1109/TPDS.2021.3107137.

[15] W. Wen, Y. Cui, T. Q. S. Quek, F.-C. Zheng, and S. Jin, "Joint optimal software caching, computation offloading and communications resource allocation for mobile edge computing," *IEEE Trans. Veh. Technol*., vol. 69, no. 7, pp. 7879–7894, 2020, DOI: 10.1109/TVT.2020.2993359.

[16] Y. Cheng, H. Zhao, and W. Xia, "Energy-aware offloading and power optimization in full-duplex mobile edge computing-enabled cellular IoT networks," *IEEE Sens. J*., vol. 22, no. 24, pp. 24 607–24 618, 2022, DOI: 10.1109/JSEN.2022.3218584.

[17] B. Liu, C. Liu, and M. Peng, "Resource allocation for energy- efficient MEC in NOMA-enabled massive IoT networks," *IEEE J. Sel. Areas Commun*., vol. 39, no. 4, pp. 1015–1027, 2021, DOI: 10.1109/JSAC.2020.3018809.

[18] J. Luo, Q. Qian, L. Yin, and Y. Yiao, "A game-theoretical ap- proach for task offloading in edge computing," in *Proc. 16th Int. Conf. Mobility, Sens. Netw. (MSN)*, 2020, pp. 756–761, DOI: 10.1109/MSN50589.2020.00129.

[19] Q. You and B. Tang, "Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things," *J. Cloud Comput*., vol. 10, no. 1, p. 41, 2021, DOI: 10.1186/s13677-021-00212-8.

[20] M. Myyara, O. Lagnfdi, A. Darif, and A. Farchane, "A new approach based on genetic algorithm for computation offloading optimization in multi-access edge computing networks," *IAES Int. J. Artif. Intell*., vol. 13, no. 4, pp. 4186–4194, 2024.

[21] Z. Yu, Y. Gong, S. Gong, and Y. Guo, "Joint task offloading and resource allocation in UAV-enabled mobile edge computing," *IEEE Internet Things J*., vol. 7, no. 4, pp. 3147–3159, 2020, DOI: 10.1109/JIOT.2020.2965898.

[22] X. Song, Q. Ma, G. Zheng, L. Li, P. Cong, and J. Zhou, "Dynamic task offloading and resource allocation for energy-harvesting end-edge- cloud computing systems," *J. Syst. Archit.,* p. 103 469, 2025, DOI: 10.1016/j.sysarc.2025.103469.

[23] C. Sonmez, A. Ozgovde, and C. Ersoy, "Fuzzy workload orchestration for edge computing," *IEEE Trans. Netw. Serv. Manag*., vol. 16, no. 2, pp. 769–782, 2019, DOI: 10.1109/TNSM.2019.2911615.

[24] V. Nguyen, T. T. Khanh, T. D. Nguyen, C. S. Hong, and E. N. Huh, "Flexible computation offloading in a fuzzy-based mobile edge orchestrator for IoT applications," *J. Cloud Comput*., vol. 9, no. 1, pp. 1–18, 2020, DOI: 10.1186/s13677-020-0170-8.

[25] H. Flores, X. Su, V. Kostakos, A. Y. Ding, P. Nurmi, S. Tarkoma, P. Hui, and Y. Li, "Large-scale off loading in the internet of things," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, 2017, pp. 479–484, DOI: 10.1109/PERCOMW.2017.7917638.

[26] M. Akhlaqi and Z. M. Hanapi, "Task offloading paradigm in mobile edge computing: Current issues, adopted approaches, and future directions," *J. Netw. Comput. Appl*., vol. 212, p. 103 568, 2023, DOI: 10.1016/j.jnca.2022.103568.

Oussama Lagnfdi received his B.Sc. in Physical Matter Science in 2020 and M.Sc. in Telecommunications Systems and Computer Networks in 2022 from Sultan Moulay Slimane University, Beni Mellal, Morocco. Currently, he is a Ph.D. candidate at the Laboratoire d'Innovation en Mathématiques et Applications et Technologies de l'Information (LIMATI), Polydisciplinary Faculty, Sultan Moulay Slimane University, Morocco. His ongoing research is focused on enhancing the performance of Internet of Things (IoT) and Mobile Edge Computing (MEC), Artificial Intelligence, Deep Learning, and Fuzzy Logic.

Marouane Myyara received his B.Sc. in Electronic and Telecommunication Engineering in 2019 and M.Sc. in Telecommunication Systems and Computer Networks in 2021 from Sultan Moulay Slimane University, Beni Mellal, Morocco. He is currently a Ph.D. candidate at the Laboratory of Innovation in Mathematics, Applications, and Information Technology (LIMATI), Polydisciplinary Faculty, Sultan Moulay Slimane University, Morocco. His current research focuses on improving the performance of Multi-access Edge Computing networks (MEC), Cloud Computing, Computation Offloading, and the Internet of Things (IoT).

Anouar Darif received the bachelor in IEEA (Informatique Électrotechnique, Électronique and Automatique) from Dhar El Mahraz Faculty of Sciences at Mohamed Ben Abdellah University Fez, Morocco in 2005. He received the Diplôme d'Etudes Supérieures Approfondies in Computer Sciences and Telecommunications from the Faculty of Sciences Rabat in 2007. He received the Ph.D. degree in Computer Sciences and Telecommunications from the Faculty of Sciences of Rabat in 2015. He is currently a Research and Teaching Associate in the Multidisciplinary Faculty at the University of Sultan Moulay Slimane Beni Mellal, Morocco. His research interests include Wireless Sensor Networks (WSN), Mobile Edge Computing (MEC), Internet of Things (IoT), Cloud Computing, and Neural Networks.