sensors

Article

Federated Reinforcement Learning-Based Dynamic Resource Allocation and Task Scheduling in Edge for IoT Applications

Saroj Mali [1], Feng Zeng [1,*], Deepak Adhikari [2], Inam Ullah [3,*], Mahmoud Ahmad Al-Khasawneh [4,5], Osama Alfarraj [6] and Fahad Alblehai [6]

[1] School of Computer Science and Engineering, Central South University, Changsha 410083, China; malisaroj@csu.edu.cn
[2] School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China; deepakadhikari@uestc.edu.cn
[3] Department of Computer Engineering, Gachon University, Seongnam 13120, Republic of Korea
[4] Hourani Center for Applied Science Research Center, Al-Ahliyya Amman University, Amman 19328, Jordan; mahmoudalkhasawneh@outlook.com
[5] School of Computing, Skyline University College, University City Sharjah, Sharjah 1797, United Arab Emirates
[6] Computer Science Department, Community College, King Saud University, Riyadh 11437, Saudi Arabia; oalfarraj@ksu.edu.sa (O.A.); falblehi@ksu.edu.sa (F.A.)
* Correspondence: fengzeng@csu.edu.cn (F.Z.); inam@gachon.ac.kr (I.U.)

Abstract: Using Google cluster traces, the research presents a task offloading algorithm and a hybrid forecasting model that unites Bidirectional Long Short-Term Memory (BiLSTM) with Gated Recurrent Unit (GRU) layers along an attention mechanism. This model predicts resource usage for flexible task scheduling in Internet of Things (IoT) applications based on edge computing. The suggested algorithm improves task distribution to boost performance and reduce energy consumption. The system's design includes collecting data, fusing and preparing it for use, training models, and performing simulations with EdgeSimPy. Experimental outcomes show that the method we suggest is better than those used in best-fit, first-fit, and worst-fit basic algorithms. It maintains power stability usage among edge servers while surpassing old-fashioned heuristic techniques. Moreover, we also propose the Deep Deterministic Policy Gradient (D4PG) based on a Federated Learning algorithm for adjusting the participation of dynamic user equipment (UE) according to resource availability and data distribution. This algorithm is compared to DQN, DDQN, Dueling DQN, and Dueling DDQN models using Non-IID EMNIST, IID EMNIST datasets, and with the Crop Prediction dataset. Results indicate that the proposed D4PG method achieves superior performance, with an accuracy of 92.86% on the Crop Prediction dataset, outperforming alternative models. On the Non-IID EMNIST dataset, the proposed approach achieves an F1-score of 0.9192, demonstrating better efficiency and fairness in model updates while preserving privacy. Similarly, on the IID EMNIST dataset, the proposed D4PG model attains an F1-score of 0.82 and an accuracy of 82%, surpassing other Reinforcement Learning-based approaches. Additionally, for edge server power consumption, the hybrid offloading algorithm reduces fluctuations compared to existing methods, ensuring more stable energy usage across edge nodes. This corroborates that the proposed method can preserve privacy by handling issues related to fairness in model updates and improving efficiency better than state-of-the-art alternatives.

Keywords: federated reinforcement learning; federated learning; reinforcement learning; edge computing; internet of things

check for updates

Academic Editor: Ashraf Uddin

Received: 21 January 2025
Revised: 25 March 2025
Accepted: 27 March 2025
Published: 30 March 2025

Citation: Mali, S.; Zeng, F.; Adhikari, D.; Ullah, I.; Al-Khasawneh, M.A.; Alfarraj, O.; Alblehai, F. Federated Reinforcement Learning-Based Dynamic Resource Allocation and Task Scheduling in Edge for IoT Applications. *Sensors* **2025**, *25*, 2197. https://doi.org/10.3390/s25072197

1. Introduction

The Internet of Things (IoT) is connected with various sensing devices that generate a deluge of sensing data from various IoT applications, such as agriculture and food industries. IoT applications in agriculture as a whole rely on interconnected devices such as sensor networks, drones, and automated machines. These devices generate an enormous volume of heterogeneous data from diverse sources, including soil moisture sensors, weather stations, satellite imagery, and farm machinery. However, these data are often redundant, noisy, or inconsistent, limiting their utility for real-time decision-making [1,2]. To address these challenges, multi-source information fusion has become a critical methodology for integrating, processing, and analyzing data from various sources to provide comprehensive, accurate, and actionable insights.

Multi-source information fusion has become a critical methodology for integrating, processing, and analyzing heterogeneous data from diverse IoT applications, enabling enhanced accuracy, robustness, and extended applications in real-time decision-making for agriculture. For example, fusion techniques such as Bayesian inference, neural networks, and fuzzy logic have shown significant improvements in optimizing resource allocation and scheduling tasks [3]. The integration of information fusion and edge computing further minimizes energy consumption and latency while maintaining data privacy, transforming traditional Agri-IoT systems into efficient, resilient, and sustainable frameworks [4].

Information fusion in IoT applications enables three key improvements:

1. Enhanced Accuracy: By merging information from varied origins, fusion decreases the impact of mistakes or disturbances from single data flows. For instance, combining soil moisture details with weather predictions and drone images grants a more exact interpretation of crop water necessities.
2. Improved robustness: The combination of information reduces reliance on just one data source. This makes sure that the system continues to work correctly and can be trusted, even if some sensors do not operate or generate incorrect data. Having this strength is very important in agricultural settings that are always changing and cannot be predicted easily.
3. Extended applications: Through the combination of different datasets, information fusion opens up new potentials. These include the detection of pests in real time, exact irrigation methods, and improved schedules for harvesting.

These improvements in combining information directly impact critical operations in agriculture IoT like resource allocation and scheduling tasks. Resource allocation makes sure that the computational resources, such as processing units, storage, and memory, are used well. For example, bringing together merged data from drones and sensors placed on the ground assists in dividing necessities like water or fertilizer exactly at the place and time when they are required. In the same way, organizing tasks includes managing farming actions like planting, watering crops, controlling pests, and gathering crops based on actual conditions. Through studying combined data, intelligent planning systems can improve these activities. Aspects such as soil state, stages of crop growth or development, availability of machinery, and forecasting about weather are considered.

The use of information fusion is very important to deal with the natural inefficiencies and privacy issues that are present in centralized Agri-IoT systems. These centralized systems frequently have delays, consume a lot of energy, and can be weak against data breaches—all these problems seriously affect the success of smart farming efforts. To extend beyond these barriers, this study brings together edge computing and information fusion along with machine learning algorithms to improve the performance, security, and sustainability of IoT systems. Edge computing handles data near its origin, reducing latency, energy consumption, and the risk of data leakage during transmission. Together with information fusion, edge computing makes it possible to conduct analysis and make decisions in real time, even in environments with limited connectivity.

IoT devices generate tremendous amounts of data that need significant computing power for analysis purposes—this kind of effective resource allocation is crucial in maintaining operational efficiency and cost-effectiveness, adapting to fluctuating environmental conditions and market dynamics to enhance productivity. Dynamic resource allocation in IoT application systems encompasses the efficient utilization of computational resources such as processing power, memory, and storage [5]. It also includes managing physical resources, including sensing devices, monitoring equipment, machinery, and other related resources.

IoT in agriculture, also known as Agriculture 4.0 [6], uses sensor networks, drones, and automated machines to gather tremendous amounts of data for optimizing agricultural outcomes [7]. Efficient management of computational and physical resources is critical for the effective operation of this technical framework. Dynamic resource allocation and task scheduling play pivotal roles in this optimization process [8]. Task scheduling is equally crucial, involving the coordination of various farm activities—planting, irrigation, pest control, and harvesting—taking into account real-time factors such as weather conditions, soil moisture levels, equipment availability, and crop growth stages. Smart task-scheduling algorithms dynamically adjust to optimize resource utilization and yields, ensuring timely and efficient farm operations [9].

This research is driven by the need to tackle the privacy issues, inefficiency, and delay problems linked with centralized Agri-IoT systems [8]. By tackling the challenges associated with dynamic resource allocation and task scheduling in edge-based IoT environments using a hybrid form of Bidirectional Long Short-Term Memory (BiLSTM) [10] and Gated Recurrent Unit (GRU) [11] for task scheduling along the attention mechanism, and Federated Reinforcement Learning for resource allocation, this research contributes to the development of more efficient, scalable, and adaptive decision-making frameworks. The integrated use of data fusion in edge computing with novel hybrid algorithms to predict and optimize dynamic task scheduling helps to reduce energy consumption. Similarly, Federated Learning integrated with Reinforcement Learning helps to augment the privacy preservation of the data, dynamic resource consumption, and user equipment selection for efficient operation and decision-making. In summary, our contributions are as follows:

1. Novel Hybrid Prediction Model: We present a novel hybrid model that merges Bidirectional Long Short-Term Memory (BiLSTM) and Gated Recurrent Unit (GRU) layers with an attention mechanism for projecting resource use in IoT applications, which can be used for predicting dynamic task scheduling.
2. Proposition of an Edge-Centric Offloading Algorithm: Our algorithm optimizes task allocation, enhances performance, and reduces energy consumption by utilizing predicted resource demands.
3. Novel Federated Learning Algorithm: We present a novel Federated Learning algorithm utilizing Deep Deterministic Policy Gradient (D4PG) for dynamic user equipment (UE) participation based on resource availability, tested against DQN, DDQN, Dueling DQN, and Dueling DDQN models using the EMNIST dataset, demonstrating its effectiveness in addressing model update fairness and enhancing overall efficiency.

The rest of the paper is structured as follows: Section 2 reviews existing approaches for resource management in Federated and Reinforcement Learning contexts, Section 3 presents the problem formulation, Section 4 presents the system design and details the proposed methodology, and Section 5 discusses the experimental results. Finally, Section 6 concludes the study and outlines potential future research directions.

2. Related Works

Paper [12] proposes an optimized system model for task offloading in 5G heterogeneous Mobile Edge Computing (MEC) networks, focusing on balancing processing time and energy consumption through a multitasking framework (MTO) and a Multitasking Evolutionary Algorithm (METOA). Paper [13] integrates IoT devices, MEC, and cloud layers, distinguishing between ordinary and VIP users. It employs differentiated services and security levels, with tasks offloaded based on real-time requirements and security needs across public and private nodes.

Edge computing and Agricultural IoT (Agri-IoT) have undergone notable progress in recent years to tackle distinct problems like instant decision-making, limited resources, and data protection [14]. This exploration investigates essential research and movements that are sculpting the world of edge computing in Agri-IoT.

In precision agriculture, edge computing was presented in [15]. It highlighted the importance of managing data locally and making decisions in real time. The method applied in [16] used pre-trained models based on a convolutional neural network (CNN) to detect plant sicknesses, adjusting hyperparameters for good functioning. A custom edge fog-cloud structure for IoT agriculture applications was suggested in [17], handling intermittent internet connections with a unique framework of edge computing [18]. The worries over security were tackled in [19]. The authors put forward a structure that brings together software-defined networking, blockchain technology, and fog computing to enhance Agri-IoT security.

Edge and fog node architectures in Agricultural IoT were studied in [20], and a five-layer hybrid IoT system architecture was introduced in [21] to improve deployment options. The idea of ad hoc networks using mobile devices for edge computing was put forward in [18], while the combination of edge computing with blockchain, AI, and virtual/augmented reality in Agri-IoT was explored in [22].

Though there is notable advancement as shown in [15–17], study [19] pointed out ongoing restrictions. Essential elements like dynamic resource allocation and task scheduling for improving Agri-IoT system performance are relatively untouched areas, as stated in studies [18,22]. The creation of adaptive resource allocation algorithms along with effective task-scheduling methods that are specifically designed for Agricultural IoT settings shows excellent potential for future investigation.

Dealing with difficulties in moving data for IoT devices, especially in situations where cellular networks are overloaded, the study ref. [23] proposed offloading schemes that depend on prediction and make use of mobility patterns. To solve the problem of resource scheduling in Federated Learning (FL) environments, study ref. [24] introduced the method called DynamicSched, which improves learning accuracy and scheduling efficiency. Paper [25] talked about how Federated Edge Learning (FEEL) was explored using multi-access edge computing (MEC). The authors used the greedy approach to select UEs, which helped with communication overheads and data privacy problems. They presented the Data-Quality Based Scheduling (DQS) algorithm. It takes into account the UE dataset variety and trustworthiness for picking participating devices and optimally distributing bandwidth. However, it does not completely solve the problem of unfair model updates, where some UEs might put more information into the global model than others, causing an imbalance that could impact the fairness and precision of the Federated Learning process. The random sampling method's stochastic nature adds variability to the training process that might influence convergence properties. Convergence may require additional tuning and monitoring.

Table 1 provides a summary of existing literature on Dynamic Resource Allocation and Task Scheduling in Edge for IoT Applications. Various research papers have suggested new frameworks to boost resource allocation and performance within distributed computing environments. An example of these is the Modified LSTM-based Congestion-Aware Edge Federated Learning (MLSTM-CEFL) model [26]. This framework focuses on congestion-aware timing and combining resources, but might neglect justice in allotting resources. This can cause individuals who receive fewer allocations to experience a decrease in their performance, which could lead to uneven results during model training processes. In the same manner, there is a BiLSTM-based cloud-edge-client Federated Learning system for Mobile Edge Computing (MEC) in 5G networks [27]. It handles conditions that change over time but does not integrate stochastic models or probabilistic methods to represent uncertainties in wireless channel states and data volumes. Pertaining to privacy and security, this TrustFedGRU framework [28] improves Federated Learning (FL) for the Industrial Internet of Things (IIoT) using dynamic update strategies. Yet it exhibits no consideration for non-uniformity in client data distributions (Non-IID) or differing computational abilities, hence indicating potential space for a more flexible strategy. Also, the FedBi-GRU algorithm [29] permits Virtual Network Functions (VNFs) to train locally while sharing model parameters, thereby boosting confidentiality and decreasing computational burden. Nonetheless, it encounters difficulties in adjusting to the changing and non-permanent nature of IoT settings, mainly dealing with abrupt shifts in traffic patterns or resource accessibility. In conclusion, the BILSTM-GRU model for predicting cloud resource use [30] presents enhancements in prediction precision and reduction time when compared to current models. However, it does not tackle fairness concerns regarding resource distribution—a vital matter within cloud settings where several users or applications share identical infrastructure. These models have been used in state-of-the-art research for resource allocation, task scheduling, and performance optimization in distributed computing environments. The chosen models represent a progression in sequence-learning architectures, from basic GRU/LSTM to more complex bidirectional and hybrid models. This allows for a fair comparison of how different architectures impact resource allocation and scheduling performance. These were chosen based on their relevance and common usage in sequence modeling tasks, particularly for time-series data, which is critical in IoT systems. Each model represents a distinct approach to handling sequential dependencies, with varying levels of complexity and performance.

Study [31] proposes an F-DDQN-based framework for energy-efficient resource allocation in D2D-assisted HetNets. It focuses on resource allocation and power control for a limited number of DDPs and CUEs. However, it does not address the scalability challenges arising from the massive number of devices in real-world D2D-assisted 6G networks. Moreover, the static reward function used for optimization does not consider real-time trade-offs among energy efficiency, latency, and throughput. It reduces its practical applicability in various scenarios. Study [32] proposes a high-precision map caching approach for ICVs with FL and Dueling-DQN to optimize vehicle participation and resource allocation. However, it does not consider the impact brought about by non-IID data that always characterizes real-world driving with diverse behaviors, environments, and sensing capabilities. Moreover, while participant selection is optimized per time slice, fairness over time is not guaranteed in the study and may result in the overuse or neglect of some vehicles. In study [33], the DQN-based method is proposed for node selection in FL over a network of data and hardware heterogeneity. Yet it does not exploit any dynamic change in the policy of node selection with respect to changes in the network, in client availability, or even regarding fluctuations in performance across devices. Paper [34] handles offloading decisions in MEC environments using DDQN agents on mobile devices while D3QN agents are on MEC servers; however, they do not cover heterogeneous mobile devices in a Federated Learning setup, considering hardware, processing power, battery life, and network connectivity to preserve the performance.

Table 1. Summary of Dynamic Resource Allocation and Task Scheduling in Edge for IoT Applications.

Framework	Key Focus	Advantages	Limitations
MLSTM-CEFL [26]	Congestion-aware timing and resource combination	Improves performance in congestion scenarios	Neglects fairness in resource allocation, leading to performance imbalances for some users
BiLSTM [27]	Mobile Edge Computing (MEC) in 5G networks	Handles changing conditions over time	Lacks integration of stochastic models for uncertainty in wireless channels and data volumes
TrustFedGRU [28]	Dynamic update strategies for FL in IIoT	Improves privacy and security	Does not account for non-IID data distributions or varying computational abilities in clients
FedBi-GRU [29]	Virtual Network Functions (VNFs) for local training	Enhances confidentiality and reduces computational burden	Struggles with the dynamic and non-permanent nature of IoT settings, including traffic and resource fluctuations
BiLSTM-GRU [30]	Cloud resource prediction model	Improves prediction precision and reduces time	Does not address fairness in resource distribution, especially in shared cloud environments
FeDRL-D2D [31]	Energy-efficient resource allocation in D2D-assisted HetNets	Focuses on power control and resource allocation for limited devices	Static reward function, lacks scalability for real-world D2D-assisted 6G networks, does not optimize real-time trade-offs
Dueling-DQN [32]	Resource allocation and vehicle participation optimization	High precision in map caching for vehicles	Does not account for non-IID data in real-world driving scenarios; fairness issues in vehicle participation
DQN [33]	Node selection in heterogeneous FL networks	Suitable for networks with data and hardware heterogeneity	Does not adapt to dynamic changes in the network or fluctuating performance across devices
DDQN [34]	Offloading decisions in MEC environments	Efficient offloading using DDQN agents	Does not consider heterogeneous mobile devices in Federated Learning, neglecting device characteristics like battery life and network connectivity

Various approaches have been proposed for client selection, each with its strengths and limitations. One notable method is the Enhanced Multi-Layer Feedforward Network (EMFFN) [35], which evaluates clients based on metrics such as Received Signal Strength Indicator (RSSI), statistical efficiency, communication efficiency, bandwidth, and energy levels. While EMFFN optimizes FL performance by reducing convergence time and improving accuracy, its reliance on static criteria may hinder adaptability to real-time changes in client performance.

To solve these problems of unfair model updates, privacy issues, and unbalanced resource utilization in edge computing, we suggest using a User-Efficient Federated Learning Algorithm for Resource Allocation in Edge Computing Networks. This can be applied to the Agricultural IoT scenario and it chooses UEs dynamically according to resource availability while also dealing with fairness problems.

3. Problem Formulation

This section initiates with an introduction to the system model, encompassing both the offloading and calculation models aided by the IoT. Formulating the Edge-Agri-IoT problem mathematically involves defining the key components, variables, and objectives of the system. We embrace the framework of edge computing, structured around three principal components: edge devices, edge nodes/servers, and clouds. Tasks are assigned to local devices and can be executed on any of these components.

We let $D = \{d_1, d_2, \ldots, d_n\}$ represent the set of edge devices deployed in the agricultural environment, $T = \{t_1, t_2, \ldots, t_m\}$ denote the set of tasks assigned on the edge devices, and $E = \{e_1, e_2, \ldots, e_n\}$ represent the set of edge servers where data can be offloaded for further processing. C represents the central cloud server. $x_{ij} \in \{0, 1\}$ represents the binary decision variable indicating whether task t_i is assigned to edge device d_j ($x_{ij} = 1$ if assigned, $x_{ij} = 0$ otherwise), and $y_{jk} \in \{0, 1\}$ represents the binary decision variable indicating whether edge device d_j offloads data to the edge server e_j ($y_{jk} = 1$ if offloaded, $y_{jk} = 0$ otherwise).

Table 2 describes the variables used in the problem formulation. The objective of our proposed method is to maximize the efficiency and performance of Agricultural IoT applications through the design of an optimized Edge-Agri-IoT architecture. The mathematical interpretation of the objective function is given below:

$$\text{Maximize } Z = \sum_{i=1}^{m} \sum_{j=1}^{n} \text{Performance}(x_{ij}) - \lambda \sum_{j=1}^{n} \text{OffloadCost}(y_{jk}) \tag{1}$$

Table 2. Notation and Variable Definitions.

Symbol	Description
$\text{Performance}(x_{ij})$	Performance metric for assigning task t_i to edge device d_j
$\text{OffloadCost}(y_{jk})$	Cost of offloading data from edge device d_j to the edge server e_k
λ	Weighting factor to balance performance and offload cost
$\text{DataLoad}(x_{ij})$	represents the data load associated with assigning task t_i to edge device d_j
$\text{DataOffload}(y_{jk})$	represents the data offload from edge device d_j to the cloud server e_j
$\text{Capacity}(d_j)$	is the capacity constraint for edge device d_j
N as the total number of UEs (user equipments)	
x_i	represents the learning progress of UE i, where $i = 1, 2, \ldots, N$
$f(x_1, x_2, \ldots, x_N)$	represents the objective function representing the learning process, which depends on the progress of all UEs
η	represents the learning rate, determining the step size in the learning process

Constraints

Constraints for the Edge-Agri-IoT problem are as follows:

1. Each task must be assigned to exactly one edge device:

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i \in \{1, 2, \ldots, m\} \tag{2}$$

2. Capacity constraint for each edge device:

$$\sum_{i=1}^{m} \text{DataLoad}(x_{ij}) \leq \text{Capacity}(d_j), \quad \forall j \in \{1, 2, \ldots, n\} \tag{3}$$

3. Offload decision constraint:

$$\sum_{j=1}^{n} y_{jk} \leq 1, \quad \forall k \in \{1\} \tag{4}$$

4. Data transfer constraint:

$$\sum_{i=1}^{m} \text{DataLoad}(x_{ij}) \cdot x_{ij} = \sum_{k=1}^{1} \text{DataOffload}(y_{jk}), \quad \forall j \in \{1, 2, \ldots, n\} \tag{5}$$

For resource allocation in edge computing networks within the Agricultural IoT environment, we need to incorporate the impact of user equipment (UE) sampling, unfair model updates, and convergence challenges.

UE Sampling: A subset K_g of UEs is randomly selected in each global round.

$$K_g \subseteq \text{UEs} \tag{6}$$

$$|K_g| \leq |\text{UEs}| \tag{7}$$

Impact of Stragglers: If some UEs are selected more frequently due to random sampling, they may not contribute equally to the learning process, causing delays in convergence. We can represent this impact by introducing factor α_i for each UE, where α_i denotes the weight or frequency of selection for UE i. Thus, the learning progress of UE i can be modified as

$$\Delta x_i = \eta \alpha_i \frac{\partial f}{\partial x_i} \tag{8}$$

Convergence Challenges: The stochastic nature of random sampling introduces variability in the updates, affecting the convergence properties. This variability can be represented by adding random noise to the learning progress updates. Let us denote ϵ_i as the random noise for UE i. Then, the learning progress update with convergence challenges can be represented as

$$\Delta x_i = \eta \frac{\partial f}{\partial x_i} + \epsilon_i \tag{9}$$

Overall, the learning progress update for each UE, considering both stragglers and convergence challenges, can be represented as

$$\Delta x_i = \eta (\alpha_i \frac{\partial f}{\partial x_i}) + \epsilon_i \tag{10}$$

These representations capture the impact of stragglers and convergence challenges in the learning process. Adjustments to the learning rate η, as well as additional tuning and monitoring, may be necessary to ensure convergence despite these challenges.

4. System Design and Algorithm

The system is designed in two parts, namely the local computing model and the edge server computation model, as shown in Figure 1.

4.1. Local Computing Model

The objective is to maximize the local computing performance while minimizing the overall energy consumption in a local computing environment. $D = \{d_1, d_2, \ldots, d_n\}$ represents the set of local computing edge devices available, $T = \{t_1, t_2, \ldots, t_m\}$ denotes the set of computing tasks that need to be processed locally, and $x_{ij} \in \{0, 1\}$ represents the binary decision variable indicating whether task t_i is assigned to device d_j ($x_{ij} = 1$ if assigned, $x_{ij} = 0$ otherwise). Table 3 describes the variables used in the local computing model.

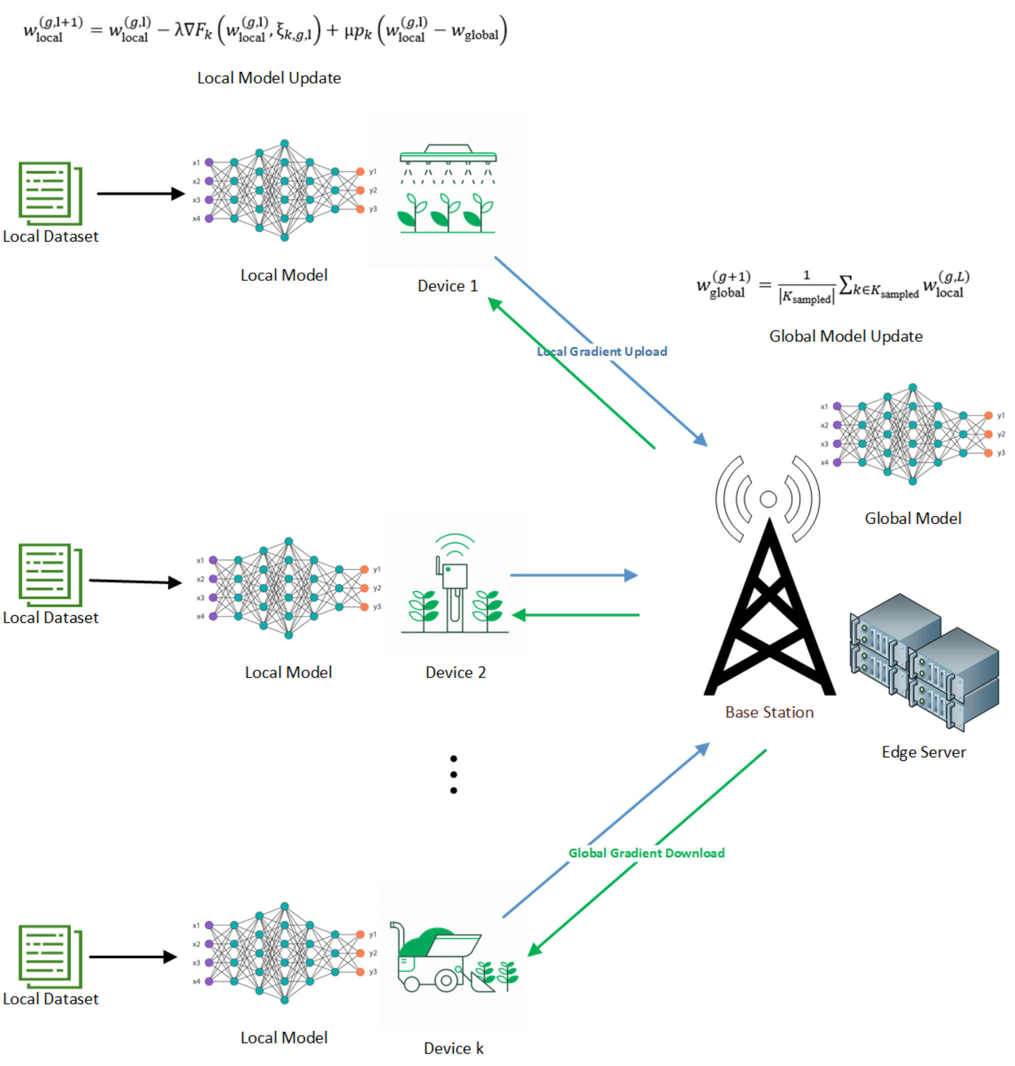
$w_{\text{local}}^{(g,l+1)} = w_{\text{local}}^{(g,l)} - \lambda \nabla F_k \left(w_{\text{local}}^{(g,l)}, \xi_{k,g,l} \right) + \mu p_k \left(w_{\text{local}}^{(g,l)} - w_{\text{global}} \right)$
Local Model Update
Local Dataset
Local Model
Device 1
$w_{\text{global}}^{(g+1)} = \frac{1}{|K_{\text{sampled}}|} \sum_{k \in K_{\text{sampled}}} w_{\text{local}}^{(g,L)}$
Global Model Update
Local Gradient Upload
Global Model
Local Dataset
Local Model
Device 2
Base Station
Edge Server
Global Gradient Download
Local Dataset
Local Model
Device k

Figure 1. Edge Computing architecture in an agricultural environmental setting.

Table 3. Notation and variable Definitions.

Symbol	Description
PerformanceMetric(x_{ij})	is a metric representing the performance of assigning task t_i to device d_j
EnergyConsumption(d_j)	is the energy consumption of device d_j
λ	is a weighting factor to balance performance and energy consumption
$\gamma_{e_j,t}$	signal-to-noise ratio (SNR) for the communication channel between the end user and the edge server for task t
P_u	transmit power from the end user
$h_{e_j,t}$	channel gain between the end user and the edge server for task t
n_t	noise power at the edge server
$h_{n,t',t}$	channel gain between the end user and the edge server for task t'
t'	belongs to the set T of all tasks except t
σ^2	noise power

Objective Function

$$\begin{aligned}&\text{Maximize Local Computing Performance: } Z\\ &= \sum_{i=1}^{m}\sum_{j=1}^{n} \text{PerformanceMetric}(x_{ij}) - \lambda \sum_{j=1}^{n} \text{EnergyConsumption}(d_j)\end{aligned} \tag{11}$$

4.2. Edge Server Computation Model

4.2.1. Signal-to-Noise Ratio (SNR)

The signal-to-noise ratio between the edge device and the edge server is given by

$$\gamma_{e_j,t} = \frac{P_u \cdot h_{e_j,t}}{\sum_{t' \in T \setminus \{t\}} P_u \cdot n_t \cdot h_{n,t',t} + \sigma^2} \tag{12}$$

This equation calculates the SNR by dividing the signal power $P_u \cdot h_{e_j,t}$ by the sum of noise and interference powers. The interference power is represented by the sum over all tasks t' in set T (excluding the current task t) of $P_u \cdot n_t \cdot h_{n,t',t}$, which captures the interference caused by other tasks on the channel. This sum is then added to the noise power σ^2.

This equation provides a measure of the quality of the communication channel between the end user and the edge server for a specific task t, considering both the signal power and the effects of noise and interference. Higher SNR values indicate better signal quality relative to noise and interference.

4.2.2. Objective Function

The objective is to maximize the edge server computation performance while minimizing the overall energy consumption:

$$Z = \sum_{i=1}^{m}\sum_{j=1}^{n} \text{PerformanceMetric}(x_{ij}) - \lambda \sum_{j=1}^{n} \text{EnergyConsumption}(e_j) \tag{13}$$

4.3. Offload to Edge Servers

The data transfer rate $R_{e_j,t}$ at which the end user offloads the computational task to the edge server is given by

$$R_{e_j,t} = B_{e_j,t} \cdot \log_2\left(1 + \gamma_{e_j,t}\right) \tag{14}$$

4.3.1. Data Transfer Delay

The data transfer delay D_{transfer} for offloading computational tasks from an end user to an edge server is

$$D_{\text{transfer}} = \frac{D_{\text{data}}}{B_{e_j,t} \cdot \log_2\left(1 + \gamma_{e_j,t}\right)} \tag{15}$$

4.3.2. Computational Delay

The computational delay D_{compute} in offloading a computational task to an edge server involves the time it takes for the edge server to process the task:

$$D_{\text{compute}} = \frac{C_{\text{t}}}{F_{e_j}} \tag{16}$$

4.3.3. Transmission Rate

The transmission rate $R_{\text{transmission}}$ of the calculated result data from the edge server back to the end devices is given by

$$R_{\text{transmission}} = B_{e_j} \cdot \log_2\left(1 + \gamma_{e_j,t}\right) \tag{17}$$

4.3.4. Return Delay

The return delay D_{return} of the calculated result data from the edge server to the end user is given by

$$D_{\text{return}} = \frac{D_{\text{result}}}{R_{\text{transmission}}} \tag{18}$$

4.3.5. Energy Consumption

The energy consumption of edge server transmission during the execution of a computational task is

$$E_{\text{transmission}} = P_i \cdot T_{n_i,j} + P_e \cdot T_{n_j,i} \tag{19}$$

4.3.6. Computational Energy Consumption

The computational energy consumption E_{compute} during the execution of a computational task by an edge server is given by

$$E_{\text{compute}} = P_{\text{compute}} \cdot C_{\text{t}} \tag{20}$$

4.3.7. Total Energy Consumption

The total energy consumption E_{Total} during the execution of a computational task by an edge server is

$$E_{\text{Total}} = E_{\text{compute}} + E_{\text{transmission}} \tag{21}$$

4.4. Objective in Federated Learning

The objective in Federated Learning involves collaborative training of a global model across distributed user devices (UEs), all without central aggregation of sensitive data. Nevertheless, an issue emerges: the potential for unfairness in model updates—certain UEs may contribute disproportionately to the training process—and this could result in biased or suboptimal global models. In response to this challenge, a solution is proposed: an algorithm for Federated Learning that dynamically determines the subset of user equipment (UEs) participating in every global iteration, guided by a Reinforcement Learning (RL) agent. The aim is to minimize the global objective function and ensure fairness in model updates, focusing on this particular objective.

The global objective function in this context aims to minimize the overall loss incurred by the system while addressing the issue of unfair model updates. Mathematically, the global objective function can be defined as follows: We let $\mathbf{L}_k(\mathbf{w}_k)$ represent the local loss function for UE k with model parameters $\mathbf{w}_k$ and dataset $\mathbf{D}_k$. Then, the global objective function can be defined as the average of the local loss functions over all selected UEs, with a regularization term to ensure fairness in sampling:

$$\min_{w} \frac{1}{|\mathcal{K}_{\text{sampled}}|} \sum_{k \in \mathcal{K}_{\text{sampled}}} \mathcal{L}_k(w_k) + \lambda \sum_{k=1}^{K_{\text{tot}}} p_k \cdot ||w_k - w_{\text{global}}||^2 \tag{22}$$

Table 4 describes the variables used in the Objective in Federated Learning.

Table 4. Notation and Variable Definitions.

Symbol	Description
$\mathbf{w}_{\text{global}}$	denotes the global model parameters
$\mathbf{w}_k$	denotes the local model parameters for UE k
p_k	denotes the sampling probability for UE k in K_{tot}
λ	denotes the regularization strength
$\mathcal{K}_{\text{sampled}}$	is the size of the sampled subset of UEs
$\mathcal{L}_k(\mathbf{w}_k)$	is the local loss function for UE k with model parameters $\mathbf{w}_k$
$\lvert\lvert\mathbf{w}_k - \mathbf{w}_{\text{global}}\rvert\rvert^2$	is the squared Euclidean distance between the local model parameters and the global model parameters

This objective function seeks to minimize the average local loss across the selected UEs while also penalizing large deviations of local models from the global model, thus promoting fairness in updates. The regularization term encourages the local models to stay close to the global model, which helps in achieving fairness in contributions across different UEs.

The objective is to minimize the global objective function subject to the given constraints:

$$\text{Minimize:} \quad J(w_{\text{global}}) = \frac{1}{|\mathcal{K}_{\text{sampled}}|} \sum_{k \in \mathcal{K}_{\text{sampled}}} \mathcal{L}_k(w_k) + \lambda \sum_{k=1}^{K_{\text{tot}}} p_k \cdot ||w_k - w_{\text{global}}||^2 \tag{23}$$

Subject to

1. $\mathbf{w}_k = \mathbf{w}_{\text{global}}$ for all k in $\mathcal{K}_{\text{sampled}}$ after local training;
2. $w_{\text{global}} = \frac{1}{|\mathcal{K}_{\text{sampled}}|} \sum_{k \in \mathcal{K}_{\text{sampled}}} w_k$ after global model aggregation.

The constraints ensure that after local training, the local models are synchronized with the global model, and after aggregation, the global model is updated based on the contributions from the selected UEs.

This optimization problem seeks to find the optimal global model parameters $\mathbf{w}_{\text{global}}$ that minimize the average local loss across the selected subset of UEs while encouraging fairness in updates by penalizing large deviations of local models from the global model. The constraints ensure that after local training, the local models are synchronized with the global model, and after aggregation, the global model is updated based on the contributions from the selected UEs.

4.5. Proposed Algorithm

4.5.1. Task Offloading Algorithm

Emphasizing the significance of Borg cells in large-scale cluster management in [36,37], our project leverages Google Cluster Traces, a dataset replete with detailed information. This indispensable resource includes CPU usage histograms, allocation set details, and job parent data that span several days of workload on a single Borg cell. Given its massive size, approximately 2.4 TiB compressed, we turn to Google BigQuery for sophisticated analysis. For the analysis, we use the instance usage table of the dataset.

Bidirectional LSTM Layer

The Figure 2 shows the architectural details of the proposed BiLSTM-GRU model. The proposed BiLSTM-GRU model (Algorithm 1) starts by initializing the weights and biases for the LSTM, GRU, and Dense layers, as well as the initial hidden and cell states for both the forward and backward LSTM layers and the initial hidden state for the GRU layer. The input sequence is now implemented into the Bidirectional LSTM layer. For every time step t, the forward LSTM calculates the forward hidden state and cell state by using the input at t along with the previous hidden state. The backward LSTM also works similarly, but it processes sequences in reverse order to compute the backward hidden state and the cell state. The output of this layer at each time step is formed by combining both forward and backward hidden states. BiLSTM is used for capturing dependencies from past and future, GRU for effective memory management, and an attention mechanism to emphasize necessary time steps making the model capable of providing high accuracy in predicting time-series tasks. The combination of these architectures can be scaled up to large datasets efficiently, particularly with real-time IoT environments. By concentrating on vital data points only through an attention mechanism, this model deals with vast input sequences effectively; at the same time, BiLSTM and GRU layers keep their potential for forming complex temporal patterns throughout larger sequences intact. In uses such as intelligent farming, health surveillance, and industry IoT, where real-time predictions are critical, this mixed structure supports swift choices based on the past and forthcoming environment.

Algorithm 1 Proposed BiLSTM-GRU with Attention Mechanism

```
Input: Sequential input data $X = \{x_1, x_2, \ldots, x_T\}$, where each $x_t$ is a feature vector at time $t$
Output: Predicted output sequence $\hat{y} = \{y_1, y_2, \ldots, y_T\}$ after applying the BiLSTM-GRU model with attention
1:  Initialization: Initialize model weights and biases: $W_{\text{lstm}}$, $U_{\text{lstm}}$, $b_{\text{lstm}}$, $W_{\text{a}}$, $b_{\text{a}}$, $W_{\text{gru}}$, $U_{\text{gru}}$, $b_{\text{gru}}$, $W^{\text{dense}}$, $b^{\text{dense}}$
2:  Initialize hidden states and cell states: $h_t^{\rightarrow}$, $h_t^{\leftarrow}$, $c_t^{\rightarrow}$, $c_t^{\leftarrow}$, $h_{t-1}^{\text{GRU}}$
3:  Bidirectional LSTM Layer:
4:  for $(t = 0, 1, \ldots, T-1)$ do
5:      Forward LSTM step: Compute forward hidden state $h_t^{\rightarrow}$ and cell state $c_t^{\rightarrow}$
6:      Backward LSTM step: Compute backward hidden state $h_t^{\leftarrow}$ and cell state $c_t^{\leftarrow}$
7:      Concatenate forward and backward hidden states: $H_{\text{BiLSTM}}[t]$
8:  end for
9:  Attention Mechanism:
10: for $(t = 0, 1, \ldots, T-1)$ do
11:     Compute attention weights for each time step: $e_t = \tanh(W_a H_{\text{BiLSTM}}[t] + b_a)$
12:     Normalize attention weights: $\alpha_t = \text{softmax}(e_t)$
13:     Compute context vector as a weighted sum of hidden states: $c_t = \sum_{i=0}^{T-1} \alpha_i \cdot H_{\text{BiLSTM}}[t]_i$
14:     Concatenate context vector with current BiLSTM hidden state: $h_{\text{att\_}t} = [c_t; H_{\text{BiLSTM}}[t]]$
15: end for
16: Gated Recurrent Unit (GRU) Layer:
17: for $(t = 0, 1, \ldots, T-1)$ do
18:     GRU forward step: Compute update gate $z_t$, reset gate $r_t$, candidate hidden state $\tilde{h}_t$
19:     Update hidden state: $h_t^{\text{GRU}}$ using $h_{\text{att\_}t}$ as input
20: end for
21: Dense Layer:
22: Perform linear transformation: $z = h_t^{\text{GRU}} \cdot W^{\text{dense}} + b^{\text{dense}}$
23: Apply activation function: $y = \text{softmax}(z)$
24: Training, Evaluation, and Prediction:
25: Train model using backpropagation and optimization algorithm
26: Evaluate model performance using appropriate metrics
27: Make predictions on new sequential data
```

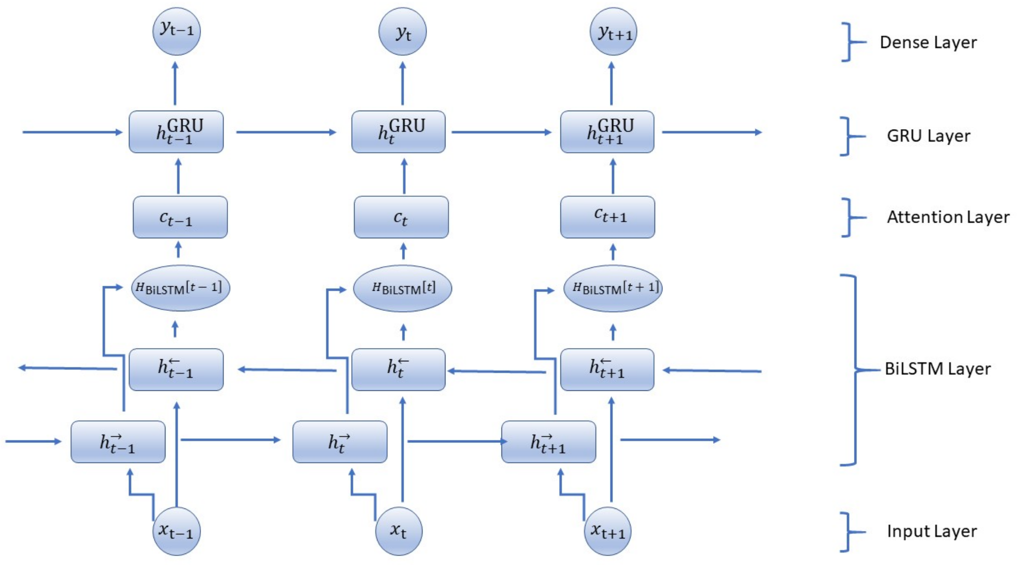

Figure 2. Hybrid model architecture.

For a given input sequence $X = (x_1, x_2, \ldots, x_T)$ of length T, a BiLSTM layer processes it in both forward ($\rightarrow$) and backward ($\leftarrow$) directions. The forward hidden states $h_t^{\rightarrow}$ and backward hidden states $h_t^{\leftarrow}$ at each time step t are computed as follows:

Forward LSTM:

$$i_t^{\rightarrow} = \sigma\left(W_{ii}^{\rightarrow} x_t + b_{ii}^{\rightarrow} + W_{hi}^{\rightarrow} h_{t-1}^{\rightarrow} + b_{hi}^{\rightarrow}\right) \tag{24}$$

$$f_t^{\rightarrow} = \sigma\left(W_{if}^{\rightarrow} x_t + b_{if}^{\rightarrow} + W_{hf}^{\rightarrow} h_{t-1}^{\rightarrow} + b_{hf}^{\rightarrow}\right) \tag{25}$$

$$g_t^{\rightarrow} = \tanh\left(W_{ig}^{\rightarrow} x_t + b_{ig}^{\rightarrow} + W_{hg}^{\rightarrow} h_{t-1}^{\rightarrow} + b_{hg}^{\rightarrow}\right) \tag{26}$$

$$o_t^{\rightarrow} = \sigma\left(W_{io}^{\rightarrow} x_t + b_{io}^{\rightarrow} + W_{ho}^{\rightarrow} h_{t-1}^{\rightarrow} + b_{ho}^{\rightarrow}\right) \tag{27}$$

$$c_t^{\rightarrow} = f_t^{\rightarrow} \odot c_{t-1}^{\rightarrow} + i_t^{\rightarrow} \odot g_t^{\rightarrow} \tag{28}$$

$$h_t^{\rightarrow} = o_t^{\rightarrow} \odot \tanh\left(c_t^{\rightarrow}\right) \tag{29}$$

Backward LSTM:

$$i_t^{\leftarrow} = \sigma\left(W_{ii}^{\leftarrow} x_t + b_{ii}^{\leftarrow} + W_{hi}^{\leftarrow} h_{t+1}^{\leftarrow} + b_{hi}^{\leftarrow}\right) \tag{30}$$

$$f_t^{\leftarrow} = \sigma\left(W_{if}^{\leftarrow} x_t + b_{if}^{\leftarrow} + W_{hf}^{\leftarrow} h_{t+1}^{\leftarrow} + b_{hf}^{\leftarrow}\right) \tag{31}$$

$$g_t^{\leftarrow} = \tanh\left(W_{ig}^{\leftarrow} x_t + b_{ig}^{\leftarrow} + W_{hg}^{\leftarrow} h_{t+1}^{\leftarrow} + b_{hg}^{\leftarrow}\right) \tag{32}$$

$$o_t^{\leftarrow} = \sigma\left(W_{io}^{\leftarrow} x_t + b_{io}^{\leftarrow} + W_{ho}^{\leftarrow} h_{t+1}^{\leftarrow} + b_{ho}^{\leftarrow}\right) \tag{33}$$

$$c_t^{\leftarrow} = f_t^{\leftarrow} \odot c_{t+1}^{\leftarrow} + i_t^{\leftarrow} \odot g_t^{\leftarrow} \tag{34}$$

$$h_t^{\leftarrow} = o_t^{\leftarrow} \odot \tanh\left(c_t^{\leftarrow}\right) \tag{35}$$

The provided equations delineate the forward and backward LSTM units' calculations essential for processing such data. First, the input, forget, output, and input gates for the forward and backward LSTMs ($i_t^{\rightarrow}, f_t^{\rightarrow}, o_t^{\rightarrow}, g_t^{\rightarrow}; i_t^{\leftarrow}, f_t^{\leftarrow}, o_t^{\leftarrow}, g_t^{\leftarrow}$) are delineated. These gates regulate the flow of information, enabling the model to selectively retain or discard information at each time step. Additionally, the equations illustrate how the cell states ($c_t^{\rightarrow} = f_t^{\rightarrow} \odot c_{t-1}^{\rightarrow} + i_t^{\rightarrow} \odot g_t^{\rightarrow}, c_t^{\leftarrow} = f_t^{\leftarrow} \odot c_{t+1}^{\leftarrow} + i_t^{\leftarrow} \odot g_t^{\leftarrow}$) are updated. The new cell states are computed by combining the previous cell states with selectively weighted information using the input and forget gates. Lastly, the equations detail the computation of hidden states ($h_t^{\rightarrow} = o_t^{\rightarrow} \odot \tanh(c_t^{\rightarrow})$, $h_t^{\leftarrow} = o_t^{\leftarrow} \odot \tanh(c_t^{\leftarrow})$), which encapsulate the model's learned representation of the input sequence. These hidden states are influenced by the cell states and are modulated by the output gates, enabling the model to capture and retain relevant information for prediction tasks. Together, these computations empower the BiLSTM to effectively process sequential data, learning intricate dependencies both forward and backward through time.

Attention Mechanism Integration

The novel aspect of adding the attention mechanism to the BiLSTM-GRU model lies in its ability to selectively focus on the most relevant time steps in the input sequence. For each time step, an attention score is computed using a learned weight matrix W_a and the hidden states from the BiLSTM layer. This score reflects how much attention the model should pay to a particular time step.

Step 1: Compute Attention Weights. First, we compute the attention weights for the hidden state output by the BiLSTM. The attention weights help the model to decide which parts of the sequence are more important at each time step.

For each hidden state $H_{\mathrm{BiLSTM}}[t]$ of the BiLSTM, we compute the attention score e_t:

$$e_t = \tanh(W_a H_{\mathrm{BiLSTM}}[t] + b_a) \tag{36}$$

where W_a and b_a are learnable parameters.

Next, we compute the attention weights α_t using a softmax function:

$$\alpha_t = \frac{\exp(e_t)}{\sum_{t'} \exp(e_{t'})} \tag{37}$$

Step 2: Compute Context Vector. Using the attention weights, we compute the context vector c_t, which is a weighted sum of the hidden states:

$$c_t = \sum_{t'} \alpha_{t'} H_{\mathrm{BiLSTM}}[t'] \tag{38}$$

Step 3: Combine Context Vector with GRU. We integrate the context vector into the GRU computations. Instead of directly feeding the hidden states from the BiLSTM to the GRU, we combine them with the context vector.

We modify the GRU equations as follows:

Update Gate (z_t):

$$z_t = \sigma\Big(W_z \cdot \Big[h_{t-1}^{\mathrm{GRU}}, c_t, H_{\mathrm{BiLSTM}}[t]\Big] + b_z\Big) \tag{39}$$

Reset Gate (r_t):

$$r_t = \sigma\Big(W_r \cdot \Big[h_{t-1}^{\mathrm{GRU}}, c_t, H_{\mathrm{BiLSTM}}[t]\Big] + b_r\Big) \tag{40}$$

Candidate Hidden State ($\widetilde{h_t}$):

$$\widetilde{h_t} = \tanh\Big(W_h \cdot \Big[r_t \odot h_{t-1}^{\mathrm{GRU}}, c_t, H_{\mathrm{BiLSTM}}[t]\Big] + b_h\Big) \tag{41}$$

Hidden State Update (h_t^{GRU}):

$$h_t^{\mathrm{GRU}} = (1 - z_t) \odot h_{t-1}^{\mathrm{GRU}} + z_t \odot \widetilde{h_t} \tag{42}$$

Dense Layer

The outcome of the last hidden state, which comes from the GRU layer at the final time step, flows into a Dense layer. Here, there is a linear change, and then the use of the softmax activation function to provide a result or output that represents the model's predictions. By performing backpropagation and using an optimization algorithm, we train the model. After this training process is finished, we evaluate it with suitable metrics before finally making predictions on fresh sequential data. The model is improved by using layers of Bidirectional LSTM and GRU together, along with the attention mechanism. This lets the model understand relationships in the past as well as future elements of the sequence, making it better at prediction.

$$z = h_t^{\mathrm{GRU}} \cdot W^{\mathrm{dense}} + b^{\mathrm{dense}} \tag{43}$$

$$y = \mathrm{softmax}(z) \tag{44}$$

The task offloading Algorithm 2 starts by setting a counter to zero. It uses a model that has already been trained for predicting the resource needs of tasks as time goes on, which we represent with the workload matrix $W(t)$. For each edge server i, it calculates forecasted resource requirements P_i by using the prediction model on present data D_i.

Algorithm 2 Task Offloading Algorithm

Input: Workload matrix $W(t)$ representing the resource requirements of tasks over time and Edge server information matrix EdgeServerInfo
Output: Predicted data, Energy consumption matrix, Delay matrix, Cost matrix

```
1:  Initialize Counter = 0
2:  Load a pre-trained model for predicting task resource requirements
3:  for each edge server $i$ do
4:      Calculate $P_i = \mathrm{Predict}(D_i)$
5:  end for
6:  for each edge server $i$ do
7:      Calculate $F_i = \mathrm{Combine}(P_i, E_i, \Delta_i, C_i)$
8:  end for
9:  Sort edge servers based on the decision factor: SortedServers = $\mathrm{Sort}(F_i)$
10: for each service do
11:     if $F_i <$ Threshold then
12:         Process the service locally
13:     else
14:         Migrate the service to the selected edge server
15:     end if
16: end for
17: for each edge server $i$ in SortedServers do
18:     if the edge server has enough capacity to host the current service and the service is
        not already hosted then
19:         Provision($i$)
20:         Increment Counter
21:         Exit the loop to ensure the service is migrated only once
22:     end if
23: end for
24: if Counter = $N$ then
25:     Halt the process
26: end if
```

Then, we calculate a decision factor F_i for every edge server. It is obtained by combining predicted resource requirements P_i, energy consumption E_i, delay Δ_i, and cost C_i. After that, the list of edge servers is sorted. The result becomes a sorted list SortedServers.

The algorithm continues by checking each service. If the decision factor F_i is less than a specific limit, then this service is handled locally on the edge server. Otherwise, it needs to move over and be processed at the chosen edge server from the sorted list.

For every edge server in the list that has been sorted, we see if it can hold the service and make sure this service is not already hosted on that server. If these conditions are fulfilled, we move the service to this edge server, increase our counter by one, and stop the loop so that migration only happens once for each service.

Lastly, if the counter equals the total number of services N, then it stops. This shows that all services have been allocated well on edge servers. This algorithm manages task offloading in a good way. It uses prediction models and thinks about many things like energy use, delay, cost, etc., to make sure resources are put into edge computing places as much as possible.

The process of the hybrid migration algorithm is supported by a discrete-event simulator called EdgeSimPy. This tool is purposely designed for modeling edge computing scenarios, making it easier to create situations that involve operations on edge servers and allowing resource management as well as analysis of system behavior [38].

The script makes use of the Pandas library to load a JSON file, which contains comprehensive data regarding edge servers. It then arranges this information into an array of dictionaries. Each dictionary represents a single server with distinct attributes such as 'cpu_demand' and 'memory_demand'. Feature engineering is carried out for predictive enhancement using scikit-learn classes. The script employs TensorFlow's Keras API to load a pre-trained Bidirectional LSTM model to predict CPU and memory usage for each edge server. It simulates decision-making logic to determine if a service should be processed locally or migrated based on predicted resource usage, optimizing edge server categorization and decision-making. If there is a need for migration, the service is provided to a server that has capacity, and migration statistics are updated.

An object made with EdgeSimPy simulates edge computing situations using the dataset. The simulation continues until a stopping point is reached, which checks whether every service has been successfully provided.

The 'hybrid_offloading_algorithm' function manages offloading and migration decisions, utilizing predictions from a pre-trained hybrid model of BiLSTM and GRU. The script employs EdgeSimPy's helper methods to iterate over all edge servers and services, implementing the hybrid migration logic and maintaining a record of migration counts.

4.6. Proposed Federated Learning Algorithm with Dynamic UEs Selection

The start of this process is marked by setting up the global model weights (w_{global}) and regularization parameters (λ), which are uniform for all user equipment (UE). An agent for D4PG starts next, having been given appropriate observation and action spaces to help it make decisions. In every overall iteration (g), the agent chooses a subset (K_{sampled}) from all available UEs (K_g), and this selection changes dynamically using policies learned to optimize the system's general performance.

Next, when the subset of UEs is chosen, the present global model (w_{global}) is sent to these sampled UEs. Each UE (k) in this set starts its local model (w_{local}) by using weights from the global model it received earlier. Then, UEs proceed with local training for L rounds. In every iteration (l), we pick one random data point ($\xi_{k,g,l}$) from the local dataset (D_k) of the UE and update its local model with the gradient descent method. The formula for updating includes a regularization term (μp_k) that helps in balancing how much the local model deviates from the global one:

$$w_{(g,l+1)_{\text{local}}} = w_{(g,l)_{\text{local}}} - \lambda \nabla F_k(w_{(g,l)_{\text{local}}}, \xi_{k,g,l}) + \mu p_k (w_{(g,l)_{\text{local}}} - w_{\text{global}}) \tag{45}$$

When the local training is finished, every UE sends its new local model to the base station. The base station combines all these local models into a fresh global model ($w_{(g+1)_{\text{global}}}$) by obtaining an average of weights from chosen UEs:

$$w_{\text{global}}^{(g+1)} = \frac{1}{\left|K_{\text{sampled}}\right|} \sum_{k \in K_{\text{sampled}}} w_{\text{local}}^{(g,L)} + \mathcal{N}(0, \sigma^2 I) \tag{46}$$

Updates to the model based on feedback about fairness and effectiveness are utilized for adjusting the policies of the Reinforcement Learning agent. Therefore, this helps in enhancing its ability to choose optimal UEs during subsequent iterations. The iterative process carries on for a fixed number of global iterations (G), gradually improving the global model via collaborative learning and adaptive sampling techniques.

4.6.1. Objective

The global objective function can be minimized by updating the global model parameters. This is important because it ensures that unfairness in model updates, where some UEs contribute more often than others, is taken care of.

Traditional Federated Learning algorithms often select a static or random subset of UEs for local training in each iteration [39–41]. In contrast, this Algorithm 3 leverages a D4PG agent to dynamically learn policies for selecting an optimal subset of UEs, Algorithm 4. The selection is based on observed states (UE availability, network conditions, resource constraints) and actions that maximize the overall training efficiency and fairness. This policy-driven approach to selection is a key differentiator from standard random or fixed strategies.

Algorithm 3 Federated Learning Algorithm with Dynamic UE Selection using D4PG

```
Input: Global model parameters $w_{\text{global}}$, UE dataset $D_k$ for each UE $k$, hyperparameter $\lambda$ for all UEs, maximum number of global iterations $G$, local training iterations $L$.
Output: Final global model parameters $w_{\text{global}}^{(G)}$, trained D4PG agent, performance metrics.
 1: Initialization: Initialize global model parameters $w_{\text{global}}$. Initialize the hyperparameter $\lambda$ for all UEs.
 2: D4PG Agent Initialization: Initialize the D4PG agent with appropriate observation and action spaces, neural networks for policy and value functions, and experience replay buffer.
 3: Dynamic Sampling: At each global iteration $g$:
 4:     D4PG agent dynamically selects a subset $K_{\text{sampled}}$ of $K_g$ UEs based on learned policies

 5: Global Model Broadcast: Broadcast the global model parameters $w_{\text{global}}$ to all UEs in $K_{\text{sampled}}$.
 6: Local Training: Each UE $k \in K_{\text{sampled}}$ performs local training:
 7:     Initialize local model $w_{\text{local}}$ with $w_{\text{global}}$.
 8: for $(l = 0, 1, \ldots, L-1)$ do
 9:         Randomly select a data point $\xi_{k,g,l}$ from local dataset $D_k$.
10:         Update local model: As shown in Equation (45)
11: end for
12: Send Local Models: Each UE sends its updated local model $w_{\text{local}}^{(g,L)}$ to the base station.
13: Global Model Aggregation: The base station aggregates the local models to update the global model: As shown in Equation (46)
14: D4PG Agent Update: Update the D4PG agent using the reward based on the performance and fairness of the aggregated model, storing experiences in the replay buffer, and training the neural networks.
15: Repeat: Repeat the process for $G$ global iterations.
```

Algorithm 4 Dynamic Sampling with D4PG

Input: Environment, replay buffer D, Q-network parameters θ_{online} and θ_{target}, exploration rate ϵ, learning rate α, discount factor γ, target network update frequency C, target network update rate τ

Output: Optimal action sequence for task offloading and resource allocation

```
1:  Initialize $\theta_{\text{online}}$ and $\theta_{\text{target}}$
2:  for each global iteration $g$ do
3:    Observe current state $s_g$
4:    Select action $a_g$:
5:    if random$(0,1) < \epsilon$ then
6:      Choose a random subset of UEs as $a_g$
7:    else
8:      Choose $a_g = \arg\max_a Q(s_g, a; \theta_{\text{online}})$
9:    end if
10:   Sample subset $K_{\text{sampled}}$ from $a_g$
11:   Record $(s_g, a_g)$ in $D$
12:   Conduct local training and obtain reward $R$
13:   Sample mini-batch $(s, a, r, s')$ from $D$
14:   Set target:
15:   if $s'$ is terminal then
16:     $target = r$
17:   else
18:     $target = r + \gamma \mathbb{E}[Q(s', a'; \theta_{\text{target}})]$
19:   end if
20:   Update $\theta_{\text{online}}$ by minimizing the loss:
          $L = (Q(s, a; \theta_{\text{online}}) - target)^2$
21:   Every $C$ steps, update $\theta_{\text{target}}$ by:
          $\theta_{\text{target}} \leftarrow \tau\theta_{\text{online}} + (1-\tau)\theta_{\text{target}}$
22: end for
```

4.6.2. D4PG Agent Initialization

- State Space: $S_s = \{s_1, s_2, \ldots, s_{|S_s|}\}$, where $|S_s|$ is the size of the state space.
- Action Space: $A_a = \{a_1, a_2, \ldots, a_{|A_a|}\}$, where $|A_a|$ is the size of the action space.
- Q-function Approximator: $Q(s, a; \theta)$ with neural network parameters θ.
  - The Q-function approximator estimates the expected future reward when taking action a in state s.
  - θ represents the parameters of the neural network.
- Experience Replay Buffer: $D = \{(s_1, a_1, r_1, s_1'), (s_2, a_2, r_2, s_2'), \ldots, (s_N, a_N, r_N, s_N')\}$, where N is the size of the replay buffer, r_i is the reward, and s_i' is the next state.

4.6.3. Dynamic Sampling Mechanism

The dynamic sampling of action subsets from the action space is a key novelty in this algorithm.

Action Selection in D4PG

In standard D4PG, the action is selected by maximizing the Q-function:

$$a_g = \arg\max_a Q(s_g, a; \theta_{\text{online}}) \tag{47}$$

This deterministic action selection works well for small action spaces.

Dynamic Sampling for Subset of Actions Instead of selecting a single action, we sample a subset $K_{\text{sampled}} \subseteq \mathcal{A}$ from the total action space $\mathcal{A}$.

- Exploration vs Exploitation (Epsilon-Greedy Policy): With probability ϵ, a random subset of actions is selected, and with probability $1 - \epsilon$, we maximize the Q-value:

$$a_g = \begin{cases} \text{Random subset of } \mathcal{A}, & \text{with probability } \epsilon \\ \arg\max_{a \in \mathcal{A}} Q(s_g, a; \theta_{\text{online}}), & \text{with probability } 1 - \epsilon \end{cases} \tag{48}$$

- Dynamic Subset Sampling: After selecting an action a_g, we sample a subset K_{sampled} from $\mathcal{A}$:

$$K_{\text{sampled}} \sim P(K \mid s_g, a_g) \tag{49}$$

  where $P(K \mid s_g, a_g)$ is a probability distribution over subsets of the action space.

Initially, the online and target Q-network parameters are set up. During each global iteration, the agent observes the current state and selects an action based on an exploration–exploitation strategy: with probability ϵ, it randomly chooses an action, and otherwise it selects the action that maximizes the Q-value from the online Q-network. The agent then samples a subset of actions and records the state-action pair in the replay buffer for training. After taking the action, the agent receives a reward and samples a mini-batch of experiences from the buffer to update the Q-network. The target Q-value is computed using the reward and the expected Q-value of the next state, and the online Q-network is updated by minimizing the loss between the predicted and target Q-values. Every C step, the target network is updated by slowly blending the online Q-network's weights. This process continues iteratively, allowing the agent to learn an optimal policy for task offloading and resource allocation, leveraging both exploration and exploitation while maintaining stable learning with the target network.

4.6.4. Experience Replay Mechanism

Experience replay helps sample past experiences from a replay buffer D to improve sample efficiency.

Standard Experience Replay In D4PG, a mini-batch $\mathcal{B}$ of past experiences is sampled:

$$L(\theta_{\text{online}}) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{B}}\left[(Q(s, a; \theta_{\text{online}}) - \text{target})^2\right] \tag{50}$$

where the target is defined as

$$\text{target} = \begin{cases} r, & \text{if } s' \text{ is terminal} \\ r + \gamma \mathbb{E}[Q(s', a'; \theta_{\text{target}})], & \text{otherwise} \end{cases} \tag{51}$$

Experience Replay with Dynamic Sampling In this algorithm, the experience tuples now contain action subsets:

$$D = \{(s_g, K_{\text{sampled}}, r_g, s'_g)\} \tag{52}$$

The mini-batch is sampled from the replay buffer, and the Q-learning loss is modified:

$$L(\theta_{\text{online}}) = \mathbb{E}_{(s,K_{\text{sampled}},r,s') \sim \mathcal{B}}\left[\Big(Q(s, K_{\text{sampled}}; \theta_{\text{online}}) - \text{target}\Big)^2\right] \tag{53}$$

With the target remaining the same:

$$\text{target} = r + \gamma \mathbb{E}\left[\max_{a'} Q(s', a'; \theta_{\text{target}})\right] \tag{54}$$

- Dynamic Subset Selection: The algorithm selects a subset of actions K_{sampled}, allowing for flexibility in handling large or combinatorial action spaces, particularly in distributed or multi-agent environments.
- Sampling Distribution: The probability distribution $P(K \mid s_g, a_g)$ introduces a dynamic and state-dependent sampling mechanism, improving exploration efficiency.
- Experience Replay with Action Subsets: The replay buffer stores subsets of actions rather than single actions, enabling richer learning from diverse experiences, which is especially useful in multi-agent scenarios.

Table 5 describes the architecture details for the BiLSTM-GRU Model with attention mechanism.

Table 5. Architecture Details for the BiLSTM-GRU Model with attention mechanism.

Name of Parameter	Details
Input layer neurons	43 (depends on input feature dimension)
BiLSTM layer neurons	512 in each direction (total 1024 combined)
Attention layer	Custom attention layer
GRU layer neurons	128
Final layer (Dense) neurons	2
Training step/epochs	[20–100]
Batch size	[16–256]
Optimizer	adam
Activation function	ReLU/Tanh/Sigmoid
Loss function	Mean squared error

5. Experimental Results

The experiment was conducted using a system equipped with Intel® Core™ Ultra 7 155H. The system was paired with an NVIDIA® GeForce RTX™ 4070 Laptop GPU, which has 8 GB of GDDR6 VRAM. Additionally, the system included 32 GB of LPDDR5X-7467MHz RAM. For comparison with state-of-the-art methods, the implementation was carried out using TensorFlow 2.14.1. The proposed methodology incorporates a BiLSTM-GRU hybrid model for task scheduling, built using Keras, along with Federated Reinforcement Learning for dynamic resource allocation. The Flower simulation framework [42] 1.6.0 and EdgeSimPy [38] v1.1.0 was used for conducting the simulations. EdgeSimPy provides a modular, scalable environment that supports real-time monitoring, making it an ideal fit for IoT-based task offloading and optimization of resource utilization and energy efficiency. Flower framework is a scalable Federated Learning framework that supports decentralized model training, real-time updates, and privacy. It integrates with various ML libraries and offers modular components for secure, scalable learning.

5.1. Ablation Study

5.1.1. Comparing the Proposed Method With and Without the Attention Mechanism

Table 6 evaluates the impact of incorporating an attention mechanism into the proposed method. The global accuracy increases from 0.5386 (without attention) to 0.70 (with attention), showing a significant performance gain. The global loss decreases from 0.0054 to 0.0034, indicating that the model learns more effectively with the attention mechanism. Client 1 experiences a substantial boost in accuracy from 0.30 to 0.90, though the loss slightly increases from 4.5385×10^{-5} to 8.5700×10^{-4}. Client 2 also sees an improvement in accuracy from 0.60 to 0.80, while its loss decreases significantly from 2.7892×10^{-4} to 1.6920×10^{-5}.

Table 6. Comparing The proposed method with and without the attention mechanism.

Model	**Train Accuracy**	**Train Loss**	**Eval Accuracy (Client 1)**	**Eval Loss (Client 1)**	**Eval Accuracy (Client 2)**	**Eval Loss (Client 2)**
Without Attention Mechanism	0.54	0.0054	0.30	4.5385×10^{-5}	0.60	2.7892×10^{-4}
With Attention Mechanism	0.70	0.0034	0.90	8.5700×10^{-4}	0.80	1.6920×10^{-5}

The proposed D4PG-based Federated Learning algorithm is analyzed on different learning rates and its effect on model accuracy. The highest accuracy of 92.86% is achieved with a learning rate of 0.001, suggesting that this value enables optimal convergence. With 0.0001, the model achieves 91.56% accuracy, indicating stable learning but possibly slower convergence. At 0.01, accuracy drops to 90.58%, which may indicate instability or ineffective learning. At 0.1, accuracy is 91.23%, which is slightly better but still lower than the optimal learning rate of 0.001.

5.1.2. Analysis for Different Min Probability Thresholds for Client Selection and Noise Levels

In our proposed method, we aim to minimize client repetition frequency while varying noise levels to enhance the robustness and generalization of the model. Performance indicator analysis from Table 7 and Figures 3 and 4 shows that a lower Min Probability Threshold offers superior validation loss and correctness under less noisy conditions, specifically at 0.01. However, when there is more noise, model performance decreases notably especially with higher thresholds applied. This means keeping a lower frequency of choosing clients not only supports having different types of clients but also assists in lessening the bad effects that noise might have on how well the model works. In essence, our method highlights how important it is to find a good mix between diversity in picking clients and managing noise. It suggests that aiming for lower repeat frequencies can result in more dependable results under differing working situations. This capacity to adapt is essential for strong performance in practical applications where client behaviors and data quality can often vary.

Table 7. Performance metrics for different Min Probability Thresholds and Noise Levels.

Min Probability Threshold	**Noise Level**	**Val Loss**	**Val Accuracy**
0.01	0.001	0.250866	0.928939
	0.01	0.248658	0.927273
	0.05	0.195635	0.935303
	0.1	0.38984	0.901212
	0.2	2.405279	0.831515
0.05	0.001	0.250337	0.932879
	0.01	0.258329	0.927273
	0.05	0.24644	0.921667
	0.1	0.352063	0.915303
	0.2	2.761476	0.801364
0.1	0.001	0.909531	0.801667
	0.01	0.950542	0.781667
	0.05	1.153836	0.764848
	0.1	3.187835	0.635303
	0.2	19.31784	0.423636

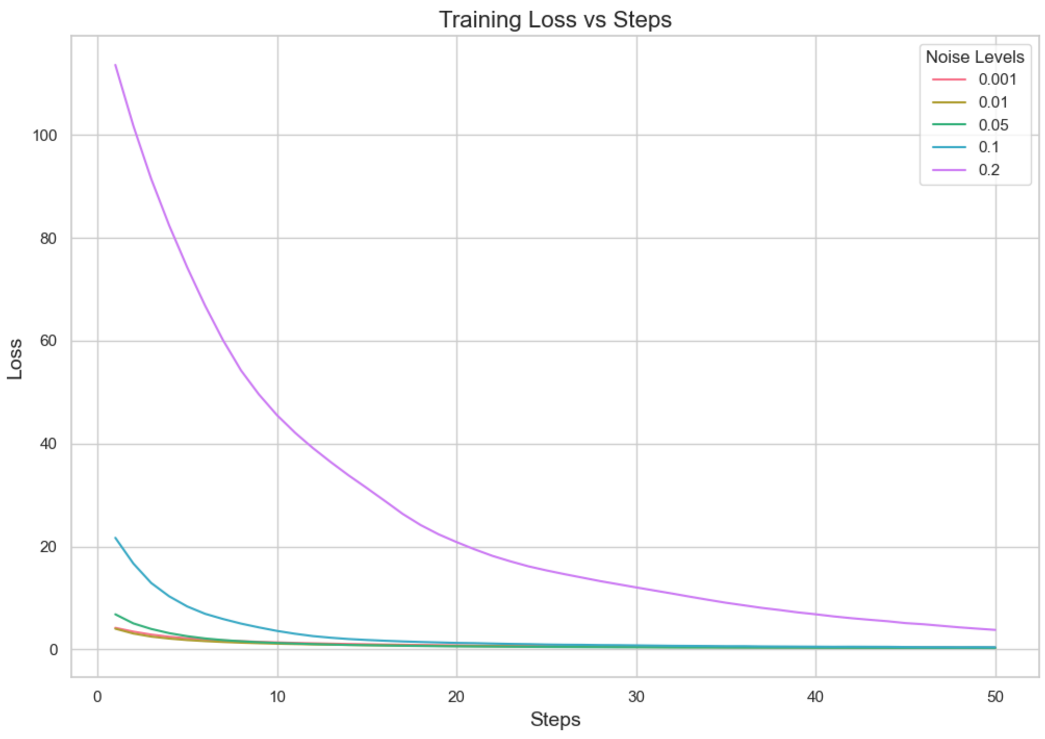

Figure 3. Noise level loss.

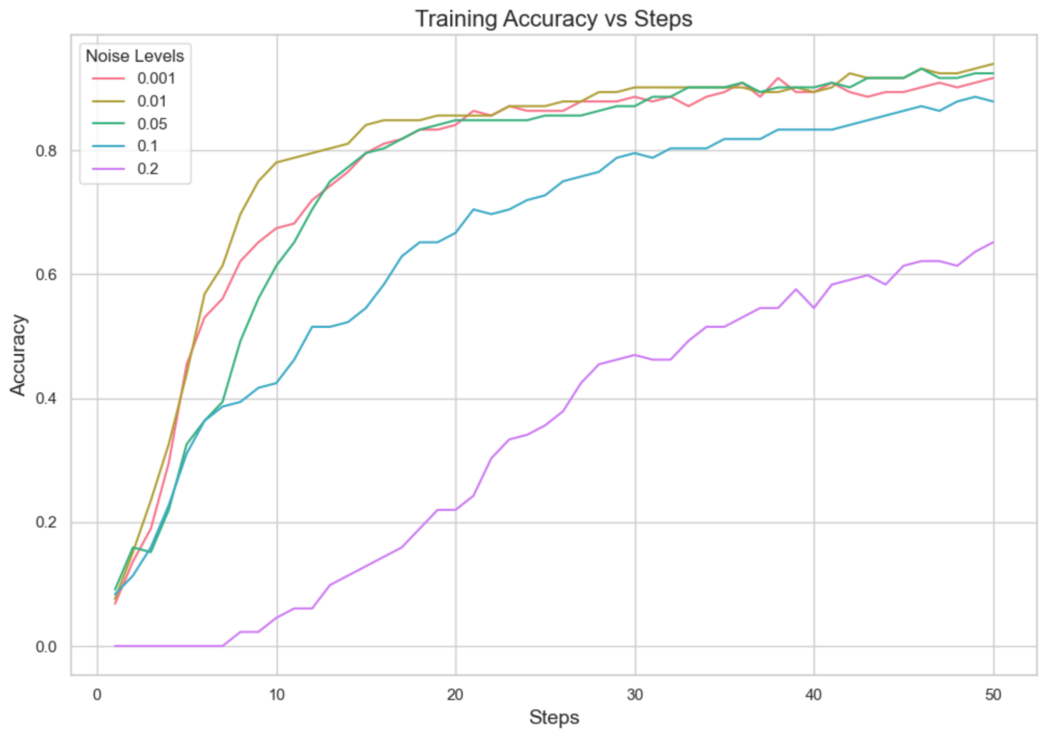

Figure 4. Noise level accuracy.

5.2. Performance Comparison of Prediction Models

The server receives updates from clients after they have trained their local models during the fit round. A fit round consists of 100 global iterations with 20 local iterations. In this case, it received three results; this implies that three out of ten sampled clients took part in this round. The server reports details of the training progress, such as loss and accuracy metrics. In the round of evaluation, the server assesses the global model's performance on data from sampled clients. Here, it sampled two out of ten clients. The log shows that the results from two clients have been received by the server. The check-up gives an idea about how well the global model can be applied to the data of various clients.

The method proposed in Table 8 produces the best outcomes among all models because it achieved a maximum Train Accuracy of 0.70, which implies a successful learning process. Also, this model produced a minimum Train Loss value of only 0.0034, showing efficient training. This same model also offers the highest Eval Accuracy result for Client 1 with a score of 0.90 and a notable Eval Accuracy result of about 0.80 for Client 2. However, the greater Eval Loss for Client 1 (8.5700×10^{-4}) shows some variation in performance. The MILSTM-CEFL model also works well, with a high Eval Accuracy (0.80) for Client 1 and very low Eval Loss for Client 2 (3.8304×10^{-6}), showing accurate predictions. In general, including attention mechanisms seems to improve model performance greatly, as shown by our proposed method.

Table 8. Performance Metrics for Prediction Models.

Model	Train Accuracy	Train Loss	Eval Accuracy (Client 1)	Eval Loss (Client 1)	Eval Accuracy (Client 2)	Eval Loss (Client 2)
TrustFedGRU [28]	0.63	0.0062	0.80	2.0606×10^{-4}	0.70	2.1974×10^{-5}
FedBi-GRU [29]	0.55	0.0048	0.60	1.8800×10^{-4}	0.50	0.0545
MILSTM-CEFL [26]	0.67	0.0069	0.80	2.4292×10^{-5}	0.70	3.8304×10^{-6}
BiLSTM [27]	0.55	0.0056	0.60	5.9681×10^{-5}	0.70	4.4949×10^{-6}
BiLSTM-GRU [30]	0.68	0.0054	0.80	2.0387×10^{-5}	0.60	1.6965×10^{-5}
Proposed Method	0.70	0.0034	0.90	8.5700×10^{-4}	0.80	1.6920×10^{-5}

5.3. *Performance Comparison of Offloading Algorithms*

In our study, we tested four offloading algorithms: Worst Fit, Best Fit, First Fit, and the Hybrid Offloading algorithm. We looked into how well these methods work for keeping power usage stable across various edge servers (Table 9). For analyzing the different algorithms, we used consistency indicators, power efficiency rate changes, and other fluctuations related to resource usage and observations specific to each algorithm.

Table 9. Showing the standard deviation of power consumption for each edge server with Hybrid Offloading, First Fit, Best Fit, and Worst Fit algorithms.

Edge Server	Algorithm			
	Hybrid	First	Worst Fit	Best Fit
EdgeServer_1	0	0	0	0
EdgeServer_2	0	0	0	0
EdgeServer_3	24.484	24.756	25.693	24.484
EdgeServer_4	24.485	24.756	25.693	26.291
EdgeServer_5	12.533	20.489	19.104	5.933
EdgeServer_6	0	0	0	10.449

5.3.1. Consistency

In terms of power consumption stability, the Hybrid Offloading Algorithm seems to be more consistent. Both EdgeServer_1 and EdgeServer_2 maintain a uniform power usage of 265 units in every step and algorithm. This implies that these servers can handle steady workloads with dependability. Moreover, EdgeServer_6 has a steady use of nearly 177.18 units for all algorithms apart from Best Fit. In this case, we can observe slight ups and downs up to approximately 200 units for the Best Fit algorithm. This suggests that hybrid offloading and first fit are more stable in handling load on EdgeServer_6 when compared against the best-fit method.

5.3.2. Power Efficiency

Regarding power efficiency, EdgeServer_5 generally has the least power consumption. It uses an average of around 74.51 units across all algorithms, showing occasional spikes in use but mostly staying at this level. A steady pattern of low usage indicates good workload management, particularly with the Hybrid Offloading and First Fit algorithms. The Hybrid Offloading Algorithm is very steady for all servers, with little variation. This makes it the best in terms of power efficiency overall. The First Fit Algorithm is almost as stable, but EdgeServer_4 always shows a bit higher power usage. The Worst Fit Algorithm is not as efficient. It displays more noticeable changes, particularly for EdgeServer_3—it has the highest power consumption when busy and lowest when idle among all servers; the case of EdgeServer_4 and 5 shows similar behavior but not to the same extent as the previous one. The Best Fit Algorithm appears to be mostly steady. However, several times it shows some unsteadiness with swings of up to 200 units in the case of EdgeServer_6.

5.3.3. Fluctuations

The power consumption differences are very small with the Hybrid Offloading Algorithm, keeping the performance stable for all servers. The First Fit Algorithm is similar and shows only slightly more variance, especially on EdgeServer_4. The Worst Fit Algorithm displays large fluctuation in power consumption, particularly on EdgeServer_3, EdgeServer_4, and EdgeServer_5. This is not as predictable and might be an inefficient distribution of workload. The Best Fit Algorithm has some ups and downs, particularly in EdgeServer_6, where power consumption increases to 200 units. It implies that there are fluctuations in how efficiently servers handle their loads.

5.3.4. Algorithm-Specific Observations

The power usage in all servers using the Hybrid Offloading Algorithm is stable and effective, with small fluctuations. It works well for situations needing steady performance. The First Fit Algorithm also shows efficient work, but only slightly less stability than the Hybrid Offloading Algorithm. The Worst Fit Algorithm displays the greatest variety and potential inefficiencies, particularly in handling workloads of EdgeServer_5. At last, even if generally stable, the Best Fit Algorithm does have a few inefficiencies, such as significant alterations in specific servers like EdgeServer_6, suggesting an enhancement requirement for workload management.

5.4. Performance Comparison of Federated Learning Algorithms

Our experiment is based on the EMNIST [43] dataset and the Crop Prediction dataset provided publicly by the Indian Chamber of Food and Agriculture (ICFA) under the license CC BY 3.0 IGO. The parameters for the model training are carried out as given in Table 10. We combine D4PG and Federated Learning in a model we propose, using decentralized training together with enhanced value estimation. In every iteration of Federated Learning, client devices calculate local updates while preserving the privacy of data. Data privacy in this model is maintained by incorporating differential privacy via the addition of Gaussian noise to the model weights during each iteration of Federated Learning as in Equation (46). Table 7 presents the impact of different noise levels on model performance, showing how higher noise levels preserve privacy but lead to a decrease in model accuracy. In Federated Learning, the Min Probability Threshold controls client selection, which also influences how noise and data are aggregated. Thus, maintaining privacy involves finding an optimal balance between noise (for privacy) and performance (accuracy) [44]. These updates are aggregated by a central server using D4PG. The training curves in Figures 5–8 were smoothed using a parameter of 0.99 in TensorBoard. This high level of smoothing was chosen to emphasize long-term trends and reduce the visual impact of short-term fluctuations.

Table 10. Parameter Settings for Training DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG Models.

Parameter	Non-IID EMNIST	IID EMNIST	Crop Prediction
G (Global rounds)	200	200	50
NUM_CLIENTS	10	10	10
BATCH_SIZE	32	32	32
L (Local training iterations)	20	20	10
LR (Learning rate)	0.05	0.05	0.001
MU (Regularization Strength)	0.01	0.01	0.01
EPSILON (Exploration rate)	0.1	0.1	0.1
TARGET_UPDATE_INTERVAL	5	5	5
MAX_BUFFER_SIZE	10,000	10,000	10,000
ALPHA (Priority exponent)	0.6	0.6	0.6
BETA (Importance sampling exponent)	0.4	0.4	0.4
GAMMA (Discount factor for future rewards)	0.99	0.99	0.99

In Figure 5, all the algorithms have a high loss value at first, but DQN and DDQN are showing a steady decrease. D4PG is performing the best as it reaches the lowest loss around 80 rounds. The rest of the algorithms converge at a slower pace, but they keep reducing their loss and come close to each other around 160 rounds.

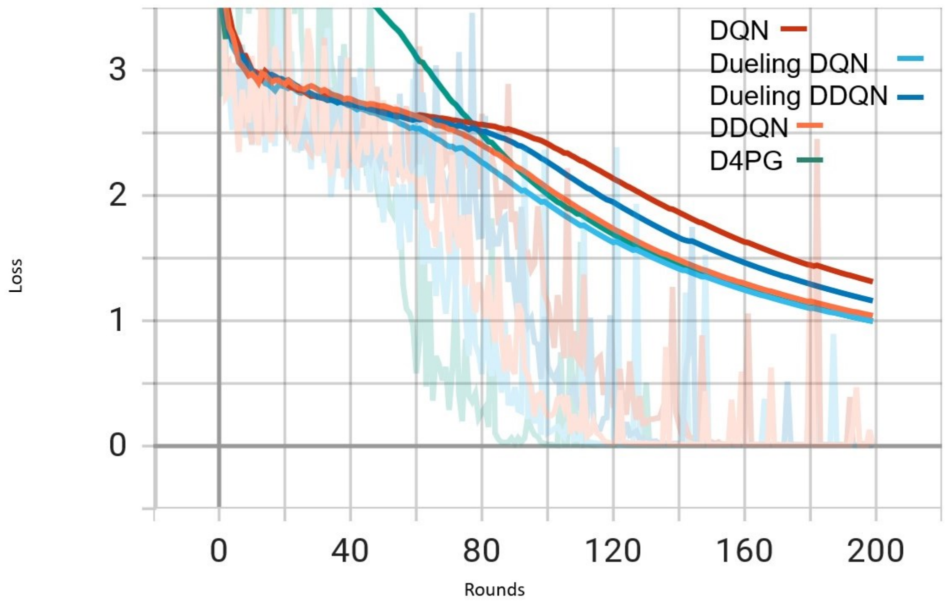

Figure 5. Comparison of loss: DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG Models on IID EMNIST Dataset.

In Figure 6, D4PG remains at the topmost position with an overall accuracy higher than 0.8. DQN and DDQN display similar patterns of accuracy improvement, reaching a steady state around 0.75 as iterations proceed further. Every algorithm shows a clear increase in accuracy, and D4PG maintains its lead during the entire training process.

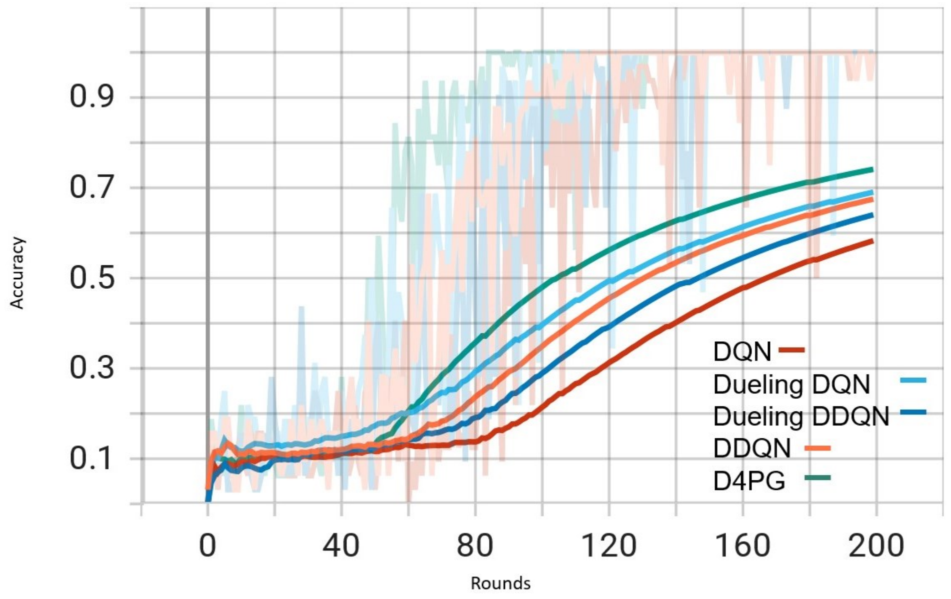

Figure 6. Comparison of accuracy: DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG Models on IID EMNIST Dataset.

Figure 7 shows that all the algorithms begin with a high loss of around 2.4. Over time, these loss values drop and become less significant as training continues. It is interesting to observe that D4PG experiences a swift and consistent decrease in its loss compared to other algorithms. Around 50 rounds, D4PG's loss stabilizes more rapidly and at a lesser value compared to the others. It shows its better performance in reducing loss soon after starting the training process. At the end of 200 rounds, D4PG keeps a lower loss value near 0.35 versus other algorithms which converge around 0.4.

In Figure 8, D4PG exhibits a better performance trajectory as well. It starts at low accuracy, but then quickly improves as it reaches higher values faster than other methods. After about 50 rounds, the accuracy of D4PG becomes better than other algorithms and keeps growing at a regular pace. When we finish 200 rounds, D4PG reaches the topmost accuracy with around 0.85 while other algorithms settle near 0.8; this implies that D4PG not only learns quicker but also attains greater final precision.

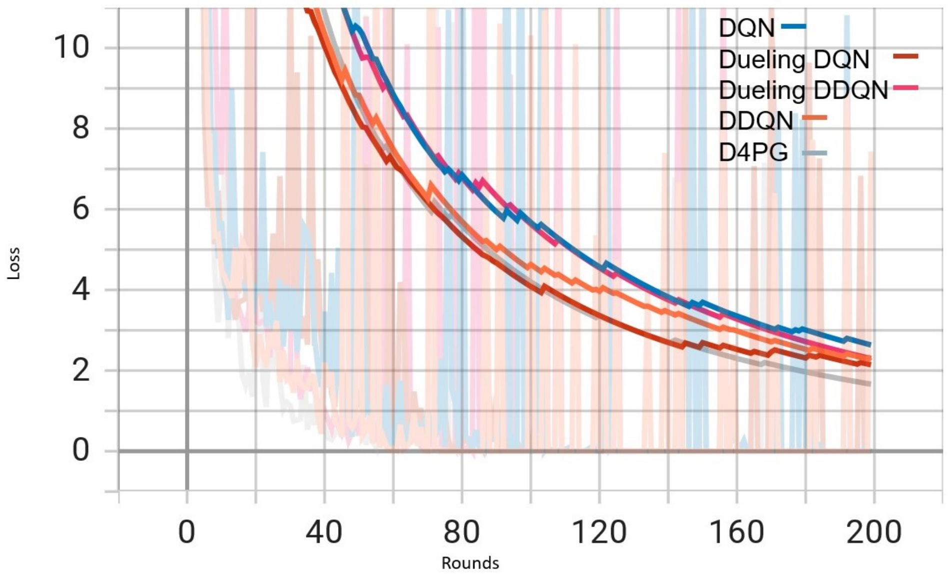

Figure 7. Comparison of loss: DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG models on Non IID EMNIST Dataset.

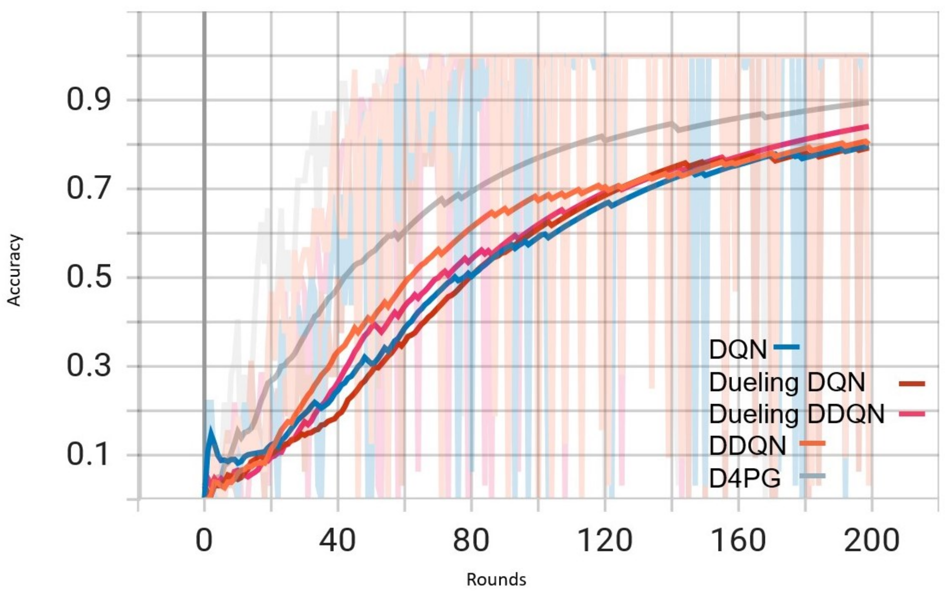

Figure 8. Comparison of accuracy: DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG models on Non IID EMNIST Dataset.

5.4.1. Analysis Between IID and Non-IID Datasets

Table 11 displays performance metrics of five varying Reinforcement Learning models, namely DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG when applied to the Non-IID EMNIST dataset. The values for performance metrics are shown in decimal numbers ranging from zero to one. DQN has the poorest performance with recall, F1 score, and accuracy of just 0.75, but it also has the highest loss at 0.84. For DDQN, we see an improvement with recall, F1 score, and accuracy, all at 0.82; loss is down to only being as high as 0.64. Dueling DQN shows nearly the same scores as DDQN, but it has a better loss of 0.60. For Dueling DDQN, the metrics are a little lower, with recall at 0.79 and loss being 0.77. D4PG performs well with a recall of 0.83 and a loss of 0.62.

Table 11. Performance Metrics for DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG Models on the Non-IID EMNIST Dataset.

Model	Recall	Precision	F1 Score	Accuracy	Loss
DDQN [31]	0.8689	0.8722	0.8689	0.8689	0.50897
Dueling DDQN [34]	0.8675	0.8693	0.8668	0.8675	0.53662
DQN [33]	0.8885	0.8945	0.8888	0.8885	0.47522
Dueling DQN [32]	0.8852	0.8898	0.8847	0.8852	0.47577
Proposed(D4PG)	0.8918	0.8941	0.8916	0.8918	0.43822

According to the table with results from different models on IID EMNIST in Table 12, the D4PG model has top recall and precision at 0.82 and 0.83, respectively, showing its outstanding ability to correctly find positive instances and keep false positives low. Additionally, it also displays the highest accuracy score of 0.82 which shows how well-rounded this model is in correctly classifying cases overall. The F1 score is at 0.82, showing the balance between precision and recall. We can see that the D4PG model has the lowest loss value of all the models listed here, 0.61. This low loss shows that the model is converging well and strongly. Together, these metrics confirm D4PG as superior; it outperforms other models, which makes D4PG the most dependable and efficient model for this task on the IID EMNIST dataset.

Table 12. Performance Metrics for DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG Models on the IID EMNIST Dataset.

Model	Recall	Precision	F1 Score	Accuracy	Loss
DQN	0.76	0.77	0.75	0.76	0.86
DDQN	0.80	0.80	0.80	0.80	0.70
Dueling DQN	0.77	0.78	0.76	0.77	0.79
Dueling DDQN	0.76	0.78	0.75	0.76	0.88
Proposed (D4PG)	0.82	0.83	0.82	0.82	0.61

5.4.2. Analysis with High Number of Classes in Non-IID EMNIST Dataset

The analysis of Deep Reinforcement Learning models—DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG—on the 62 classes of the Non-IID EMNIST Dataset reveals distinct performance characteristics across various metrics. The accuracy versus training steps graph Figure 9 shows that all models initially improve rapidly, but D4PG consistently achieves the highest accuracy as training progresses. Also, the loss graph Figure 10 shows the consistent decrease in model losses in the D4PG model. This observation aligns with the performance metrics Table 13, where D4PG demonstrates superior results with a recall of 0.6106, precision of 0.5811, F1 score of 0.5754, and accuracy of 0.6106 while also maintaining the lowest loss value of 1.6810. These results highlight D4PG's robustness and adaptability, likely due to its ability to manage continuous actions effectively, making it well-suited for complex, non-IID datasets. In contrast, DDQN and Dueling DDQN exhibit slightly less robust performance, with DDQN achieving a recall and accuracy of 0.5569 and a higher loss of 1.9808. Dueling DDQN also fails to significantly outperform its basic counterpart, suggesting that the benefits of the dueling architecture are not fully realized in this scenario. Meanwhile, Dueling DQN shows better performance than DDQN, with an accuracy of 0.5842 and a loss of 1.8222, indicating some advantage of the dueling architecture, although this advantage is context-dependent and may require further optimization. DQN, the simplest model, serves as a benchmark, showing reasonable recall and precision but higher loss (1.7372), indicating less effective optimization compared to D4PG. This analysis underscores the importance of advanced techniques such as prioritized replay and continuous action handling, as demonstrated by D4PG, in enhancing learning outcomes in challenging Federated Learning contexts. Overall, the differences in model performance emphasize the critical role of architectural choices in effectively addressing complex data scenarios.

Table 13. Performance Metrics for DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG Models on the 62 classes Non-IID EMNIST Dataset.

Model	Recall	Precision	F1 Score	Accuracy	Loss
DQN	0.6061	0.5659	0.5659	0.6061	1.7372
DDQN	0.5569	0.5386	0.5210	0.5569	1.9808
Dueling DQN	0.5842	0.5603	0.5496	0.5842	1.8222
Dueling DDQN	0.5640	0.5438	0.5377	0.5640	2.0092
Proposed(D4PG)	0.6106	0.5811	0.5754	0.6106	1.6810

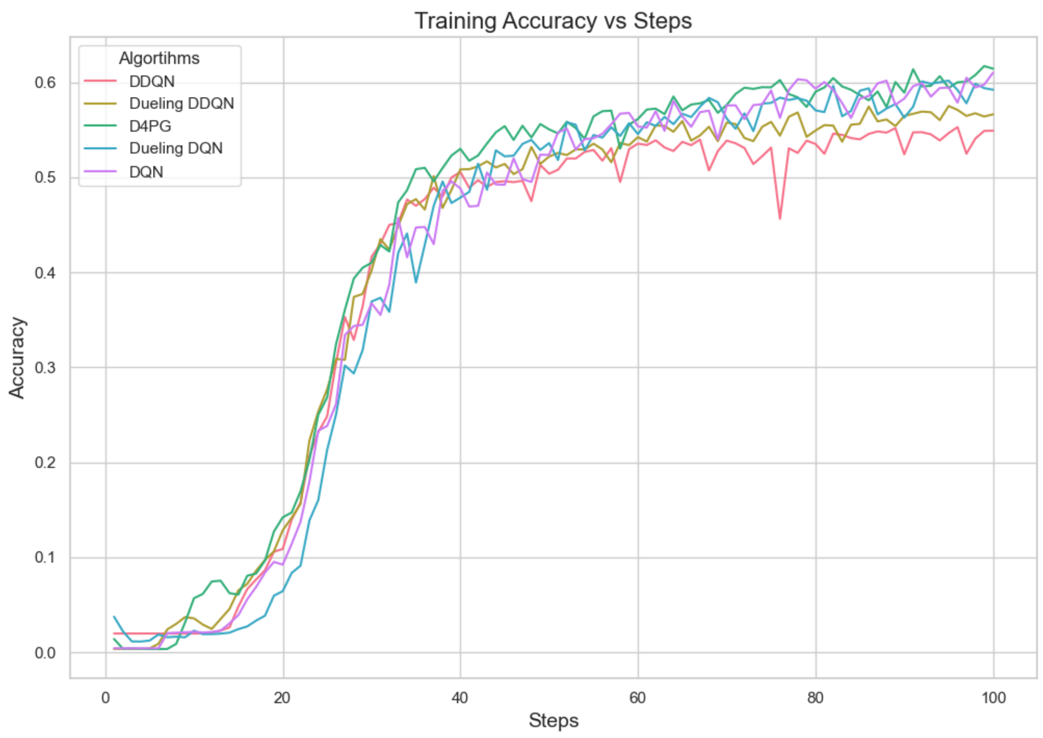

Figure 9. Comparison of accuracy: DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG models on 62 classes Non-IID EMNIST Dataset.

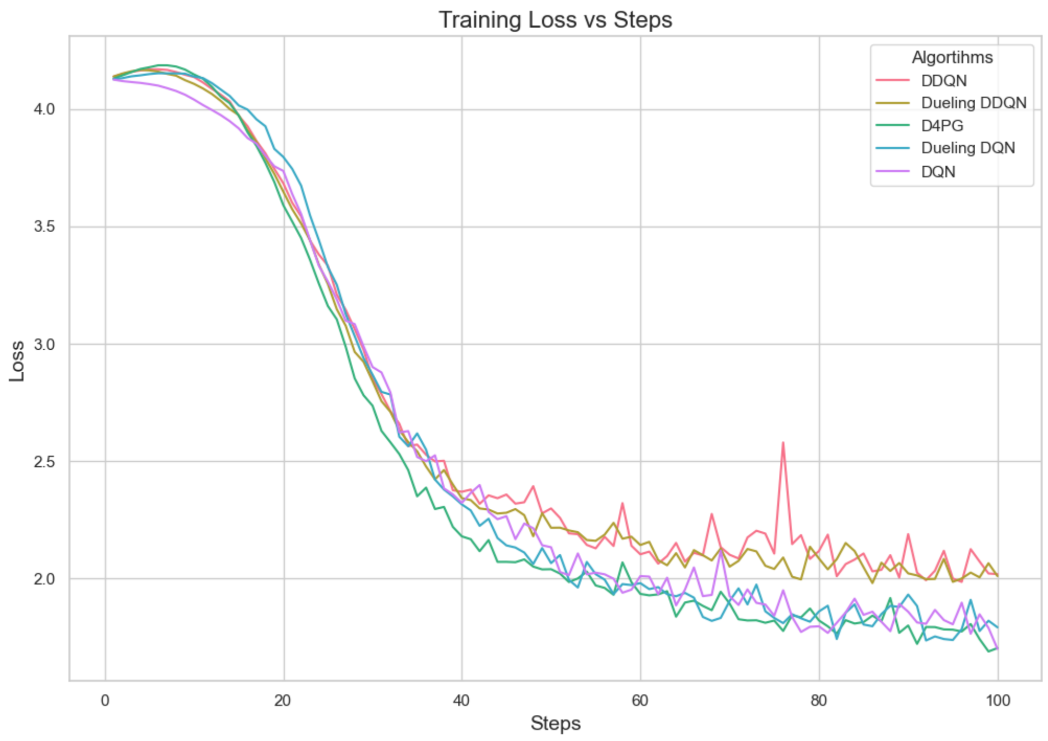

Figure 10. Comparison of loss: DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG models on 62 classes Non-IID EMNIST Dataset.

5.4.3. Analysis of Different Number of Clients

Figures 11–16 show the analysis of training accuracy and training loss in different numbers of clients. The analysis of the performance metrics (Table 14) for the DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG models on the Non-IID EMNIST dataset with varying numbers of clients (10, 20, and 50) reveals significant insights into how these models handle Federated Learning scenarios with differing data distributions. The DQN model shows impressive performance with 10 and 20 clients, achieving high accuracy and low loss, indicating its robustness in dealing with smaller client pools. However, its performance slightly declines with 50 clients, suggesting challenges in managing increased data heterogeneity. The DDQN model performs reasonably well with 10 and 20 clients but struggles with 50 clients, showing a noticeable drop in accuracy and an increase in loss, highlighting its limitations in scaling effectively with more clients. The Dueling DQN model consistently performs well across all client numbers, especially excelling with 50 clients, where it achieves high accuracy and the lowest loss among all models. This demonstrates its ability to leverage larger, diverse datasets effectively. Conversely, the Dueling DDQN model starts strong but shows a slight dip in performance with 20 clients, stabilizing with 50 clients but not achieving the same level of effectiveness as the Dueling DQN model. The D4PG model emerges as the most adaptable across different client sizes, maintaining strong performance with 10 clients and achieving top results with 20 and 50 clients, where it records the highest accuracy and minimal loss. This suggests that the D4PG model is particularly well suited for Federated Learning environments, effectively handling the challenges posed by non-IID data distributions across a large number of clients. In summary, while all models have their strengths, the D4PG and Dueling DQN models stand out for their robustness and adaptability, making them ideal choices for Federated Learning applications that require managing diverse and large datasets.

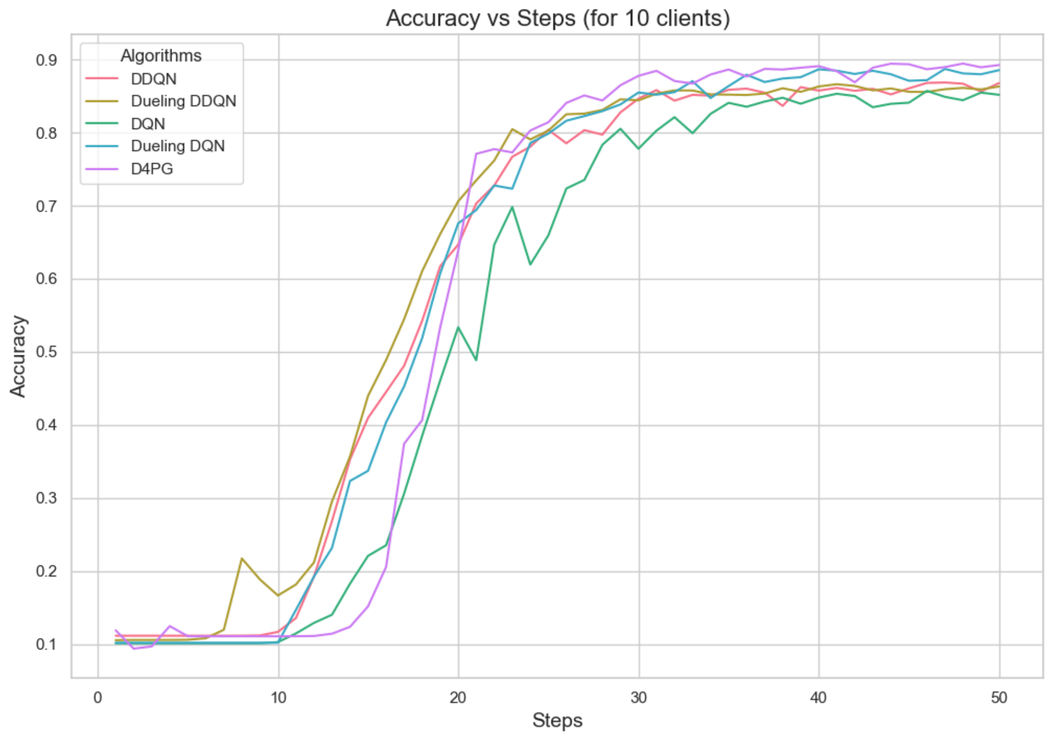

Figure 11. Accuracy for 10 clients.

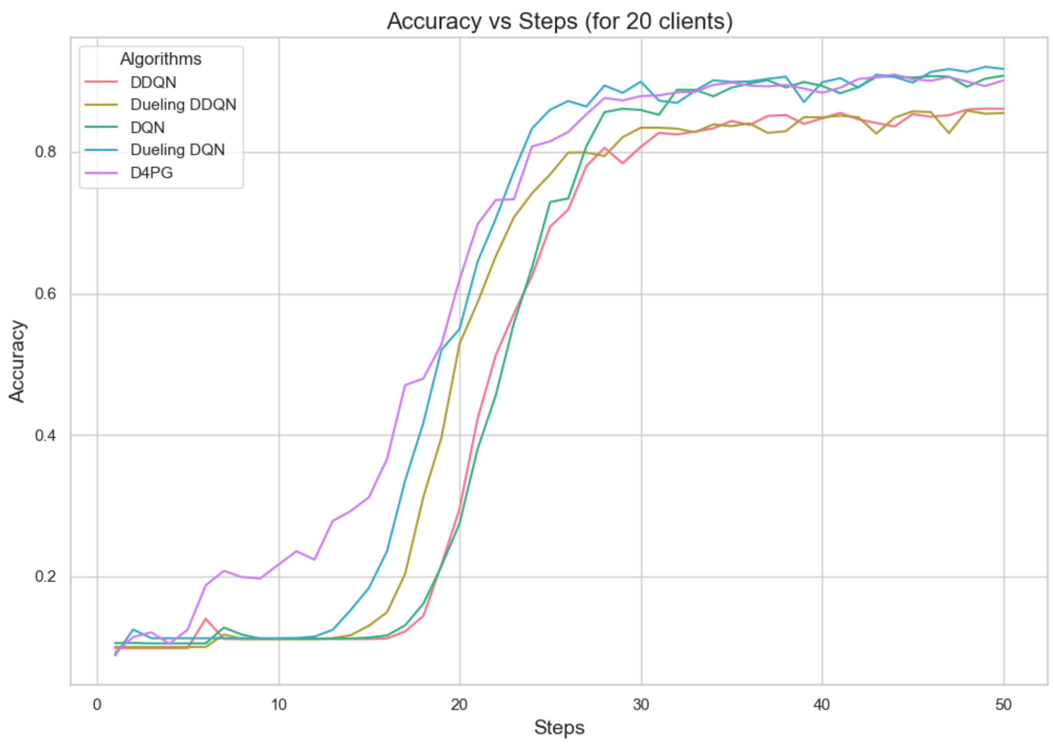

Figure 12. Accuracy for 20 clients.

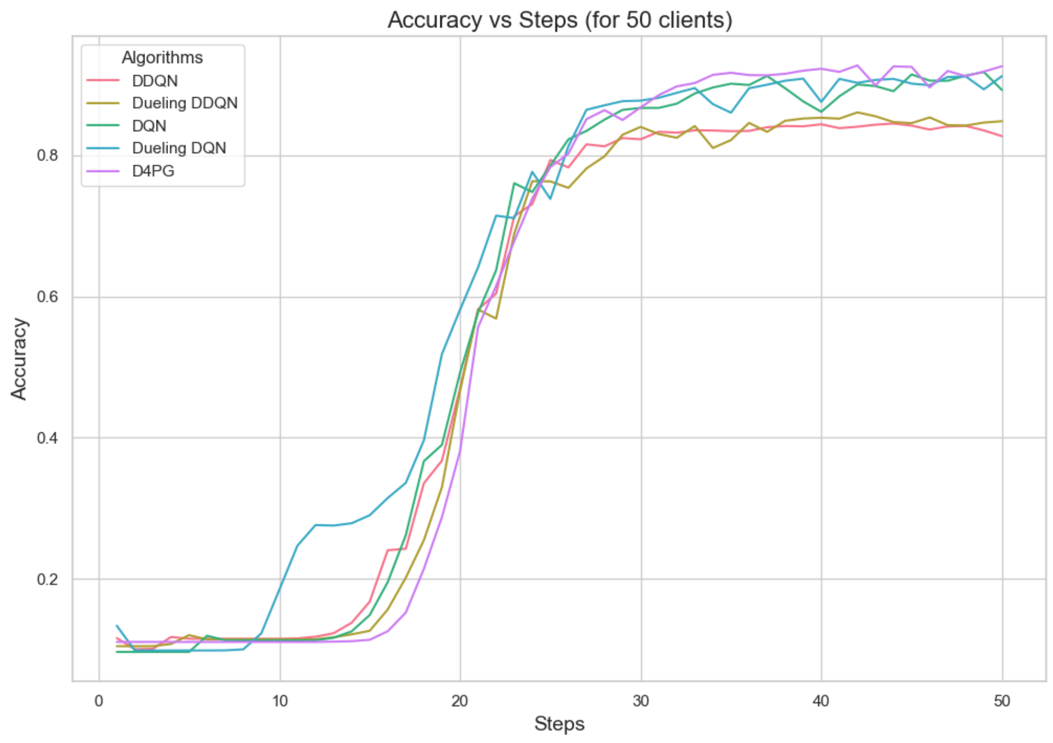

Figure 13. Accuracy for 50 clients.

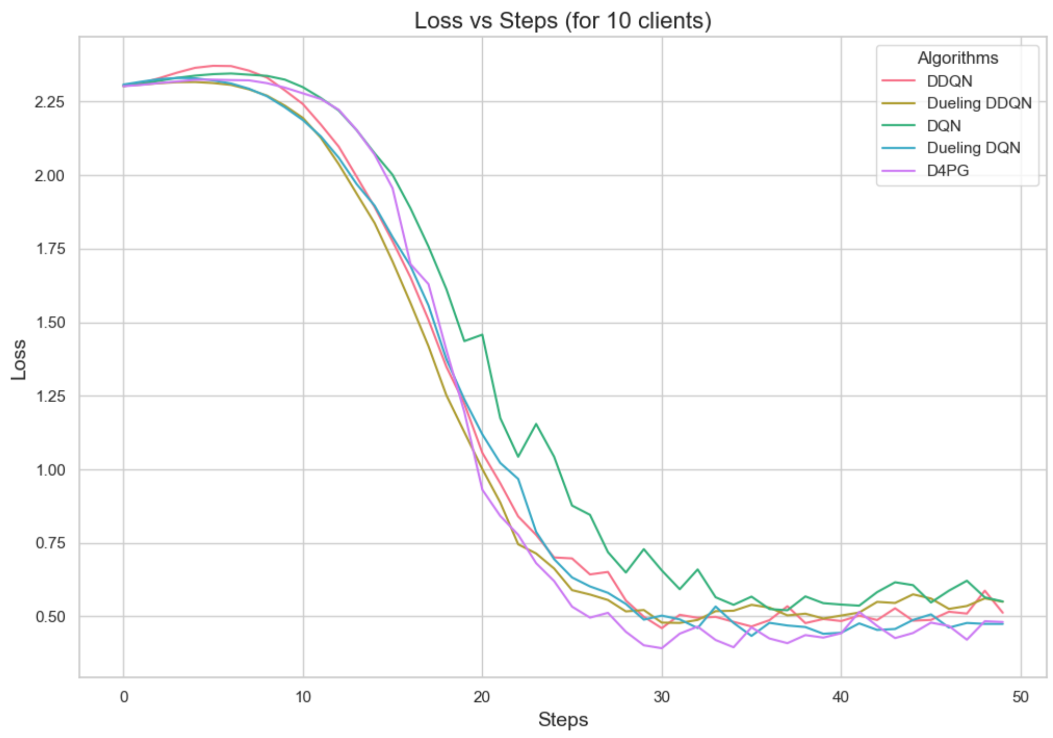

Figure 14. Loss for 10 clients.

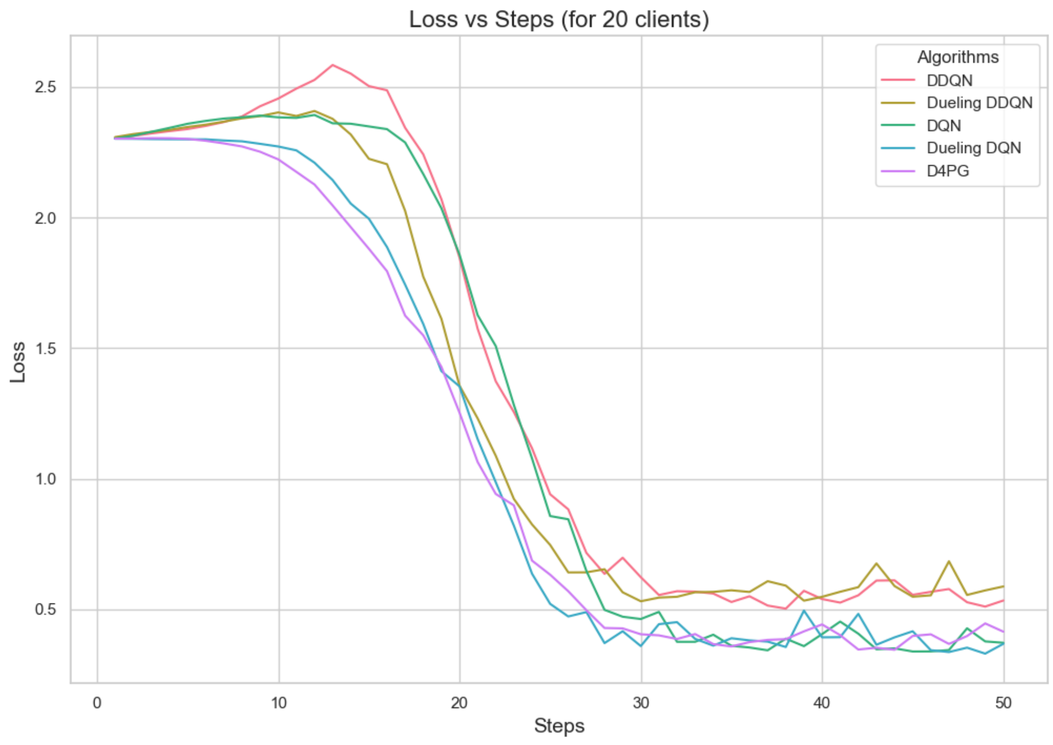

Figure 15. Loss for 20 clients.

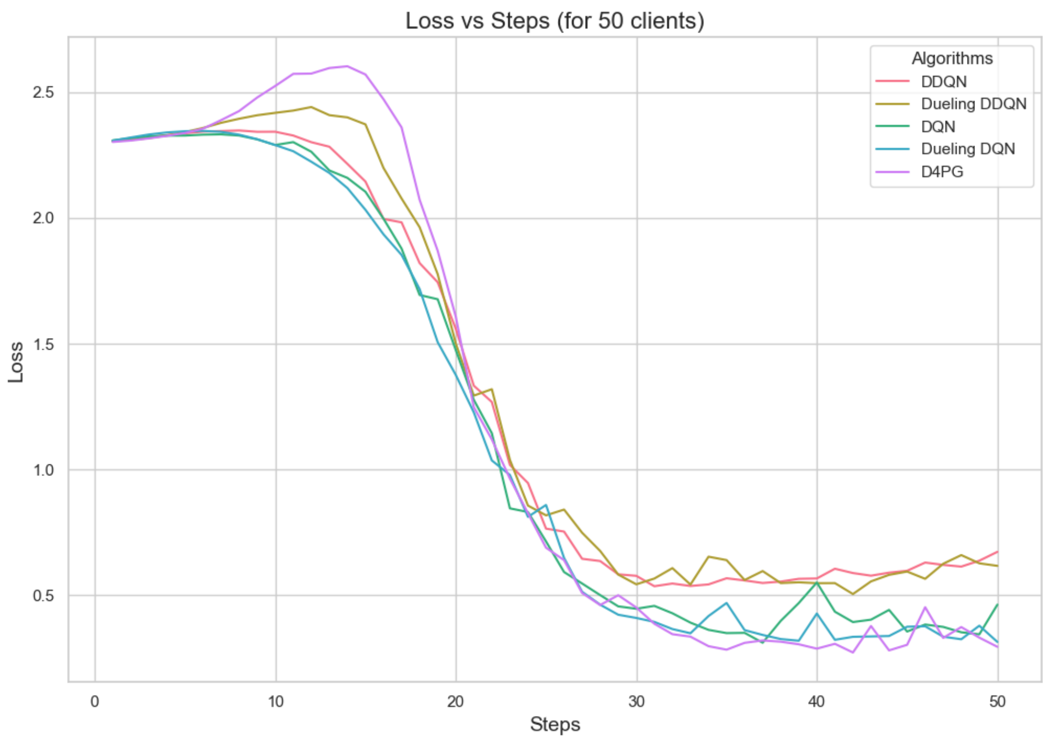

Figure 16. Loss for 50 clients.

Table 14. Performance Metrics for DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG Models on the different number of clients Non-IID EMNIST Dataset.

Model	Client	Recall	Precision	F1 Score	Accuracy	Loss
DDQN	10	0.8689	0.8722	0.8689	0.8689	0.50897
DDQN	20	0.8692	0.8710	0.8687	0.8692	0.50758
DDQN	50	0.8285	0.8568	0.8336	0.8285	0.64657
Dueling DDQN	10	0.8675	0.8693	0.8668	0.8675	0.53662
Dueling DDQN	20	0.8505	0.8542	0.8504	0.8505	0.60313
Dueling DDQN	50	0.8494	0.8532	0.8486	0.8494	0.61122
DQN	10	0.8885	0.8945	0.8888	0.8885	0.47522
DQN	20	0.9026	0.9043	0.9019	0.9026	0.40192
DQN	50	0.8788	0.8907	0.8769	0.8788	0.53618
Dueling DQN	10	0.8852	0.8898	0.8847	0.8852	0.47577
Dueling DQN	20	0.9001	0.9121	0.9023	0.9001	0.40195
Dueling DQN	50	0.9161	0.9195	0.9164	0.9161	0.30606
D4PG (proposed)	10	0.8918	0.8941	0.8916	0.8918	0.43822
D4PG (proposed)	20	0.9095	0.9149	0.9095	0.9095	0.40091
D4PG (proposed)	50	0.9192	0.9223	0.9192	0.9192	0.32927

5.4.4. Analysis with Crop Prediction Dataset

Further, we tested the proposed model on the Crop prediction dataset. The Figures 17 and 18 show the model training accuracy and training loss in the crop prediction dataset. In these models that were tested, it is shown (Table 15) that D4PG has the best performance with a 92.86% accuracy rate and the highest weighted average F1-score of 0.93. This shows that D4PG not only has the greatest total accuracy, but it also keeps up its performance in different classes.

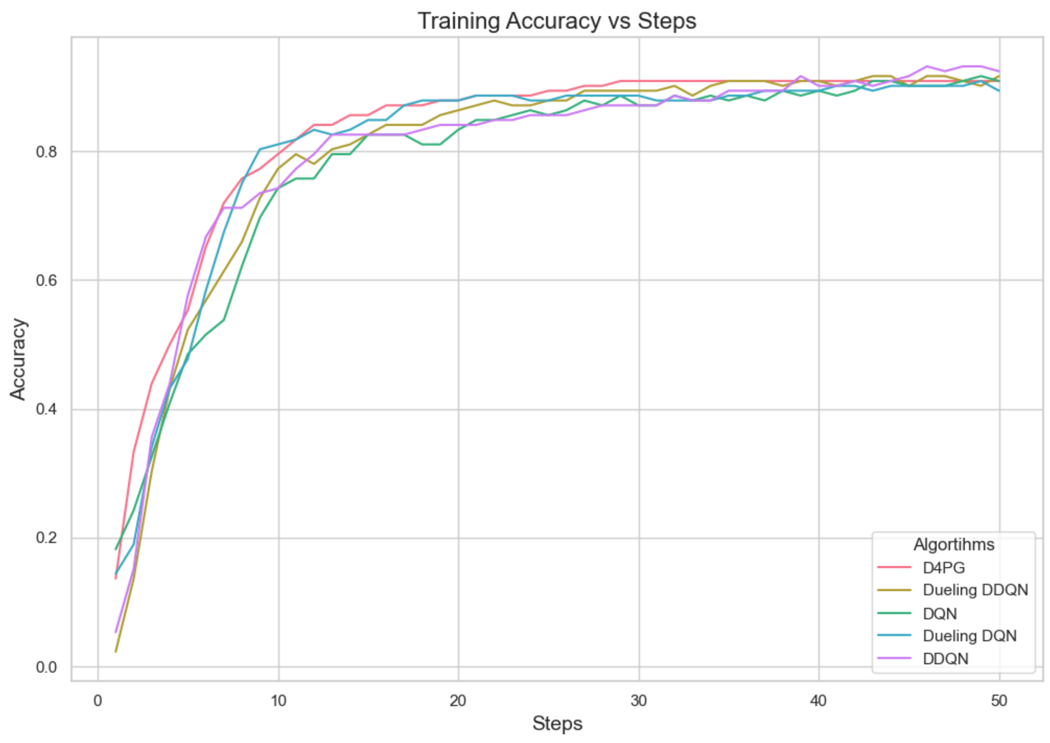

Figure 17. Training accuracy for crop prediction dataset.

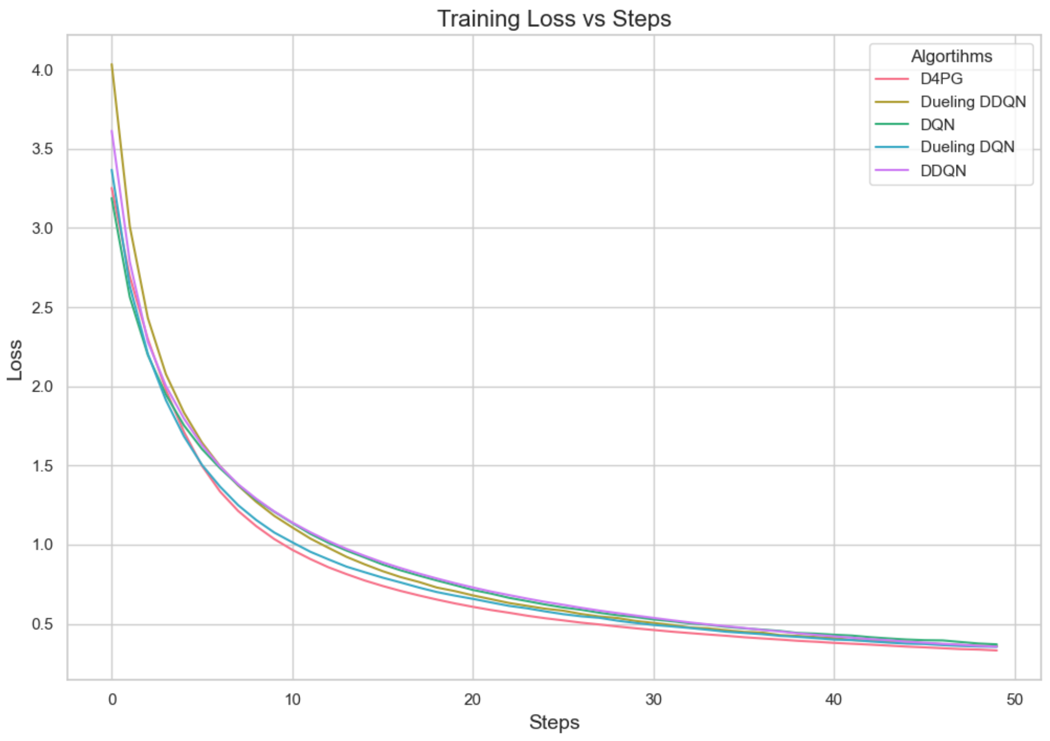

Figure 18. Training loss for crop prediction dataset.

Dueling DDQN and DQN perform well, too, having accuracies of 91.88% and 91.56%, respectively, with weighted average F1-scores of about 0.92 and 0.91. This shows that these models are also dependable but a bit less effective than the D4PG model. DDQN performs well: it has an accuracy of 90.91% and a weighted average F1-score of about 0.91.

On the other hand, the Dueling DQN model has the worst performance among all tested models. It shows an accuracy rate of 89.29% and a weighted average F1-score of 0.88. The drops in performance for some classes are noticeable, especially class number

13, which affects its overall functioning power but still holds relatively high accuracy and performance in other classes, too.

Table 15. Performance Metrics for DQN, DDQN, Dueling DQN, Dueling DDQN, and D4PG Models on Crop Prediction Dataset.

Model	**Accuracy**	**Macro Avg**			**Weighted Avg**	
		Precision	**Recall**	**F1-Score**	**Precision**	**Recall**
DDQN	90.91%	0.91	0.91	0.90	0.92	0.91
Dueling DDQN	91.88%	0.92	0.93	0.92	0.92	0.92
DQN	91.56%	0.92	0.92	0.91	0.92	0.92
Dueling DQN	89.29%	0.90	0.90	0.89	0.90	0.89
Proposed(D4PG)	92.86%	0.93	0.93	0.93	0.93	0.93

5.4.5. Analysis of the Reward per Episode Graph

The reward is computed out of a combined reward score that takes into account fairness, resource usage, and accuracy. The fairness part of the reward is calculated by obtaining the class distribution data. Next, it uses these data to find out how fairly resources or outcomes are distributed among different classes. The resource reward uses the average of resources used, showing how efficient the use of all resources is. After calculating these two rewards, they are added together with the accuracy score. Each is multiplied by its weight. The total reward comes from adding up these weighted parts, which gives us an even measurement for performance, taking into account fairness, usage of resources, and accuracy in prediction results. Figure 19 displays reward per episode for 200 episodes and shows the learning process of different Reinforcement Learning algorithms in a Federated Learning setting.

D4PG and Dueling DDQN present the best and steady rewards that show they are superior in performance and learning capability. The graph shows that the D4PG algorithm has the maximum rewards. It starts slightly above 3.0 and increases consistently to about 4.8, which implies strong learning performance with moderate volatility. The Dueling DDQN algorithm also experiences significant improvement. It starts slightly below 3.5 and gradually increases to approximately 4.5, indicating effective learning over time with a bit more oscillations compared to D4PG. The DDQN demonstrates an obvious increase from below 3.0 up until just above 4.0, indicating good performance with moderate stability.

The rewards of DQN and Dueling DQN are lower with more fluctuations, indicating that learning and stability are not as effective. The DQN and Dueling DQN algorithms have a beginning point with lower rewards around 2.0 and more turbulent behavior. DQN increases to about 3.0 while the Dueling DQN reaches nearly 3.5; both show considerable changes, suggesting unstable learning occurs. The polynomial trend lines support these observations, pointing out the overall trends and consistent performance of every algorithm across rounds.

In the evaluation of reward per episode for Crop Prediction Dataset Figure 20, the Dueling DDQN algorithm is the best. It reaches an episode maximum of 23.12 and the topmost value of reward per step at 0.463, showing the increased capability of gathering rewards compared to other algorithms in both the overall performance and on a per-step basis. The D4PG algorithm is the next best, with an episode reward of 23.02 and a reward per step value of 0.460, performing quite impressively. The DDQN algorithm is performing quite well. It has an episode reward of 22.55 and a reward per step of 0.452, thus being efficient in gaining rewards. The DQN algorithm seems to be less efficient in this area, with lower numbers both for episode reward 21.78 and the lowest reward every step 0.436. This points to its relative inefficiency compared to the other two methods. On average, Dueling DDQN reaches the top performance among these models, but also shows slightly worse performance than average sometimes, too, because its best model does not always perform equally well.

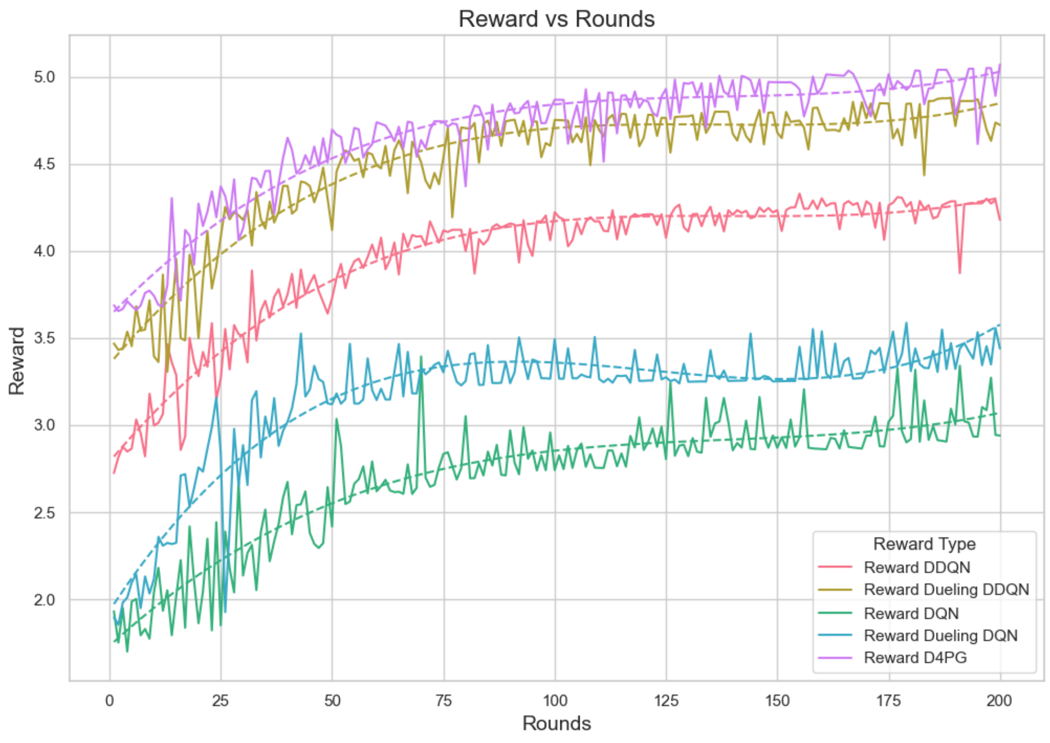

Figure 19. Analysis of the Reward Per Episode Graph for different Reinforcement Learning algorithms in Federated Learning setup.

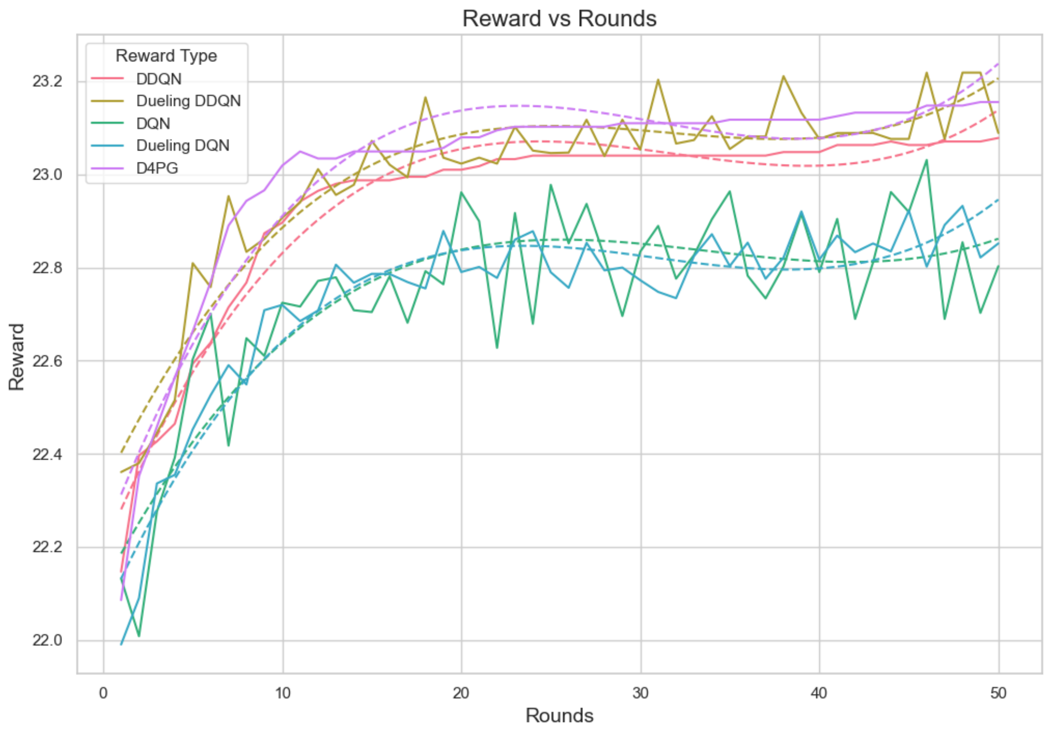

Figure 20. Analysis of the Reward Per Episode Graph for different Reinforcement Learning algorithms in Federated Learning setup for crop prediction dataset.

6. Discussion and Conclusions

This study presents a robust framework for efficient resource management and task scheduling in Agricultural IoT by integrating machine learning-based forecasting with Federated Reinforcement Learning. The proposed hybrid BiLSTM-GRU with the attention mechanism model significantly enhances resource usage prediction, enabling adaptive task scheduling that minimizes energy consumption across edge servers. The ability of our model to optimize resource allocation and task offloading in real time ensures that the system can handle an increasing number of devices without compromising performance. This approach contributes to optimizing resource utilization, offering a promising solution for dynamic task distribution in edge computing environments. The integration of a hybrid offloading algorithm optimizes performance by dynamically distributing tasks based on real-time resource availability. Experimental results validate the effectiveness of this approach, demonstrating notable improvements in scalability, energy efficiency, and system reliability over existing methods. Additionally, we also propose the D4PG based on a Federated Learning algorithm for adjusting the participation of dynamic User Equipment (UE) according to resource availability and data distribution. Among the Reinforcement Learning techniques explored, D4PG outperforms others by reducing loss and increasing accuracy, while the Dueling architectures show exceptional performance during the early stages of learning. While DQN and DDQN exhibit steady performance, they converge more slowly compared to the advanced models. The theoretical insights derived from this research underline the importance of combining machine learning, Reinforcement Learning, and edge computing in solving complex IoT challenges, particularly in dynamic and resource-constrained environments. Furthermore, the proposed hybrid model's ability to handle non-IID data distributions and ensure privacy preservation in federated settings provides a significant advancement in the field of Federated Learning for IoT applications. Our proposed model is particularly well-suited for Agricultural IoT applications due to its ability to process large-scale time-series data and dynamically optimize resource allocation and task scheduling in real time. In smart farming, IoT devices continuously collect data from various sources, such as soil moisture sensors, weather stations, and crop health monitoring systems. Efficiently managing and analyzing this data is crucial for improving crop yield, minimizing resource waste, and ensuring sustainable farming practices. The D4PG in our framework enhances decision-making by enabling adaptive task scheduling and resource allocation across distributed edge nodes in an agricultural setting. This allows for real-time processing and reduced latency, which is essential for applications like automated irrigation, pest control, and precision agriculture. Additionally, Federated Learning ensures data privacy and security by keeping sensitive farm data localized while collaboratively training global models. Beyond Agricultural IoT, our model has broader applicability in various IoT domains where dynamic resource allocation and intelligent task scheduling are critical. For example, efficient traffic management, energy distribution, and public infrastructure monitoring; predictive maintenance, autonomous manufacturing processes, and real-time monitoring of supply chains; remote patient monitoring, resource allocation in hospitals, and adaptive scheduling of medical IoT devices. By enabling scalable, privacy-preserving, and adaptive resource management, this study contributes to the advancement of intelligent edge computing for next-generation IoT applications, providing a solid foundation for future research and real-world deployment.

From the results, we can corroborate that the scalability of our proposed framework is backed by a Federated Learning framework. This allows decentralized training on multiple devices without needing data transfer to a central server. It lowers communication overhead and lets the system manage an increasing number of networked devices efficiently. The edge computing structure used in our study also aids the distribution of computation tasks, lessening individual device strain and avoiding bottlenecks while more devices join up. The hybrid forecasting model, using BiLSTM and GRU along with the attention mechanism techniques, ensures the prediction of resource use remains accurate even when the network size increases. Edge resource competition, where multiple devices simultaneously request processing power, is a critical challenge in edge computing environments. Our approach addresses this issue by utilizing dynamic task offloading strategies that prioritize tasks based on real-time resource availability and predicted workload. The D4PG Reinforcement Learning model adapts according to changes in resource conditions and manages the involvement of User Equipment (UE) based on the demand-supply balance for resources, thus minimizing competition over resources. Furthermore, using a hybrid prediction model guarantees effective resource allocation. It prevents the overloading of edge servers and balances the resource demands across devices. The proposed architecture uses Federated Learning to enhance coordination among distributed devices without compromising privacy. Federated Learning allows each device to train models locally while only sharing model updates; this ensures coordination between edge devices without centralizing data. The D4PG model offers improvement to this coordination. It ensures each device can adjust its behavior based on global learning. This leads to more efficient task scheduling and resource utilization. Plus, the attention mechanism in the prediction model makes coordination better by letting the system concentrate on relevant features when predicting resource usage. This helps allocate tasks more precisely.

While the approach demonstrates good performance in smaller agricultural IoT environments, the system's performance might be impacted by the complexity and heterogeneity of large networks. Although the proposed method effectively manages dynamic resource allocation, real-time adaptation to highly unpredictable network conditions, such as unexpected network failures, sudden spikes in task load, or varying edge device availability, may require further optimization. While our approach improves resource efficiency and scheduling, extreme fluctuations in network conditions could introduce delays or suboptimal allocation decisions. Future enhancements could incorporate adaptive learning mechanisms, such as Reinforcement Learning-based optimization or self-adjusting heuristics, to further enhance the system's responsiveness and robustness in handling highly volatile environments.

In summary, the proposed Federated Reinforcement Learning-based task offloading algorithm optimizes resource utilization, reducing energy consumption while maintaining system performance in edge-based IoT environments. BiLSTM-GRU with the attention model accurately predicts resource usage, enabling dynamic and adaptive scheduling in IoT applications. The D4PG-based Federated Learning approach enhances fairness in model updates while preserving user privacy, which is crucial for distributed IoT systems. Experimental results demonstrate that the proposed method outperforms existing algorithms (best-fit, first-fit, worst-fit) and deep RL baselines (DQN, DDQN, Dueling DQN, Dueling DDQN).

Author Contributions: Conceptualization, S.M.; Data curation, S.M. and D.A.; Formal analysis, S.M. and D.A.; Funding acquisition, M.A.A.-K., O.A., F.A., I.U. and D.A.; Investigation, S.M., D.A. and I.U.; Methodology, S.M. and D.A.; Project administration, F.Z., D.A. and I.U.; Resources, F.Z.; Software, S.M.; Supervision, F.Z., D.A. and I.U.; Validation, S.M.; Visualization, S.M., M.A.A.-K., O.A. and F.A.; Writing—original draft, S.M., D.A. and I.U.; Writing—review and editing, S.M., F.Z., D.A., I.U., M.A.A.-K., O.A. and F.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded by the Researchers Supporting Project number (RSP2025R102) King Saud University, Riyadh, Saudi Arabia. This work was also supported by the IITP (Institute of Information & Communications Technology Planning & Evaluation)-ITRC (Information Technology

Research Center) grant funded by the Korea government (Ministry of Science and ICT) (IITP-2025-RS-2023-00259004).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The datasets generated and/or analyzed and coded on simulations during the current study can be found online.

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

References

1. Adhikari, D.; Jiang, W.; Zhan, J.; Rawat, D.B.; Bhattarai, A. Recent advances in anomaly detection in Internet of Things: Status, challenges, and perspectives. *Comput. Sci. Rev.* **2024**, *54*, 100665. [CrossRef]
2. Adhikari, D.; Jiang, W.; Zhan, J.; Assefa, M.; Khorshidi, H.A.; Aickelin, U.; Rawat, D.B. A lightweight window portion-based multiple imputation for extreme missing gaps in iot systems. *IEEE Internet Things J.* **2023**, *11*, 3676–3689. [CrossRef]
3. Mei, Q.; Guo, W.; Zhao, Y.; Nie, L.; Adhikari, D. Blockchain-based privacy-preserving incentive scheme for internet of electric vehicle. *Inf. Fusion* **2025**, *115*, 102732. [CrossRef]
4. Zhao, Y.; Zhang, H.; Liu, Q. Edge computing and IoT data fusion: New opportunities and challenges. *Inf. Fusion* **2020**, *61*, 99–111.
5. Liu, X.; Yu, J.; Wang, J.; Gao, Y. Resource Allocation With Edge Computing in IoT Networks via Machine Learning. *IEEE Internet Things J.* **2020**, *7*, 3415–3426. [CrossRef]
6. Boursianis, A.D.; Papadopoulou, M.S.; Diamantoulakis, P.; Liopa-Tsakalidi, A.; Barouchas, P.; Salahas, G.; Karagiannidis, G.; Wan, S.; Goudos, S.K. Internet of Things (IoT) and Agricultural Unmanned Aerial Vehicles (UAVs) in smart farming: A comprehensive review. *Internet Things* **2022**, *18*, 100187. [CrossRef]
7. Elijah, O.; Rahman, T.A.; Orikumhi, I.; Leow, C.Y.; Hindia, M.N. An Overview of Internet of Things (IoT) and Data Analytics in Agriculture: Benefits and Challenges. *IEEE Internet Things J.* **2018**, *5*, 3758–3773. [CrossRef]
8. de la Parte, M.S.E.; Martínez-Ortega, J.F.; Castillejo, P.; Lucas-Martínez, N. Spatio-temporal semantic data management systems for IoT in agriculture 5.0: Challenges and future directions. *Internet Things* **2024**, *25*, 101030. [CrossRef]
9. Alonso, R.S.; Sittón-Candanedo, I.; García, O.; Prieto, J.; Rodríguez-González, S. An intelligent Edge-IoT platform for monitoring livestock and crops in a dairy farming scenario. *Ad Hoc Netw.* **2020**, *98*. [CrossRef]
10. Graves, A.; Fernández, S.; Schmidhuber, J. Bidirectional LSTM networks for improved phoneme classification and recognition. In Proceedings of the International Conference on Artificial Neural Networks, Warsaw, Poland, 11–15 September 2005; Springer: Berlin/Heidelberg, Germany, 2005; pp. 799–804.
11. Ahmadzadeh, E.; Kim, H.; Jeong, O.; Kim, N.; Moon, I. A Deep Bidirectional LSTM-GRU Network Model for Automated Ciphertext Classification. *IEEE Access* **2022**, *10*, 3228–3237. [CrossRef]
12. Yang, C.; Chen, Q.; Zhu, Z.; Huang, Z.A.; Lan, S.; Zhu, L. Evolutionary Multitasking for Costly Task Offloading in Mobile-Edge Computing Networks. *IEEE Trans. Evol. Comput.* **2024**, *28*, 338–352. [CrossRef]
13. Wu, H.; Geng, J.; Bai, X.; Jin, S. Deep reinforcement learning-based online task offloading in mobile edge computing networks. *Inf. Sci.* **2024**, *654*, 119849. [CrossRef]
14. Idoje, G.; Dagiuklas, T.; Iqbal, M. Survey for smart farming technologies: Challenges and issues. *Comput. Electr. Eng.* **2021**, *92*, 107104. [CrossRef]
15. Lloret, J. Edge Computing in Precision agriculture. In Proceedings of the 2022 Seventh International Conference on Fog and Mobile Edge Computing (FMEC), Paris, France, 12–15 December 2022; p. 1. [CrossRef]
16. Eunice, J.; Popescu, D.E.; Chowdary, M.K.; Hemanth, J. Deep Learning-Based Leaf Disease Detection in Crops Using Images for Agricultural Applications. *Agronomy* **2022**, *12*, 2395. [CrossRef]
17. Alharbi, H.A.; Aldossary, M. Energy-Efficient Edge-Fog-Cloud Architecture for IoT-Based Smart Agriculture Environment. *IEEE Access* **2021**, *9*, 110480–110492. [CrossRef]
18. Nadig, D.; El Alaoui, S.; Ramamurthy, B.; Pitla, S. ERGO: A Scalable Edge Computing Architecture for Infrastructureless Agricultural Internet of Things. In Proceedings of the 2021 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN), Virtually, 12–14 July 2021; pp. 1–2. [CrossRef]
19. Padhy, S.; Alowaidi, M.; Dash, S.; Alshehri, M.; Malla, P.P.; Routray, S.; Alhumyani, H. AgriSecure: A Fog Computing-Based Security Framework for Agriculture 4.0 via Blockchain. *Processes* **2023**, *11*, 757. [CrossRef]
20. Ferrández-Pastor, F.J.; García-Chamizo, J.M.; Nieto-Hidalgo, M.; Mora-Martínez, J. Precision Agriculture Design Method Using a Distributed Computing Architecture on Internet of Things Context. *Sensors* **2018**, *18*, 1731. [CrossRef]
21. Gia, T.N.; Qingqing, L.; Queralta, J.P.; Zou, Z.; Tenhunen, H.; Westerlund, T. Edge AI in Smart Farming IoT: CNNs at the Edge and Fog Computing with LoRa. In Proceedings of the 2019 IEEE AFRICON, Accra, Ghana, 25–27 September 2019; pp. 1–6. [CrossRef]

22. Zhang, X.; Cao, Z.; Dong, W. Overview of Edge Computing in the Agricultural Internet of Things: Key Technologies, Applications, Challenges. *IEEE Access* **2020**, *8*, 141748–141761. [CrossRef]
23. Ghosh, A.; Khalid, O.; Rais, R.N.B.; Rehman, A.; Malik, S.U.R.; Khan, I.A. Data offloading in IoT environments: Modeling, analysis, and verification. *EURASIP J. Wirel. Commun. Netw.* **2019**, *2019*, 1–23. [CrossRef]
24. Zaw, C.W.; Pandey, S.R.; Kim, K.; Hong, C.S. Energy-Aware Resource Management for Federated Learning in Multi-Access Edge Computing Systems. *IEEE Access* **2021**, *9*, 34938–34950. [CrossRef]
25. Shirke, A.; Chandane, M.M. Performance Modelling and Analysis of IoT Based Edge Computing Policies. *Wirel. Pers. Commun.* **2022**, *127*, 2553–2568. [CrossRef]
26. Tam, P.; Kang, S.; Ros, S.; Kim, S. Enhancing QoS with LSTM-Based Prediction for Congestion-Aware Aggregation Scheduling in Edge Federated Learning. *Electronics* **2023**, *12*, 3615. [CrossRef]
27. Zhang, X.; Wu, W.; Wang, J.; Liu, S. BiLSTM-based Federated Learning Computation Offloading and Resource Allocation Algorithm& in MEC. *ACM Trans. Sen. Netw.* **2023**, *19*, 68. [CrossRef]
28. Zhao, R.; Yang, L.T.; Liu, D.; Lu, W.; Yang, X. Lightweight Tensor-Enabled GRU for Trustworthy and Communication Efficient Federated Learning in Industrial IoT. *IEEE Trans. Ind. Inform.* **2024**, *21*, 2043–2052. [CrossRef]
29. Liu, Q.; Tang, L.; Wu, T.; Chen, Q. Deep Reinforcement Learning for Resource Demand Prediction and Virtual Function Network Migration in Digital Twin Network. *IEEE Internet Things J.* **2023**, *10*, 19102–19116. [CrossRef]
30. Li, X.; Wang, H.; Xiu, P.; Zhou, X.; Meng, F. Resource Usage Prediction Based on BILSTM-GRU Combination Model. In Proceedings of the 2022 IEEE International Conference on Joint Cloud Computing (JCC), Fremont, CA, USA, 15–18 August 2022; pp. 9–16. [CrossRef]
31. Muhammad Fahad Noman, H.; Dimyati, K.; Ariffin Noordin, K.; Hanafi, E.; Abdrabou, A. FeDRL-D2D: Federated Deep Reinforcement Learning- Empowered Resource Allocation Scheme for Energy Efficiency Maximization in D2D-Assisted 6G Networks. *IEEE Access* **2024**, *12*, 109775–109792. [CrossRef]
32. Li, C.; Zhang, Y.; Luo, Y. A Federated Learning-Based Edge Caching Approach for Mobile Edge Computing-Enabled Intelligent Connected Vehicles. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 3360–3369. [CrossRef]
33. Xu, T.; Liu, Y.; Ma, Z.; Huang, Y.; Liu, P. A DQN-Based Multi-Objective Participant Selection for Efficient Federated Learning. *Future Internet* **2023**, *15*, 209. [CrossRef]
34. Liu, S.; Yang, S.; Zhang, H.; Wu, W. A Federated Learning and Deep Reinforcement Learning-Based Method with Two Types of Agents for Computation Offload. *Sensors* **2023**, *23*, 2243. [CrossRef]
35. Abdulqadder, I.H.; Aziz, I.T.; Zou, D. DT-Block: Adaptive vertical federated reinforcement learning scheme for secure and efficient communication in 6G. *Comput. Netw.* **2024**, *254*, 110841. [CrossRef]
36. Tirmazi, M.; Barker, A.; Deng, N.; Haque, M.E.; Qin, Z.G.; Hand, S.; Harchol-Balter, M.; Wilkes, J. Borg: The next generation. In Proceedings of the Fifteenth European Conference on Computer Systems, New York, NY, USA, 27–30 April 2020; EuroSys '20, pp. 1–14. [CrossRef]
37. Verma, A.; Pedrosa, L.; Korupolu, M.; Oppenheimer, D.; Tune, E.; Wilkes, J. Large-scale cluster management at Google with Borg. In Proceedings of the Tenth European Conference on Computer Systems, Bordeaux, France, 21–24 April 2015; pp. 1–17. [CrossRef]
38. Souza, P.S.; Ferreto, T.; Calheiros, R.N. EdgeSimPy: Python-based modeling and simulation of edge computing resource management policies. *Future Gener. Comput. Syst.* **2023**, *148*, 446–459. [CrossRef]
39. Dinh, C.T.; Tran, N.H.; Nguyen, M.N.; Hong, C.S.; Bao, W.; Zomaya, A.Y.; Gramoli, V. Federated learning over wireless networks: Convergence analysis and resource allocation. *IEEE/ACM Trans. Netw.* **2020**, *29*, 398–409.
40. Tran, N.H.; Bao, W.; Zomaya, A.; Nguyen, M.N.; Hong, C.S. Federated learning over wireless networks: Optimization model design and analysis. In Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; IEEE: New York, NY, USA, 2019; pp. 1387–1395.
41. Yang, H.H.; Liu, Z.; Quek, T.Q.; Poor, H.V. Scheduling policies for federated learning in wireless networks. *IEEE Trans. Commun.* **2019**, *68*, 317–333. [CrossRef]
42. Beutel, D.J.; Topal, T.; Mathur, A.; Qiu, X.; Fernandez-Marques, J.; Gao, Y.; Sani, L.; Kwing, H.L.; Parcollet, T.; Gusmão, P.P.B.; et al. Flower: A Friendly Federated Learning Research Framework. *arXiv* **2020**, arXiv:2007.14390.
43. Cohen, G.; Afshar, S.; Tapson, J.; Schaik, A.V. EMNIST: Extending MNIST to handwritten letters. In Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017. [CrossRef]
44. Wei, K.; Li, J.; Ding, M.; Ma, C.; Yang, H.H.; Farokhi, F.; Jin, S.; Quek, T.Q.S.; Poor, V. Federated Learning With Differential Privacy: Algorithms and Performance Analysis. *IEEE Trans. Inf. Forensics Secur.* **2020**, *15*, 3454–3469. [CrossRef]