Online-Journals.org

iJIM International Journal of Interactive Mobile Technologies

iJIM | eISSN: 1865-7923 | Vol. 19 No. 10 (2025) | OPEN ACCESS

https://doi.org/10.3991/ijim.v19i10.54067

PAPER

Enhancing Adaptive Behavior in Federated Learning: The AdaptiveMesh Algorithm for Interactive Mobile and Bandwidth-Limited Resource-Constrained Wireless Environments

Lamir Shkurti[1](✉), **Mennan Selimi**[1,2]

[1]Faculty of Contemporary Sciences and Technologies, South East European University, Tetovo, North Macedonia

[2]Max van der Stoel Institute, South East European University, Tetovo, North Macedonia

ls29773@seeu.edu.mk

ABSTRACT

Federated learning (FL) enables decentralized model training while preserving data privacy, but its application in resource-constrained environments faces challenges, including computational load, slowing down CPU speed, and network bandwidth limitations. This paper introduces and evaluates the AdaptiveMesh (Adaptive Federated Learning for Wireless Mesh Environments) algorithm, an adaptive approach designed to optimize resource usage in FL scenarios, with a focus on resource-constrained Internet of Things (IoT) devices, mobile applications, and commercial cloud systems. We compare the performance of AdaptiveMesh with a naive non-adaptive algorithm, focusing on key metrics such as CPU utilization, device temperature, accuracy, training time, and network bandwidth. Our study examines these algorithms' effectiveness on both physical devices, including Raspberry Pis, mini PCs, and laptops, as well as virtual machine instances in the Google Cloud Platform. The results demonstrate that the AdaptiveMesh algorithm dynamically adjusts training parameters in response to real-time device metrics, thereby optimizing resource usage, preventing device overload, and reducing training time under constrained conditions, whereas the naive approach often leads to device inefficiencies and potential temporary shutdowns, as devices attempt to manage heat or workload by throttling performance. This comparison underscores the importance of adaptive mechanisms in FL, especially in resource-limited environments.

KEYWORDS

adaptive federated learning, wireless mesh networks, embedded machine learning, edge computing

Shkurti, L., Selimi, M. (2025). Enhancing Adaptive Behavior in Federated Learning: The AdaptiveMesh Algorithm for Interactive Mobile and Bandwidth-Limited Resource-Constrained Wireless Environments. *International Journal of Interactive Mobile Technologies (iJIM)*, 19(10), pp. 32–55. https://doi.org/10.3991/ijim.v19i10.54067

Article submitted 2025-01-02. Revision uploaded 2025-03-03. Final acceptance 2025-03-03.

1 INTRODUCTION

Federated learning (FL) is a popular approach in machine learning (ML) that allows models to be trained across several distributed devices, guaranteeing data privacy by keeping the data on the local devices and sending only the model updates to a central server [1]. This technique has shown promise in various applications, including mobile devices, Internet of Things (IoT) networks, and distributed sensor networks [2]. However, deploying FL in heterogeneous and resource-constrained environments has significant challenges [3]–[5]. These challenges include limited computational power, network bandwidth constraints, and managing heterogeneous data distributions across devices.

Devices with limited resources, such as IoT and mobile devices, often have challenges with long training tasks due to limited energy and computational capabilities. As training tasks increase in size or complexity, the computational demands also increase, potentially causing inefficiencies as devices attempt to manage heat or workload by throttling performance, which leads to inefficiencies [6]. Non-adaptive algorithms typically fail to manage these constraints effectively, leading to temporary shutdowns or significant performance drops due to resource overload.

Addressing these inefficiencies requires adaptive strategies that dynamically respond to resource constraints. Energy-aware computing techniques, such as task offloading, low-power hardware, and dynamic resource allocation, are essential for increasing the longevity and reliability of IoT and edge devices [7]. Additionally, there is a critical trade-off between training accuracy and resource utilization. While increasing training rounds and the size of the training dataset can improve accuracy, it can also prolong the training time and elevate the risk of device overuse. Studies have highlighted this trade-off, emphasizing the need to manage resource and power constraints to prevent performance degradation [8]. Implementing adaptive mechanisms that adjust the training workload based on real-time device conditions, such as CPU utilization thresholds, can reduce these risks, though potentially at the cost of reduced accuracy [9].

Another critical challenge in FL is fluctuating network bandwidth, which can disrupt data transmission, extend training times, or drop connections. Standard FL algorithms with fixed parameters are unable to adapt to these variations, resulting in inefficiencies in both computational and network resources. Recent studies highlight the need for adaptive FL algorithms that can dynamically respond to the limitations of resource-limited environments [10] [11]. Adaptive FL is crucial because it helps models perform well with different types of data on various devices with different hardware capacities. Since data on each device can be unique, adaptive algorithms allow models to adjust and perform better despite these differences [12]. This adaptability ensures that the learning process remains efficient even with varying device capabilities, internet connectivity, and resource constraints like CPU load [13].

However, the effectiveness and efficiency of FL algorithms can vary significantly based on several critical factors, including computational load, memory usage, environmental temperature, battery consumption, and training time.

To address these challenges, we introduce the AdaptiveMesh algorithm. AdaptiveMesh dynamically adjusts training parameters based on client characteristics and network conditions. It evaluates CPU utilization, CPU temperature, training time, and bandwidth, adapting the training load to prevent overuse and ensure optimal performance [5]. This approach is evaluated on both physical and virtual platforms, providing insights into its effectiveness across diverse environments. By maintaining device longevity and stable operation under varying conditions, it enhances the efficiency and scalability of federated learning.

The main contributions of this paper are:

1. We design and implement the AdaptiveMesh, an adaptive FL algorithm. AdaptiveMesh dynamically adapts to device conditions in real time, adjusting epochs and training images to prevent performance degradation, offering an advantage over non-adaptive algorithms like FedAvg and FedProx [5]. Other than state-of-the-art approaches, AdaptiveMesh has been augmented to include network bandwidth during FL training, which is critical for resource-constrained networks.
2. We test and evaluate the AdaptiveMesh algorithm in a real wireless environment, including mobile and resource-constrained machines such as Raspberry Pis, Mini PCs (Intel Atom) etc.
3. To understand the impact of the wireless network environment on training ML models on resource-constrained devices, we also evaluate AdaptiveMesh in a commercial cloud environment, such as Google Cloud Platform. Our analysis offers a deeper understanding of the practical trade-offs in FL and provides recommendations for selecting and adapting algorithms to maintain a balance between accuracy and resource conservation in physical and virtual environments.

The rest of the paper is organized as follows: Section 2 reviews the related work, providing context and background. In Section 3, we present the AdaptiveMesh algorithm in detail. Section 4 outlines the experimental setup, including the testbed and dataset employed. The results from the testbed experiments are discussed in Section 5, while Section 6 covers the outcomes observed in the cloud environment. Section 7 outlines the main findings and limitations of our work. Finally, Section 8 summarizes the findings and suggests directions for future research.

2 RELATED WORK

Federated learning faces several challenges, especially in resource-constrained IoT devices, particularly in wireless environments where computational power, network bandwidth, and energy resources are limited.

Recent studies have extensively addressed the challenges and opportunities of FL in resource-constrained environments, particularly in the context of the IoT. Imteaj et al. [14], in their work, provide a comprehensive analysis of existing FL methods and the challenges of implementing them on IoT devices with limited processing power, low bandwidth, and restricted storage capacity. Their work emphasizes the need for optimizing FL algorithms to handle these constraints effectively and suggests future research directions to improve FL performance in such environments. This study is foundational for understanding how FL can be adapted for IoT devices with limited resources, aligning closely with the goals of this study.

Authors in [15] explore the implementation of FL in a resource-constrained environment using Raspberry Pi devices within a wireless mesh network. The focus is on training a convolutional neural network (CNN) with medical data, specifically chest X-ray images, without compromising data privacy. The research highlights that increasing the number of training epochs improves model accuracy but also leads to higher CPU usage. Their findings underscore the trade-offs between performance and resource consumption, emphasizing the potential of FL in privacy-sensitive applications such as healthcare. This work contributes practical insights into optimizing FL for low-capacity devices in real-world scenarios, offering valuable guidance for similar future implementations.

The study by Zhang et al. [16] provides an in-depth exploration of the interplay between energy consumption and resource allocation in ML processes, particularly in IoT devices. The work emphasizes the significance of balancing the computational load to prevent overheating and excessive energy usage, which is critical for the sustainability and efficiency of IoT networks. The paper introduces methodologies to optimize resource allocation dynamically based on the workload and temperature feedback, ensuring that devices do not overheat while maintaining high performance levels. This approach is particularly relevant for scenarios involving continuous ML tasks, where overheating and excessive energy consumption are common concerns.

The research [17] introduces the P2FLF framework, a new approach for ensuring privacy in FL in fog computing environments. The framework balances data protection with maintaining model accuracy by using fog nodes for local data processing and incorporating methods like differential privacy and encryption. The results show that it reduces communication delays and latency, making it a promising solution for IoT applications.

Authors in [18] also explored FL for applications like detecting depression through smartphone sensing. This method keeps user data on personal devices while allowing collaborative training of ML models, showing that privacy can be preserved without sacrificing usability. The study also underscores the trade-offs between centralized and FL systems in healthcare.

The study [19] highlights advancements in healthcare systems, such as the Smart Healthcare Monitoring System, which uses sensors and microcontrollers like Raspberry Pi to monitor health metrics in real time. By integrating cloud storage and mobile apps, these systems enable continuous health tracking and faster medical decision-making, showcasing the transformative impact of IoT in modern healthcare.

Federated learning has evolved with the development of adaptive algorithms aimed at optimizing performance in resource-constrained and heterogeneous environments. Traditional FL approaches, such as those based on static training parameters, often struggle in dynamic environments where device capabilities and network conditions vary significantly. To address these challenges, recent research has focused on introducing adaptability to FL models.

One research study in this field is FedAda [20], which presents an FL algorithm designed for mobile edge computing environments. The main aim of FedAda is to improve the speed of convergence by allocating workloads across devices with different capabilities. This method becomes significant in situations where device abilities vary and quick convergence is essential for maintaining system performance.

FedAda can adjust to device capabilities and network conditions to achieve convergence and optimize resource usage effectively. A crucial feature for applications in environments with limited resources.

FedALA [21] introduces an aggregation approach that enhances the learning process by dynamically modifying local updates to accommodate different heterogeneous client capabilities effectively. This technique proves beneficial in tailoring FL for clients with resource limitations like IoT devices or mobile edge nodes. FedALA significantly reduces communication overhead and improves convergence speed by optimizing local aggregation weights. It guarantees that even devices with resources can make contributions without impeding the overall system performance significantly. By managing device heterogeneity effectively, FedALA addresses challenges like high CPU load and bandwidth fluctuations, aligning well with the goals of this study in optimizing FL for real-world applications.

Another contribution is the study [22] on communication-efficient FL with adaptive aggregation in edge-cloud environments. The study focuses on reducing communication in FL through the use of aggregation methods. Its aim is to strike a balance between communication efficiency and learning outcomes optimization in edge cloud environments with resources. By reducing communication costs and upholding precision levels simultaneously, this initiative enhances the system performance proving beneficial in situations where bandwidth limitations are a concern.

Moreover, FedLC [23] puts forward a structure to speed up FL through handling device diversity and cutting down on the delay between updates. This study is focused on advancing FL by dealing with device diversity and lessening the waiting duration between updates. By introducing methods, the FedLC framework effectively manages updates, enhancing the learning pace and productivity of federated systems.

This is especially crucial in environments where devices have varying computational capabilities, as the framework helps preserve learning accuracy while reducing resource strains.

The research discussed in the publication [24] introduces an FL model specifically designed for IoT settings characterized by devices with resources. The innovative FedPARL model adopts a three-layered strategy to tackle FL obstacles like variations in systems and statistics among clients and constraints related to capacity and the potential inclusion of inaccurate client data updates.

3 ADAPTIVEMESH ALGORITHM

The AdaptiveMesh algorithm dynamically adapts to device conditions in real time, adjusting epochs and training images to prevent performance degradation, offering an advantage over non-adaptive algorithms. The algorithm highlighted in Figure 1 is initially configured with key parameters such as global and local learning rates, number of epochs, number of images, batch size, and other training-related parameters. The algorithm proceeds with the following steps:

1. **Initialization:** Clients are registered within the system to participate in the collaborative training process.
2. **Initialization of training:** The training process begins with the server distributing initial configurations to heterogeneous clients.
3. **Monitoring:** In each round, during training, a thread is launched in the background and monitors CPU utilization, CPU temperature, memory utilization, bandwidth, training time, and accuracy metrics and stores them for subsequent analysis.
4. **Adaptive mechanism activation:** After three rounds of training (round > 3), an adaptive mechanism is employed to assess whether adjustments to the training parameters are necessary.
5. **Adaptation process:** The adaptive component evaluates whether the thresholds for training time, bandwidth, CPU temperature, and the difference in CPU utilization between consecutive training rounds have been exceeded to determine if adaptation is required. If the predefined thresholds exceed the specified limits, the algorithm dynamically adjusts the number of epochs and training images to ensure efficient resource utilization.
   a) When adaptation is required and resources are available *(e.g., CPU < 80%, CPU temperature < 85°C, and bandwidth usage < 2 Mbps)*, the number of training images is increased, with a maximum of up to 112 images.

b) When adaptation is required and resources are not available (e.g., CPU utilization > 80%, CPU temperature > 85°C, or bandwidth usage > 2 Mbps), the number of training images is decreased.

In cases where these thresholds are exceeded, AdaptiveMesh dynamically reduces the number of epochs and training images to maintain continuous stability in the distributed learning process.

The adaptive decision-making process of the AdaptiveMesh algorithm is crucial for optimizing model performance across heterogeneous clients in wireless mesh environments. This adaptive mechanism ensures that resource allocation is dynamically adjusted at the clients to meet the evolving demands of model training. Additionally, the algorithm's setup for FL demonstrates a thoughtful approach to coordinating shared learning, which is vital for enhancing distributed ML in practical applications.

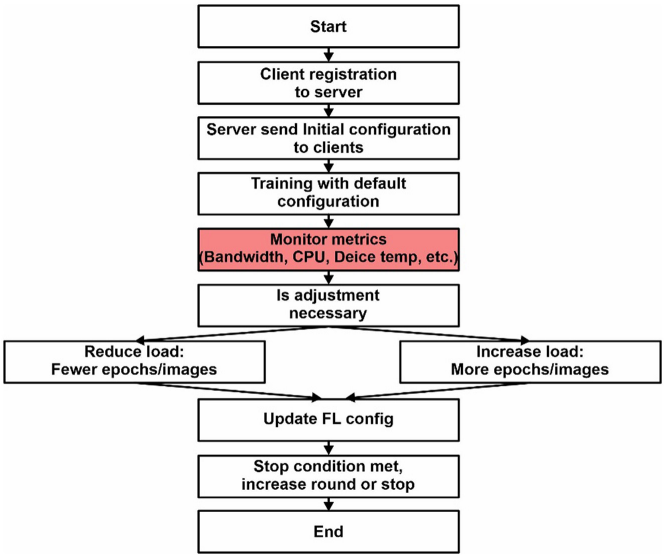

Fig. 1. AdaptiveMesh's flowchart algorithm

4 EXPERIMENTAL SETUP AND RESULTS

In this section, we will first describe the wireless testbed utilized, followed by a detailed discussion of the dataset employed and the two algorithms used in our experiments.

4.1 Experimental setup

The experimental setup consists of a distributed FL system that includes physical IoT devices interconnected via a wireless network. We differentiate two types of nodes: the server and clients. The client devices include: 1) a MiniPC NEO Z83-4 device featuring an Intel Atom x5-Z8350 CPU, 4 GB DDR3 RAM, and Windows 10 OS; 2) a laptop with an Intel(R) Core (TM) i5-1035G1 CPU, 8 GB RAM, and Windows 11 OS; and 3) four Raspberry Pi 4 Model B units, namely RPI4 and RPI5, each equipped with a Quad Core Cortex-A72 CPU, 8 GB RAM, 128 GB storage, and running the Linux Ubuntu 22.04.4 LTS OS.

The server role is performed by a laptop with an Intel(R) Core (TM) i5-8250U CPU, equipped with 8 GB RAM running Windows 11 OS. Figure 2 illustrates the wireless testbed used for the experiments.

We installed Python and PyTorch on all client and server devices. The system was developed entirely in Python, with distinct implementations for the client and server functionalities. We developed Docker images for both the client and server components. The client image was installed on the Raspberry Pi devices, the MiniPC, and one of the laptops acting as client devices, while the server image was deployed on the laptop serving as the central server. To train ML models, we used the Keras API to implement a 6-layer CNN model. The CNN was trained on the client devices using the Chest X-Ray Images (Pneumonia) dataset.

Adaptive FL operates by training models on decentralized devices, where each device uses its local data to update the model. The process begins with an initial model distributed from a central server to all participating devices. Each device trains its model using local data and adapts the training process based on resource conditions and thresholds set by the algorithm. After completing the local training, each device sends its model updates (not raw data) back to the central server. The central server aggregates these updates to improve the global model. This improved model is then redistributed to the participating devices for the next learning round, repeating the cycle until satisfactory performance is achieved. This approach ensures that sensitive data remains local, maintaining privacy and training efficiency in a heterogeneous and resource-constrained environment.

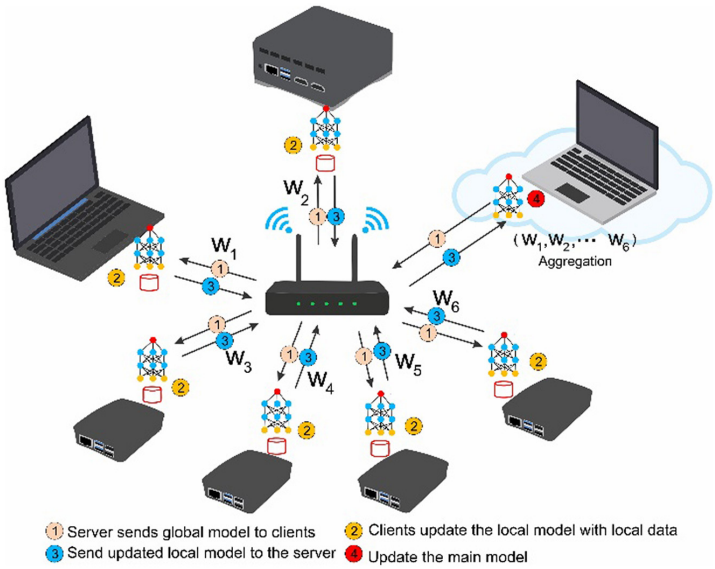

Fig. 2. Testbed nodes used for the experimentation

4.2 Dataset

For our experimental evaluations, we utilized a dataset comprising 5,863 X-ray images in JPEG format. This dataset is divided into two primary categories: pneumonia and normal. The images were obtained from young patients, aged between one and five years, who were treated at the Guangzhou Women's and Children's Medical Center in Guangzhou [24]. Medical represents a crucial field where data privacy is paramount, and FL provides an effective approach by enabling collaborative

model training without sharing sensitive patient data. By using a dataset of chest X-rays, this study demonstrates the applicability of AdaptiveMesh in healthcare scenarios, where IoT devices or mobile health monitoring devices often operate under constrained computational and energy resources. High-resolution images require substantial computational power for model training, and the need for accurate classification (e.g., identifying pneumonia) directly impacts patient outcomes. This aligns closely with the objectives of this study, showcasing how AdaptiveMesh can balance resource utilization and model performance in environments with limited hardware capabilities.

4.3 Comparing adaptive and non-adaptive FL: AdaptiveMesh vs. NAIVE

The comparison of AdaptiveMesh adaptive FL algorithms with the NAIVE FL algorithm involves evaluating their performance based on several important metrics. Common performance metrics include resource consumption, such as CPU load and CPU temperature, model accuracy, loss, training time, and network load.

To ensure a fair comparison, experiments will be conducted under the same conditions using the same dataset, the same number of training rounds, and the same network configuration. AdaptiveMesh's configuration includes 30 training rounds, a learning rate of 0.001, and a batch size of 2, training images start at 6 and increase up to a maximum of 112, with epochs starting at 1 and increasing gradually. The bandwidth is set at 2 Mbps for all 4 RPIs and 5 Mbps for the Mini PC and laptop. The CPU threshold is set at 80%, with a CPU thermal temperature of 85%, and the maximum training time is 30 minutes. In contrast, the *NAIVE* approach has no CPU load, CPU thermal temperature, bandwidth, or training time thresholds. NAIVE algorithm also runs for 30 training rounds, learning rate of 0.001, and a batch size of 2, training images start at 6 and increase up to a maximum of 112, with epochs starting at 1 and increasing gradually.

The result analysis involves collecting and analyzing performance data for each algorithm and comparing them by using statistical methods to evaluate any significant differences. This analysis will focus on CPU load, CPU temperature, accuracy, loss, training times, network load, and resource consumption.

4.4 Experiments

The goal of the experimentation in this study is to:

1. Evaluate the performance of the AdaptiveMesh adaptive algorithm: The experimentation aims to assess how well AdaptiveMesh optimizes resource usage (such as CPU utilization, temperature management, and bandwidth efficiency) in FL environments compared to a NAIVE non-adaptive FL algorithm.
2. Compare the effectiveness of AdaptiveMesh and NAIVE algorithms across different environments: The experimentation seeks to compare how these algorithms perform on physical devices (e.g., Raspberry Pis, mini PCs, and laptops) versus virtual machine instances in Google Cloud. This comparison is intended to highlight the differences between deploying FL algorithms in physical versus virtual environments, providing insights into the practical implications and trade-offs involved in each scenario.

5 RESULTS

5.1 Performance metrics

To evaluate the performance of the AdaptiveMesh and NAIVE algorithms, we analyzed several critical metrics:

- **CPU utilization (%):** Represents the percentage of processor resources used during the training process.
- **CPU temperature (C):** Measures the processor's temperature during training.
- **Accuracy (%):** Indicates the accuracy of the trained model, reflecting the overall effectiveness of the model training process.
- **Training time (seconds):** The total time required to complete one round of training. This metric evaluates the time efficiency of the algorithm.
- **Bandwidth utilization (Mbps):** The amount of network bandwidth consumed during communication between clients and the server, highlighting the algorithm's impact on network resource usage.

CPU utilization: Figure 3 (left) illustrates the CPU utilization percentage during various training rounds for different devices. A horizontal red dashed line at 80% marks the CPU load limit to avoid overloading. The graph demonstrates that the laptop device maintains a consistent and low CPU load throughout the training rounds, unlike other devices, which show significant increases in CPU load, sometimes exceeding 80%. This is managed by the AdaptiveMesh algorithm, which automatically optimizes the training process by reducing the resources when the CPU load exceeds 80%, ensuring stable performance and preventing device overloading. This analysis highlights the effectiveness of the AdaptiveMesh algorithm in *managing CPU load* during FL across different devices, optimizing performance, and avoiding conditions that could lead to reduced performance or cause the device to slow down.

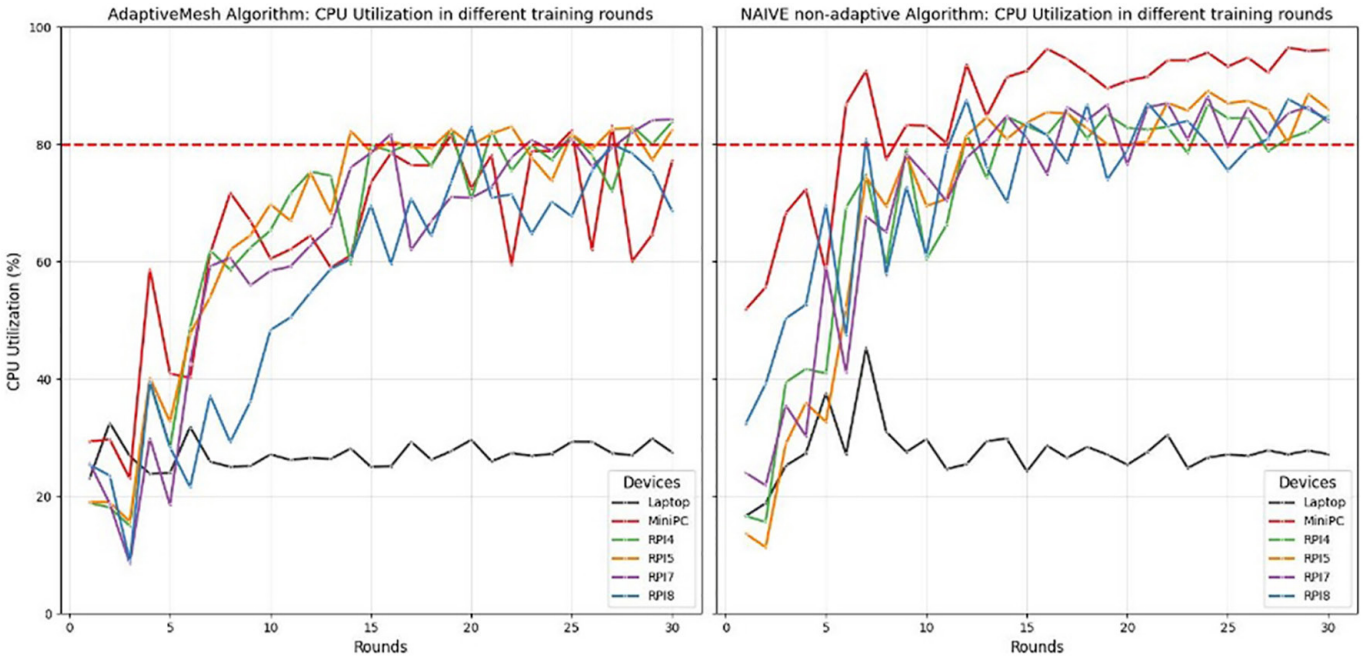

Fig. 3. CPU utilization during training with AdaptiveMesh (left) and NAIVE (right)

Note: AdaptiveMesh keeps usage below 80%, avoiding overload, while NAIVE exceeds it, risking instability.

Figure 3 (right) depicts CPU utilization percentages during various training rounds for the same devices without adaptive CPU load management, where resources are increased with each round. Unlike the AdaptiveMesh approach, this method, referred to as, does not adapt CPU load. The graph reveals that, except for the laptop device, all other devices exceed the 80% CPU utilization threshold as the number of images and epochs increases. The increase in tasks may require more energy, shortening battery life or affecting the device's performance. The device may start to operate more slowly or be unable to handle tasks. Additionally, increased CPU usage can lead to device overloading, indicating the importance of adaptive management in maintaining optimal performance. Comparing the two graphs highlights the effectiveness of the AdaptiveMesh algorithm in managing CPU load during FL. While the *NAIVE approach leads to excessive CPU utilization and potential overheating*, AdaptiveMesh ensures a balanced load, maintaining device performance and longevity.

CPU temperature: Figure 4 (left) shows the CPU temperatures during various training rounds for the same devices using the AdaptiveMesh algorithm. With a red horizontal dashed line at 85°C marking the temperature limit. This graph highlights that when the CPU temperature exceeds 85°C, AdaptiveMesh reduces the workload by decreasing the resources, effectively lowering the CPU temperature. This adaptive resource management ensures that devices, particularly those with weaker hardware performance, are not overloaded, thereby avoiding performance degradation and conserving battery life. The *consistent temperature management* shown in the graph underscores AdaptiveMesh's ability to dynamically allocate resources based on temperature, ensuring optimal device performance and longevity.

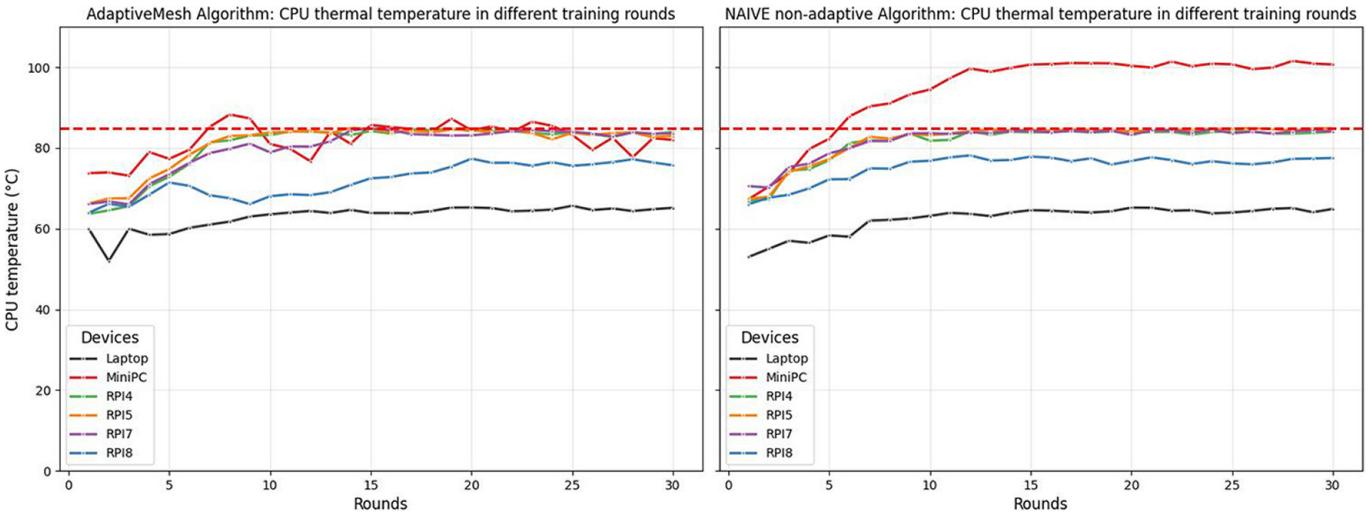

Fig. 4. CPU temperature during training with AdaptiveMesh (left) and NAIVE (right)

Note: AdaptiveMesh keeps temperatures below 85°C, while NAIVE often causes overheating in constrained devices.

Figure 4 (right) depicts the CPU temperatures during various training rounds for different devices using a NAIVE approach, where resources are increased with each round without adaptive management. Figure 4 (right) reveals that model training necessitates substantial computational power and reveals that, except for the laptop, all other devices experience significant increases in CPU temperature, often surpassing the 85°C threshold as the number of images and epochs increases. The MiniPC, in particular, consistently exceeds this limit. The lack of adaptive management in the NAIVE approach results in *higher CPU loads and temperatures*, highlighting the necessity of adaptive algorithms like AdaptiveMesh to ensure optimal device performance.

Training accuracy: Figure 5 (top) shows the accuracy of different devices during training rounds when using an adaptive approach. It is observed that as the rounds increase, the accuracy also increases. The laptop achieved the highest accuracy of 95.1%. The RPI4, RPI5, RPI7, and RPI8 maintained accuracy around 80–83%. The MiniPC showed the lowest accuracy of 57.9%. Adaptive management improved the accuracy and maintained device performance by adjusting training based on predefined thresholds. The AdaptiveMesh algorithm demonstrates a clear trade-off between maintaining model accuracy and optimizing resource usage. The arrows indicate that AdaptiveMesh intervened when the CPU load exceeded 80%, the CPU temperature exceeded 85°C, the bandwidth exceeded 2 Mbps, and the training time limit was exceeded, such as when the MiniPC accuracy dropped to 57.9% after exceeding the thermal threshold. Similarly, Raspberry Pi devices, including RPI4 and RPI5, exhibited accuracy fluctuations around 80–83%, with drops observed during adaptive adjustments triggered by bandwidth exceeding 2 Mbps or training time exceeding 30 minutes. However, despite the slight loss in accuracy, the primary goal is to prevent excessive CPU load, reduce device slowdowns, and minimize battery consumption. Overall, the adaptive approach helps maintain a balance between accuracy and system performance, preventing overloading, as shown by the annotations.

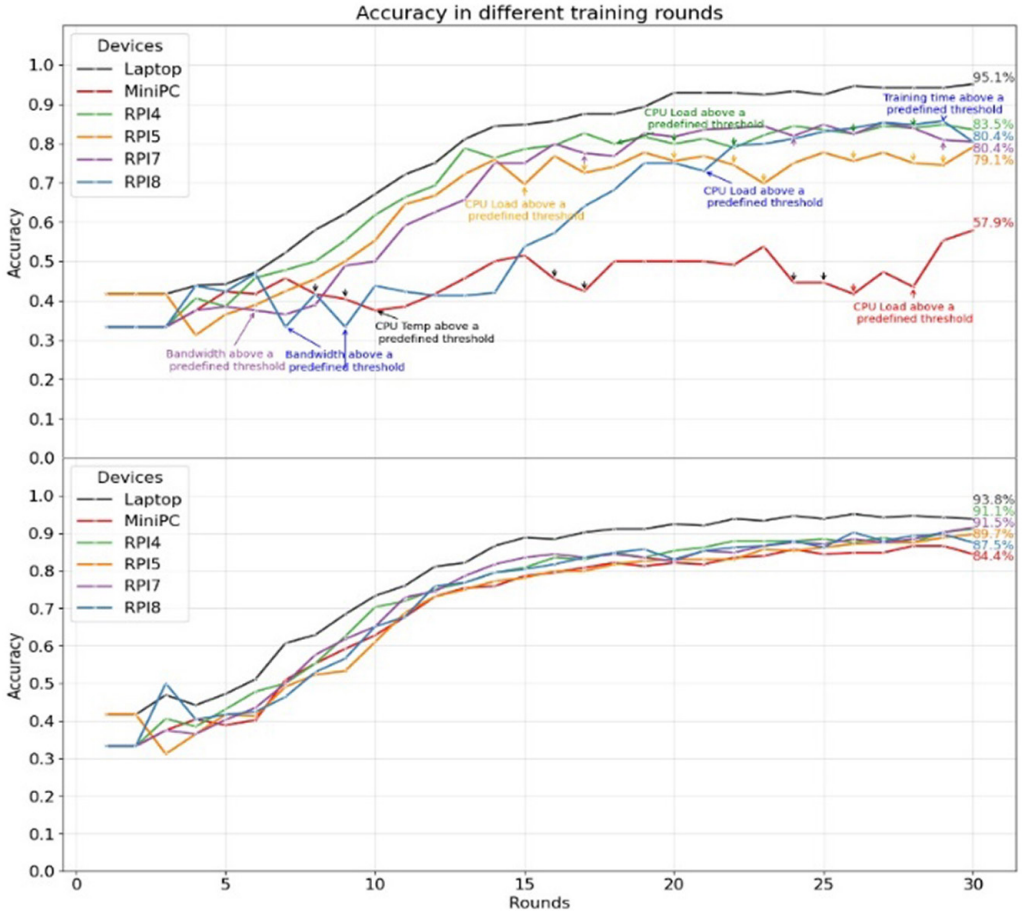

Fig. 5. Accuracy during training with AdaptiveMesh (top) and NAIVE (bottom)

Note: AdaptiveMesh sacrifices up to 5% accuracy to stabilize devices, while NAIVE prioritizes accuracy and potentially training times will be longer.

Figure 5 (bottom) presents the accuracy progression over different training rounds using the NAIVE approach for the same devices. Across all devices, accuracy improves as the number of training rounds increases. This trend is expected since more rounds involve more training data and epochs, enhancing the model's learning capability. The laptop device consistently achieves the highest accuracy by the end of the training rounds. In contrast, the MiniPC shows a lower peak accuracy of 84.4%, likely due to its lower hardware capacity. Devices like RPI4, RPI5, and RPI7 show similar convergence trends, stabilizing around 87–91%. This suggests that these devices, while having different hardware capabilities, perform similarly under the NAIVE approach given enough rounds. The steady increase in accuracy comes at a cost. The NAIVE approach's continuous demand for more resources imposes a significant workload on devices, especially those with lower hardware capacities. This results in higher CPU load, increasing training time and temperature, potentially causing faster battery depletion and reduced device longevity. The increasing training time is a notable drawback, highlighting the need for a balanced approach to maintain accuracy without overburdening system resources.

Epochs per device: Figure 6 (left) displays the number of local epochs per device across different training rounds. It illustrates how the number of epoch changes between training rounds for each device used. It is crucial to highlight the role of the AdaptiveMesh algorithm in this variation. This algorithm automatically reacts not only when the CPU load exceeds 80% but also when the CPU temperature is above 85%, the bandwidth is above 2 Mb/s, and the training time exceeds 30 minutes. The AdaptiveMesh algorithm reduces the resources to lower the load, ensuring stable performance. In the graph, the red arrows indicate moments when this reduction occurs, leading to an immediate decrease in the number of local epochs. The graph analysis demonstrates the effectiveness of the AdaptiveMesh algorithm in managing CPU load and optimizing federated model training. The automatic reduction in the number of epochs and training images ensures a good balance while optimizing bandwidth usage and training time.

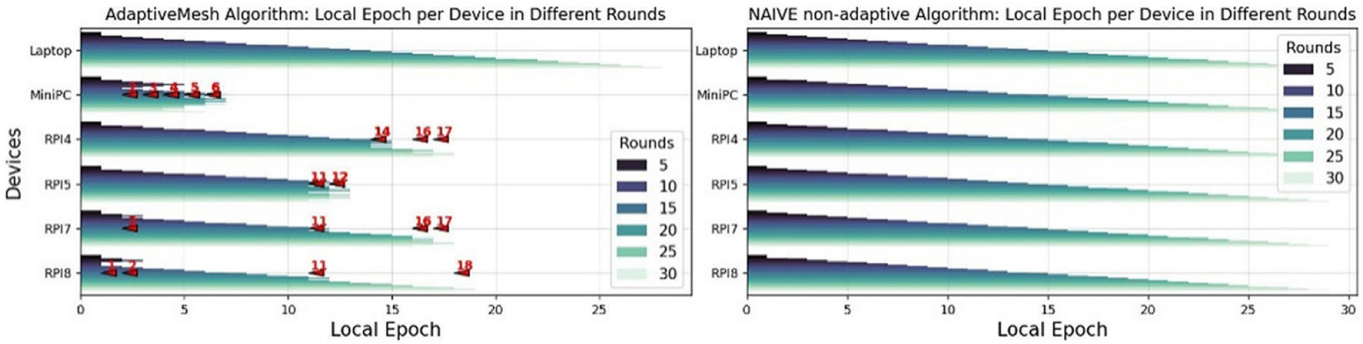

Fig. 6. Number of epochs per client during training with AdaptiveMesh (left) and NAIVE (right)

Note: AdaptiveMesh adjusts epochs to stay within CPU load and temperature limits, while NAIVE increases epochs unchecked, causing inefficiencies and overload.

Figure 6 (right) displays the number of local epochs per device across different training rounds under a NAIVE approach. It illustrates how the number of epochs increases continuously without any reaction to hardware performance or device load. Unlike the AdaptiveMesh algorithm, which dynamically adjusts the number of epochs based on CPU load, temperature, bandwidth, and training time, the naive approach continuously increases the number of epochs regardless of the device's hardware performance or load. This approach does not incorporate any

mechanism that can affect the overall efficiency of the training process, especially on resource-constrained devices.

Trained images per device: Figure 7 (left) illustrates the distribution of training images across various devices over different rounds of training. The AdaptiveMesh algorithm dynamically adjusts the allocation of training resources based on the predetermined threshold, the algorithm reduces the number of training images assigned to devices with lower hardware performance to ensure efficient resource utilization and prevent overloading. The red arrows indicate points where the AdaptiveMesh algorithm has intervened to reduce the number of training images. The mini PC for example, experiences reductions in training images at multiple stages, indicating that these devices frequently encounter threshold exceeding, prompting the AdaptiveMesh algorithm to adjust their workloads. Devices such as the laptop can handle a higher number of training images across more rounds without as frequent interventions, reflecting their relatively better hardware performance.

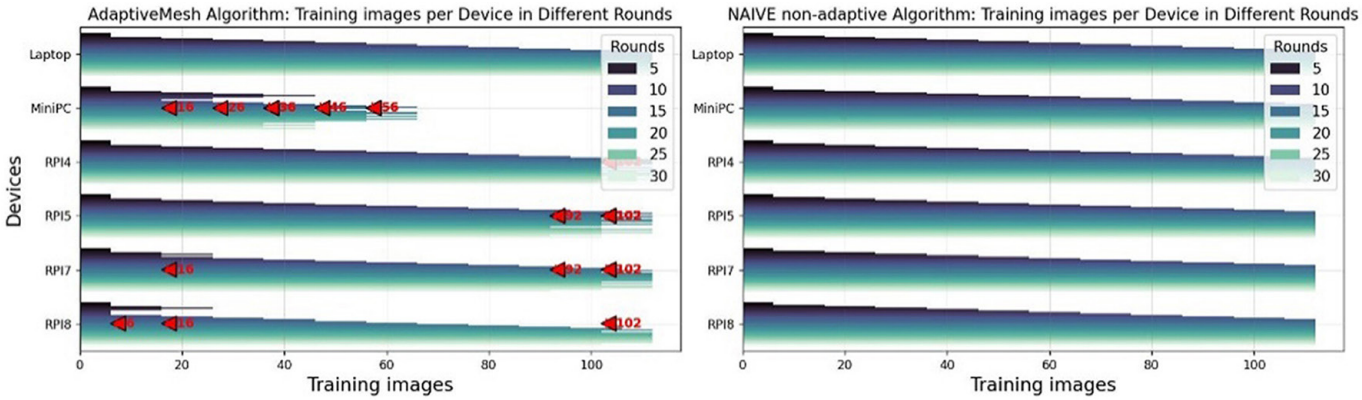

Fig. 7. The number of trained images per client, for AdaptiveMesh (left) and NAIVE (right) algorithms

Note: AdaptiveMesh adapts the workload based on device load to prevent overloading, while NAIVE assigns the same workload to all devices, causing inefficiencies and overload in devices.

This Figure 7 (right) demonstrates the distribution of training images across various devices over different rounds of training using the NAIVE approach. Unlike the AdaptiveMesh algorithm, the NAIVE approach does not adapt or react based on the performance metrics of the devices. As a result, all devices are assigned the same number of training images regardless of their individual hardware capabilities or performance constraints. In the NAIVE approach, each device handles the same workload, which could lead to inefficiencies and potential overloads on devices with lower hardware performance. Unlike the AdaptiveMesh algorithm, there are no interventions or adjustments in the number of training images based on device performance, which results in suboptimal training times and resource utilization. The NAIVE approach can lead to overloading less capable devices, whereas AdaptiveMesh aims to optimize resource allocation by reducing the workload on such devices when necessary.

Training time: Figure 8 (left) depicts the training time for various devices across different training rounds. It shows the adaptive behavior of the AdaptiveMesh algorithm in response to specific thresholds of CPU load, CPU temperature, bandwidth, and training time. Arrows are placed at points where the threshold of CPU load, CPU temperature, bandwidth, and training time is exceeded. The laptop maintains a relatively stable training time. mini PC shows several instances where CPU temperature and load exceed the thresholds, triggering adjustments to reduce training time.

RPI4, RPI5, RPI7, and RPI8 experience small variations in training time, with adaptive measures and other metrics reaching critical levels. This adaptive approach ensures efficient resource usage and optimizes training performance in real-time across different rounds and devices, enhancing the overall efficiency of the FL process. The AdaptiveMesh algorithm's ability to dynamically adjust training parameters in response to hardware performance metrics is crucial for maintaining optimal training times and preventing device overload.

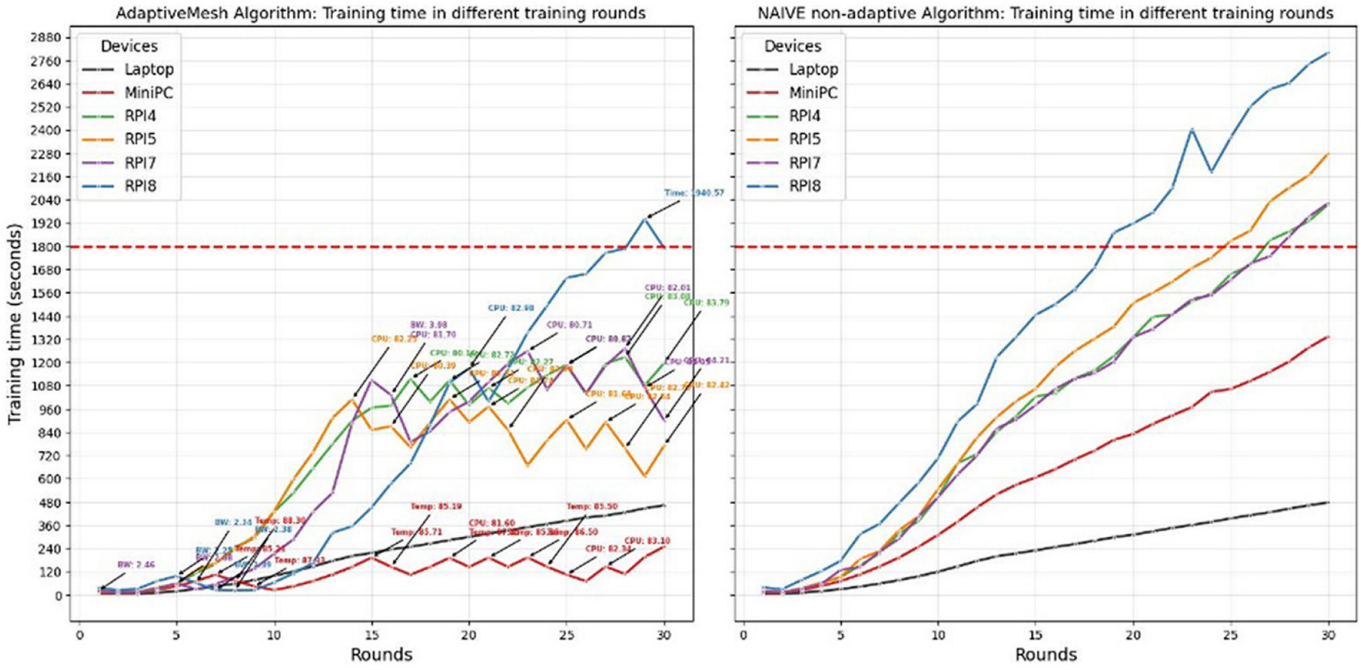

Fig. 8. Training time per client, for AdaptiveMesh (left) and NAIVE (right)

Note: AdaptiveMesh optimizes training time by dynamically adjusting resources, while NAIVE increases training time due to the lack of dynamic resource adaptation.

Figure 8 (right) showcases the training time for each device over 30 training rounds under a naive, non-adaptive approach. Each device's training time increases steadily as the number of training rounds progresses. The lines represent the cumulative training time for each round, reflecting the constant resource allocation without any adaptive measures. The absence of adaptive adjustments leads to continuously increasing training times, potentially leading to inefficiencies and device overloads in resource-constrained environments.

Bandwidth usage: In Figure 9, visualizes the bandwidth utilization across different rounds and devices using the adaptive AdaptiveMesh algorithm. Figure 9 (top left plot) shows the density distribution of rounds. Most rounds are concentrated around the midpoint. In the same figure (top right plot) displays the distribution of devices across different rounds. Each dot represents a device at a specific round. Bottom left plot illustrates the relationship between bandwidth utilization and the number of rounds. Different colors indicate different devices. Finally, the bottom right plot represents the density distribution of bandwidth utilization for each device. This helps to identify the bandwidth usage patterns across different devices. The pair plot effectively visualizes the adaptive nature of the AdaptiveMesh algorithm by showing how bandwidth utilization changes across different rounds and devices. The adaptive approach allows for efficient management of bandwidth, preventing overload and optimizing performance across all devices.

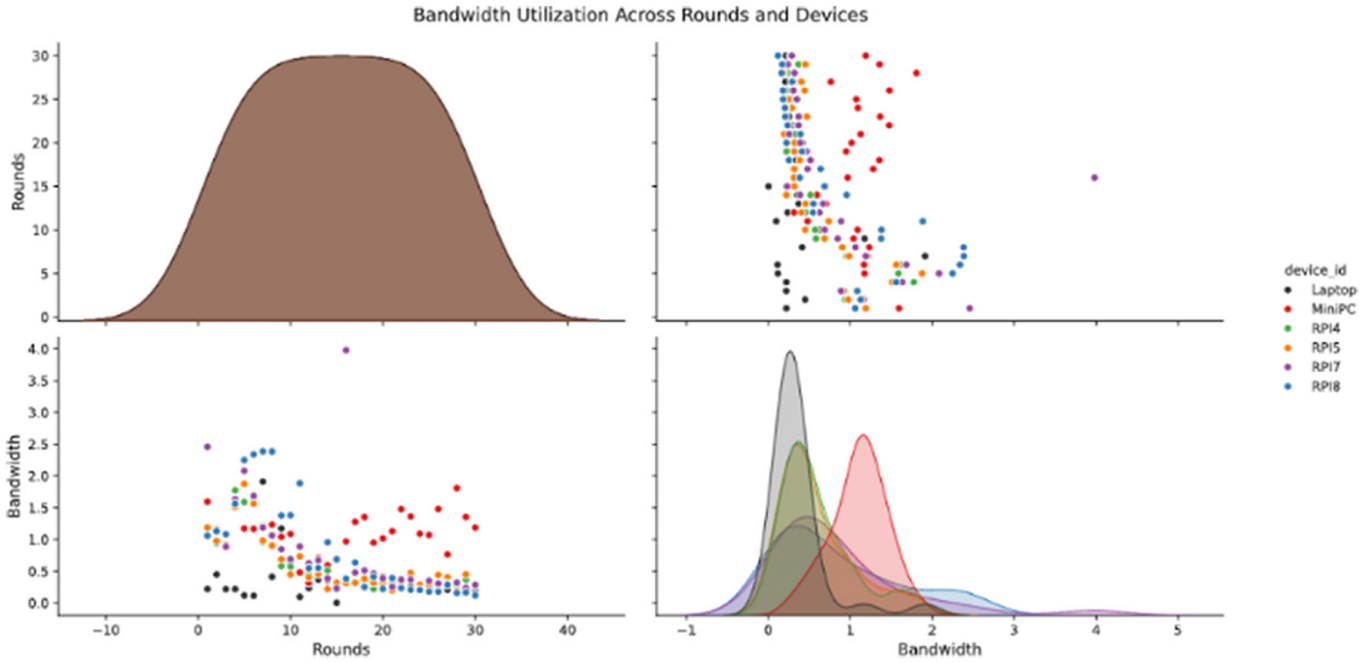

Fig. 9. Bandwidth utilization across rounds and devices using the AdaptiveMesh algorithm

Note: AdaptiveMesh optimizes bandwidth by dynamically adjusting usage, preventing network overload.

In Figure 10, it can be observed that the non-adaptive version shows less uniformity in bandwidth distribution across rounds and devices. The bandwidth utilization is concentrated more in the earlier rounds, suggesting that the non-adaptive algorithm does not regulate bandwidth usage as effectively over time. This could potentially lead to inefficiencies and higher loads on the network early in the process.

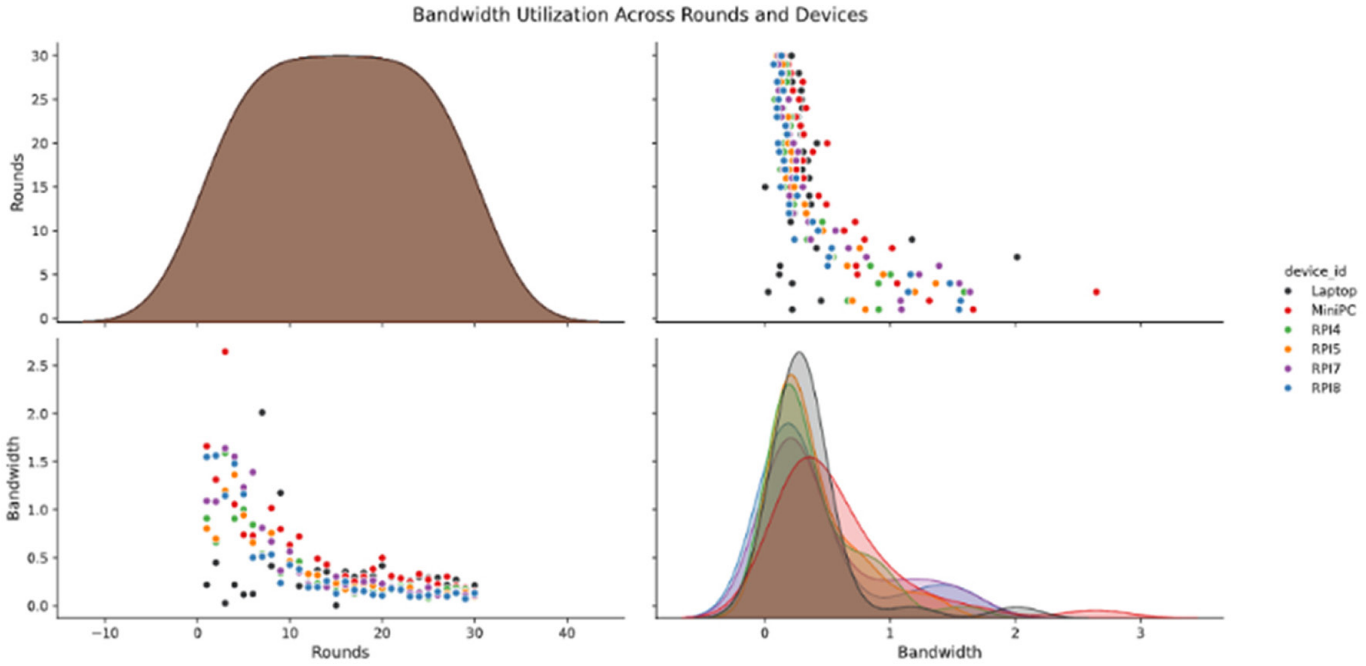

Fig. 10. Bandwidth utilization across rounds and devices (NAIVE Algorithm)

Note: NAIVE leads to suboptimal performance due to unregulated bandwidth usage.

5.2 Mathematical model for regulating context switching

This section presents a mathematical model to regulate context switching, enabling each device to adapt its parameters locally based on real-time resource usage, including CPU load, temperature, and bandwidth. The model builds on the adaptive mechanism illustrated in the flowchart, where each device independently monitors its resource utilization and dynamically adjusts its workload to prevent overloading.

CPU load calculation – At any given training round t, the CPU usage history of a device is maintained for the duration of the round. At the end of each training round, the average CPU usage $\bar{U}_i$ for device i is calculated:

$$\bar{U}_i = \frac{1}{T}\sum_{j=1}^{T} U_{ij}$$

where:

- T is the total number of samples during the training round,
- U_{ij} is the CPU usage of device i at sample j.

This average CPU usage $\bar{U}_i$ is compared against the predefined threshold (80%):

$$\bar{U}_i > \text{Threshold}.$$

Context-switching trigger – If the average CPU usage $\bar{U}_i$ exceeds the threshold, the system reduces both the local training epochs E_i and the number of processed images I_i to balance the computational load. The adjustments are defined as follows:

$$E' = \begin{cases} E_i - \Delta E & \text{if } \bar{U}_i > \text{Threshold}, \\ E_i + \Delta E & \text{otherwise}, \end{cases}$$

$$I' = \begin{cases} I_i - \Delta I & \text{if } \bar{U}_i > \text{Threshold}, \\ I_i + \Delta I & \text{otherwise}, \end{cases}$$

Here:

- ΔE represents a fixed reduction in local training epochs,
- ΔI represents a fixed reduction in the number of processed images.

In addition to CPU usage thresholds (e.g., 80%), the model ensures regulation when:

- Temperature rises above a critical threshold (e.g., 85°C),
- Bandwidth usage exceeds a predefined limit (e.g., 2 Mbps).

By formalizing these thresholds mathematically, the model allows devices to adapt key parameters, such as the number of local training epochs and the quantity of processed data, ensuring stable performance under varying conditions.

5.3 Comparison of AdaptiveMesh, FedAda, and FedALA

We compared AdaptiveMesh with two existing algorithms, FedAda and FedALA, which are used in different FL scenarios for resource-constrained devices.

Table 1. Comparison of AdaptiveMesh, FedAda, and FedALA

Algorithm	Adaptability to Device Load	CPU Usage Optimization	Dynamic Bandwidth Adjustment	Model Accuracy	Training Time
AdaptiveMesh	Yes (based on CPU usage and temperature)	Yes (reduces overload)	Yes (adjusts network usage in real-time)	Slightly lower to maintain device performance	Shorter, as it prevents excessive workload
FedAda	Yes (uses an adaptive workload distribution method)	Not always (focuses on speed of convergence, no mechanism for temperature control)	No (does not adjust bandwidth dynamically)	Higher, but may increase device workload	Improves training speed, but may increase load on devices with limited resources, as it prioritizes accuracy
FedALA	Yes (adapts local aggregation for heterogeneous clients)	Partially (Reduces communication overhead, but does not monitor CPU load in real time)	No (It uses adaptive aggregation to reduce communication overhead, but does not adjust bandwidth dynamically)	Higher on powerful devices but not always for IoT	Improves training speed by reducing communication overhead.

FedAda focuses on distributing training workloads based on device computing power but lacks a specific mechanism for managing CPU overload and temperature. On the other hand, FedALA adapts the aggregation process for heterogeneous clients but does not optimize network bandwidth or training time. In contrast, AdaptiveMesh offers better resource adaptation, dynamically adjusting the number of epochs and training dataset size based on CPU usage, temperature, and bandwidth constraints. This makes it well-suited for resource-constrained environments where stability and resource management are critical.

6 GOOGLE CLOUD VIRTUAL ENVIRONMENT

In order to measure the impact of the wireless environment when training FL models on resource-constrained devices, we also tested the AdaptiveMesh in a Google Cloud environment. Google Cloud is a set of cloud computing services that use the same infrastructure as Google's end-user applications, including Google Search, Gmail, and YouTube.

In Google Cloud, we have created instances that closely resemble our physical hardware, such as a Raspberry Pi 4, a mini PC NEO Z83-4, or a laptop with specific CPU and RAM requirements. For a laptop with an Intel(R) Core (TM) i5-8250U CPU and 8 GB of RAM, we configured a Google Cloud instance using the N2 Series with 4 core 8 logical processors, 8 GB of RAM, and an SSD Persistent Disk. For the mini PC NEO Z83-4, we selected an instance from the E2 Custom Series with 2 cores 4 logical processor, 4 GB of RAM, and a Standard Persistent Disk in Google Cloud.

Google Cloud offers instances with ARM processors under its Tau T2A series. These instances use Ampere Altra processors, which are based on ARM architecture and are similar to the Cortex-A72 in terms of architecture. To create instances comparable to our physical devices, such as Raspberry Pi 4 Model B units with a Quad

Core Cortex-A72 CPU, we chose a T2A instance with 4 cores and 8 logical processors, along with a Standard Persistent Disk to meet basic storage needs, similar to how an SD card is used in a Raspberry Pi.

Figure 11 (left) depicts CPU utilization (%) during different training rounds across various devices simulated as virtual machines in Google Cloud. The dashed red line marks the 80% CPU threshold, which is a limit where AdaptiveMesh starts adapting if this limit is exceeded. As seen in the figure, the laptop exhibits the lowest CPU utilization, starting around 20% and gradually increasing to about 40%, staying well below the 80% threshold. The RPI4, RPI5, RPI7, and RPI8 devices show similar behavior, with varying levels of CPU utilization fluctuating between 60–80%. The MiniPC experiences more significant fluctuations, often surpassing the 80% threshold. However, when the CPU exceeds this limit, the algorithm allocates fewer resources to the device, resulting in a subsequent decrease in CPU utilization. This illustrates the algorithm's effectiveness in managing CPU resources dynamically to prevent overloading during intensive training processes. From this, we observe that CPUs with 4 cores and 8 logical processors handle the training well, while processors with 2 cores and 4 logical processors experience CPU utilization exceeding 80%, which could lead to reduced performance to avoid overheating.

Figure 11 (right) depicts CPU utilization percentages during various training rounds, where the resources are increased with each round without adaptive techniques to prevent CPU overload. In this scenario, CPUs with fewer cores and logical processors (e.g., 2 cores, 4 threads) are likely showing higher CPU utilization more consistently above the 80% threshold, which could indicate increased the risk of overloading.

This lack of adaptation would cause more strain on devices, particularly those with limited resources. AdaptiveMesh adjusts the CPU load during training, ensuring stable performance and extending the lifespan of devices. Unlike the NAIVE approach, which leads to excessive CPU usage and increases the risk of overloading, AdaptiveMesh dynamically manages the workload to prevent such issues.

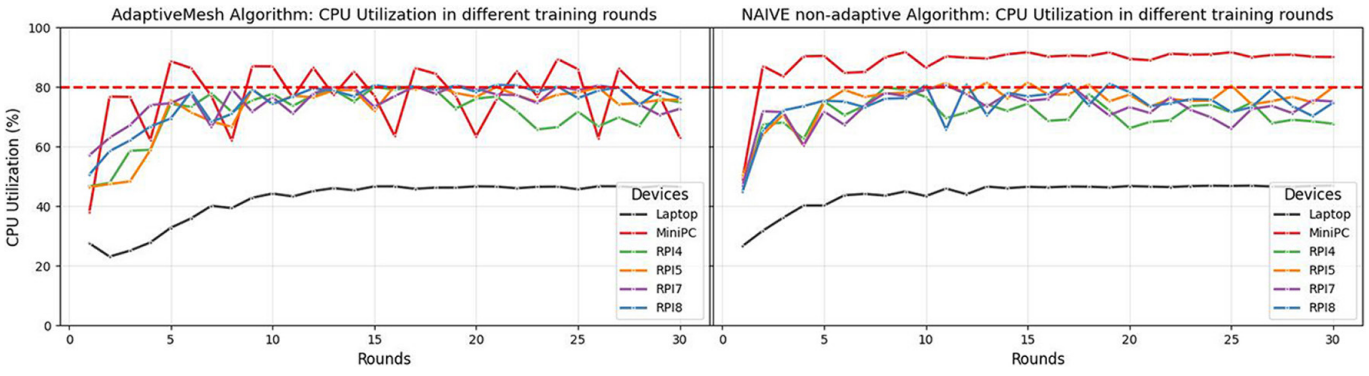

Fig. 11. CPU utilization in different training rounds using AdaptiveMesh (left) and NAIVE (right)

Note: AdaptiveMesh prevents CPU overload in virtual machines by keeping utilization below 80%, even on resource-limited devices. NAIVE allows CPU usage to exceed this threshold, raising the risk of overloading.

Figure 12 (top) illustrates the accuracy with threshold violations (CPU and bandwidth) over 30 rounds using the adaptive method. The initial accuracy is lower compared to the non-adaptive graph, with the lowest accuracy reaching 51.0%. The red arrows indicate where the CPU exceeded its threshold, while the blue arrows show bandwidth threshold violations. These threshold breaches contributed to the reduction in accuracy. Exceeding predefined CPU or bandwidth limits negatively impacted

the training process, but these interventions are crucial for avoiding performance throttling and preventing system overloads.

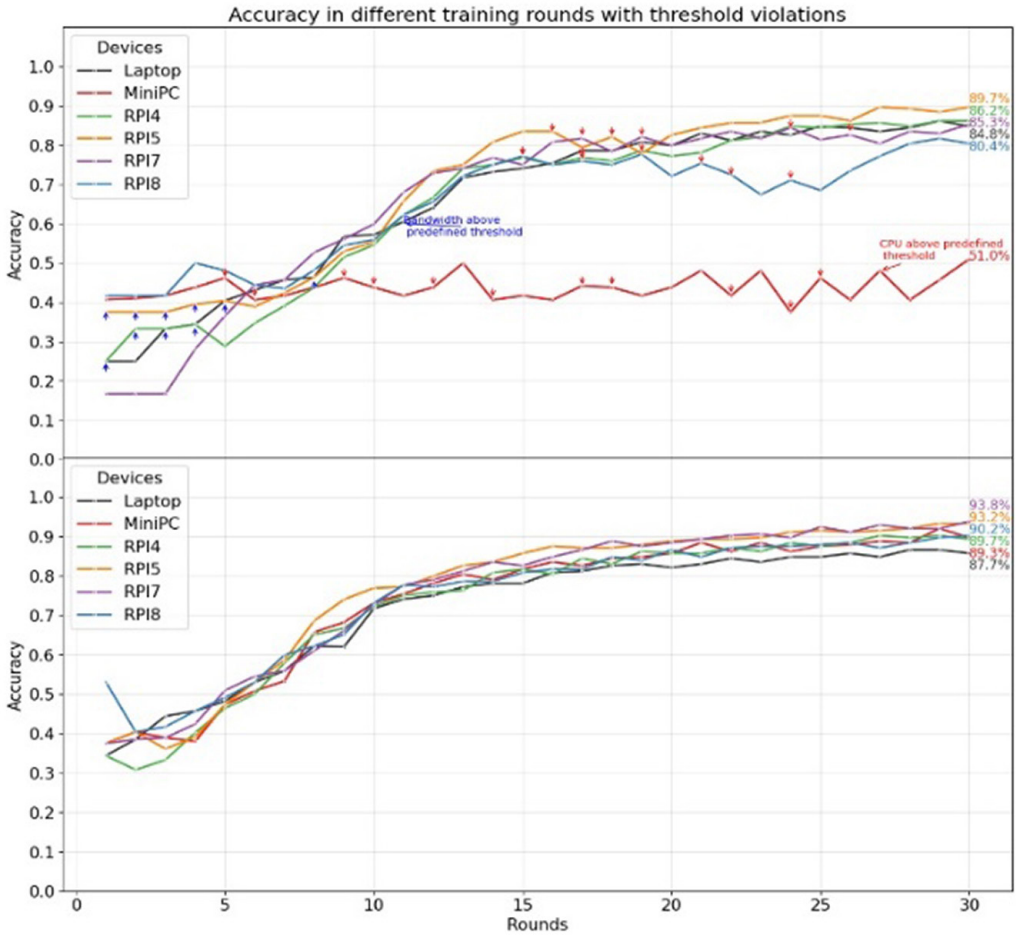

Fig. 12. Accuracy in different training rounds using AdaptiveMesh (top) and NAIVE (bottom) in Google Cloud

Note: AdaptiveMesh maintains stable accuracy by adjusting resources dynamically, while NAIVE achieves higher accuracy at the cost of increased resource strain, risking instability in resource-constrained devices.

Figure 12 (bottom) shows the accuracy over 30 rounds of training for different devices (laptop, mini PC, RPI4, RPI5, RPI7, RPI8) without any adaptive mechanisms. The trend indicates that as training continues, accuracy slightly increases across all devices. The accuracy improvement occurs because resources are increased in each round, allowing the model to learn from more data and perform better over time.

However, continuously increasing rounds, epochs, images, and other resources without adaptation can lead to several issues. First, the CPU load may become excessively high, potentially resulting in performance throttling as the devices attempt to manage the heat generated by the increased processing demands. This can slow down the devices, particularly those with limited hardware resources. Additionally, the prolonged training time due to the increase in training data can lead to inefficiencies, particularly on lower-performance devices. High CPU utilization also increases power consumption, which may deplete battery life faster on portable or IoT devices. Ultimately, without adaptive mechanisms, the constant increase in computational load can cause performance degradation and potentially lead to system instability or restarts.

Similarly, AdaptiveMesh could support industrial IoT applications where sensor data from machinery is analyzed to detect potential failures. By dynamically managing resource allocation, AdaptiveMesh ensures timely analysis of data streams while preventing device overloads in environments with limited computational capacity. These examples highlight the adaptability of AdaptiveMesh across various domains, emphasizing its potential to enhance efficiency and reliability in critical real-world applications.

7 DISCUSSION

The experimental results demonstrate a clear advantage of the AdaptiveMesh adaptive algorithm over the non-adaptive NAIVE approach in both physical and virtual environments. The adaptive nature of AdaptiveMesh allows it to dynamically manage critical resources such as CPU load, temperature, and bandwidth, ensuring efficient training while preventing device overheating and overuse. In contrast, the NAIVE approach, which lacks such mechanisms, consistently leads to higher resource consumption, increased CPU load, and the risk of overheating, particularly on resource-constrained devices such as Raspberry Pis and mini PCs.

The adaptive management in AdaptiveMesh proves essential for devices with limited computational power, as shown by the significant differences in performance across various metrics. For instance, devices such as the Mini PC and some Raspberry Pi benefited greatly from the real-time adjustments in resources, which prevented the CPU from exceeding the predefined limits. This ensured stable and continuous training, even in rounds where the computational load increased. On the other hand, the NAIVE algorithm's lack of adaptive mechanisms caused these devices to frequently surpass critical thresholds, leading to the potential risk of system instability and device failure.

Moreover, the results highlight the difference in performance between physical and virtual environments. AdaptiveMesh performed robustly on physical devices, maintaining lower CPU loads and temperatures, thanks to its ability to adjust the training process. However, in virtual environments such as Google Cloud Platform, where computational resources are more flexible, the adaptive mechanisms of AdaptiveMesh provided even more consistent and efficient performance. This suggests that the AdaptiveMesh algorithm is well-suited for diverse deployment scenarios, offering a flexible solution for environments with varying resource constraints. The findings also reveal that virtual environments show similar behavior to physical devices when resource thresholds are exceeded. However, a key difference lies in the ability to increase computational resources in the cloud. Additional cores can be allocated if a virtual instance struggles to handle training workloads despite AdaptiveMesh interventions, reducing performance issues. This comparison underlines that while AdaptiveMesh effectively maintains stability across both physical and virtual environments, the scalability of cloud resources offers an advantage in high-demand scenarios.

However, the AdaptiveMesh algorithm has some limitations that need to be considered. The balance between accuracy and resource usage was another critical outcome of this study. Although AdaptiveMesh sometimes sacrificed accuracy to preserve system stability, the overall trade-off between model performance and device safety was favorable. Temporary drops in accuracy by approximately 3–5% were observed when thresholds for CPU load (80%), CPU temperature (85°C), or bandwidth (2 Mbps) were exceeded. These reductions occurred because the algorithm

dynamically reduced the number of epochs and training images to prevent device overload. Despite these reductions, the overall benefits of AdaptiveMesh in maintaining system longevity and preventing device failures outweigh the slight decrease in accuracy. This trade-off is particularly important in real-world applications, where ensuring continuous device functionality often takes precedence over marginal gains in model performance.

The NAIVE approach, while yielding slightly higher accuracy in some instances, did so at the cost of excessive resource consumption and overheating risks, which could compromise the long-term functionality of the devices.

The reliance on predefined thresholds for adaptation presents challenges, as selecting optimal values can be dependent on context and affect overall performance. Additionally, while continuous monitoring of resource metrics introduces a small computational overhead, this is offset by the significant improvements in resource efficiency and system stability achieved by AdaptiveMesh. This lightweight monitoring process is optimized to minimize energy consumption, making it suitable even for resource-constrained devices. These aspects also present opportunities for future research to further refine monitoring mechanisms' efficiency.

AdaptiveMesh can be important for devices operating in wireless networks, where limited bandwidth and fluctuating CPU loads directly impact overall performance.

By reacting in real time to these variations, AdaptiveMesh can be deployed for different applications such as mobile healthcare services or industrial monitoring systems, where devices are particularly sensitive to resource overload. The practical relevance of AdaptiveMesh becomes evident in its ability to enhance critical real-world applications. In healthcare, AdaptiveMesh can optimize resource usage in patient monitoring systems that rely on wearable devices or portable medical equipment. For instance, in remote monitoring of patients with chronic conditions, it dynamically adjusts workloads to balance computational demands and battery life. Similarly, in mobile fitness apps, AdaptiveMesh efficiently processes real-time sensor data, such as heart rate and step count, while improving device performance and battery longevity. In industrial IoT, the algorithm supports predictive maintenance by analyzing sensor data in resource-constrained environments without overloading devices. These examples demonstrate AdaptiveMesh's potential to enhance efficiency, reliability, and device longevity across diverse domains.

8 CONCLUSION

In conclusion, this study provides compelling evidence of the effectiveness of the AdaptiveMesh algorithm in optimizing FL in both physical and virtual environments. AdaptiveMesh's adaptive management of CPU load, temperature, and bandwidth significantly reduces the risk of device overheating and ensures a more efficient allocation of resources during the training process. By dynamically adjusting the resources based on real-time performance metrics, AdaptiveMesh outperforms the NAIVE approach, particularly in resource-constrained environments.

The comparative analysis between physical devices and virtual machines also underscores the algorithm's versatility. AdaptiveMesh not only maintains system stability on devices with limited computational power but also capitalizes on the flexibility offered by cloud environments, delivering superior performance across diverse settings. Moreover, its applicability to mobile environments highlights its potential to enhance real-time applications, such as personalized learning platforms, mobile healthcare services, and industrial monitoring systems, where device efficiency is critical.

In summary, AFL approaches like AdaptiveMesh are crucial for the efficient deployment of FL in real-world applications, particularly in heterogeneous and resource-constrained environments. The insights from this study provide a strong foundation for future work in developing more sophisticated adaptive algorithms that can further enhance FL performance while minimizing resource strain on devices.

Future research could focus on further enhancing adaptive mechanisms. One important area for future development is the creation of a system that not only monitors real-time performance but also accurately predicts resource load variations using machine ML learning techniques. This could be achieved by integrating ML techniques such as neural networks or reinforcement learning to predict CPU and bandwidth usage based on historical data and workload patterns. For instance, predictive models could analyze device-specific metrics to predict resources, enabling AdaptiveMesh to preemptively allocate or adjust resources before thresholds are exceeded. This would enable a higher degree of adaptation and further expand the use of FL in more advanced and larger-scale applications.

9 ACKNOWLEDGMENT

This work was funded by South East European University through the project: "Assessing the Nexus Between Air Pollution and Human Mobility in the City of Tetova".

10 REFERENCES

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, vol. 54, 2017, pp. 1273–1282. https://proceedings.mlr.press/v54/mcmahan17a.html

[2] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine,* vol. 37, no. 3, pp. 50–60, 2020. https://doi.org/10.1109/MSP.2020.2975749

[3] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. Vincent Poor, "Federated learning for internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials,* vol. 23, no. 3, pp. 1622–1658, 2021. https://doi.org/10.1109/COMST.2021.3075439

[4] L. Shkurti and M. Selimi, "AdaptiveMesh: Adaptive federate learning for resource-constrained wireless environments," *Int. J. Onl. Eng.*, vol. 20, no. 14, pp. 22–37, 2024. https://doi.org/10.3991/ijoe.v20i14.50559

[5] L. Shkurti and M. Selimi, "Baca: Bandwidth and cpu-aware adaptive federated learning for wireless environments," in *2024 13th Mediterranean Conference on Embedded Computing (MECO)*, 2024, pp. 1–5. https://doi.org/10.1109/MECO62516.2024.10577810

[6] A. Imteaj, K. Mamun Ahmed, U. Thakker, S. Wang, J. Li, and M. H. Amini, "Federated learning for resource-constrained IoT devices: Panoramas and State of the Art," in *Federated and Transfer Learning*, R. Razavi-Far, B. Wang, M. E. Taylor, and Q. Yang, Eds., Springer, Cham, 2023, pp. 7–27. https://doi.org/10.1007/978-3-031-11748-0_2

[7] M. H. Alsharif *et al.*, "A comprehensive survey of energy-efficient computing to enable sustainable massive IoT networks," *Alexandria Engineering Journal,* vol. 91, pp. 12–29, 2024. https://doi.org/10.1016/j.aej.2024.01.067

[8] G. Rjoub, O. A. Wahab, J. Bentahar, R. Cohen, and A. S. Bataineh, "Trustaugmented deep reinforcement learning for federated learning client selection," *Information Systems Frontiers*, vol. 26, pp. 1261–1278, 2024. https://doi.org/10.1007/s10796-022-10307-z

[9] S. Samarakoon, M. Bennis, W. Saad, and M. Debbah, "Distributed federated learning for ultra-reliable low-latency vehicular communications," *IEEE Transactions on Communications*, vol. 68, no. 2, pp. 1146–1159, 2020. https://doi.org/10.1109/TCOMM.2019.2956472

[10] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "Federated learning for resource-constrained IoT devices," *arXiv preprint arXiv.2002.10610*, 2020.

[11] P. Kairouz *et al.*, "Advances and open problems in federated learning," *Found. Trends Mach. Learn*, vol. 14, pp. 1–210, 2019.

[12] C. T. Dinh, N. H. Tran, M. N. H. Nguyen, C. S. Hong, W. Bao, and A. Y. Zomaya, "Federated learning over wireless networks: Convergence analysis and resource allocation," *IEEE/ACM Trans. Netw*, vol. 29, no. 1, pp. 398–409, 2021. https://doi.org/10.1109/TNET.2020.3035770

[13] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–19, 2019. https://doi.org/10.1145/3298981

[14] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "A survey on federated learning for resource-constrained IoT devices," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 1–24, 2022. https://doi.org/10.1109/JIOT.2021.3095077

[15] L. Shkurti, M. Selimi, and A. Besimi, "Federatedmesh: Collaborative federated learning for medical data sharing in mesh networks," in *Collaborative Computing: Networking, Applications and Worksharing*, H. Gao, X. Wang, and N. Voros, Eds., Springer, Cham, vol. 563, 2024, pp. 154–169. https://doi.org/10.1007/978-3-031-54531-3_9

[16] H. Zhang, Z. Xie, R. Zarei, T. Wu, and K. Chen, "Adaptive client selection in resource constrained federated learning systems: A deep reinforcement learning approach," *IEEE Access*, vol. 9, pp. 98423–98432, 2021. https://doi.org/10.1109/ACCESS.2021.3095915

[17] B. Ankayarkanni, N. K. Pani, M. Anand, V. Malathy, and Bhupati, "P2FLF: Privacy-preserving federated learning framework based on mobile fog computing," *Int. J. Interact. Mob. Technol.*, vol. 17, no. 17, pp. 72–81, 2023. https://doi.org/10.3991/ijim.v17i17.42835

[18] N. Tabassum, M. Ahmed, N. J. Shorna, M. M. U. R. Sowad, and H. M. Z. Haque, "Depression detection through smartphone sensing: A federated learning approach," *Int. J. Interact. Mob. Technol.*, vol. 17, no. 1, pp. 40–56, 2023. https://doi.org/10.3991/ijim.v17i01.35131

[19] B. G. Mohammed and D. S. Hasan, "Smart healthcare monitoring system using IoT," *Int. J. Interact. Mob. Technol.*, vol. 17, no. 1, pp. 141–152, 2023. https://doi.org/10.3991/ijim.v17i01.34675

[20] J. Zhang *et al.*, "FedAda: Fast-convergent adaptive federated learning in heterogeneous mobile edge computing environment," *World Wide Web*, vol. 25, pp. 1971–1998, 2022. https://doi.org/10.1007/s11280-021-00989-x

[21] J. Zhang *et al.*, "Fedala: Adaptive local aggregation for personalized federated learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023, vol. 37, no. 9, pp. 11237–11244. https://doi.org/10.1609/aaai.v37i9.26330

[22] L. Luo, C. Zhang, H. Yu, G. Sun, S. Luo, and S. Dustdar, "Communication-efficient federated learning with adaptive aggregation for heterogeneous client-edge-cloud network," *IEEE Transactions on Services Computing*, vol. 17, no. 6, pp. 3241–3255, 2024. https://doi.org/10.1109/TSC.2024.3399649

[23] Y. Xu, Z. Ma, H. Xu, S. Chen, J. Liu, and Y. Xue, "Fedlc: Accelerating asynchronous federated learning in edge computing," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 5327–5343, 2024. https://doi.org/10.1109/TMC.2023.3307610

[24] A. Imteaj and M. H. Amini, "Fedparl: Client activity and resource-oriented lightweight federated learning model for resource-constrained heterogeneous IoT environment," *Frontiers in Communications and Networks*, vol. 2, 2021. https://doi.org/10.3389/frcmn.2021.657653

[25] Paul Mooney, "Chest X-Ray Images (Pneumonia)," *Kaggle*, 2018. https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia

11 AUTHORS

Lamir Shkurti is a Ph.D. candidate at the Faculty of Contemporary Sciences and Technologies, Southeast European University in North Macedonia. He is also employed as a Lecturer at the University for Business and Technology in Kosovo. His study interests include federated learning for resource-constrained platforms, embedded machine learning, Internet of Things, Wireless Mesh Networks, Cloud Computing, virtualization technology, software development and algorithm optimization (E-mail: ls29773@seeu.edu.mk).

Mennan Selimi is an Associate Professor at South East European University, North Macedonia. He is the head of the Distributed Systems and Data Science research group at Max van der Stoel Institute. Previously, he was a Postdoctoral Research Associate at University of Cambridge working with the N4D Lab. He has a Phd in Distributed Computing (with Distinction and Honors) from UPC BarcelonaTech and IST Lisbon. His study interests focus on decentralized cloud infrastructures looking at topics ranging from machine learning and blockchain to network economics. More: https://mvdsi.seeu.edu.mk/mselimi/.