information

Article

# Real-World Implementation and Performance Analysis of Distributed Learning Frameworks for 6G IoT Applications

David Naseh *, Mahdi Abdollahpour and Daniele Tarchi

Department of Electrical, Electronic and Information Engineering "Guglielmo Marconi", University of Bologna, 40126 Bologna, Italy; mahdi.abdollahpour@unibo.it (M.A.); daniele.tarchi@unibo.it (D.T.)
* Correspondence: david.naseh2@unibo.it

**Abstract:** This paper explores the practical implementation and performance analysis of distributed learning (DL) frameworks on various client platforms, responding to the dynamic landscape of 6G technology and the pressing need for a fully connected distributed intelligence network for Internet of Things (IoT) devices. The heterogeneous nature of clients and data presents challenges for effective federated learning (FL) techniques, prompting our exploration of federated transfer learning (FTL) on Raspberry Pi, Odroid, and virtual machine platforms. Our study provides a detailed examination of the design, implementation, and evaluation of the FTL framework, specifically adapted to the unique constraints of various IoT platforms. By measuring the accuracy of FTL across diverse clients, we reveal its superior performance over traditional FL, particularly in terms of faster training and higher accuracy, due to the use of transfer learning (TL). Real-world measurements further demonstrate improved resource efficiency with lower average load, memory usage, temperature, power, and energy consumption when FTL is implemented compared to FL. Our experiments also showcase FTL's robustness in scenarios where users leave the server's communication coverage, resulting in fewer clients and less data for training. This adaptability underscores the effectiveness of FTL in environments with limited data, clients, and resources, contributing valuable information to the intersection of edge computing and DL for the 6G IoT.

**Keywords:** federated learning; transfer learning; virtual machines; Raspberry Pi; Odroid; proof-of-concept

check for updates

**Citation:** Naseh, D.; Abdollahpour, M.; Tarchi, D. Real-World Implementation and Performance Analysis of Distributed Learning Frameworks for 6G IoT Applications. *Information* **2024**, *15*, 190. https://doi.org/10.3390/info15040190

Academic Editor: Shingo Yamaguchi

Received: 29 February 2024
Revised: 28 March 2024
Accepted: 29 March 2024
Published: 29 March 2024

## 1. Introduction

The imminent arrival of 6G technology heralds a new era marked by intelligence, full connectivity, and digitalization, achieved through the extensive deployment of distributed intelligent networks. This technological advancement is poised to provide a diverse array of intelligent services and applications tailored to specific demands. The success witnessed in 5G solutions underscores the pivotal role of Internet of Things (IoT) technology, which is expected to play an equally crucial role in the 6G society. As the IoT seamlessly integrates into various wireless scenarios, the 6G landscape anticipates the deployment of numerous sensory nodes capable of sensing their surroundings and generating copious amounts of high-quality data, crucial to enabling intelligent solutions in 6G networks.

Machine learning (ML) plays a central role in the 6G vision, specifically in the empowerment of intelligent services. The deployment of various ML methods across different wireless scenarios is expected to facilitate intelligent solutions by analyzing 6G network data and extracting valuable insights. However, the conventional centralized learning (CL) method, which concentrates training data on a single server, is inefficient when viewed from a 6G perspective. Given the need for next-generation networks to manage vast amounts of information that is predominantly generated at the network's edge, centralizing data incurs communication overheads and network traffic, limiting its applicability in the resource-constrained and time-critical 6G environment.

An efficient alternative to CL is distributed learning (DL), emphasizing decentralization of training. This approach carries out the training phase directly on end devices or strategically positioned nodes at the edge of the network. DL enables the training of models by processing local data on each device, harnessing large amounts of heterogeneous data across the network. Beyond improving privacy and reducing communication overhead, DL distributes the computational load between devices, leading to improvements in energy management.

Within the realm of DL, federated learning (FL) has emerged as a successful approach to building high-quality ML models based on dispersed wireless data. The traditional FL approach involves devices with their datasets and a central server node. While offering advantages such as reduced data transmission costs and enhanced data privacy, FL faces challenges in the form of heterogeneous nodes with varying capabilities and the time required for model convergence, particularly in latency-critical 6G use cases.

Recognizing the challenges posed by limited client resources in the FL framework, transfer learning (TL) is introduced as a strategic solution to fine-tune deep neural networks (DNNs). Implementing FL directly on clients for all training processes becomes impractical due to resource constraints, making it challenging to achieve satisfactory accuracy. TL leverages pre-trained models and adapts them to the specific task at hand, offering an efficient hybrid approach that overcomes the limitations of resource constraints and time challenges in the 6G landscape. This incorporation of TL within FL promises to improve the feasibility and effectiveness of training DNNs in the context of 6G networks.

### *1.1. Technological Background*

ML technology has attracted significant attention for empowering intelligent solutions in various wireless networks, spanning mobile communication networks [1], wireless sensor networks [2], transportation systems [3], and non-terrestrial networks (NTNs) [4]. It proves instrumental in addressing intricate challenges within wireless communication, such as resource management [5], data offloading to edge networks [6], spectrum management [7], routing [8], and user server allocation [9], among others. Various ML techniques offer effective solutions to these complex problems. In particular, FL stands out as a widely adopted DL approach that provides efficient solutions. In a specific case, a paper [10] proposes energy-efficient FL solutions customized for wireless communication environments. Furthermore, research in [11] explores the applicability of FL solutions in smart city environments, demonstrating their versatility. The utility of FL extends to solving challenges within vehicular networks, particularly in edge computing environments [12], digital twin-based vehicular networks [13], resource allocation, and task offloading [14], and finds applications in diverse satellite networks [15]. Despite the extensive analyses of FL performance in various wireless settings, the existing body of work often lacks considerations for practical implementations.

Recently, various scholars have explored the implementation of a learning testbed that uses various sensory nodes to evaluate the performance of ML solutions. Raspberry Pi devices have become a common choice for assessing the efficacy of ML solutions that address various challenges in wireless networking. For instance, in [16], reinforcement learning (RL)-based solutions were proposed to optimize routing processes in software-defined wireless sensor networks, with Raspberry Pi devices serving as sensor nodes for performance analysis. Another study [17] presented deep learning-based solutions for speed bump detection in intelligent vehicular systems, employing Raspberry Pi devices and associated camera modules for testing.

In the realm of healthcare monitoring, an approach called distributed incremental learning (DIL) was introduced in [18] to mitigate *catastrophic forgetting*. However, the application of this approach faces challenges on Raspberry devices due to the large model size (49 KB), and the absence of details regarding data batch quality poses obstacles to convergence. Communication-efficient FL solutions, proposed in [19], were tested using Raspberry Pi devices in edge computing environments. Furthermore, the researchers in [20] delved into the analysis of FL performance in the presence of heterogeneous clients, providing valuable information for the integration of different IoT subsystems with heterogeneous nodes in the context of 6G networks.

FL has witnessed numerous advancements in recent years, extending its applicability to various domains and addressing various challenges. In particular, FL has been integrated with wireless channel properties, allowing devices to transmit model updates simultaneously and enabling automatic model aggregation through the wireless channel. This paradigm, known as unit-modulus over-the-air computation [21], improves the efficiency and scalability of FL in wireless communication settings.

Moreover, advancements in FL have led to the development of hierarchical federated learning frameworks tailored for complex applications, such as end-to-end autonomous driving [22]. These hierarchical approaches enable distributed model training across multiple layers of a hierarchical architecture, facilitating collaborative learning among interconnected components while ensuring efficient information exchange and decision-making in dynamic environments.

Additionally, recent research has explored the integration of task-oriented sensing, computation, and communication in federated learning settings [23]. By aligning sensing, computation, and communication resources with specific tasks or objectives, these integrated FL frameworks optimize resource allocation and coordination, thus improving the effectiveness and adaptability of federated learning systems in real-world applications.

Despite these contributions, to the best of our knowledge, the existing literature lacks a comprehensive study focusing on the implementation and performance analysis of a real-world DL scenario with limited resources. Such scenarios require the utilization of TL for DNNs on end-users and clients, as exemplified by Raspberry Pis. This deficiency in the literature motivates our experimental analysis of the FL process in the presence of diverse sets of clients with varying resource capacities. This study aims to bridge the gap in understanding the practical implications and performance aspects of DL scenarios in resource-constrained environments, shedding light on the effectiveness of TL applications.

### *1.2. Contributions and Novelties*

This study presents several significant contributions and novel insights into the realm of DL frameworks, particularly focusing on federated transfer learning (FTL) and FL methodologies. The key contributions and novelties of our research are outlined as follows:

1. **Implementation and efficiency analysis**: We deploy FTL and FL frameworks on a real-world platform comprising heterogeneous devices, including Raspberry Pis, Odroid, and virtual machines. Through comprehensive efficiency assessments that include accuracy, convergence rate, and loss metrics, we demonstrate the superiority of FTL. In particular, FTL exhibits improved accuracy and reduced loss over shorter training durations compared to FL.
2. **Dynamic measurement and comparison of technical parameters:** Our study uses dynamic measurement techniques to evaluate key technical parameters, such as load average, memory usage, temperature, power, and energy consumption, in DL methodologies. The findings reveal that FTL exhibits a lower average load on processing units and memory usage, while consuming less energy and power. This aspect is particularly crucial in emerging 6G scenarios characterized by resource-limited devices.
3. **Scalability analysis:** We perform scalability analyses to explore the impact of varying user counts and dataset sizes on the performance of the DL framework. Our results confirm the robustness and reduced sensitivity of the FTL to these variations, highlighting its adaptability and reliability in diverse operational scenarios.
4. **Innovative implementation strategies:** We introduce novel implementation strategies to address the practical challenges encountered during the implementation of the DL framework. In particular, we employ an asymmetric data distribution to closely emulate real-world scenarios, enhancing the applicability and robustness of our findings. Furthermore, we implemented cooling systems using fans and heat sinks to mitigate overheating and processing capability degradation, thus optimizing training

performance and ensuring the reliability of our DL frameworks. Additionally, the introduction of memory swap functionality, which addresses the limited memory capacities of Raspberry Pis, represents a novel approach to enhancing the scalability and efficiency of DL frameworks.

### *1.3. Limitations*

Although our study offers valuable information on the efficiency and performance of the FTL and FL frameworks, several limitations warrant acknowledgment. One notable limitation is the relatively limited number of users involved in our implementations. This constraint may restrict the generalizability of our findings to larger-scale DL environments. Additionally, while the introduction of memory swap functionality represents a novel approach, further research is warranted to explore its implications on system performance and scalability in more extensive DL scenarios. Future research efforts could address these limitations by incorporating larger user cohorts and investigating alternative memory management strategies to further enhance the robustness and applicability of DL frameworks.

This paper explores DL frameworks, specifically FL and FTL, and implements them on IoT devices within the context of 6G technology. The paper begins with an introduction that outlines the objectives and necessity of DL frameworks in the evolving landscape of 6G and the IoT. Section 2 delves into the fundamentals of FL and extends the discussion to FTL, highlighting their applications and challenges. Following this, Section 3 details the implementation of the system, focusing on server and client configurations and functionalities. Experimental results and performance evaluations are presented in Section 4, with Experiment 1 analyzing DL with five heterogeneous clients and Experiment 2 exploring DL with three heterogeneous clients in order to analyze the scalability of the system. Through experimentation, the paper demonstrates the superiority of FTL over FL in terms of performance and efficiency. Finally, the conclusion summarizes the key findings and contributions of the study, emphasizing the added benefits and importance of FTL in heterogeneous IoT environments.

## **2. Distributed Learning (DL)**

The traditional machine learning (ML) method functioned within a centralized framework, where distributed wireless nodes, acting as potential data origins, sent their data samples to a central, more powerful node with substantial storage and computational capabilities. With increasing interest in the 5G system and the upcoming 6G technology, the wireless environment is filled with small devices capable of sensing the environment and producing vast amounts of high-quality data. Managing such a large volume of data using a conventional centralized approach could lead to significant communication overhead. Conversely, collecting data from multiple distributed nodes to create a comprehensive dataset at the central server node could result in notably higher training expenses. Furthermore, the emergence of new intelligent services and applications sets strict demands in terms of latency, privacy, and reliability, creating obstacles for centralized ML model training in wireless settings.

Recent progress in hardware and software technologies has significantly enhanced the onboard functionalities of end devices. With this enhanced capacity, these devices are now able to independently train complex machine learning models using their own datasets, removing the necessity of transmitting data over long distances and decreasing the extra training workload. Moreover, devices can interact with each other and server nodes to improve machine learning models, thereby boosting performance. This has given rise to a new approach in machine learning model training called distributed learning, which seeks to overcome the constraints of conventional centralized techniques [24]. Different types of distributed learning methods have been investigated recently.

There are two main strategies for implementing distributed learning: distributing data in parallel or distributing models [25]. The former method involves spreading out the training data across multiple servers, while the latter method involves splitting the model's parameters among different servers. However, the complexity of dividing machine learning models into separate parameter groups poses a challenge when applying the parallel model approach. As a result, most distributed machine learning systems primarily rely on distributing data. Some prominent deep learning techniques include federated learning (FL), collaborative learning, multi-agent reinforcement learning (MARL), and split learning (SL) [26]. Among these, FL and its variations, like FTL, have become widely adopted in wireless networks due to their effectiveness in facilitating intelligent solutions.

## *2.1. Federated Learning (FL)*

FL acts as a decentralized structure for distributed machine learning, ensuring privacy by conducting learning directly on the end devices [27]. This deep learning approach allows the collaborative training of models across decentralized data sources by iteratively exchanging model updates while maintaining the raw data on individual devices. The FL algorithm, illustrated in Algorithm 1, functions in numerous rounds, with each round comprising several steps:

1. **Initialization:** At the beginning of the process, a global model ($\theta$) is initialized. This global model serves as the starting point for training across all devices (Line 3).
2. **Iteration (round):** The algorithm iterates for a predetermined number of rounds. This iterative process aims to minimize a specific function for efficient FL [28]. Each round involves the following steps:
    (a) **Device interaction loop:** Within each round, the algorithm interacts with each device (representing different data sources) individually (Lines 5–10).
        - **Broadcasting model:** The current global model ($\theta$) is sent to each device participating in the process (Line 6).
        - **Local training:** Each device trains a local model ($\theta_i$) using its own local dataset ($D_i$) (Line 7). This ensures that models are trained using data that remain on the respective devices, preserving privacy.
        - **Local model update:** After training, each device computes a local gradient update ($\Delta\theta_i$) based on its local loss function ($L(\theta_i, D_i)$) (Line 8).
        - **Sending update:** The devices send their respective gradient updates ($\Delta\theta_i$) to a central server for aggregation (Line 9).

    (b) **Aggregation:** The central server receives the gradient updates from all devices and aggregates them to update the global model ($\theta$) (Line 11). This aggregation step ensures that the insights gained from local data across all devices contribute to improving the global model. The update is performed using a learning rate ($\eta$) to control the step size in the gradient descent process.
3. **End of iteration:** The algorithm repeats this process for a fixed number of rounds or until convergence criteria are met (Lines 4–12). Each round allows the global model to learn from diverse data sources without compromising individual data privacy.

This approach promotes privacy preservation, communication efficiency, and collaborative learning in ML tasks [29].

In FL, various algorithms were proposed to address different challenges and optimize the performance of the model on decentralized devices. Some commonly used FL algorithms include FedAvg, FedAvgM, FedIR, FedNova, FedCurv, MOON, and SCAFFOLD. Each algorithm has its unique characteristics and advantages, such as privacy preservation, communication efficiency, and robustness to heterogeneity. FedAvg, which is introduced in Algorithm 1 and we use in this study, is a widely adopted algorithm that averages local model updates to form a global model while mitigating privacy risks. FedAvgM extends FedAvg by incorporating momentum to accelerate convergence [30]. FedIR focuses on addressing imbalanced data distribution among clients by adjusting the learning rate based on the inverse of the local gradient norm [31]. FedNova introduces adaptive learning rate scaling to enhance convergence in non-convex optimization problems [32]. FedCurv uses local curvature information to determine adaptive step size, improving optimization efficiency [33]. MOON and SCAFFOLD are more recent advances that aim to enhance communication efficiency and robustness to adversarial attacks, respectively [34]. It is important to note that the choice of the FL algorithm may significantly impact the performance and convergence behavior of federated learning systems, making algorithm selection a critical consideration in FL research and deployment. Further comparative studies are warranted on different FL algorithms to elucidate their relative strengths and limitations in various application scenarios.

**Algorithm 1** Federated learning.

```
1:  **Input**: Global model $\theta$, data subsets $D_1, D_2, \ldots, D_n$
2:  **Output**: Updated global model $\theta$
3:  **Initialization**: Initialize $\theta$
4:  for $t = 1$ to $T$ do
5:    for $i = 1$ to $n$ do
6:      Broadcast $\theta$ to device $i$
7:      Train local model $\theta_i$ on $D_i$
8:      Compute local update $\Delta\theta_i = \nabla L(\theta_i, D_i)$
9:      Send $\Delta\theta_i$ to central server
10:   end for
11:   Aggregate updates: $\theta \leftarrow \theta - \eta \sum_{i=1}^{n} \Delta\theta_i$
12: end for
```

To ensure the successful implementation of FL, it is crucial to thoroughly examine several factors. These factors encompass the choice of learning devices, differences in performance levels between users, handling diverse training data, possible algorithms for combining local models, deciding on the aggregation approach on the server, and managing resource distribution [35].

When choosing devices, it is essential to assess factors like the quality and amount of local data, the efficiency of the connection to the central server, and the computational capacity of the client. In a diverse setting with devices of different onboard capabilities, it is crucial to consider the variations in computing power, as clients with limited capabilities could potentially hinder the overall learning process [36].

Two commonly accepted FL methodologies for DL are synchronous FL and asynchronous FL [37]. Synchronous FL involves all devices engaging in training local models concurrently, presenting difficulties in heterogeneous environments with diverse client capabilities. In contrast, asynchronous FL allows devices to train models at their individual speeds and transmit parameters to the server for aggregation, making it better suited for varying device capabilities.

In Section 4, an investigation is conducted into the effectiveness of various DL methods, with a specific focus on the application of asynchronous FL in compensating heterogeneous clients. This aims to overcome issues stemming from the varying performance capabilities of Raspberry Pis and virtual machines, with the goal of achieving equilibrium.

The variability in data between FL devices can have a notable impact on the effectiveness of FL [38]. In [39], the concept of federated continual learning was introduced to enhance performance with non-IID data, but the issue of scalability was not addressed. Convergence challenges may arise during the fusion of models created locally, influencing the selection of an appropriate aggregation method. Conventional methods such as federated average (FedAvg) may demonstrate restricted efficacy, prompting the exploration of weighted means or device selection strategies guided by model performance.

These scenarios emphasize the necessity of establishing a suitable FL framework in diverse environments that are tailored to the features and conditions of the FL device. Employing a uniform FL approach may lead to diminished performance, underscoring the significance of evaluating the FL model's performance across different situations and opting for a suitable FL model [40].

### 2.2. Federated Transfer Learning (FTL)

Recent developments in the field of DL have elevated TL to a central position as an essential method for enhancing FL capabilities. The adoption of TL in FL is driven by the practical obstacles that arise when users confront resource constraints, making it impractical to perform extensive FL training on these devices. Integrating TL seeks to improve deep neural networks by leveraging existing knowledge, providing a tactical resolution to resource limitations experienced by users [41].

FTL is an extension of FL that incorporates the advantages of TL within a decentralized learning structure. In FTL, every client leverages their own data to develop a model, benefiting from the knowledge of a pre-existing global model instead of commencing the learning procedure anew. This method effectively reduces the need for computationally intensive training on separate devices, offering a valuable solution, especially in resource-constrained environments [42].

In the FTL algorithm depicted in Algorithm 2, the process begins with each client loading a pre-trained global model ($\theta$) (Line 5). This pre-trained model contains knowledge gained from previous tasks or domains. Subsequently, each client fine-tunes its local model ($\theta_i$) on its respective local data subset ($D_i$) using insights from the pre-trained global model (Line 6). The local model is then used to compute a local update ($\Delta\theta_i$) based on the local loss function ($L(\theta_i, D_i)$) (Line 7), which is sent to a central server for aggregation (Line 8). The central server aggregates these updates (Line 10) and updates the global model accordingly. This iterative process repeats for a predetermined number of rounds ($T$), enabling collaborative learning between decentralized networks while leveraging the benefits of TL.

**Algorithm 2** Federated transfer learning.

```
1:  **Input**: Pre-trained global model $\theta$, data subsets $D_1, D_2, ..., D_n$
2:  **Output**: Updated global model $\theta$
3:  **for** $t = 1$ to $T$ **do**
4:    **for** $i = 1$ to $n$ **do**
5:      Load pre-trained global model $\theta$
6:      Fine-tune local model $\theta_i$ on $D_i$
7:      Compute local update $\Delta\theta_i = \nabla L(\theta_i, D_i)$
8:      Send $\Delta\theta_i$ to central server
9:    **end for**
10:   Aggregate updates: $\theta \leftarrow \theta - \eta \sum_{i=1}^{n} \Delta\theta_i$
11: **end for**
```

In the context of FTL, the distinction lies in the initialization of the global model. Unlike FL, where the global model is typically initialized from scratch, in FTL, the global model is initialized with parameters pre-trained on a source domain dataset. This pre-trained global model serves as a starting point for learning in the federated setting. During the FTL process, the global model is collaboratively fine-tuned across the decentralized devices using the available local data from the target domain. Using the knowledge encoded in the pre-trained global model, FTL aims to accelerate convergence and enhance performance, especially in scenarios where data in the target domain are limited or imbalanced.

In our paper, Algorithms 1 and 2 illustrate the procedural flow of FL and FTL, respectively. Algorithm 1 outlines the steps involved in standard FL, where the global model is initialized without any prior knowledge of external datasets. Conversely, Algorithm 2 delineates the modified procedure for FTL, where the global model is pre-initialized with parameters pre-trained on a source domain dataset before being fine-tuned in the federated learning setting. In summary, while FL focuses on collaborative model training across decentralized data sources, FTL introduces the concept of transfer learning by leveraging pre-trained models to enhance learning performance in federated settings. The distinction between "Global model" and "Pre-trained global model" underscores the initialization strategy employed in FL and FTL, respectively, reflecting their unique approaches to federated learning.

FTL presents several benefits [43]. To begin with, it overcomes the challenges related to limited resources by allowing clients to utilize a common knowledge base, removing the need for complex model training on individual devices. This speeds up the learning process and enhances the overall effectiveness of the FL framework [44]. Secondly, FTL enhances the model's ability to generalize. By transferring knowledge from a pre-trained DNN to users, FTL establishes a foundational set of generalized features that can be adjusted based on each client's specific data characteristics. This flexibility is crucial for building strong and efficient ML models [45]. Additionally, FTL aids in better model convergence, particularly in situations involving diverse devices [46]. The shared knowledge promotes a more coherent learning process, ensuring that models on different devices converge more effectively. This is especially important in environments where there is significant variation in device capabilities [3].

In summary, the integration of TL into FL to create FTL presents a hopeful opportunity to address resource constraints and enhance the effectiveness, adaptability, and convergence of ML models in decentralized settings. The seamless merging of FL and TL concepts presents a holistic approach for DL in situations marked by varied and resource-limited users [47].

## 3. Implementation of the System

The main aim of this research is to develop, evaluate, and compare two detailed and systematically organized DL frameworks, known as FL and FTL, for training ML algorithms using a federated approach involving various distributed clients. Here, we outline the crucial steps taken in implementing both FL and FTL approaches.

The envisioned DL system utilizes a client–server setup, with a Windows PC acting as the server and a variety of Raspberry Pi and Odroid devices functioning as distributed users. Additionally, the integration of virtual clients enhances the distributed framework by introducing diversity. Incorporating both hardware and software clients into the federated framework offers significant benefits, especially in light of the importance of network programmability and virtualization in the realm of 5G/6G networks. Communication between devices is facilitated through the local network, with all hardware nodes connected via Wi-Fi.

The primary goal is to evaluate the effectiveness of these deep learning frameworks in settings with limited and varied resources. A standard task of image classification is conducted to develop a proficient DNN using the designated DL frameworks. The CIFAR10 dataset is used as a basis for training the model. It is important to highlight the intentional creation of data diversity among FL clients by dividing the original dataset into different subsets. Although our study focuses on a specific ML task using predefined datasets, we recognize the potential of these frameworks to address general ML challenges. The implementation of FL and FTL frameworks is carried out using the Python programming language, with the PyTorch library for training the DNN model and the Flower library, designed for federated machine learning, to facilitate efficient automation of client–server interactions [48].

Later on, we elaborate on the complex setups utilized in both the client and server aspects of the examined distributed system. Figure 1 illustrates the essential elements of the DL framework under review, including a collection of Raspberry Pi and Odroid devices, virtual machines, and a centralized server responsible for consolidating parameters from the user community.

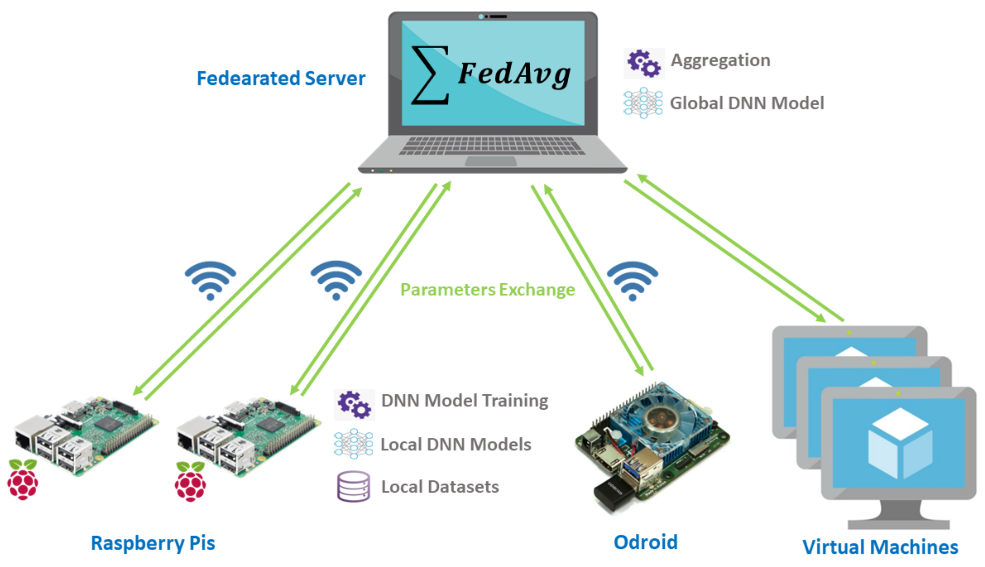

**Figure 1.** Considered distributed framework with heterogeneous clients.

### *3.1. Server Configurations and Functionalities*

In this section, we describe the fundamental procedures for setting up the server aspect of the FTL system and provide information on the software employed. The FTL server runs on a powerful Windows machine, namely an Asus K556U equipped with an Intel Core i7-6500U processor and 12 GB DDR4–2133 MHz RAM. Furthermore, it utilizes Intel Wireless-AC 9560 for Wi-Fi connectivity, which supports 802.11 a/b/g/n/ac standards.

The project opted for Anaconda as the development platform as it provides sophisticated package and library control for Python applications. The project utilized the standard Anaconda setup with Python 3.9.13 alongside PyTorch 2.0.0 and Flower 1.3, which were acquired and set up through the built-in terminal. The project architecture is based on three Python scripts: one for server operations, another for client tasks, and a third for specifying neural network training procedures. Importantly, the system functions solely within the internal network, exchanging data among devices using the procedures outlined in the scripts.

The FL server plays various essential roles in facilitating FTL. Firstly, it initiates contact with a specified group of clients via the local network. Next, it sets up the global DNN model by incorporating pre-trained models tailored to particular situations. Afterward, it distributes the network parameters to all clients, ensuring a consistent foundation for federated training.

During the federated training procedure, participants train their individual models locally for a set number of iterations before sending their model parameters to the central server. The server combines these parameters to form a unified neural network using the FedAvg algorithm. FedAvg is a privacy-conscious FL algorithm that trains a shared model among various participants while protecting the privacy of their data. The process and mathematical representation of FedAvg can be found in Algorithms 1 and 2.

The performance of the combined model is evaluated based on accuracy and losses by utilizing a test dataset available on the server. The outcomes are saved in a CSV file. Subsequently, the improved combined model is sent back to the clients to start the next round of federated learning. This repetitive procedure persists until reaching the designated number of iterations, with each component playing a vital part in enabling the federated learning process.

### *3.2. Client Configurations and Functionalities*

In this section, we provide more details on the variety of client devices used in our FTL framework, including Raspberry Pi devices, Odroid-N2L, and virtual clients created as virtual machines on a PC. Having a variety of client setups expands the range of capabilities and resources available in our IoT environment.

### 3.2.1. Raspberry Pis

Our client devices include Raspberry Pi 3B+ units, which were chosen for their cost-effectiveness and high customizability. The Raspberry Pi 3B+ features a Broadcom BCM2837B0 processor, a Cortex A53 (ARMv8) 64-bit SoC running at 1.4 GHz, 1GB LPDDR2 SDRAM, and 2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac wireless LAN. Additional specifications include Bluetooth 4.2, BLE, Gigabit Ethernet over USB 2.0, a 40-pin GPIO header, full-size HDMI, USB ports, DSI, a Micro SD slot, and a 5V/2.5A DC power supply.

To support our experiments, we installed the 64-bit Raspberry Pi OS, which is crucial for PyTorch compatibility. The configuration involved an SSH setup for the virtual network computing (VNC) service, which allows remote desktop access. This facilitated real-time monitoring of simulations from a connected Windows computer, offering seamless interaction with the Raspberry Pi desktop.

The client-side code, executed in Python 3.9.2, utilized the PyTorch and Flower libraries. Clients ran the client.py script, inheriting contents from cifar.py for the generation, training, and evaluation of the neural network. This configuration allowed efficient participation in the FTL process, optimizing training time and computational costs.

Here, we also take advantage of swap memory, also known as swap space, which is a section of a computer’s hard disk or SSD that the operating system (OS) uses to store inactive data from random access memory (RAM). It involves moving data between the RAM and the dedicated space on the hard disk. When the RAM of a device like a Raspberry Pi is not enough to handle all the running processes, the OS can move some of the data to the swap space. This allows larger and multiple processes to run concurrently. The kernel may also decide to move data that has not been touched for a while to the swap space, freeing up RAM for other applications or cache. This makes better use of the available RAM and can improve system performance. However, using swap has its disadvantages; accessing data from the hard disk is significantly slower than accessing it from RAM. This can lead to slower response times for the system and applications if processes are too aggressively moved out of memory. We have made 3 GB of swap memory for the Raspberry Pis.

### 3.2.2. Odroid

We introduce Odroid-N2L devices, which feature a quad-core Cortex-A73 (2.2 GHz) and dual-core Cortex-A53 (2 GHz) Amlogic S922X processor with ARMv8-A architecture. Equipped with 4GB of LPDDR4 RAM, these devices have enhanced computational capabilities compared to Raspberry Pi devices. Their inclusion adds diversity to our client pool, accommodating various resource capacities within the IoT ecosystem.

### 3.2.3. Virtual Clients

In addition to physical devices, we include as many as two virtual clients created as virtual machines on a personal computer. Although all virtual clients were set up on a single PC for testing purposes, the system can be expanded to multiple-PC setups to enhance client variety. These virtual clients, utilizing the same scripts as the physical clients, provide additional computational capabilities [49]. Each of the virtual clients was configured to utilize one of the CPU cores of the hosting hardware, which was an ASUS K556U Core i7 2.7 GHz with 12 GB RAM memory. The virtual clients were allocated access to all system resources, ensuring consistent performance and enabling seamless integration with the federated learning framework. The specifications of the hardware used in this experiment are summarized in Table 1, which compares the capabilities of all different client types, including CPU speed, RAM size, storage capabilities, and other relevant specifications.

In order to avoid overheating in Raspberry Pi devices, a cooling fan was utilized throughout all experiments to maintain peak performance. The efficiency of this cooling method is further investigated in Section 4.

**Table 1.** Hardware specifications.

| HW | RAM | SWAP | CPU | No. of Cores | Memory |
|---|---|---|---|---|---|
| PI | 1 GB LPDDR2 SDRAM | 4 GB | Cortex-A53 (1.4 GHz) | 4 | 16 GB Micro SD |
| Odroid | 4 GB LPDDR4 RAM | 0 | Quad-core Cortex-A73 (2.2 GHz), Dual-core Cortex-A53 (2 GHz) | 6 | 32 GB eMMC |
| PC | 12 GB DDR4 SDRAM | 0 | Core i7-6500U (3.1 GHz) | 2 | 1 TB Hard Drive |

In our FTL framework, the client nodes perform a series of functions. Communication channels are established with the server, global model parameters are received at the beginning of each round, and local training is performed on client devices. These trained models are evaluated locally, and the metrics are transmitted back to the server. The entire process is detailed in Figure 2, providing insight into the experimental setup that uses Odroid and Raspberry Pi nodes, virtual machines, and an FL server installed on a Windows PC.

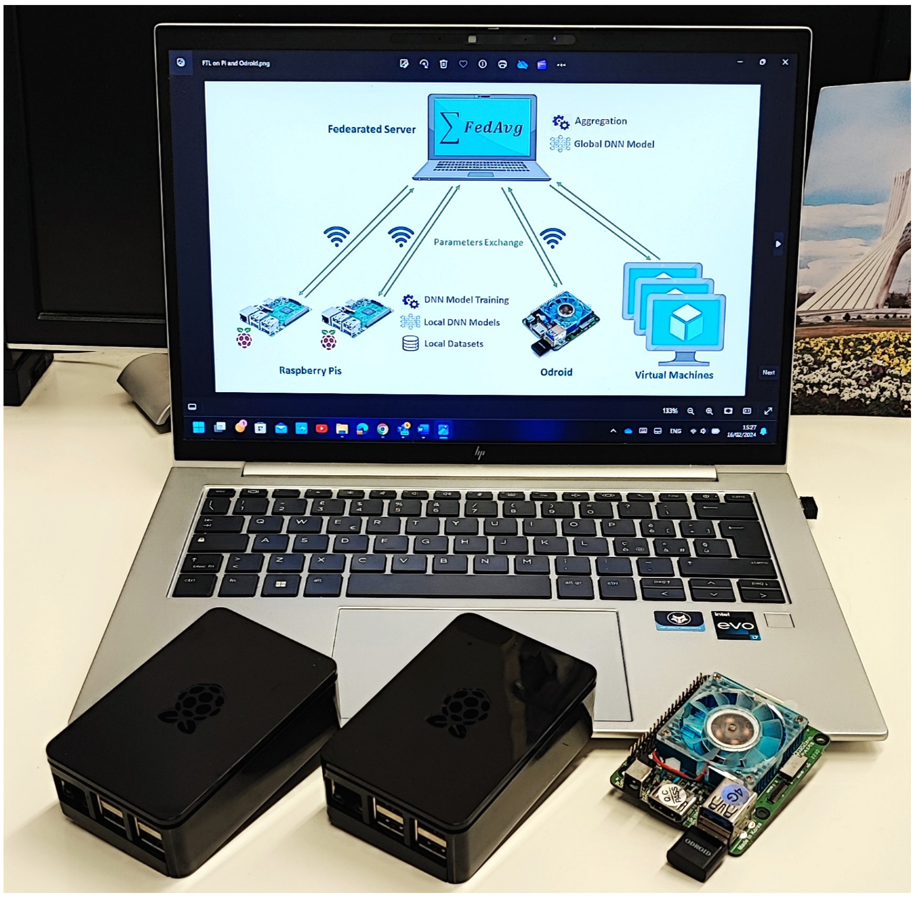

**Figure 2.** Experimental setup used during the first DL implementations.

## 4. Experimental Results and Performance Evaluations

In this section, the performance analysis of the proposed FTL framework is performed using the Python simulation environment. The evaluation focuses on image classification, using the CIFAR10 dataset, which comprises 60,000 colorful pictures with a resolution of 32 × 32 pixels, classified into 10 classes, each containing 6000 images. The dataset is partitioned into 50,000 training samples and 10,000 verification samples. A random arrangement of images is ensured while maintaining a uniform distribution of classes. We experiment in two different settings, one with five heterogeneous clients and the other with three heterogeneous clients.

### *4.1. Experiment 1: DL with Five Heterogeneous Clients*

The first experimental setup involves one Odroid, two Raspberry Pi devices, and two virtual clients, as shown in Figure 2. Each virtual client is assigned an equitable share of CPU resources on the host machine, ensuring a fair and consistent experimental environment. This resource allocation strategy guarantees a balanced and representative simulation, devoid of the influence of uneven resource distribution among virtual clients, providing a reliable basis for experimental findings. The initial distribution of the CIFAR10 training set among clients is followed by 30 federated iterations, comprising three local training epochs. The test data, consisting of 10,000 CIFAR10 samples, are used throughout the simulation.

The accuracy of the model is measured using a metric that calculates the percentage of correct predictions out of all predictions. The accuracy values presented are normalized between 0 and 1. To accelerate the training process, the number of training iterations and the overall size of the data are limited, setting an upper bound on DNN performance, specifically ResNet in our case. The experimental results indicate that with FTL, the DNN achieves an accuracy of up to 83%, compared to approximately 60% with FL. Improved performance can be achieved with additional resources such as data samples, devices, and training iterations.

Figure 3 illustrates the performance of the FTL and FL frameworks in basic settings, where users possess varying amounts of uncorrelated data and onboard capabilities. The simulation comprises 10% of randomly selected data for Raspberry Pi 1, 15% for Raspberry Pi 2, 25% for Odroid, 20% for Virtual Machine 1, and 30% for Virtual Machine 2, totaling five clients. Both the FTL and FL models undergo training for 30 global federated rounds. The results reveal that the FTL framework surpasses FL in terms of performance and accuracy, achieving higher accuracy in a shorter time, indicative of its faster DNN training capabilities.

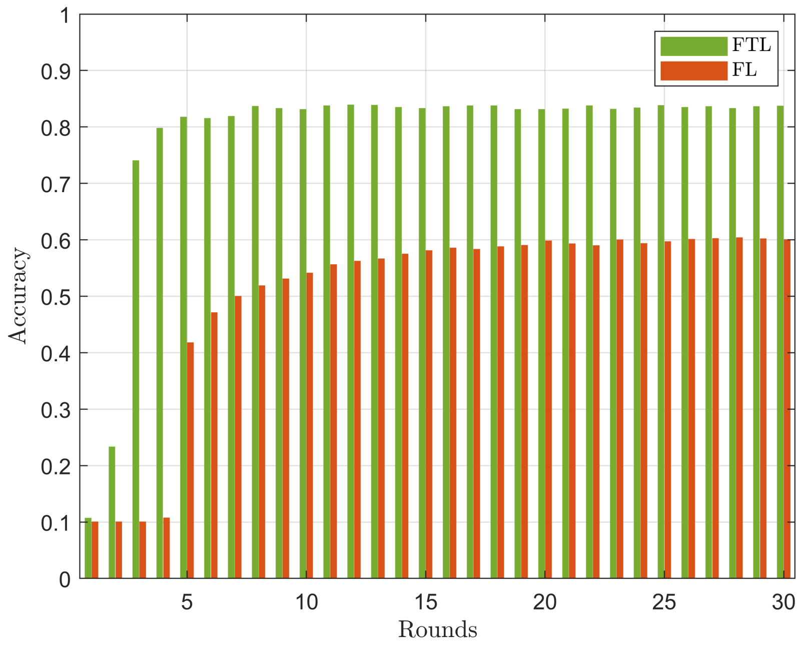

**Figure 3.** Accuracy of FTL for 5 clients, 2 Raspberry Pis, 1 Odroid, and 2 virtual machines, compared to ordinary FL vs. rounds.

To quantify this superior performance in terms of latency, a settling time of 2% is considered, representing the time it takes for accuracy to reach or fluctuate within 2% of the steady-state accuracy after 30 rounds. Figure 4 illustrates a settling time of approximately 9720 s for FTL compared to 27,300 s for FL, indicating a three-fold faster convergence for FTL. Moreover, FTL exhibits potential advantages in terms of reduced communication overhead and enhanced data privacy, which are crucial aspects in emerging wireless scenarios, particularly in 6G.

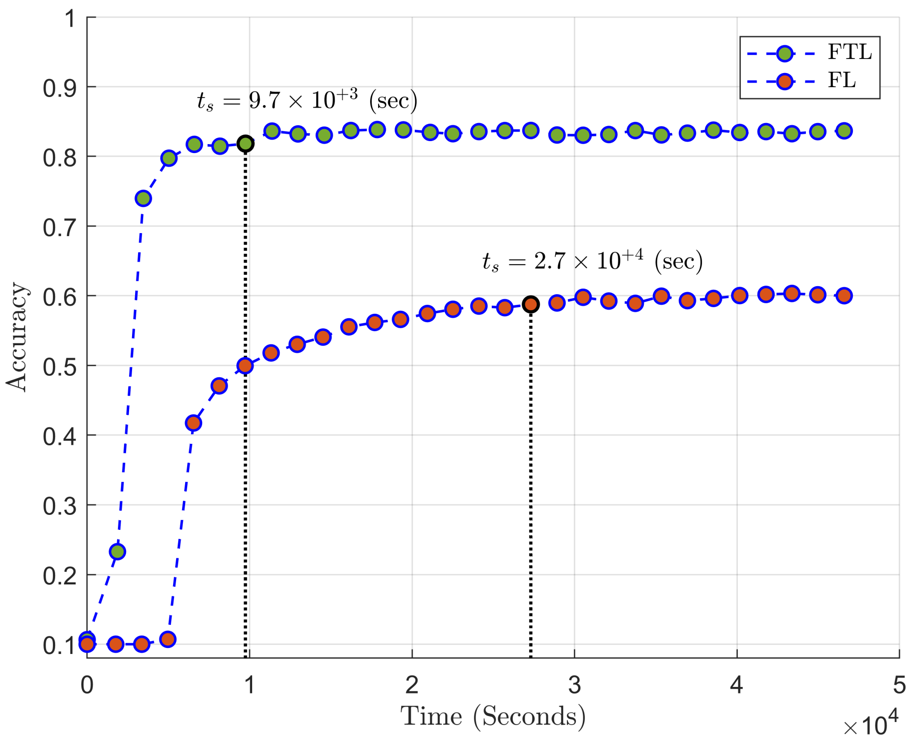

**Figure 4.** Accuracy of FTL for 5 clients, 2 Raspberry Pis, 1 Odroid, and 2 virtual machines, compared to ordinary FL vs. time.

In the case of FTL, the global model is pre-trained using a large dataset from a related domain, such as ImageNet, before being fine-tuned on the federated dataset. This pre-training step initializes the global model with knowledge learned from the source domain, allowing it to capture generic features and patterns that can facilitate faster convergence and improved performance when adapting to the target federated dataset. In our experiment, we employed a ResNet DNN architecture, which is commonly pre-trained on large-scale image datasets like ImageNet due to its effectiveness in feature extraction and representation learning. By initializing the ResNet model with weights learned from ImageNet, the model already possesses a strong foundation of feature representations, enabling it to learn more efficiently and effectively from the federated CIFAR10 dataset during the fine-tuning phase. This approach leverages the transferability of features learned from ImageNet to the task of classifying CIFAR10 images, ultimately leading to enhanced performance and accelerated convergence in the federated learning process.

In our implementation, we used the categorical cross-entropy loss function as a metric to assess the performance of the FTL framework against conventional FL. This loss function measures the difference between the true probability distribution of the class labels and the predicted probability distribution generated by the neural network model. Mathematically, the categorical cross-entropy loss $L(y, \hat{y})$ for a single sample is defined as

$$L(y, \hat{y}) = -\sum_{i} y_i \log(\hat{y}_i)$$

Here, $y_i$ denotes the $i$th element of the true class distribution vector $y$, and $\hat{y}_i$ represents the $i$th element of the predicted probability distribution vector $\hat{y}$.

Next, in Figures 5 and 6, the loss incurred by FTL across five clients—two Raspberry Pis, one Odroid, and two virtual machines—is compared to that of the conventional FL approach over multiple training rounds and versus time. Our analysis reveals a markedly superior performance of FTL in terms of loss minimization. Specifically, an examination of these figures elucidates that the steady-state loss attained by FTL is merely one-third of that encountered with the FL methodology. Consequently, our approach significantly advances towards the objective of minimizing the loss cost function inherent in this multi-class classification problem facilitated by DNNs.

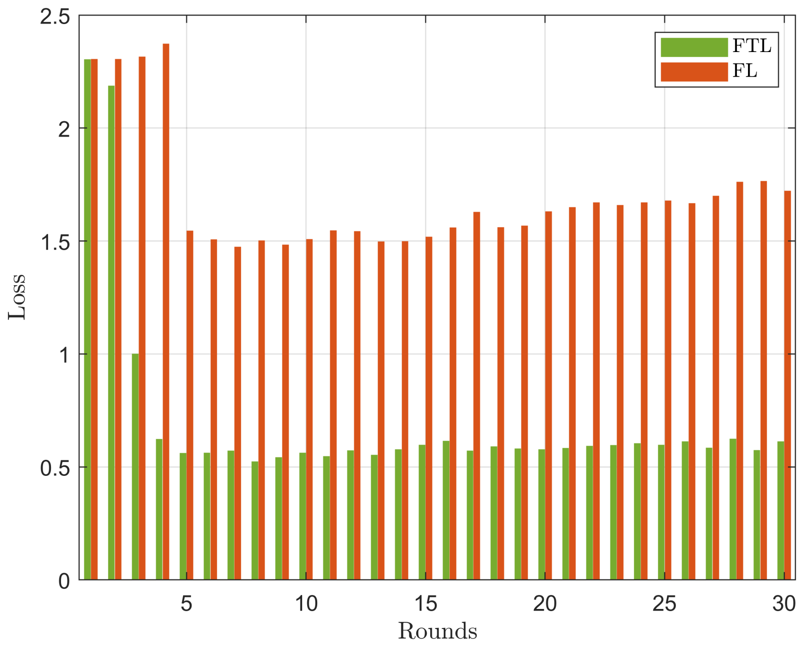

**Figure 5.** Loss of FTL for 5 clients, 2 Raspberry Pis, 1 Odroid, and 2 virtual machines, compared to ordinary FL vs. rounds.

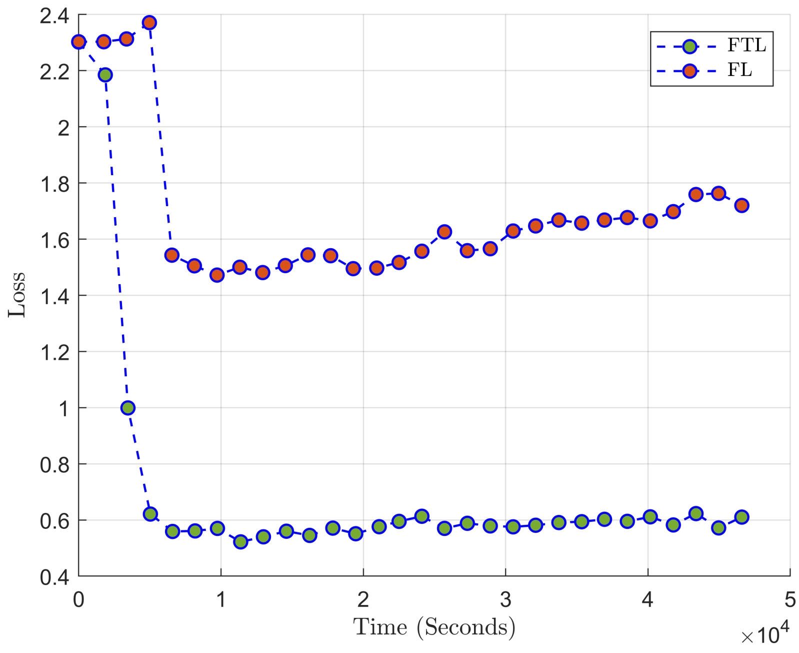

**Figure 6.** Loss of FTL for 5 clients, 2 Raspberry Pis, 1 Odroid, and 2 virtual machines, compared to ordinary FL vs. time.

Subsequently, we utilize *htop*, a command-line utility renowned for its real-time monitoring capabilities of system resources and server processes, to dive deeper into dynamic parameters such as memory usage, average CPU load, and temperature. Using *htop*'s interactive interface, we obtain crucial insights into the operational status of our distributed clients.

Our initial focus is to examine the average load exerted on the processing units of the client devices. The load average, a metric indicative of the computational workload borne by the CPU, is of paramount interest. For example, a load average of 1.0 on a single-core CPU signifies full utilization, while a load average of 2.0 on a dual-core CPU equals 100% CPU usage. Comprising three values—a one-minute average, a five-minute average, and a fifteen-minute average—the load average affords a comprehensive snapshot of CPU utilization trends over varying time intervals. We meticulously measured these load average values across different clients, including Raspberry Pis and Odroid, culminating in the generation of error bar charts, as depicted in Figure 7.

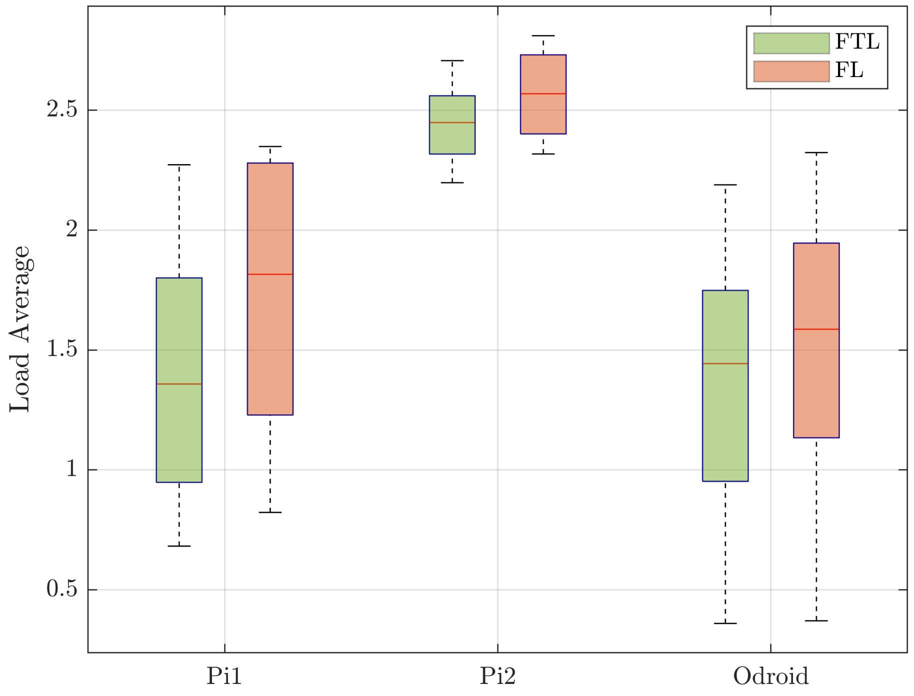

**Figure 7.** Average load on the Raspberry Pis and Odroid when running FTL and FL with 5 clients.

In our experimental setup, we utilized Raspberry Pis with four cores and Odroid with six cores. These hardware configurations significantly influence the interpretation of load average values, as depicted in Figure 7. Load average, as a metric indicative of CPU workload, becomes more comprehensible in light of the processor counts of the client devices. For instance, a load average of 1.0 on a single-core CPU indicates full utilization, while the same load average on a quad-core Raspberry Pi suggests only a quarter of the CPU resources are utilized, and on a six-core Odroid suggests one-sixth of the CPU resource utilization. Therefore, understanding the processor count of each device is crucial for contextualizing the load average values accurately. By meticulously measuring load average values across different clients and considering their respective processor counts, our analysis aims to provide a nuanced understanding of CPU utilization trends and their implications for federated learning performance.

In particular, our observations unveiled lower load average values when executing FTL on clients as opposed to employing FL to train ResNet models. This disparity can be attributed to the utilization of pre-trained networks within the FTL framework, obviating the need for initialization and training from scratch, thereby alleviating the computational burden on the CPU.

A nuanced examination of the load average dynamics across different client devices further elucidated intriguing trends. In particular, the load average exhibited a descending order from Odroid to Raspberry Pi 1 and Raspberry Pi 2. This order can be attributed to the varying processing capabilities inherent in each device. Specifically, Odroid, characterized by superior computational capacity compared to Raspberry Pis, demonstrated the lowest load average due to its prompt completion of training tasks. On the contrary, Raspberry Pi 2, tasked with a larger dataset for ResNet training, endured a prolonged training duration compared to its counterparts, consequently registering a higher load average.

In Figure 7, the load average is defined based on the percentages of CPU core usage, which reflect the system's utilization over a period of time. Since the VMs in our study were implemented on the same CPU, each assigned to a distinct single core, the load average parameter becomes less relevant for VMs as their resource utilization is inherently tied to the CPU core they are allocated to. Therefore, including VMs in the load average plot would not provide meaningful insights into their resource utilization patterns.

Adjacent to our investigation of the dynamic temperature profiles of devices during DL training sessions, as depicted in Figure 8, a discernible correlation emerges between the temperature dynamics and the distribution of workload and computational resources between devices. In particular, the observed trend closely parallels that identified in the measurement of load averages, indicating a coherent relationship between computational workload, training efficiency, and thermal management.

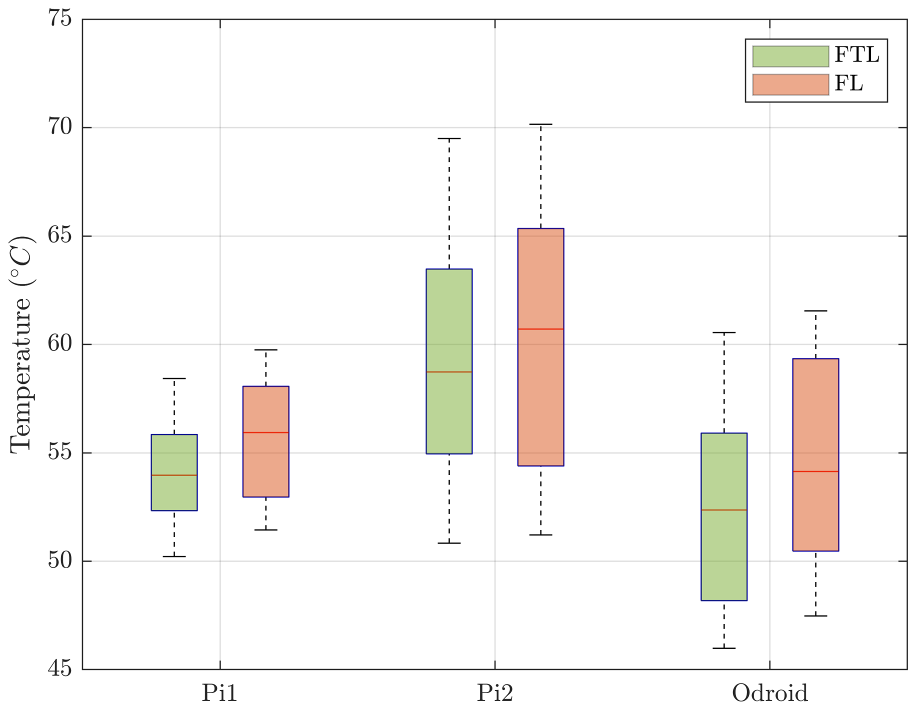

**Figure 8.** Raspberry Pis and Odroid temperature when running FTL compared to FL with 5 clients.

Specifically, devices allocated with lighter training burdens demonstrate faster training completion times, giving them ample opportunity to dissipate heat and maintain cooler operating temperatures. This can also be attributed to the fact that the Odroid device has a built-in cooling mechanism of a fan and a heatsink. Consequently, the incidence of overheating is significantly mitigated, underscoring the importance of workload optimization to promote thermal stability within DL environments.

Furthermore, our analysis reveals a notable distinction in the temperature dynamics between the FTL and FL methodologies in identical client configurations, data distributions, and environmental conditions. Interestingly, devices that execute FTL exhibit lower operating temperatures compared to those that implement FL. This disparity underscores the efficacy of FTL in minimizing thermal stress on client devices, thus enhancing operational reliability and longevity.

Subsequently, we conducted an assessment of memory utilization across various client devices, encompassing Raspberry Pis, Odroid, and virtual machines. The results of this investigation are depicted by error bar graphs, as illustrated in Figure 9. As expected, the disparity in memory usage values between the FL and FTL frameworks is not pronounced. However, FTL exhibits a discernible alleviation of memory burden, attributable to intervals of inactivity following each round and the utilization of pre-trained parameters rather than commencing training anew.

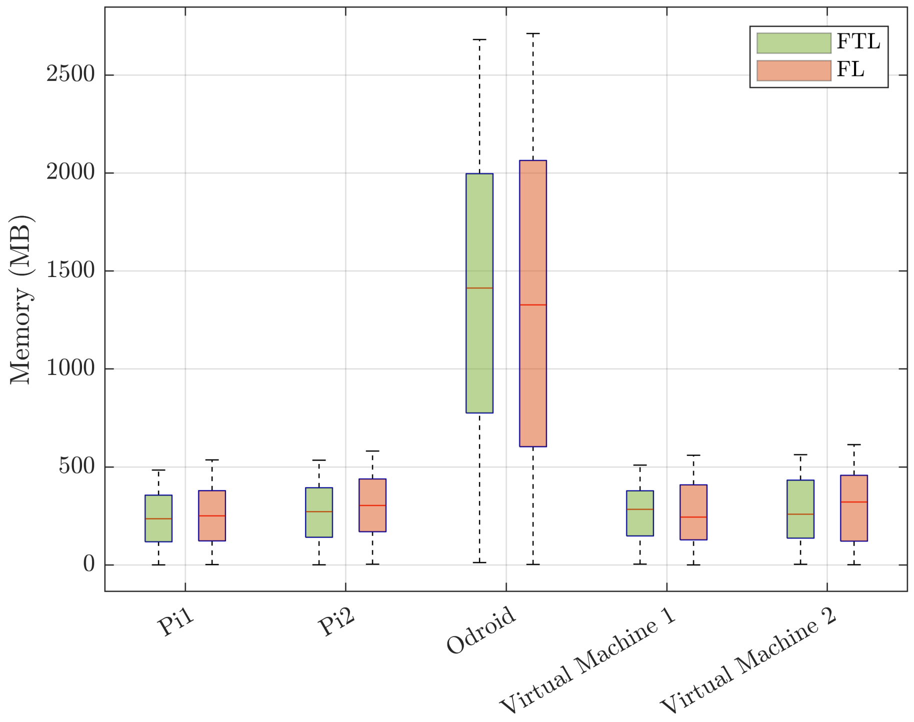

**Figure 9.** Memory usage of FTL compared to ordinary FL for 5 clients.

In particular, each client device maximally utilizes approximately half of its available memory when subjected to full-load training of its respective DNN. Specifically, Odroid, with 25% of the dataset allocated to ResNet training, exceeds the halfway mark in memory usage. On the contrary, Raspberry Pis 1 and 2, with data allocations of 10% and 15%, respectively, consume less than half of their total memory. This observed trend in memory utilization aligns proportionally with the assigned dataset volumes, in which Raspberry Pi 2, endowed with a larger dataset compared to Raspberry Pi 1, exhibits increased memory usage.

Further scrutiny of memory usage patterns reveals a nuanced interplay between computational load and processing efficiency. In particular, Raspberry Pi 1 completes its training tasks before Raspberry Pi 2 in each training round, giving it additional time to recover memory before the next round. Consequently, Raspberry Pi 1 tends to exhibit lower memory consumption relative to Raspberry Pi 2, a phenomenon reflective of the inter-device variability in training durations.

This comprehensive examination elucidates the intricacies of memory management in the context of DL scenarios, shedding light on the nuanced interplay between device specifications, dataset allocations, and training dynamics.

Subsequently, we performed an exhaustive analysis of power consumption across client devices, as illustrated in Figure 10. This assessment was facilitated using a digital multimeter capable of dynamically recording and visualizing variations in power consumption over time. In particular, the maximum power consumption corresponds to periods of full-load training, whereas the minimum power consumption occurs during idle intervals when clients wait for the completion of training by other participants.

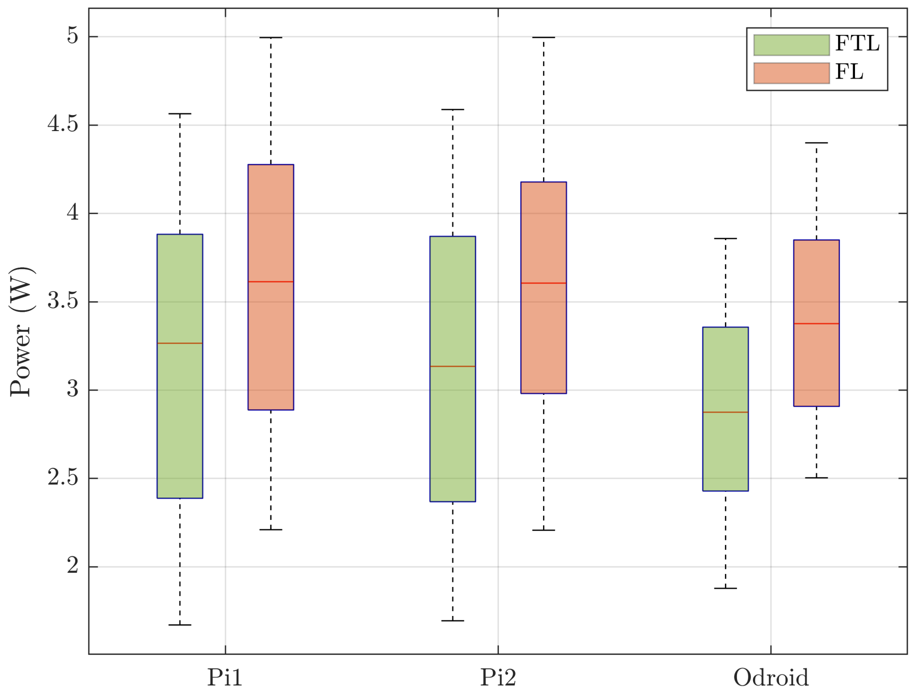

**Figure 10.** Power consumption of Raspberry Pis and Odroid when running FTL and FL with 5 clients.

It is worth mentioning that, in Figures 8 and 10, which depict temperature and power consumption, respectively, the absence of VMs is attributed to the practical limitation of measuring these metrics separately for each VM. In our experimental setup, the VMs, along with the FL server, were implemented on the same CPU, making it challenging to isolate and measure the temperature and power consumption of individual VMs. Consequently, these figures focus solely on the temperature and power consumption of other parts of the system, i.e., Pis and Odroid, without distinguishing between the FL server and VMs.

Finally, we extrapolated these power consumption measurements to derive energy consumption profiles for all devices, as depicted in Figures 11–13. Intriguingly, we observed a direct correlation between training effort and energy consumption, whereby intervals of client idleness exhibited a linear increase in energy consumption at a rate equivalent to the minimum power consumption. Conversely, periods of active training were characterized by a steeper increase in energy consumption, correlating with the maximum power consumption.

Furthermore, our analysis revealed a notable difference in total energy consumption between the FTL and FL methodologies. Due to the inherently reduced training effort associated with FTL compared to FL, the total energy consumption incurred by devices utilizing the FTL approach was significantly lower than that observed with FL. This observation underscores yet another advantage of FTL over FL, which is particularly pertinent in scenarios featuring resource-constrained devices, such as Raspberry Pi, and in mobile or vehicular 6G communication scenarios where energy efficiency is paramount. These findings underscore the importance of energy-aware design considerations in optimizing the performance and sustainability of DL systems.

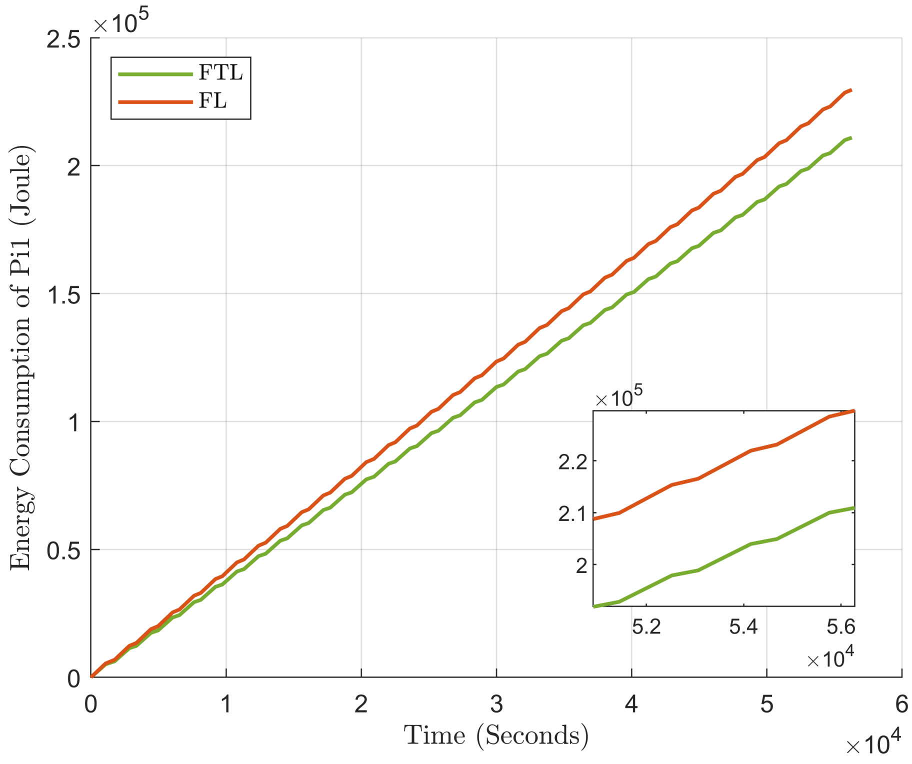

**Figure 11.** Energy consumption of Raspberry Pi 1 when running FTL and FL.

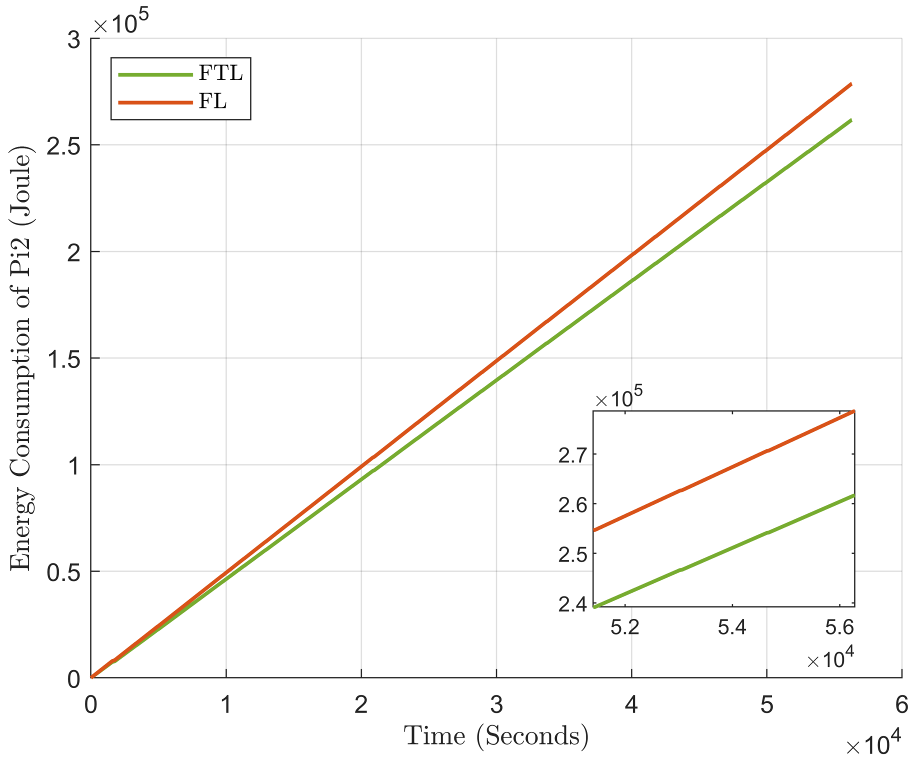

**Figure 12.** Energy consumption of Raspberry Pi 2 when running FTL and FL.

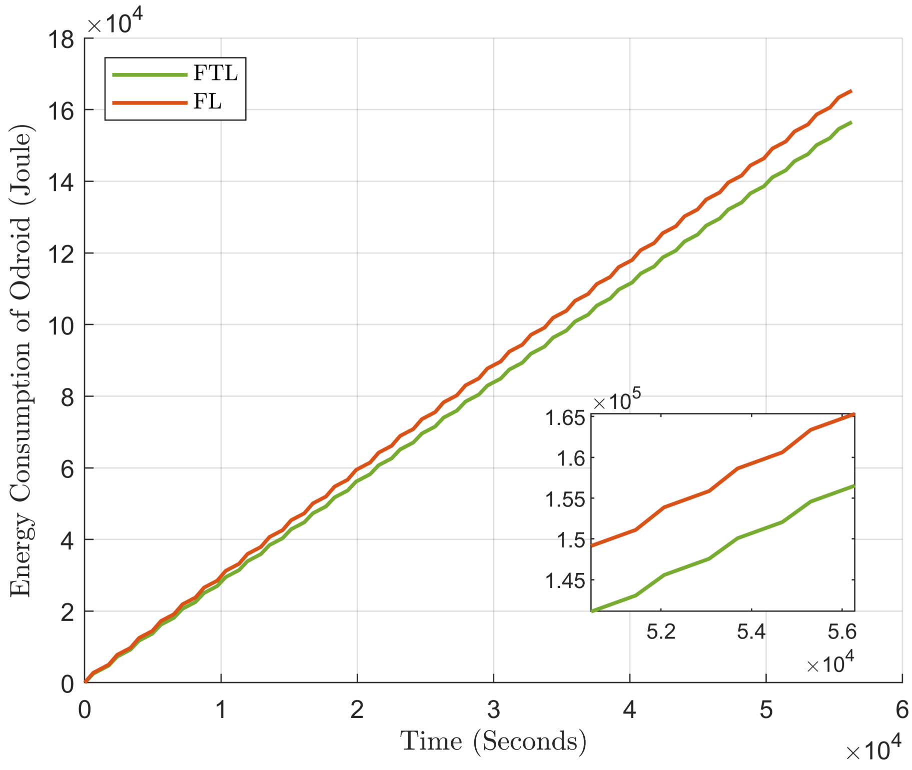

**Figure 13.** Energy consumption of Odroid when running FTL and FL.

### 4.2. Experiment 2: DL with Three Heterogeneous Clients

DL methods are inherently influenced by factors such as the number of participants and the volume of training data within the dataset. Therefore, it is imperative to explore the impact of these parameters on both the FL and FTL frameworks. Through this investigation, we aim to elucidate the relative sensitivity of FL and FTL to variations in these crucial parameters, highlighting the potential advantages conferred by FTL over conventional FL methodologies.

To this end, we conducted an experiment in which Raspberry Pi 1 and Virtual Client 2, along with their corresponding training datasets, were removed, thus reducing the number of participants to three: one Raspberry Pi, one Odroid, and one virtual machine. Despite this reduction in the user base, the distribution of the data remained consistent with the previous experiment, allocating 15% of the data to the Raspberry Pi, 25% to the Odroid, and 20% to the virtual machine. We experiment with this setting because it mirrors real-world scenarios, such as mobile or vehicular 5G/6G networks, where users may lose connection to the federated server or go out of coverage of the server.

The accuracy and training dynamics measured from these DL approaches were analyzed against the number of training rounds and time, as shown in Figures 14 and 15, respectively. A comparative analysis with the previous experiment involving five clients (Figures 2 and 3) revealed interesting information. Despite the reduction in participants, the final accuracy of the FTL approach experienced a slight decrease, from 83% to 81%, while the final accuracy of the FL approach exhibited a more pronounced decrease, from 60% to 55%. This observation underscores the superior robustness of FTL in mitigating the adverse effects of reduced user participation and reduced training data volume, an inherent advantage of FTL over FL.

Furthermore, analysis of the settling time revealed that while the settling time for FTL experienced a modest increase from 9700 to 11,400 s, the FL approach demonstrated a more substantial rise, escalating from 27,000 to 32,100 s. This increased sensitivity of FL to changes in user count and training data volume underscores the inherent resilience of FTL in maintaining stability and performance consistency under varying conditions.

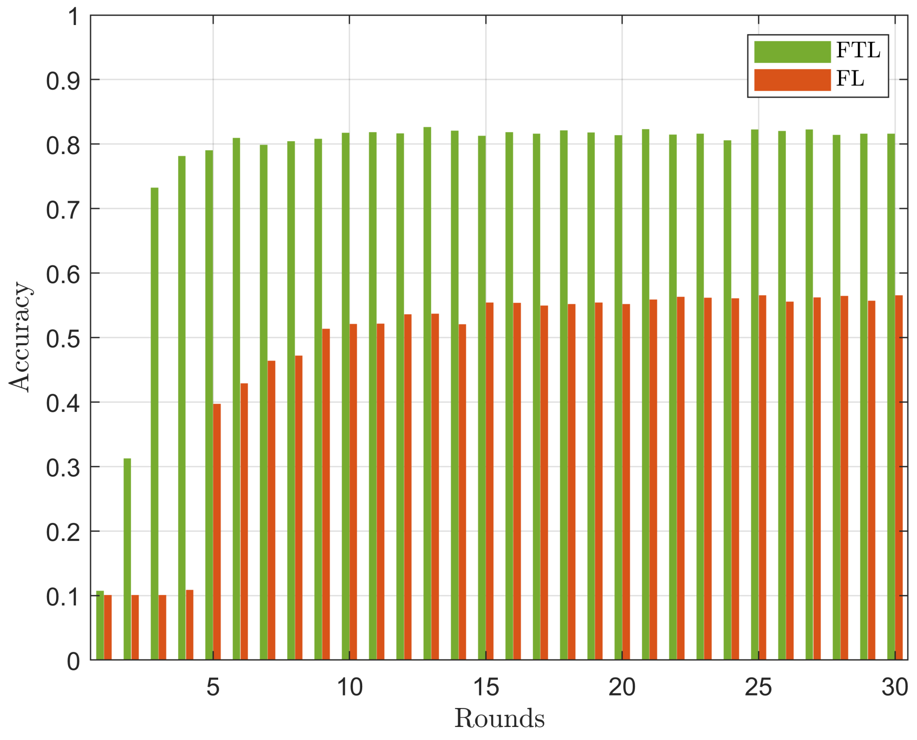

**Figure 14.** Accuracy of FTL for 3 clients, a Raspberry Pi, an Odroid, and a virtual machine, compared to ordinary FL vs. rounds.

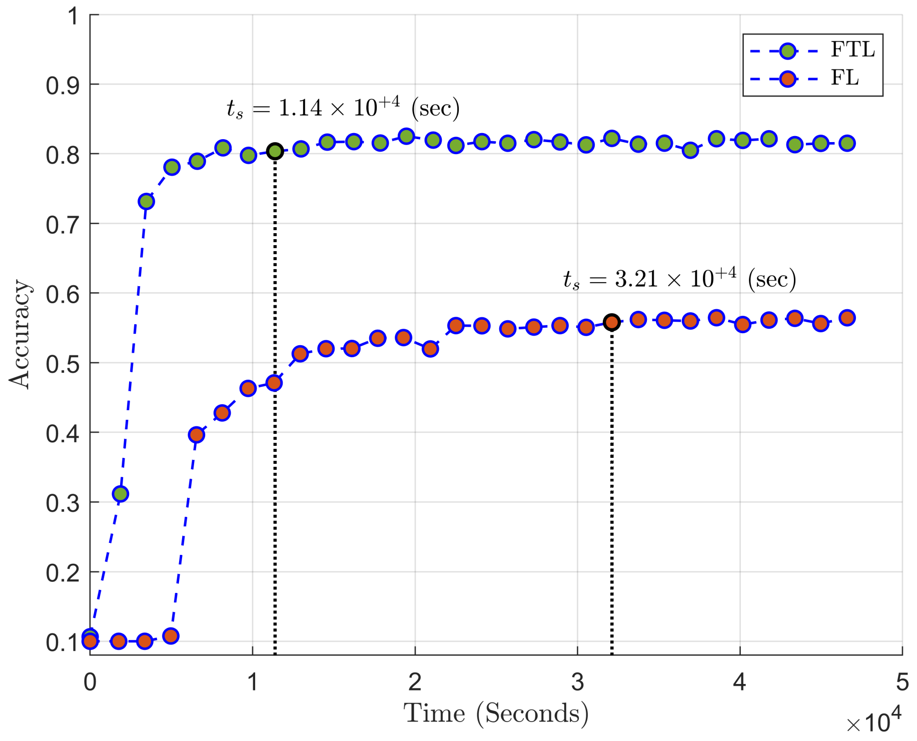

**Figure 15.** Accuracy of FTL for 3 clients, a Raspberry Pi, an Odroid, and a virtual machine, compared to ordinary FL vs. time.

These findings are corroborated by the examination of loss dynamics, in which the categorical cross-entropy loss function is used to compare the performance of FTL and FL across different training rounds and elapsed time, which is demonstrated in Figures 16 and 17. The superior efficacy of the FTL approach in minimizing the loss function is evident, owing to its utilization of prior training knowledge and avoidance of training from scratch, a distinct advantage of FTL over the FL methodology.

In summary, our comprehensive analysis underscores the robustness and stability of FTL in the face of fluctuating participant counts and training data volumes, reaffirming its potential as a reliable and efficient framework for DL applications.

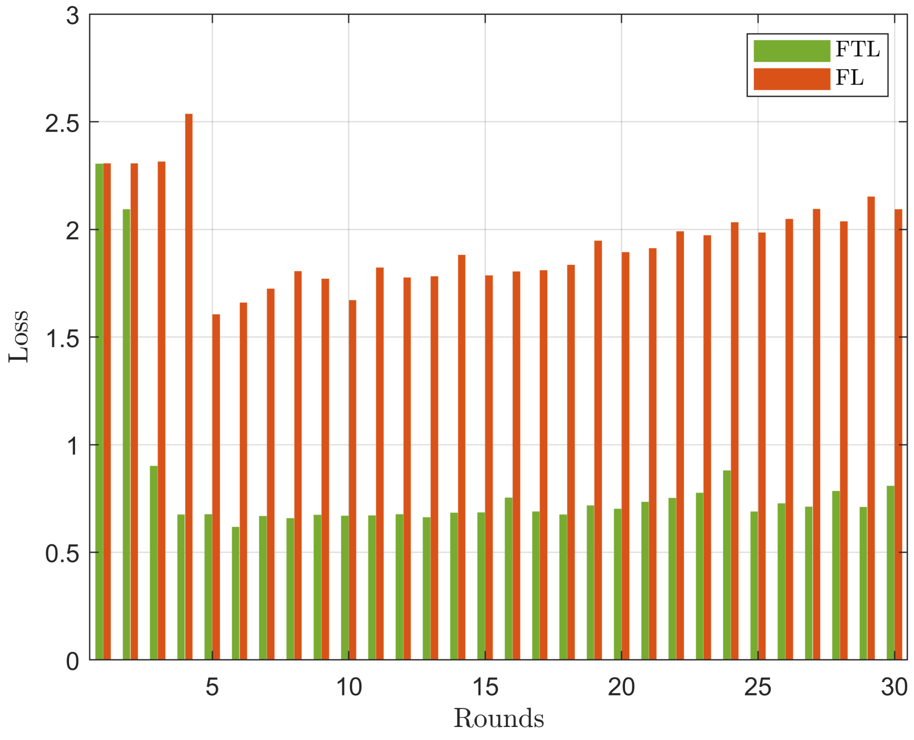

**Figure 16.** Loss of FTL for 3 clients, a Raspberry Pi, an Odroid, and a virtual machine, compared to ordinary FL vs. rounds.

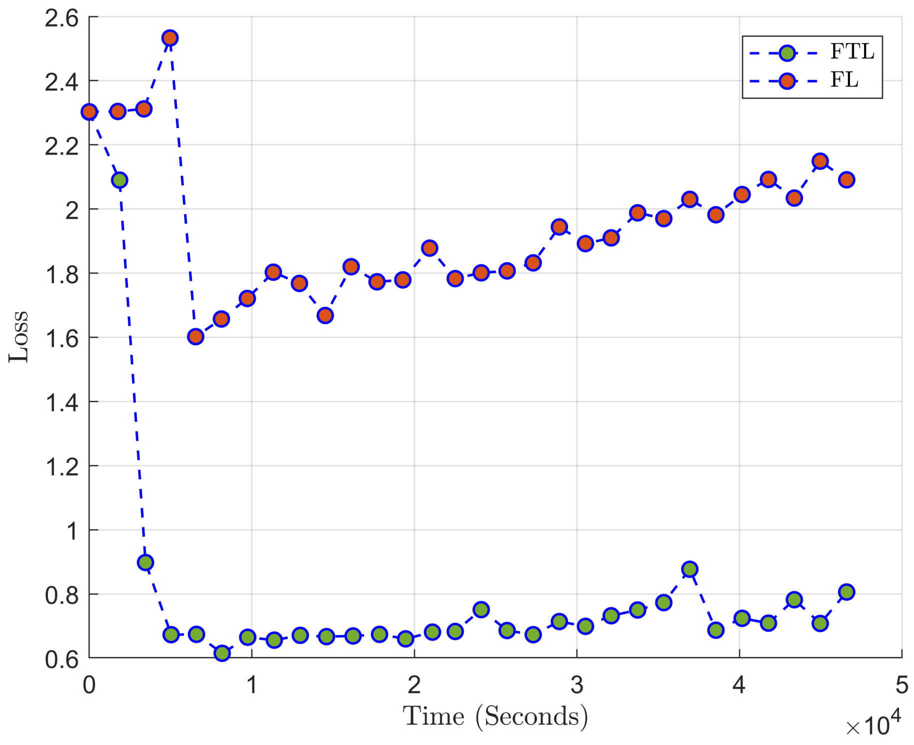

**Figure 17.** Loss of FTL for 3 clients, a Raspberry Pi, an Odroid, and a virtual machine, compared to ordinary FL vs. time.

Subsequently, we conducted a comprehensive assessment of key performance metrics, namely load average, temperature, and memory usage, on various client devices during FTL and FL training sessions. The results of this investigation are presented through error bar charts in Figures 18–20.

Consistent with previous observations, the utilization of FTL manifests in superior performance metrics compared to FL, attributed primarily to the integration of pre-trained networks within the FTL framework. This strategic utilization of pre-existing knowledge mitigates the computational overhead associated with initializing and training models from scratch, consequently enhancing overall efficiency and effectiveness.

Moreover, our analysis revealed notable disparities in load average and temperature dynamics between the Odroid and Raspberry Pi devices. The Odroid, characterized by superior computational resources relative to the Raspberry Pi, exhibited lower averages and load temperatures. This difference can be attributed to the accelerated training completion times facilitated by the Odroid's enhanced processing capabilities, affording it additional idle time for relaxation and cooling.

Furthermore, an examination of the memory usage patterns revealed that the Odroid device consistently utilized more than half of its available memory, while the Raspberry Pi device consumed less than half of its total memory capacity. This observation underscores the efficient utilization of computational resources across devices, with the Odroid leveraging its enhanced capabilities to support intensive computational tasks, while the Raspberry Pi demonstrates a more conservative approach to memory utilization.

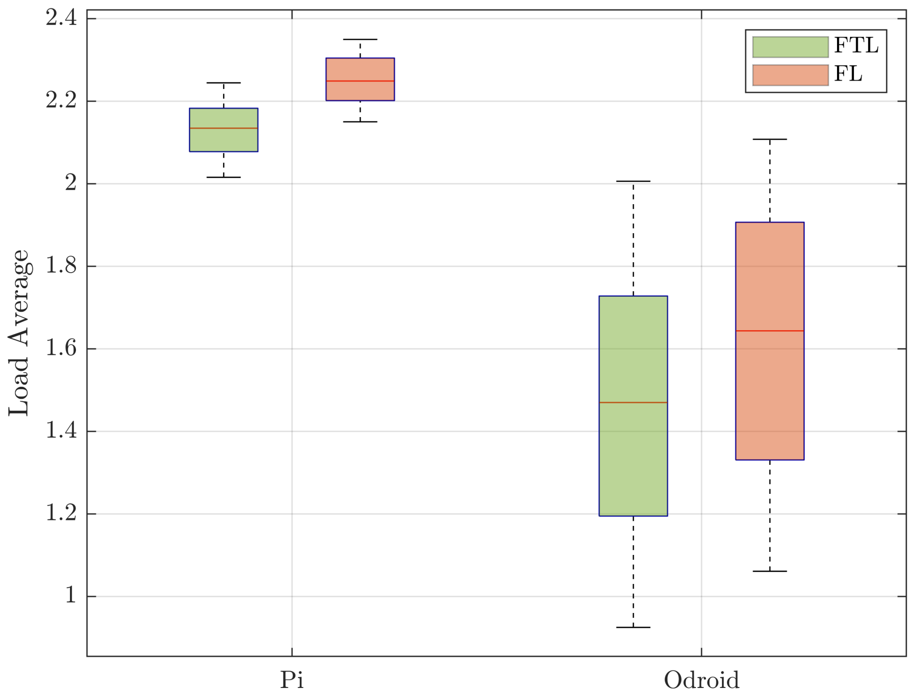

**Figure 18.** Average load on the Raspberry Pi and Odroid when running FTL and FL with 3 clients.

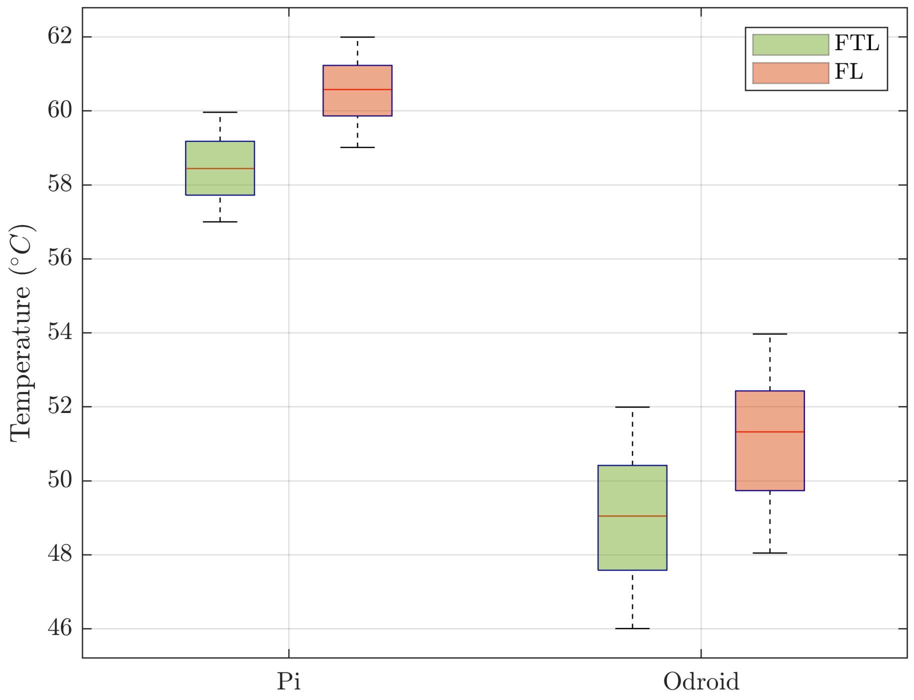

**Figure 19.** Raspberry Pi and Odroid temperature when running FTL compared to FL with 3 clients.

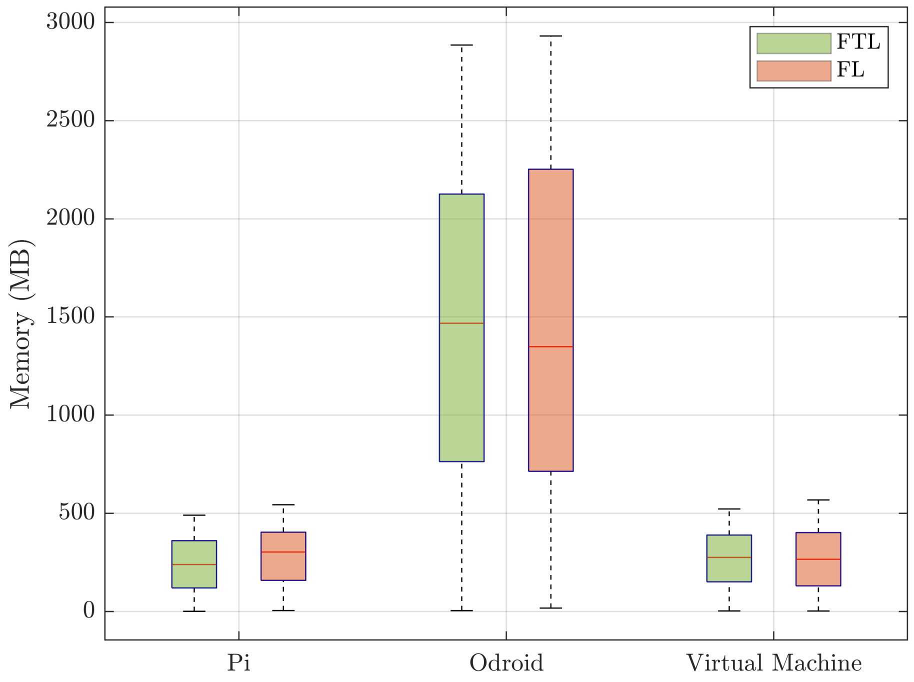

**Figure 20.** Memory usage of FTL compared to ordinary FL for 3 clients.

Finally, we meticulously monitored and graphed the power consumption of client devices, as shown in Figure 21. Using a digital multimeter with dynamic recording capabilities, we captured fluctuations in power use over time. In particular, peak power consumption coincided with full-load training periods, while minimal power draw occurred during idle intervals as clients waited for the completion of training rounds by other participants.

Following this, we depicted the energy consumption profiles for all devices, as shown in Figures 22 and 23. We observe a direct correlation between energy consumption and training activity, with idle intervals characterized by a gradual increase in energy consumption at the minimum power rate. On the contrary, the active training phases exhibited a steep rise in energy consumption, in line with maximum power usage. Furthermore, our analysis revealed that FTL required less training effort compared to FL, resulting in a reduction in overall energy consumption. This underscores FTL's advantages, which are particularly significant in resource-constrained environments such as IoT, mobile, or vehicular 6G communication scenarios, emphasizing its potential to enhance energy efficiency in DL frameworks.

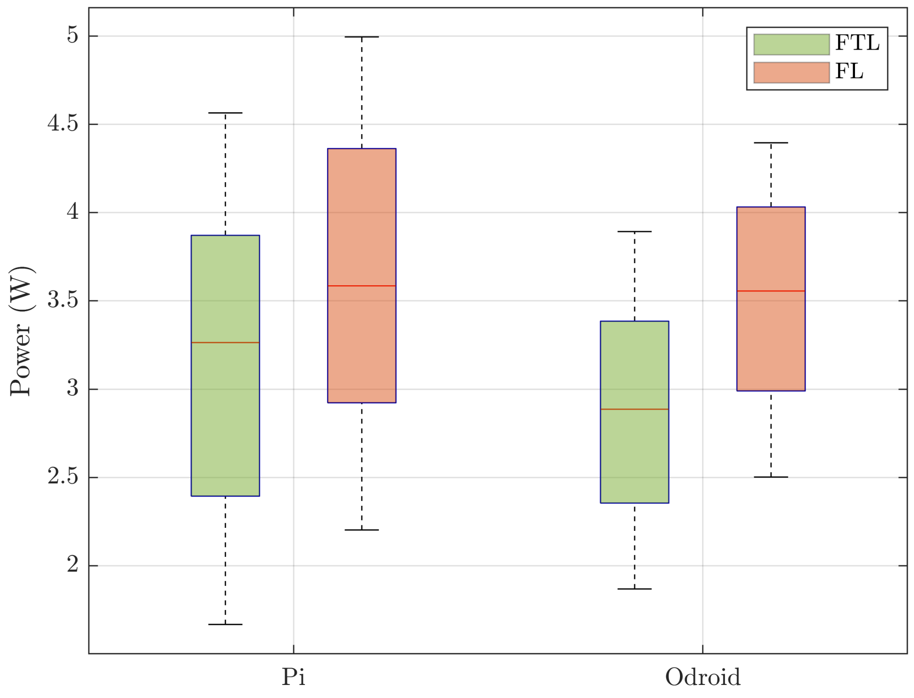

**Figure 21.** Power consumption of Raspberry Pi and Odroid when running FTL and FL with 3 clients.

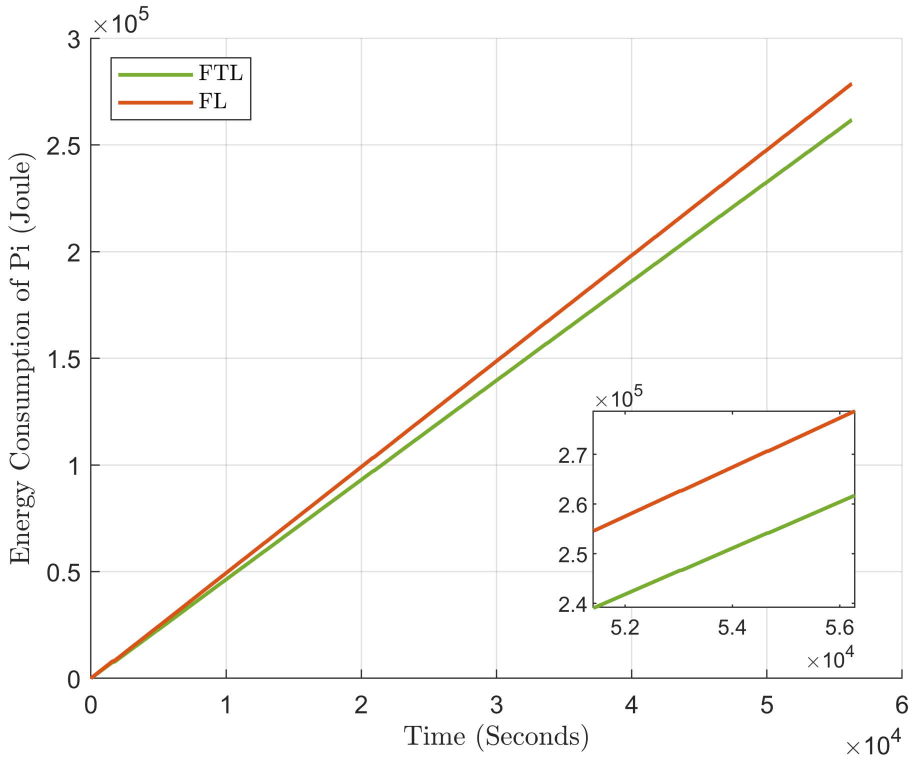

**Figure 22.** Energy consumption of Raspberry Pi when running FTL and FL.

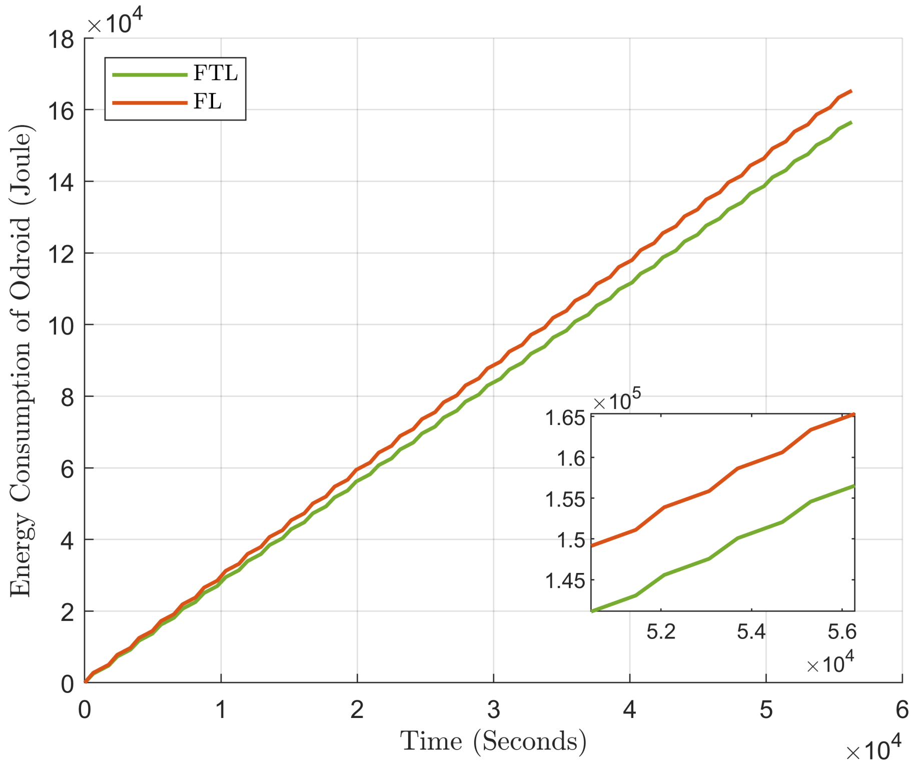

**Figure 23.** Energy consumption of Odroid when running FTL and FL.

## 5. Conclusions

In conclusion, this study conducted a comprehensive exploration of DL frameworks in response to the evolving landscape of 6G technology and the imperative for a fully connected distributed intelligence network for IoT devices. Recognizing the inherent challenges posed by the heterogeneous nature of clients and data, we embarked on an investigation of FTL as an alternative to traditional FL techniques. Through meticulous design, implementation, and evaluation, we elucidated the efficacy and superiority of FTL, particularly tailored to the diverse constraints of IoT platforms.

Our findings underscore the notable performance advantages offered by FTL over FL, particularly characterized by faster training and higher accuracy, facilitated by the incorporation of TL methodologies. Real-world measurements further substantiated these assertions, revealing improved resource efficiency with lower average load, memory usage, temperature, power, and energy consumption in FTL implementations compared to their FL counterparts.

Moreover, our experiments unveiled FTL's resilience in scenarios characterized by sporadic client participation, where users depart from the server's communication coverage, resulting in a reduced number of clients and training data availability. This adaptability underscores FTL's effectiveness in environments with limited data, clients, and resources, offering valuable insights into the convergence of edge computing and DL for 6G IoT applications.

By addressing practical challenges through innovative implementation strategies, such as memory swap functionality and asymmetric data distribution, we have advanced the understanding and applicability of DL frameworks in resource-constrained environments. Looking ahead, future research endeavors could further explore the scalability and optimization of FTL methodologies, fostering the continued evolution of distributed intelligence networks in the era of 6G technology.

**Author Contributions:** Conceptualization, D.T. and D.N.; methodology, D.T., D.N. and M.A.; software, M.A.; validation, D.N. and M.A.; formal analysis, D.N. and M.A.; investigation, D.N. and M.A.; resources, D.T.; data curation, M.A.; writing—original draft preparation, D.N.; writing—review and editing, D.N., D.T. and M.A.; visualization, D.N. and M.A.; supervision, D.T.; project administration, D.T.; funding acquisition, D.T. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work has been partially supported by the ECOSISTER project funded under the National Recovery and Resilience Plan (NRRP), Mission 04 Component 2 Investment 1.5—NextGenerationEU, call for tender No. 3277, dated 30 December 2021, award number 0001052, dated 23 June 2022, and by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001—program "RESTART").

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Data available on request due to restrictions, e.g., privacy or ethical.

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

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| IoT | Internet of Things |
| ML | Machine learning |
| CL | Centralized learning |
| DL | Distributed learning |
| FL | Federated learning |
| TL | Transfer learning |
| DNNs | Deep neural networks |
| NTNs | Non-terrestrial networks |
| RL | Reinforcement learning |
| DIL | Distributed incremental learning |
| FTL | Federated transfer learning |
| MARL | Multi-agent reinforcement learning |
| SL | Split learning |
| FedAvg | Federated average |
| VNC | Virtual network computing |
| OS | Operating system |
| RAM | Random access memory |

## References

1. Morocho-Cayamcela, M.E.; Lee, H.; Lim, W. Machine Learning for 5G/B5G Mobile and Wireless Communications: Potential, Limitations, and Future Directions. *IEEE Access* **2019**, *7*, 137184–137206. [CrossRef]
2. Praveen Kumar, D.; Amgoth, T.; Annavarapu, C.S.R. Machine learning algorithms for wireless sensor networks: A survey. *Inform. Fusion* **2019**, *49*, 1–25. [CrossRef]
3. Naseh, D.; Shinde, S.S.; Tarchi, D. Enabling Intelligent Vehicular Networks Through Distributed Learning in the Non-Terrestrial Networks 6G Vision. In Proceedings of the European Wireless 2023; 28th European Wireless Conference, Rome, Italy, 2–4 October 2023.
4. Fontanesi, G.; Ortíz, F.; Lagunas, E.; Baeza, V.M.; Vázquez, M.; Vásquez-Peralvo, J.; Minardi, M.; Vu, H.; Honnaiah, P.; Lacoste, C.; et al. Artificial Intelligence for Satellite Communication and Non-Terrestrial Networks: A Survey. *arXiv* **2023**, arXiv:2304.13008.
5. Lee, H.; Lee, S.H.; Quek, T.Q.S. Deep Learning for Distributed Optimization: Applications to Wireless Resource Management. *IEEE J. Sel. Areas Commun.* **2019**, *37*, 2251–2266. [CrossRef]
6. Huang, J.; Wan, J.; Lv, B.; Ye, Q.; Chen, Y. Joint Computation Offloading and Resource Allocation for Edge-Cloud Collaboration in Internet of Vehicles via Deep Reinforcement Learning. *IEEE Syst. J.* **2023**, *17*, 2500–2511. [CrossRef]
7. Song, H.; Liu, L.; Ashdown, J.; Yi, Y. A Deep Reinforcement Learning Framework for Spectrum Management in Dynamic Spectrum Access. *IEEE Internet Things J.* **2021**, *8*, 11208–11218. [CrossRef]

8. Nayak, P.; Swetha, G.; Gupta, S.; Madhavi, K. Routing in wireless sensor networks using machine learning techniques: Challenges and opportunities. *Measurement* **2021**, *178*, 108974. [CrossRef]
9. Liu, E.; Zheng, L.; He, Q.; Lai, P.; Xu, B.; Zhang, G. Role-Based User Allocation Driven by Criticality in Edge Computing. *IEEE Trans. Serv. Comput.* **2023**, *16*, 3636–3650. [CrossRef]
10. Yang, Z.; Chen, M.; Saad, W.; Hong, C.S.; Shikh-Bahaei, M. Energy Efficient Federated Learning Over Wireless Communication Networks. *IEEE Trans. Wirel. Commun.* **2021**, *20*, 1935–1949. [CrossRef]
11. Jiang, J.C.; Kantarci, B.; Oktug, S.; Soyata, T. Federated Learning in Smart City Sensing: Challenges and Opportunities. *Sensors* **2020**, *20*, 6230. [CrossRef]
12. Wu, Q.; Wang, X.; Fan, Q.; Fan, P.; Zhang, C.; Li, Z. High stable and accurate vehicle selection scheme based on federated edge learning in vehicular networks. *China Commun.* **2023**, *20*, 1–17. [CrossRef]
13. Khan, L.U.; Mustafa, E.; Shuja, J.; Rehman, F.; Bilal, K.; Han, Z.; Hong, C.S. Federated Learning for Digital Twin-Based Vehicular Networks: Architecture and Challenges. *IEEE Wirel. Commun.* **2023**, 1–8. [CrossRef]
14. Sun, Z.; Sun, G.; Liu, Y.; Wang, J.; Cao, D. BARGAIN-MATCH: A Game Theoretical Approach for Resource Allocation and Task Offloading in Vehicular Edge Computing Networks. *IEEE Trans. Mob. Comput.* **2024**, *23*, 1655–1673. [CrossRef]
15. Matthiesen, B.; Razmi, N.; Leyva-Mayorga, I.; Dekorsy, A.; Popovski, P. Federated Learning in Satellite Constellations. *IEEE Netw.* **2023**, 1–16. [CrossRef]
16. Younus, M.U.; Khan, M.K.; Bhatti, A.R. Improving the Software-Defined Wireless Sensor Networks Routing Performance Using Reinforcement Learning. *IEEE Internet Things J.* **2022**, *9*, 3495–3508. [CrossRef]
17. Dewangan, D.K.; Sahu, S.P. Deep Learning-Based Speed Bump Detection Model for Intelligent Vehicle System Using Raspberry Pi. *IEEE Sens. J.* **2021**, *21*, 3570–3578. [CrossRef]
18. Cicceri, G.; Tricomi, G.; Benomar, Z.; Longo, F.; Puliafito, A.; Merlino, G. DILoCC: An approach for Distributed Incremental Learning across the Computing Continuum. In Proceedings of the 2021 IEEE International Conference on Smart Computing (SMARTCOMP), Irvine, CA, USA, 23–27 August 2021; pp. 113–120. [CrossRef]
19. Mills, J.; Hu, J.; Min, G. Communication-Efficient Federated Learning for Wireless Edge Intelligence in IoT. *IEEE Internet Things J.* **2020**, *7*, 5986–5994. [CrossRef]
20. Ridolfi, L.; Naseh, D.; Shinde, S.S.; Tarchi, D. Implementation and Evaluation of a Federated Learning Framework on Raspberry PI Platforms for IoT 6G Applications. *Future Internet* **2023**, *15*, 358. [CrossRef]
21. Wang, S.; Hong, Y.; Wang, R.; Hao, Q.; Wu, Y.C.; Ng, D.W.K. Edge federated learning via unit-modulus over-the-air computation. *IEEE Trans. Commun.* **2022**, *70*, 3141–3156. [CrossRef]
22. Kou, W.B.; Wang, S.; Zhu, G.; Luo, B.; Chen, Y.; Ng, D.W.K.; Wu, Y.C. Communication resources constrained hierarchical federated learning for end-to-end autonomous driving. In Proceedings of the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, MI, USA, 1–5 October 2023; pp. 9383–9390.
23. Wen, D.; Liu, P.; Zhu, G.; Shi, Y.; Xu, J.; Eldar, Y.C.; Cui, S. Task-oriented sensing, computation, and communication integration for multi-device edge AI. *IEEE Trans. Wirel. Commun.* **2023**, 23, 2486–2502. [CrossRef]
24. Chen, M.; Gündüz, D.; Huang, K.; Saad, W.; Bennis, M.; Feljan, A.V.; Poor, H.V. Distributed Learning in Wireless Networks: Recent Progress and Future Challenges. *IEEE J. Sel. Areas Commun.* **2021**, *39*, 3579–3605. [CrossRef]
25. Farkas, A.; Kertész, G.; Lovas, R. Parallel and Distributed Training of Deep Neural Networks: A brief overview. In Proceedings of the 2020 IEEE 24th International Conference on Intelligent Engineering Systems (INES), Reykjavik, Iceland, 8–10 July 2020; pp. 165–170. [CrossRef]
26. Naseh, D.; Shinde, S.S.; Tarchi, D. Network Sliced Distributed Learning-as-a-Service for Internet of Vehicles Applications in 6G Non-Terrestrial Network Scenarios. *J. Sens. Actuator Netw.* **2024**, *13*, 14. [CrossRef]
27. Liu, Z.; Guo, J.; Yang, W.; Fan, J.; Lam, K.Y.; Zhao, J. Privacy-Preserving Aggregation in Federated Learning: A Survey. *IEEE Trans. Big Data* 2022, *early access*. [CrossRef]
28. Li, T.; Sahu, A.K.; Talwalkar, A.; Smith, V. Federated Learning: Challenges, Methods, and Future Directions. *IEEE Signal Process. Mag.* **2020**, *37*, 50–60. [CrossRef]
29. Rafi, T.H.; Noor, F.A.; Hussain, T.; Chae, D.K. Fairness and privacy preserving in federated learning: A survey. *Inform. Fusion* **2024**, *105*, 102198. [CrossRef]
30. Hsu, T.M.H.; Qi, H.; Brown, M. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv* **2019**, arXiv:1909.06335.
31. Zhou, H.; Cheng, J.; Wang, X.; Jin, B. Low rank communication for federated learning. In Proceedings of the Database Systems for Advanced Applications. DASFAA 2020 International Workshops: BDMS, SeCoP, BDQM, GDMA, and AIDE, Jeju, Republic of Korea, 24–27 September 2020; pp. 1–16.
32. Wang, J.; Liu, Q.; Liang, H.; Joshi, G.; Poor, H.V. A novel framework for the analysis and design of heterogeneous federated learning. *IEEE Trans. Signal Process.* **2021**, *69*, 5234–5249. [CrossRef]
33. Yao, X.; Sun, L. Continual local training for better initialization of federated models. In Proceedings of the 2020 IEEE International Conference on Image Processing (ICIP), Abu Dhabi, United Arab Emirates, 25–28 October 2020; pp. 1736–1740.
34. Li, Q.; He, B.; Song, D. Model-contrastive federated learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 19–25 June 2021; pp. 10713–10722.

35. Dinh, C.T.; Tran, N.H.; Nguyen, M.N.H.; Hong, C.S.; Bao, W.; Zomaya, A.Y.; Gramoli, V. Federated Learning Over Wireless Networks: Convergence Analysis and Resource Allocation. *IEEE/ACM Trans. Netw.* **2021**, *29*, 398–409. [CrossRef]
36. Li, Z.; He, Y.; Yu, H.; Kang, J.; Li, X.; Xu, Z.; Niyato, D. Data Heterogeneity-Robust Federated Learning via Group Client Selection in Industrial IoT. *IEEE Internet Things J.* **2022**, *9*, 17844–17857. [CrossRef]
37. Jiang, K.; Cao, Y.; Song, Y.; Zhou, H.; Wan, S.; Zhang, X. Asynchronous Federated and Reinforcement Learning for Mobility-Aware Edge Caching in IoVs. *IEEE Internet Things J.* 2024, *early access*. [CrossRef]
38. Amiri, S.; Belloum, A.; Nalisnick, E.; Klous, S.; Gommans, L. On the impact of non-IID data on the performance and fairness of differentially private federated learning. In Proceedings of the 2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Los Alamitos, CA, USA, 27–30 June 2022; pp. 52–58. [CrossRef]
39. Zhang, Z.; Zhang, Y.; Guo, D.; Zhao, S.; Zhu, X. Communication-efficient federated continual learning for distributed learning system with Non-IID data. *Sci. China Inf. Sci.* **2023**, *66*, 122102. [CrossRef]
40. Hong, C.S.; Khan, L.U.; Chen, M.; Chen, D.; Saad, W.; Han, Z. *Federated Learning for Wireless Networks*; Springer: Singapore, 2022. [CrossRef]
41. Girelli Consolaro, N.; Shinde, S.S.; Naseh, D.; Tarchi, D. Analysis and Performance Evaluation of Transfer Learning Algorithms for 6G Wireless Networks. *Electronics* **2023**, *12*, 3327. [CrossRef]
42. Liu, Y.; Kang, Y.; Xing, C.; Chen, T.; Yang, Q. A Secure Federated Transfer Learning Framework. *IEEE Intell. Syst.* **2020**, *35*, 70–82. [CrossRef]
43. Yang, H.; He, H.; Zhang, W.; Cao, X. FedSteg: A Federated Transfer Learning Framework for Secure Image Steganalysis. *IEEE Trans. Netw. Sci. Eng.* **2021**, *8*, 1084–1094. [CrossRef]
44. Sharma, S.; Xing, C.; Liu, Y.; Kang, Y. Secure and Efficient Federated Transfer Learning. In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; pp. 2569–2576. [CrossRef]
45. Wang, A.; Zhang, Y.; Yan, Y. Heterogeneous Defect Prediction Based on Federated Transfer Learning via Knowledge Distillation. *IEEE Access* **2021**, *9*, 29530–29540. [CrossRef]
46. Gao, D.; Liu, Y.; Huang, A.; Ju, C.; Yu, H.; Yang, Q. Privacy-preserving Heterogeneous Federated Transfer Learning. In Proceedings of the 2019 IEEE International Conference on Big Data (Big Data), Los Angeles, CA, USA, 9–12 December 2019; pp. 2552–2559. [CrossRef]
47. Yan, Z.; Li, D. Performance Analysis for Resource Constrained Decentralized Federated Learning Over Wireless Networks. *IEEE Trans. Commun.* 2024, *early access*. [CrossRef]
48. Beutel, D.J.; Topal, T.; Mathur, A.; Qiu, X.; Fernandez-Marques, J.; Gao, Y.; Sani, L.; Li, K.H.; Parcollet, T.; de Gusmão, P.P.B.; et al. Flower: A Friendly Federated Learning Research Framework. *arXiv* **2022**, arXiv:2007.14390.
49. Shiraz, M.; Abolfazli, S.; Sanaei, Z.; Gani, A. A study on virtual machine deployment for application outsourcing in mobile cloud computing. *J. Supercomput.* **2013**, *63*, 946–964. [CrossRef]