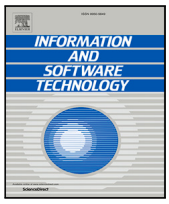
Enabling efficient and low-effort decentralized federated learning with the EdgeFL framework

Information and Software Technology 178 (2025) 107600

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Enabling efficient and low-effort decentralized federated learning with the EdgeFL framework

Hongyi Zhang [a,*], Jan Bosch [a], Helena Holmström Olsson [b]

[a] Chalmers University of Technology, Gothenburg, Sweden
[b] Malmö University, Malmö, Sweden

ARTICLE INFO

Keywords:
Federated learning
Machine learning
Software engineering
Decentralized architecture
Information privacy

ABSTRACT

Context: Federated Learning (FL) has gained prominence as a solution for preserving data privacy in machine learning applications. However, existing FL frameworks pose challenges for software engineers due to implementation complexity, limited customization options, and scalability issues. These limitations prevent the practical deployment of FL, especially in dynamic and resource-constrained edge environments, preventing its widespread adoption.
Objective: To address these challenges, we propose EdgeFL, an efficient and low-effort FL framework designed to overcome centralized aggregation, implementation complexity and scalability limitations. EdgeFL applies a decentralized architecture that eliminates reliance on a central server by enabling direct model training and aggregation among edge nodes, which enhances fault tolerance and adaptability to diverse edge environments.
Methods: We conducted experiments and a case study to demonstrate the effectiveness of EdgeFL. Our approach focuses on reducing weight update latency and facilitating faster model evolution on edge devices.
Results: Our findings indicate that EdgeFL outperforms existing FL frameworks in terms of learning efficiency and performance. By enabling quicker model evolution on edge devices, EdgeFL enhances overall efficiency and responsiveness to changing data patterns.
Conclusion: EdgeFL offers a solution for software engineers and companies seeking the benefits of FL, while effectively overcoming the challenges and privacy concerns associated with traditional FL frameworks. Its decentralized approach, simplified implementation, combined with enhanced customization and fault tolerance, make it suitable for diverse applications and industries.

1. Introduction

Federated Learning (FL), a revolutionary machine learning approach, enables model training across decentralized data sources while upholding the fundamental concepts of data privacy and security. A frequent technique in traditional machine learning is to centralize all relevant data into a single location for model training. However, this traditional strategy frequently confronts formidable obstacles. These obstacles cover a wide range of issues, from data privacy concerns and severe regulatory requirements to computationally complex demands. To address these restrictions, FL provides a novel approach by allowing models to be trained locally on the devices or servers from which the data originated, eliminating the need to send data to a central repository [1].

The most significant advantage of FL is its data privacy preservation. This strategy assures that data remains on client devices or servers, eliminating any concerns or suspicions about data leakage or the unintended exchange of critical information [2]. Instead of sending raw, unaltered data, FL relies on the communication of only model parameter changes, which are frequently encrypted or anonymized. This decentralized training paradigm empowers organizations, researchers, and individuals to form collaborations in the pursuit of model improvements without sacrificing the integrity and security of their data resources.

In addition to data privacy protection, FL adds a new level to collaborative model training. The approach extracts collective knowledge that is more powerful than the sum of its individual parts by encapsulating data and knowledge scattered across several nodes or devices [3]. The federated method supports the integration of various sources of data, resulting in the construction of models that are not only more accurate but also robust, which can be adaptable to a wide range of real-world scenarios.

FL can be applied to applications in various domains, such as

* Corresponding author.
E-mail address: hongyiz@chalmers.se (H. Zhang).

https://doi.org/10.1016/j.infsof.2024.107600
Received 11 May 2024; Received in revised form 9 October 2024; Accepted 10 October 2024
Available online 17 October 2024

healthcare, finance, automotive, Internet of Things (IoT), and edge computing [4–7][8,9]. One of its advantages is its ability to enable enterprises to exploit the collective knowledge contained within distributed datasets without violating severe privacy requirements or exposing sensitive information to unnecessary risks. This capacity to combine knowledge while adhering to data privacy regulations is especially important in fields such as healthcare and finance, where patient or financial data confidentiality is crucial [10]. Furthermore, FL appears as a powerful tool for reducing reliance on costly and bandwidth-intensive data transfers.

1.1. Challenges of FL development

However, while FL offers significant advantages, it also poses distinctive challenges for AI engineers, as highlighted in the research by Lwakatare et al. [11]. These challenges revolve around the intricate design and implementation of distributed systems. Engineers are tasked with creating systems capable of efficiently managing communication, coordination, and synchronization between numerous clients and a central server. This involves overcoming challenges such as scalability and fault tolerance to accommodate extensive deployments [12].

Existing FL frameworks, whether from academia or industry, tend to be complex to use. FL involves various components like distributed systems, optimization algorithms, privacy techniques, and machine learning models [13,14]. When these elements are combined into a single framework, it can result in intricate systems with numerous dependencies and configurations. This complexity can be challenging for users, especially those without a strong background in distributed systems or machine learning [15].

Additionally, FL is applied in diverse domains and use cases, each with its unique requirements and constraints [16]. Designing a platform or framework that can accommodate this diversity can add to the complexity. It becomes a challenge to strike a balance between providing flexibility for customization and maintaining simplicity for ease of use [17]. As indicated in previous research [18], software engineers must understand and implement FL algorithms and optimization techniques such as Federated Averaging and adaptive learning rate strategies. The development of algorithms that efficiently converge towards high-quality models while consuming the least amount of resources represents an overwhelming task that software engineers must face within the context of FL. These diverse duties highlight the complexities of FL and the skill set required for its successful implementation [19].

Last but not least, software engineers may face difficulties because of the lack of comprehensive tools and frameworks designed expressly for FL. To meet their special objectives, they may need to adapt current tools, create customized solutions, or contribute to open-source initiatives [20,21]. Constructing simplified workflows, putting in place effective debugging mechanisms, and creating monitoring tools that fit FL systems are all challenging tasks that require careful attention and expertise. This diverse landscape emphasizes the complexities of integrating FL with edge devices and IoT contexts, where resource efficiency and adaptability are critical.

In summary, the challenges can be concluded into four aspects:

- Scalability and Decentralization in FL: Existing FL frameworks predominantly rely on centralized aggregation, which introduces a single point of failure and limits scalability. The challenge lies in designing a decentralized architecture that can effectively eliminate the central server while maintaining model accuracy and reducing communication overhead.
- Customization and Flexibility in Aggregation Functions: Current FL frameworks often lack the flexibility needed to customize aggregation functions to suit specific use cases. The gap here involves developing an approach that allows for adaptable aggregation functions, which can be fine-tuned to optimize performance across different domains and data distributions.
- Efficient Model Evolution and Latency Reduction: The latency associated with weight updates and model evolution in FL systems has been a significant bottleneck. The gap is in devising methods that can reduce this latency while ensuring rapid model convergence, especially in scenarios with unequally distributed datasets.
- Complexity of Implementation: Implementing FL systems is often complex due to the need to integrate multiple components such as distributed systems, optimization algorithms, and privacy techniques. The challenge lies in simplifying this implementation process to make FL more accessible to engineers without deep expertise in these areas.

1.2. Contributions

To tackle these challenges, in this paper, we introduce EdgeFL, an efficient and low-effort FL framework tailored for edge computing environments. EdgeFL addresses the challenges associated with centralized aggregation by adopting an edge-only model training and aggregation approach. This eliminates the need for a central server and allows for seamless scalability across various use cases. The framework offers a simple integration process, requiring only four lines of code (LOC) for software engineers to add FL functionalities to their AI products. Moreover, EdgeFL supports the customization of aggregation functions, giving engineers the flexibility to adapt them to their specific needs. Therefore, the contributions of this paper are as follows:

(1) We identify the challenges by evaluating existing FL frameworks in six categories, including line of code, system design, architecture settings, aggregation type and deployability.

(2) To tackle these challenges, we introduce EdgeFL,[1] a scalable and low-effort edge-only FL framework. To accomplish easy-implementation and scalable model training capacity, simple API design and learning flow abstraction are built.

(3) We propose a decentralized FL architecture and learning method that enables asynchronous model training to speed up industrial FL training and support large-scale node communication along edges. The architecture can be used as a template for future edge-only FL development and research.

(4) We validate EdgeFL using two well-known datasets, namely MNIST and CIFAR-10 and compared them with five existing FL frameworks/algorithms. Furthermore, a real-world case, the distributed sentiment analysis on the IMDB Dataset of 50K Movie Reviews, is constructed and used to validate the empirical deployment of the EdgeFL framework.

The remainder of this paper is structured as follows. In Section 2, we introduce the background and related work of this study. Section 3 details our research method, including the implementation, data distribution, machine learning methods applied and evaluation metrics. Section 4 presents the system design of our proposed EdgeFL. Section 5 evaluates our proposed framework and compares it with existing FL frameworks. Section 6 outlines the discussion on our observed results. Finally, Section 7 presents conclusions and future work.

2. Background and related work

2.1. Federated learning

FL is a machine learning approach that enables multiple decentralized devices or entities to collaboratively train a shared model without sharing their raw data [22–24]. In traditional machine learning methods, a central server or data aggregator gathers and stores training data from different sources and then trains a global model using this

[1] https://github.com/HarryME-zh/EdgeFL.git.

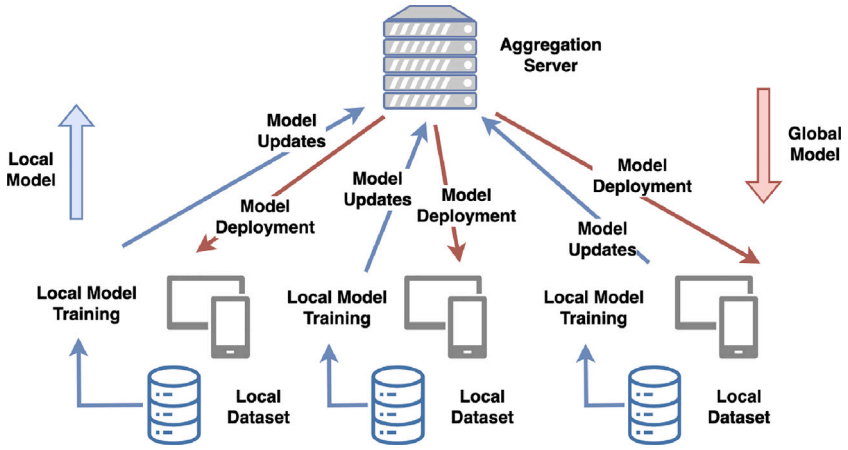


Fig. 1. Diagram of FL training process.

combined data. However, this centralized approach raises concerns related to data privacy, security, and the practicality of transferring large data volumes to a central location [18].

In the context of the growing use of edge computing and distributed data sources, there is an increasing need to utilize data that is distributed across various devices or entities while respecting data ownership and privacy [25]. FL addresses this challenge by allowing local devices, such as smartphones, IoT devices, or edge servers, to train a shared model using their locally stored data [26]. This process is illustrated in Fig. 1 and typically involves the following steps:

(1) Initialization: In the first step, the central server kicks off the process by initializing a global model and distributing it to the participating devices or entities.

(2) Local Model Training: Each device or entity then takes the initiative to train the global model using its own local data independently. Importantly, this is done without sharing the raw data. The local model undergoes training using techniques like gradient descent or other optimization methods. This involves updating the model parameters based on the device's individual data.

(3) Model Aggregation: Following the local training phase, the devices or entities transmit their locally computed model updates, which can include gradients, to the central server. The central server collects and aggregates these updates using methods such as Federated Averaging or Secure Aggregation, resulting in an improved global model.

(4) Iterative Process: The model training and aggregation process iterates over multiple rounds, allowing the global model to improve over time. Each round typically includes local model training, model aggregation, and communication between devices and the central server.

FL utilizes the power of collaborative learning from a diverse range of devices or entities, each with its unique data distribution and characteristics. This diversity plays a crucial role in enhancing the global model's generalization and robustness by capturing a more comprehensive representation of the data.

2.2. Existing FL frameworks

There are several existing FL frameworks available that facilitate the development and deployment of FL systems:

1. TensorFlow Federated (TFF): Google's open-source research-centric framework that is designed to extend TensorFlow's capabilities into the field of FL. It excels in providing a flexible programming model and a rich set of APIs, making it a strong choice for researchers experimenting with FL algorithms [27]. However, TFF's focus on research and simulation may limit its practical deployment in industrial settings, particularly in edge environments where scalability and ease of deployment are crucial.
2. PySyft: PySyft focuses on privacy-preserving FL and secure multi-party computation and is built on top of PyTorch. Its high-level API is excellent for leveraging differential privacy and secure aggregation, making it suitable for applications where data privacy is important. PySyft facilitates collaborative learning with remote data and models while preserving privacy [28]. Nevertheless, PySyft's complexity in setting up and its emphasis on privacy might make it less straightforward for developers who prioritize ease of use and broad scalability.
3. FATE: FATE is an industrial-grade FL framework developed by Webank Research, that provides a secure and adaptable infrastructure for collaborative model training across distributed entities. This framework places a strong emphasis on data privacy. It is particularly well-suited for industries like finance and healthcare that require strict data privacy controls. However, FATE's comprehensive feature set and its focus on data privacy can add to its complexity, potentially making it challenging to integrate with simpler, lightweight FL applications, especially in edge computing scenarios [29].
4. LEAF: LEAF, developed by NVIDIA, is an experimental FL framework that provides tools for evaluating FL algorithms. While LEAF is valuable for benchmarking and experimenting, it lacks the production-ready features required for deploying FL in real-world, large-scale environments. The absence of strong support for edge deployments and limited flexibility in customization can be considered as weaknesses in comparison to our proposed EdgeFL framework [30].
5. PaddleFL: PaddleFL is a deep-learning framework that supports large-scale FL models with a focus on distributed infrastructure. It offers various optimization algorithms and communication protocols, making it a strong candidate for large-scale deployments. However, PaddleFL may not be as straightforward to implement in heterogeneous environments or where the need for custom aggregation functions is critical [31].

Compared to the existing frameworks, EdgeFL is designed specifically to address practical challenges that are not fully resolved by the above-mentioned frameworks:

- Scalability and Ease of Integration: Unlike TFF and FATE, which may be complex to deploy in large-scale, real-world applications, EdgeFL is designed to be lightweight and easily integrable, requiring only four lines of code to implement.
- Decentralization and Edge Suitability: While frameworks like PySyft and FATE emphasize privacy, they often rely on centralized components for aggregation, which can be a bottleneck. EdgeFL eliminates the need for a central server, enhancing scalability and making it more suitable for edge environments where decentralized operations are necessary.

Table 1
Comparison between existing FL frameworks and our proposed EdgeFL.

	TFF	PySyft	FATE	LEAF	PaddleFL	EdgeFL[1] (Proposed framework)
Line of Code (LOC) for FL Feature	~50	~150	~100	~350	~100	**4**
Centralized server required	Yes	Yes	Yes	Yes	Yes	**No**
Asynchronous node join	No	No	No	No	No	**Yes**
Heterogeneous environment support	No	No	No	No	Limited	**Yes**
Customizable aggregation	No	No	No	No	No	**Yes**
Containerized edge deployability	No	No	Limited	No	Limited	**Yes**

- Customization and Flexibility: Unlike PaddleFL and LEAF, which may have limited support for custom aggregation functions and edge deployments, EdgeFL offers customizable aggregation mechanisms and supports asynchronous node joining, making it adaptable to a wide range of use cases.

While many decentralized FL models discussed in the literature focus primarily on aggregation algorithms and model training strategies [25,32,33], EdgeFL distinguishes itself by offering a comprehensive, implementation-ready framework. Unlike these models, which often lack complete infrastructure support, EdgeFL provides a robust API and end-to-end infrastructure that facilitates node communication, decentralized learning, and real-world deployment. This focus on practical application and ease of integration makes EdgeFL a valuable tool for software engineers, allowing them to implement FL systems tailored to their specific requirements, while also addressing challenges related to scalability, fault tolerance, and efficient model updates.

2.3. Problem description

Despite the various functionalities of these FL frameworks, the current FL frameworks available in academia and industry are often complex to use, demanding a thorough understanding of FL concepts. The majority of these systems demand a deep understanding of FL concepts and require the implementation of over 100 lines of code (LOC) to deploy an FL application. This often limits flexibility in terms of customizing aggregation functions and lacks support for asynchronous communication schemes, as summarized in Table 1. These constraints raise considerable challenges for software engineers aiming to seamlessly integrate FL into production environments.

Moreover, the complexity of existing FL frameworks extends beyond just the initial deployment phase. Maintenance and adaptation become obstacles as well. With numerous dependencies and configurations, updating or modifying an existing FL system can be a daunting task [12,15]. This complexity not only increases the risk of introducing errors or vulnerabilities but also impedes the agility needed to respond to evolving requirements or emerging research advancements in the FL domain.

Additionally, the steep learning curve associated with these frameworks can deter potential adopters, particularly those without a strong background in distributed systems or machine learning [34]. The intricate interplay between various components such as distributed systems, optimization algorithms, and privacy techniques necessitates a deep understanding of FL concepts, making it challenging for newcomers to navigate and harness the full potential of FL technology.

Furthermore, the lack of standardization across existing FL frameworks exacerbates the problem. Each framework may have its own set of APIs, data formats, and communication protocols, further complicating interoperability and hindering the development of reusable components or libraries [34]. This fragmentation not only fragments the community but also impedes collaboration and knowledge sharing, slowing down the overall progress in the field of FL.

3. Research method

In this research, we adopted the empirical methodology and learning procedure outlined by Zhang [35] to conduct a comprehensive quantitative measurement and evaluation of our proposed EdgeFL framework in comparison to existing FL frameworks [36]. The empirical methodology is particularly well-suited for evaluating the performance and practical utility of software frameworks. It allows for systematic observation, measurement, and comparison in controlled environments, ensuring that the results are both robust and generalizable. The methods we employed includes a combination of experiments and a case study [37–39]. These methods were chosen as they offer a comprehensive way to evaluate the performance and practicality of EdgeFL in both controlled and real-world settings. Firstly, we conducted a series of controlled experiments to systematically evaluate EdgeFL's performance against existing FL frameworks. The experiments were designed to simulate real-world conditions, focusing on key metrics such as accuracy, communication overhead, computational efficiency, and latency. These experiments allowed us to quantitatively measure and compare the scalability, efficiency, and ease of use of EdgeFL and its competitors. The choice of these metrics is critical because they directly reflect the core challenges in FL deployment, particularly in edge computing environments. To complement the experimental findings, we implemented a case study involving a real-world machine learning task—sentiment analysis on the IMDB dataset. The case study was chosen to demonstrate how EdgeFL performs in a practical application, highlighting its integration process, usability, and overall effectiveness in a real-world scenario. The combination of these methods provides a robust evaluation process. The experiments offer precise, repeatable measurements in a controlled setting, while the case study provides practical insights into EdgeFL's applicability and effectiveness in real-world situations. Together, they ensure that our findings are both rigorous and practically relevant. In the subsequent sections of this paper, we present the details of the selection process and criteria for the selected frameworks. We provide a detailed insight into our implementation approach. This includes our methodology for dataset partitioning and distribution, which is crucial for conducting heterogeneous simulations. We also present the metrics that were utilized to evaluate the performance of our framework. Additionally, we provide an in-depth exploration of the machine-learning techniques employed during our experimental analysis.

3.1. Search process

We employed a structured search strategy to identify all existing FL frameworks. Initially, we conducted a literature review, searching academic papers, conference proceedings, technical documentation, and industry reports related to FL technologies [40]. Key search terms included "Federated Learning Frameworks", "FL Platforms", "Distributed Machine Learning", and "Collaborative Learning Systems". This search strategy aimed to capture a diverse range of frameworks used in both research and industry settings.

Building upon insights gained from the literature review, we formulated criteria and metrics to guide the evaluation process. These criteria were designed to assess the functionality, privacy preservation capabilities, scalability, ease of use, and adoption/community support of FL

frameworks but also the comprehensiveness of their infrastructure and implementation readiness. Those frameworks demonstrating robust features, privacy preservation mechanisms, scalability, user-friendliness, substantial adoption in research or industry, and integration with other application development and implementation environments were prioritized for further evaluation [41]. Additionally, emphasis was placed on evaluating frameworks that provided a complete infrastructure with APIs that allow software engineers to implement FL solutions tailored to specific requirements, including node communication, decentralized learning, and deployment. By considering these key aspects, we aimed to ensure that the selected frameworks would be suitable for addressing the research objectives effectively and the comparison with our proposed EdgeFL framework.

3.2. Implementation

To comprehensively assess the performance and capabilities of the EdgeFL framework, we conducted a series of experiments using two machine learning applications: digit recognition and object recognition. For these experiments, we leveraged the MNIST and CIFAR-10 datasets, which are extensively applied in the research field. To enable deep learning training and testing, the PyTorch backend is utilized. One of the features of the EdgeFL framework is its simplicity in integration. With the integration of EdgeFL, the FL functionality is seamlessly integrated into these machine-learning applications with the addition of only four lines of code (LOC). Furthermore, we containerized the applications, enabling easy deployment on edge devices while maintaining their functionality and performance.

The flexibility of the EdgeFL framework allows it to be constructed on various container orchestration clusters, such as Kubernetes, Docker Swarm, etc. For the purposes of this study, we adopted Docker Swarm as our preferred cluster management solution [42]. Docker Swarm provides an efficient and scalable environment for handling containerized applications. The services within Docker Swarm facilitate smooth communication among containers, and an internal DNS resolver ensures seamless peer node service communication. By leveraging Docker Swarm's capabilities, we were able to establish a robust and scalable deployment environment for the EdgeFL framework, ensuring its suitability for edge devices and distributed computing scenarios.

3.3. Dataset distribution

For the purpose of this study, we used two kinds of edge data distribution to analyze system performance for heterogeneous simulation.

3.3.1. Uniform distribution

Within this experimental setup, the training data samples were distributed among the edge nodes following a uniform distribution. This distribution ensured an equal likelihood of data samples from each target class.

3.3.2. Normal distribution

Within this configuration, the number of samples in each class within each edge node follows a normal density function. Mathematically, this can be expressed as: $X \sim \mathcal{N}(\mu, \sigma^2)$ where μ and σ are defined as: $\mu = \frac{k \times N}{K}$, $\sigma = 0.2 \times N$

In the above equations, k represents the ID of each edge node, K denotes the total number of edge nodes, and N corresponds to the total number of target classes in the training data. This configuration aims to provide varied distributions and different numbers of samples among different edge nodes, allowing each class to have a probability of having the majority of samples in a specific node.

3.4. Machine learning method

The implementation of the models in this study utilized Python and relied on the following libraries: torch 1.6.0 [43], torchvision 0.7.0 [44], and scikit-learn [45], which were applied in model construction and evaluation.

To achieve satisfactory classification results, two distinct convolutional neural networks (CNN) [46] were trained for the MNIST and CIFAR-10 datasets. For the MNIST application, the CNN architecture comprised two 5×5 convolutional layers (with 10 output channels in the first layer and 20 in the second), each followed by 2×2 max pooling. Additionally, a fully connected layer with 50 units employing the ReLU activation function and a linear output layer were included.

For the CIFAR-10 application, the CNN architecture featured four 5×5 convolutional layers (with 66 output channels in the first layer, 128 in the second with a stride of 2, 192 in the third, and 256 in the fourth with a stride of 2). Furthermore, two fully connected layers utilizing the ReLU activation function, with 3000 and 1500 units respectively, were incorporated along with a linear output layer.

3.5. Evaluation metrics

To assess the effectiveness of EdgeFL, three key metrics were selected: weights update latency, model evolution time, and model classification performance [41,47]. The choice of these metrics is grounded in their ability to provide comprehensive insights into the performance and efficiency of the system while addressing the characteristics of EdgeFL's decentralized architecture.

3.5.1. Weights update latency

Weights update latency was chosen as a primary metric due to its direct relevance to the network and communication aspects of the FL process. In traditional FL systems using centralized architectures, where a central aggregation server collects and processes model updates, the measure of latency is relatively straightforward. However, in the case of EdgeFL, which adopts a decentralized architecture, the aggregation function is moved to the edge, and a peer-to-peer model update exchange is facilitated. Measuring the time it takes for these updates to be transmitted across edge nodes is a key indicator of the system's efficiency and responsiveness.

This metric offers insights into the network conditions, communication overhead, and overall efficiency of different FL architecture options. It helps determine how quickly the system can adapt to new data and how effectively it can disseminate model updates. By calculating the average weights update latency across all edge nodes during a training round, EdgeFL's performance in a dynamic, decentralized environment is effectively evaluated. This metric is measured by comparing the sending and receiving timestamps in all model receivers, shedding light on the speed at which EdgeFL can accommodate changing data patterns.

3.5.2. Model evolution time

Model evolution time was included as a critical metric to gauge the speed at which local edge devices can adapt and update their knowledge. In the context of FL, it is essential for systems to quickly evolve their models to adapt to rapidly changing environments or data distributions. This is especially crucial for real-time applications and industries where up-to-date information is important.

Similar to weights update latency, the average model evolution time across all edge nodes during one training round is measured. This metric provides insights into the responsiveness of EdgeFL, indicating how swiftly the deployed models at the edge nodes are updated. By examining the timestamps of model deployment, the system's ability to keep pace with evolving data patterns is effectively quantified.

3.5.3. Model classification performance

Model classification performance serves as a fundamental metric to assess the quality of the trained model, a vital consideration in machine learning applications of our testbed.

This metric evaluates the percentage of correctly recognized images among the total number of testing images. The evaluation is performed on each edge device using their updated models, ensuring that the test sample distribution aligns with the training samples. The average classification performance across all edge nodes is reported, offering a comprehensive measure of how well EdgeFL's decentralized learning approach fares in terms of model quality and accuracy.

4. System design of EdgeFL

In this section, we offer a complete and in-depth explanation of the system design of EdgeFL, providing a full insight into its architectural design and operational structure. This part also includes a thorough presentation of the extensive set of APIs and functions available to EdgeFL users. It goes into the EdgeFL learning life-cycle, explaining the steps and processes that define its functioning, and providing a comprehensive understanding of the framework's usefulness and adaptability.

4.1. System design

Within the FL process, EdgeFL provides a seamless path to achieving scalability, fault tolerance, and customized flexibility. This framework is made up of two main components: FL Edge Nodes and Registration Nodes, inspired by the architectural principles that underpin successful systems like Skype [48]. The FL Edge Nodes allow users to connect directly, thus minimizing the reliance on a central server for data transmission. In EdgeFL, the FL Edge Nodes collaborate seamlessly, harnessing their collective computing power and data while preserving the privacy of each individual contributor. Parallel to this, the Registration Nodes act as critical coordination centers, facilitating the connection and interaction of FL Edge Nodes. They enable the decentralized architecture of EdgeFL by providing the necessary mechanisms for Edge Nodes to identify and communicate with each other independently. This decentralized architecture enables Edge Nodes to identify and interact with one another independently, enabling a durable and flexible ecosystem for FL processes.

- FL Edge Nodes: The FL Edge Nodes, which are strategically deployed on edge devices, play an important function within the FL framework, with each Edge Node serving as an active participant contributing to the efficacy of the FL process. These nodes play an important role in executing the FL training process, permitting model exchanges with other nodes, and performing critical local model updates. In practice, the FL Edge Node code is developed using the Flask framework, which is well-known for its efficiency and dependability. This code architecture provides machine-learning model files on demand, enabling smooth and efficient interactions across all peer nodes.
- Registration Nodes: Registration Nodes serve as essential coordinators for participated FL edge nodes. Their primary responsibility is to manage a catalogue of active peers, as well as to provide important services for the registration, unregistration, and retrieval of relevant peer information. With their functionalities, these Registration Nodes act as catalysts, allowing FL edge nodes to discover and engage with one another in a fully decentralized manner. The registration nodes are built on a solid architecture with the Flask framework, which is well-known for its dependability and efficiency. This infrastructure exposes a variety of APIs, each precisely designed to streamline and improve the FL process, hence strengthening the framework's operational efficiency and effectiveness.

Table 2
APIs and services of EdgeFL.

Name	Endpoint	Method
Registration API	/register	POST
Unregistration API	/unregister	POST
Peer Information Retrieval API:	/peers	GET
Model File Serving API	/latest_model	GET

4.2. APIs and services

Table 2 summarizes the most important APIs and services for EdgeFL, including edge node registration and registration, peer information retrieval and model file serving.

- Registration API: FL Edge Nodes initiate the registration process by sending a registration request via this API to the Registration Nodes. The hostname of the FL Edge Node is included in the request. After successful registration, the Registration Nodes add the newly added peer information to their lists of active peers.
- Unregistration API: When FL Edge Nodes stop participating in the FL process, they use this API to send unregistration requests to the Registration Nodes. The hostname of the outgoing FL Edge Node is specified in these requests. In reaction, the Registration Nodes remove the related peer information from their lists of active participants immediately.
- Peer Information Retrieval API: FL Edge Nodes can employ this API to query the Registration Nodes for the list of active peers. The Registration Nodes promptly respond with the desired active peer information. This functionality empowers FL Edge Nodes to seamlessly discover and establish communication channels with their peers.
- Model File Serving API: FL Edge Nodes expose this API to handle requests for machine-learning model files. When a peer requests the latest model, the responding FL Edge Node delivers the requisite machine-learning model file via an HTTP response, ensuring the efficient sharing of critical model data.

The example function usage has been listed in Appendix. The EdgeFL framework simplifies the integration of FL capabilities into AI applications, requiring only four lines of code (LOC) for implementation. This ease of use allows software engineers to quickly deploy FL functionality without substantial code modifications or complex re-engineering. The framework provides a streamlined process, beginning with the creation of a Peer instance, which manages configuration, registration, and communication with other FL nodes. The peer instance facilitates key functions such as model aggregation through $peer.aggregation\_func()$ and ensures orderly participation and withdrawal from the FL system via $peer.start()$ and $peer.unregister\_peer()$, which allows seamless participation in the EdgeFL ecosystem.

4.3. EdgeFL learning life-cycle

The life cycle of the EdgeFL framework contains several essential steps for an individual edge node to seamlessly become part of the collaborative process, including joining, model training, knowledge sharing, model aggregation, and optionally departing from the FL process. Algorithm 1 outlines the detailed FL learning process of an individual edge node. Here's a detailed description of each step:

1. Edge Node Joining: The process begins with the edge node initializing itself by creating an instance of the Peer class and a background instance for handling model requests. The node then connects to the registration nodes, essentially announcing its presence and status as an active participant. In return, it receives information about other peers and their participation in the FL process.

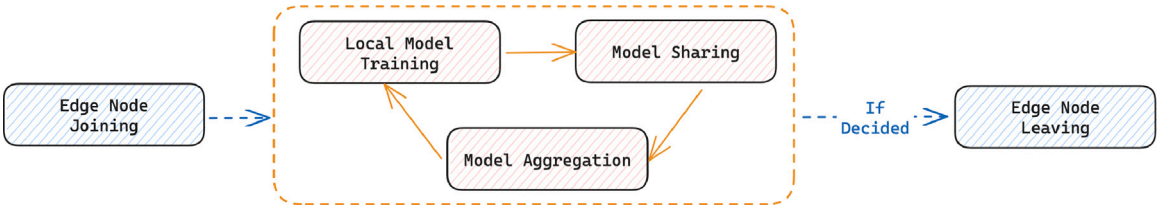


Fig. 2. The learning life-cycle of EdgeFL, including joining the FL process, model training, sharing, aggregation, and eventual node leaving. These stages collectively define the operational flow of EdgeFL within an edge node.

2. Model Training: With its registration complete, the edge node dives into the FL training process. It begins model training using its locally stored dataset. Through iterative updates, the node refines its local model to enhance its performance.
3. Sharing Models: The FL client node takes an active role in knowledge sharing, retrieving models from fellow peers within the FL framework. It identifies these peers through the registration nodes, fetching the most recent models. These models are then seamlessly integrated into their own local model updates. Notably, this model retrieval process operates asynchronously, ensuring that ongoing local model training remains uninterrupted. This asynchronous model aggregation mechanism allows the FL client node to simultaneously contribute to and benefit from the collaborative learning process.
4. Model Aggregation: The FL client node performs an aggregation function, which combines locally updated models with models provided by other peers. The aggregation function combines several models to create a new aggregated model that incorporates the collective knowledge of all participating nodes. It is crucial to note that the EdgeFL framework's aggregation function can be tailored to individual analysis and case requirements, giving software engineers the freedom to define and implement various aggregation algorithms that meet their specific requirements. For general performance analysis, this study employs a default averaging function.
5. Edge Node Leaving: In the event that an edge node decides to exit the EdgeFL system, the FL client node notifies the registration node of its intent to leave, providing its hostname for identification. The registration node promptly updates the active participant list, removing the departing edge node. It is worth noting that some edge nodes may choose to stay in the system even after completing their learning process. By choosing to stay, they contribute by providing their completed learning models, which proves beneficial for newcomers entering the EdgeFL framework. This approach ensures that the system retains the availability of fully learned models, simplifying the onboarding process for new participants.

This life cycle is repeated as additional edge nodes join the FL framework, contribute to the training and aggregation processes, exchange their models, and eventually leave when they decide to discontinue their involvement (see Fig. 2). While protecting the privacy and autonomy of individual edge nodes, the EdgeFL enables continual collaborative learning and model development.

4.4. Containerization and scalable deployment

To facilitate effortless deployment on a wide range of edge devices, the EdgeFL framework underwent containerization, employing the robust containerization technology known as Docker. This process involved the encapsulation of all essential components, dependencies, and configurations of EdgeFL into a lightweight and portable container image. This containerization approach delivers the flexibility for EdgeFL to be deployed seamlessly across diverse edge devices, regardless of the specific underlying operating system or hardware architecture.

Algorithm 1: EdgeFL - In the system, α represents the ratio of aggregated peers; C is the active peer list; B is the local mini-batch size; E represents the number of local epochs, and γ is the learning rate.

Initialize w_0
Initialize α as the ratio of aggregated peers
Initialize C as the active peer list
Function `Server_Function()`:
  for each round t = 1, 2, ... **do**
    Node_Training(w_t)
    Retrieval active peer list C
    $m \longleftarrow max(len(C) \times \alpha, 1)$;
    $N_t \longleftarrow$(random set of m peers from C);
    for each node $k \in N_t$ **do**
      Threads.start()
      Fetch w_{t+1}^k
      Threads.end()
    end for
    $w_{t+1} \longleftarrow \sum_{k=1}^{K} \frac{1}{K} w_{t+1}^k$;
  end for

Function `Node_Training(w)`:
  $\beta \longleftarrow$(split P_k into batches of size B);
  for each local epoch i from 1 to E **do**
    for batch $b \in \beta$ **do**
      $w \longleftarrow w - \gamma \nabla l(w; b)$;
    end for
  end for
  return w for model sharing

The architectural diagram showcased in Fig. 3 provides a representation of the streamlined and scalable deployment architecture of the EdgeFL framework. Within this architectural framework, each edge node container includes a range of services, such as model training, model aggregation, and model serving. Concurrently, the registration node container is equipped with services for handling registration and peer discovery tasks. This interconnected setup ensures smooth communication among all nodes within the EdgeFL framework. An important feature to highlight is the ability to expand the number of registration nodes in alignment with the growth of participating edge nodes. This scalability enables efficient coordination and management within the EdgeFL framework, ensuring smooth coordination.

By containerizing EdgeFL, software engineers benefit from the ease of distributing and deploying the framework on edge devices, alleviating concerns about intricate installation procedures or compatibility issues. The containerized EdgeFL image includes all the necessary software libraries, frameworks, and configurations, creating a self-contained environment for running the FL client nodes. Furthermore, containerization ensures that the EdgeFL framework remains isolated and independent, effectively preventing conflicts with other software components residing on the edge device. This approach not only simplifies deployment but also improves the stability and reliability of the EdgeFL framework in diverse edge computing environments.

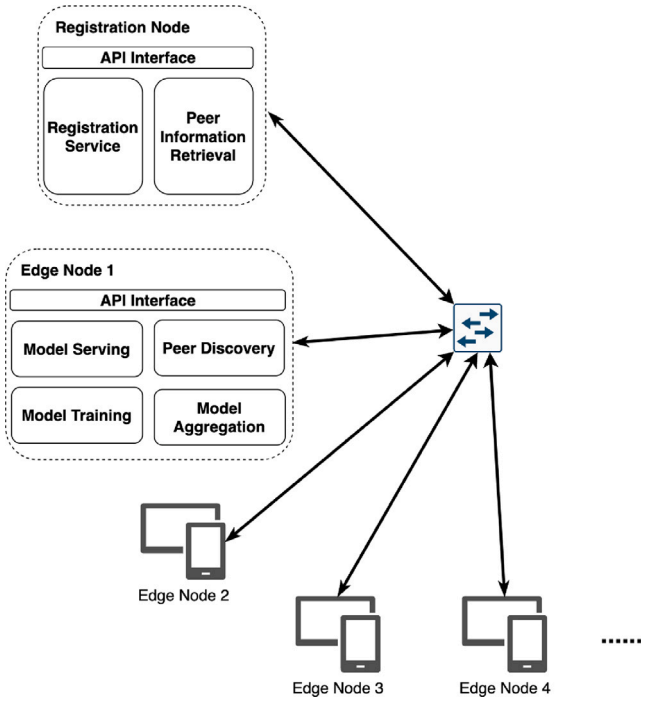


Fig. 3. Containerized architecture for seamless and scalable deployment of the EdgeFL framework.

4.5. Comparison to existing FL framework

EdgeFL employs a decentralized architecture where FL Edge Nodes collaborate directly, eliminating reliance on a central server and significantly enhancing fault tolerance. This decentralized approach reduces the risk of single points of failure and provides a robust, flexible framework suited to dynamic edge environments. In contrast, frameworks like TensorFlow Federated and FATE rely on a central server for managing model aggregation and coordination, which can become a bottleneck and a potential single point of failure [29,49]. PySyft supports decentralized learning but often still involves some central coordination, while LEAF's reference implementations may also be central-server dependent [28,30,50]. PaddleFL, while supporting decentralized setups, typically defaults to a central coordination model, which can limit fault tolerance compared to EdgeFL's fully decentralized approach [31].

In addition, EdgeFL is designed with scalability in mind, allowing edge nodes to operate independently and interact directly with one another. This architecture supports a scalable ecosystem that can grow dynamically with minimal overhead. Registration Nodes facilitate peer discovery and management in a decentralized manner, adapting well to varying network conditions and device capabilities. On the other hand, frameworks like TFF and FATE face scalability challenges due to their central server-based models, which can become bottlenecks and affect performance in large-scale, heterogeneous networks [51,52]. PySyft's scalability may be constrained by its reliance on central coordination, and LEAF's benchmark-oriented design does not inherently address scalability in real-world deployments. PaddleFL's central coordination model can similarly limit its scalability and flexibility compared to the decentralized design of EdgeFL [53–55].

EdgeFL is tailored to operate effectively in heterogeneous edge environments, accommodating devices with varying computational power, storage, and connectivity. Its support for asynchronous communication and model aggregation ensures efficient performance across diverse conditions [56,57]. The containerization of EdgeFL further enhances its adaptability, enabling seamless deployment across different hardware and software platforms. Conversely, traditional frameworks like TFF and FedProx may struggle in heterogeneous settings due to their reliance on synchronous updates and assumptions of homogeneous device capabilities. PySyft's adaptability is enhanced by privacy-preserving techniques but may still face challenges in highly diverse environments [15,58]. FATE's robustness in privacy comes with the cost of less flexibility in heterogeneous settings, while LEAF's benchmark focus might not address the nuances of real-world adaptability. PaddleFL's adaptability is also affected by its default central coordination model [59].

Last but not least, EdgeFL emphasizes ease of use with its streamlined implementation and user-friendly APIs, allowing for rapid integration into existing AI solutions with minimal re-engineering. This simplicity is particularly advantageous in fast-paced development environments [53,60]. In contrast, other FL frameworks often require more extensive setup and configuration. For instance, TensorFlow Federated and FATE can involve complex configurations and integration efforts. PySyft, while offering privacy features, may necessitate additional setup for privacy and security configurations. LEAF, being a benchmarking tool, does not prioritize ease of integration, and PaddleFL, despite its robust features, can involve a more involved setup process compared to the straightforward integration offered by EdgeFL.

5. Evaluation results

This section presents the experimental results obtained from the EdgeFL framework, with a specific focus on three aspects outlined in Section 3.5. (1) Weights Update Latency: This metric quantifies the time required to transmit model weights from one node to another. (2) Model Evolution Time: This aspect measures the duration it takes to acquire a new version of the model. (3) Classification Accuracy: The evaluation of model performance on the edge dataset is a key element of this analysis. To ensure that our experiments provide robust and meaningful results, we conducted simulations with a total of 10 nodes. In these simulations, all nodes actively participated in the training procedure for both MNIST and CIFAR10 applications. This setup was designed to ensure an adequate number of samples on each edge node, facilitating a comprehensive analysis and evaluation of the EdgeFL framework and offering insights into the real-world performance and effectiveness of EdgeFL in practical scenarios.

Firstly, we examine two performance metrics: weights update latency and model evolution time. Our experimental findings, as presented in Table 3, showcase the comparisons among the EdgeFL framework and existing FL frameworks across both MNIST and CIFAR10 applications.

In the domain of weights update latency, EdgeFL consistently demonstrates its efficiency. It exhibits reduced delays in transmitting model weights between edge nodes, underscoring the efficacy of its decentralized architecture. These improvements in weights update latency can be attributed to the streamlined communication processes inherent to EdgeFL. The decentralized nature of the framework allows for rapid sharing of updated models among nodes.

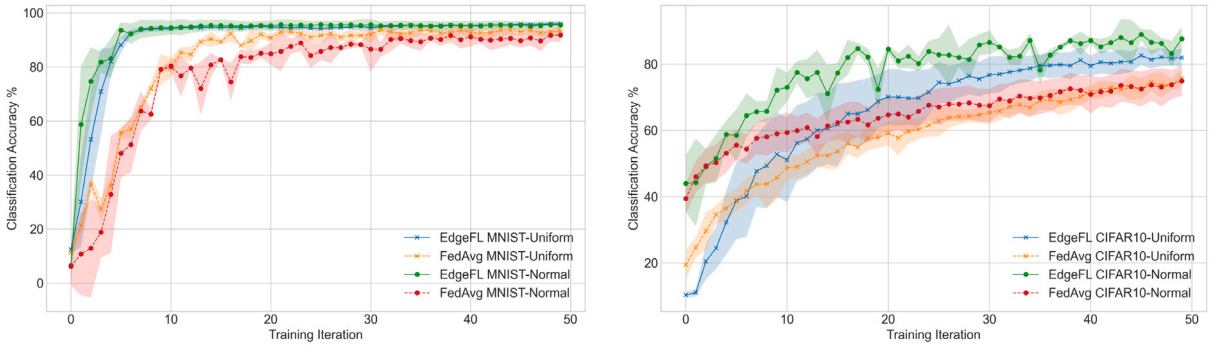
EdgeFL also excels at achieving rapid model evolution in scenarios with unequally distributed datasets. EdgeFL leverages its decentralized architecture and a pull-based model-sharing mechanism to facilitate quick model evolution. This mechanism empowers edge nodes to promptly enhance their local models, effectively adapting to the available data. Consequently, EdgeFL reduces the time required for model evolution, especially when dealing with imbalanced dataset distributions.

These findings demonstrate the advantages of EdgeFL over traditional FL methods. Its better performance in terms of model delay and evolution time can be attributed to its efficient communication and aggregation processes. By utilizing the potential of edge computing, EdgeFL minimizes network overhead and streamlines the model update process, leading to quicker and more responsive knowledge sharing

Table 3
Comparison of weight update latency and model evolution time by leveraging existing FL frameworks and our proposed EdgeFL framework.

FL Frameworks	MNIST		CIFAR10	
	Weights update latency (s)	Model evolution time (s)	Weights update latency (s)	Model evolution time (s)
TFF[a]	–	7.76	–	16.372
PySyft	0.0247	9.05	0.0311	18.292
FATE	0.0326	13.868	0.0473	31.689
LEAF[a]	–	10.239	–	27.362
PaddelFL	0.0258	11.667	0.0412	25.581
EdgeFL	**0.0092**	**5.093**	**0.0148**	**10.753**

[a] TFF and LEAF frameworks do not include actual server and client implementations but rather provide simulations of the FL process. Therefore, measuring weight latency is not feasible within these frameworks.



(a) MNIST (b) CIFAR-10

Fig. 4. The comparison of classification accuracy by utilizing FedAvg (commonly used by existing FL frameworks) and EdgeFL framework.

across the edge network. The observed enhancements in model delay and evolution time carry substantial implications for real-world applications. Reduced delays enable the quick exchange of updated models, ensuring timely access to the latest knowledge throughout the edge network. Furthermore, the reduced evolution time enables edge devices to promptly adapt to evolving data characteristics. These characteristics make EdgeFL well-suited for use cases that demand rapid model evolution and responsiveness to dynamic environmental changes.

In addition to evaluating weights update delay and model evolution time, we conducted extensive accuracy comparisons between EdgeFL's decentralized averaging and the widely used FedAvg algorithm [22] found in existing FL frameworks. Fig. 4 illustrates the accuracy comparisons. The results demonstrate that EdgeFL's decentralized averaging approach outperforms the centralized FedAvg method when it comes to evaluating models on edge devices. The average accuracy achieved through EdgeFL's decentralized averaging is approximately 2% higher for MNIST and 5% higher for CIFAR-10 datasets. These findings underscore the accuracy improvements made by EdgeFL's decentralized averaging approach. The observed increase in accuracy demonstrates the effectiveness of EdgeFL's decentralized averaging mechanism in enhancing model performance. By utilizing the collective knowledge and insights gained from distributed edge devices, EdgeFL facilitates improved model convergence.

Fig. 5 illustrates the model distribution latency of the decentralized EdgeFL algorithm in comparison to other algorithms as the number of edge nodes increases from 10 to 100. From the figure, PySyft and PaddleFL exhibit slightly higher latency, particularly with more nodes. FATE records the highest latency, especially in larger networks, which results in greater communication overhead. EdgeFL's latency shows a gradual rise with node expansion, indicating manageable overhead for larger networks due to decentralized characteristics. This scalability is achieved because each node can establish equal connections, distributing server workload more evenly to the edge, and effectively balancing updating traffic. Consequently, EdgeFL demonstrates the most modest growth rate among the compared algorithms, showcasing its scalable

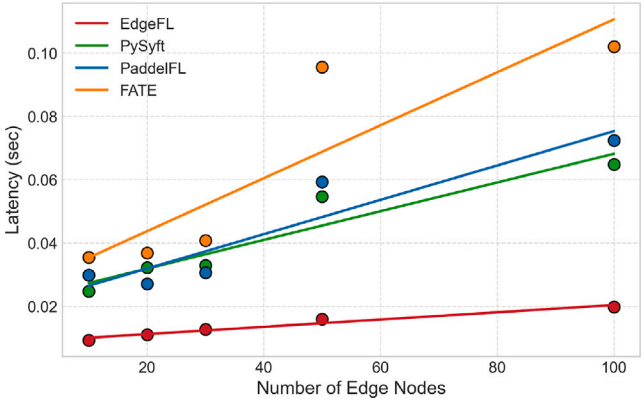


Fig. 5. Model distribution latency with the increasing number of edge nodes.

performance, particularly in large-scale machine learning applications and edge computing scenarios.

Moreover, our study demonstrates the effectiveness of EdgeFL's asynchronous join feature, which enables new nodes to seamlessly integrate into the existing system and swiftly acquire the latest knowledge without necessitating a full retraining process from scratch. As depicted in Fig. 6, our observations reveal that when new nodes (node id: 11, 12) join the system midway through the training process, they rapidly achieve the same level of accuracy as the overall system. This outcome illustrates EdgeFL's capability to facilitate efficient knowledge transfer and rapid model convergence for newly joined nodes.

The asynchronous join functionality within EdgeFL offers advantages in terms of scalability and time-to-adaptability. By allowing new nodes to immediately benefit from the collective intelligence of the system without the need for extensive training, EdgeFL reduces the

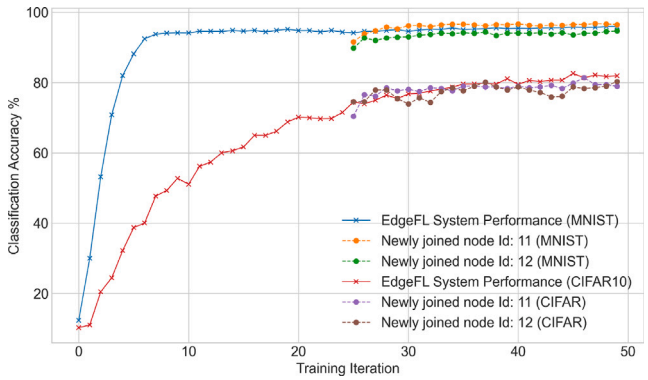


Fig. 6. Midway joined node classification performance in both MNIST and CIFAR-10 application.

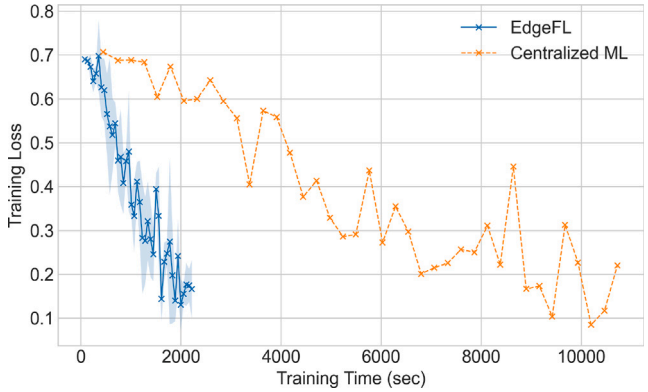


Fig. 7. Training Loss with training time consumed by EdgeFL and Centralized ML.

computational burden and time required for onboarding new participants. This attribute proves its value in dynamic environments where nodes frequently join and depart from the system.

This demonstration of the asynchronous join capability within EdgeFL underlines its potential for real-world deployments, especially in scenarios where rapid knowledge transfer and swift integration of new nodes are crucial. By leveraging the existing knowledge base and facilitating the seamless assimilation of new nodes, EdgeFL empowers FL systems to adapt and evolve efficiently over time. These findings underscore the advantages of EdgeFL's asynchronous join mechanism, highlighting its potential to enhance the scalability and flexibility of FL within dynamic edge computing environments.

6. Case study

In addition to the comparisons, we conduct a case study to apply the EdgeFL framework to a practical machine learning application. In this case study, we leverage the EdgeFL framework to tackle the task of sentiment analysis on the IMDB Dataset of 50K Movie Reviews, a well-known benchmark for binary sentiment classification [61]. This dataset comprises 50,000 movie reviews for training and testing, providing a substantial amount of data for our analysis. Our objective is to perform sentiment classification, distinguishing between positive and negative sentiments within the movie reviews. To address this task, a Long Short-Term Memory (LSTM) network is utilized, which is known for its ability to capture sequential dependencies and nuances within textual data [62,63]. LSTM networks are particularly well-suited for natural language processing tasks like sentiment analysis.

We simulate ten distinct movie review sites, each represented by an individual user end node. This decentralized setting not only allows

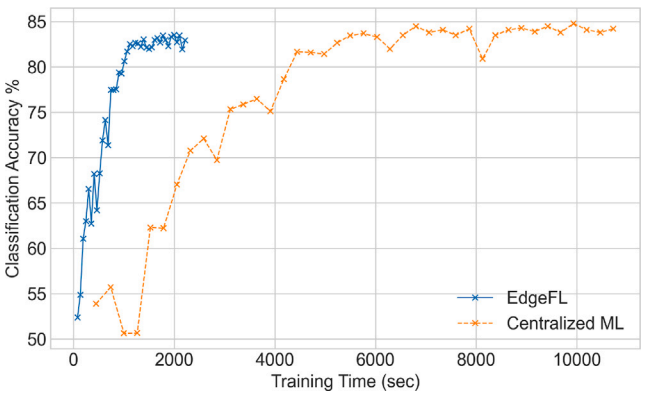


Fig. 8. Test accuracy change with training time consumed by EdgeFL and centralized ML.

for parallel processing and training on diverse data sources locally. The decentralized architecture of EdgeFL ensures that each server node trains an LSTM model locally, making updates based on the reviews available at its respective sites. These local models are then collaboratively aggregated with the function "$peer.aggregation\_func()$" to create a global model that encapsulates insights from all nodes. This FL process enables the collective model to benefit from the diversity of reviews across different movie sites, resulting in a more robust sentiment analysis model.

Figs. 7 and 8 illustrated the result of the sentiment analysis application with the EdgeFL framework compared to traditional centralized machine learning. For the result of training loss, the figure illustrates a notable efficiency gain achieved by the EdgeFL framework in comparison to traditional centralized machine learning. Specifically, EdgeFL demonstrates approximately five times greater efficiency in terms of training time. This efficiency is evident in the faster convergence of the training loss curve, indicating that the model improves significantly in less time.

Fig. 8 depicts the comparative performance of sentiment analysis training using two approaches: EdgeFL and traditional centralized machine learning. The test accuracy of EdgeFL is the average value among all participated end nodes. The results illustrate that EdgeFL achieves faster training times, approximately five times more efficient when performing rapid model training and inference, indicating its superiority in scenarios involving decentralized data sources or resource-constrained environments.

Through decentralized training across ten server nodes, we achieve robust sentiment classification while respecting data privacy and security, making this approach highly suitable for large-scale, distributed natural language processing tasks, especially in scenarios involving user-generated content and edge computing environments. In this case, we build a practical prototype of a decentralized FL system with the EdgeFL framework that can be seamlessly deployed to container orchestration platforms such as Docker Swarm, Kubernetes, etc.

7. Discussion

This paper focuses on analyzing and interpreting the results obtained from the experiments conducted with the EdgeFL framework. We evaluated EdgeFL using a range of metrics. Firstly, the EdgeFL outperformed existing FL frameworks in terms of weights update delay [64]. EdgeFL managed to reduce weights update delay by roughly 50%, surpassing centralized alternatives. When dealing with unevenly distributed datasets, EdgeFL demonstrated its benefit as well. EdgeFL's decentralized architecture addresses key limitations of traditional FL frameworks, such as centralized aggregation and scalability challenges [23,65]. Unlike FedAvg and similar approaches that rely on a central server [66], EdgeFL employs decentralized model training and

aggregation, enhancing fault tolerance and reducing latency. This approach aligns with the broader shift towards decentralized systems in distributed computing, offering a new perspective on how FL can be implemented in edge environments. The implications of this work include a rethinking of model aggregation strategies in FL, moving away from server-dependent methods to a more resilient, peer-to-peer system. EdgeFL's pull-based model-sharing mechanism introduces a dynamic way for nodes to access the latest updates, contributing to faster convergence and adaptation in non-IID environments [54,67], which could inspire further research into adaptive and asynchronous FL methods that better accommodate the heterogeneity of real-world data and network conditions.

Compared to previous FL frameworks, EdgeFL's contributions lie in its ability to achieve faster weight updates and model convergence without sacrificing accuracy. The framework's decentralized averaging technique consistently outperforms FedAvg, particularly in scenarios with unevenly distributed datasets [54,66], demonstrating that EdgeFL can achieve more efficient and accurate learning in diverse edge environments. Furthermore, EdgeFL enables faster model training and evolution, making it a solution for applications where data is generated and processed locally, such as IoT devices, smart cities, and autonomous systems [68]. The ability of EdgeFL to seamlessly integrate new nodes without requiring complete retraining further underscores its scalability and adaptability, which are crucial for real-world deployment. Additionally, the case study on sentiment analysis with the IMDB dataset demonstrates how EdgeFL can be practically applied to enhance the performance and efficiency of machine learning tasks, particularly in scenarios with decentralized data sources.

However, throughout our studies, it is important to note that, despite the multiple benefits provided by EdgeFL, we observed two major limits that deserve consideration.

Firstly, we observed a notable increase in bandwidth cost as the number of nodes in our decentralized network grew. This escalation in connectivity demands is a natural consequence of decentralization, and while it can be partially managed through parameters such as the connection parameter α (which determines the number of models shared in each round), it is important to note that adjusting this parameter may have an effect on the final quality of the model. Addressing this trade-off is crucial because excessive bandwidth consumption can limit the scalability of EdgeFL and increase operational costs, particularly in environments with constrained network resources. Research focused on optimizing communication protocols, such as developing efficient data compression techniques, adaptive bandwidth management strategies, and novel aggregation methods, is essential. These advancements would enable EdgeFL to maintain high model accuracy while minimizing bandwidth demands, making it more practical and cost-effective for large-scale, real-world deployments.

Second, we encountered computational constraints while training large neural networks, particularly when dealing with resource-constrained edge devices. The inherent limitations of these edge devices, such as low processing power and memory, caused difficulties in carrying out complicated training operations. These constraints can severely impact the performance and feasibility of FL processes on such devices. The need to perform intensive computations and manage large models can lead to inefficiencies and slowdowns, which in turn affect the overall effectiveness of the FL system. These highlight the importance of ongoing research efforts focused on reducing computation complexity and developing novel ways for accommodating resource-constrained contexts. By addressing these computational constraints, EdgeFL can become more versatile and capable of operating effectively on a wider range of edge devices, thus enhancing its practical value and usability in diverse edge scenarios.

8. Conclusion

This paper introduces EdgeFL, a novel decentralized FL framework designed exclusively for efficiency and low-effort implementation. EdgeFL effectively addresses scalability, integration, and efficiency challenges by utilizing an edge-only model training and aggregation approach, eliminating the need for a central server. The framework ensures seamless scalability across a diverse range of use cases. The framework offers a straightforward integration process, requiring only four lines of code (LOC) for software engineers to incorporate FL functionalities into their AI products. Furthermore, EdgeFL offers engineers the flexibility to customize aggregation functions to meet their specific needs, enhancing the framework's adaptability and versatility. Our experiments and evaluations demonstrate the strengths and advantages of EdgeFL. In numerous aspects, EdgeFL outperforms existing FL frameworks. It excels in reducing weight update latency and model evolution time by 50%, enhancing classification accuracy by 2% for the MNIST dataset and 5% for the CIFAR dataset compared to other FL frameworks. In a real-world case study, EdgeFL demonstrated remarkable efficiency gains, achieving training times approximately five times faster than traditional centralized machine learning, all while maintaining a comparable level of model quality. These findings highlight EdgeFL's potential in practical applications, particularly in industrial settings where timely and accurate model updates are crucial.

Our future work aims to validate and extend the capabilities of EdgeFL with a broader range of use cases. We also plan to explore resource optimization techniques, such as model compression and quantization, to enhance communication efficiency for edge devices in EdgeFL. Additionally, we will investigate adaptive aggregation strategies that dynamically adjust the aggregation process based on network conditions, device capabilities, and data heterogeneity.

CRediT authorship contribution statement

Hongyi Zhang: Writing – review & editing, Writing – original draft, Validation, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Jan Bosch:** Writing – review & editing, Supervision, Investigation, Funding acquisition, Conceptualization. **Helena Holmström Olsson:** Writing – review & editing, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was funded by the Chalmers AI Research Center and Software Center. The simulations were performed on resources at Chalmers Centre for Computational Science and Engineering (C3SE) provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

Appendix. EdgeFL function details and example usage

The proposed EdgeFL framework provides software engineers with an easy way to integrate FL capabilities into their AI solutions. In contrast to the complexity commonly associated with FL frameworks, EdgeFL offers a concise implementation that requires only four lines of code (LOC). This ease of use enables software engineers to quickly include FL functionality into their existing AI applications, with no need for substantial code changes or complex re-engineering efforts. The following Listing 1 demonstrates an example with which EdgeFL can be deployed.

```
# --- Continue from node training part ---
# --- Initialize peer instance ---
peer = Peer(config)# --- Start peer instance ---
peer.start()for epoch in range(number_of_epochs):

    # --- Pull model from active peers and start
     aggregation
    w_latest = peer.aggregation_func()
    model.load_state_dict(w_latest)

    train(model, torch.device("cpu"),
    train_loader, optimizer, epoch)
    test(model, torch.device("cpu"), test_loader
    )
    scheduler.step()

    torch.save(model.state_dict(), "model-latest
    .pth")

# --- unregister from the registration node if
    leave
peer.unregister_peer()
```

Listing 1: Usage example of EdgeFL.

peer = Peer(configs): The procedure begins with the creation of an instance of the Peer class, a component of the EdgeFL architecture that represents an active participant in the system. During this initialization, an array of configurations is created, which includes elements such as registration node addresses and specific settings for the aggregation function. This phase ensures that the FL edge node is precisely configured, allowing for flawless connection with registration nodes and competent involvement in FL training and aggregation efforts. Once the Peer class instance is established, the Peer class instance serves as a handle, supporting interactions between the FL edge node and its peer entities. The FL edge node gains the ability to do a variety of actions via this peer object, including model retrieval, registration with registration nodes, and model aggregation.

peer.start(): This function initiates the execution for the FL edge node's active participation in the EdgeFL framework. When called, it triggers a series of mandatory steps that prepare the FL edge node for seamless participation in the FL process. These steps include the registration of the FL edge node with the registration nodes, the establishment of connections with its peer entities, and the activation of a background instance ready to handle asynchronous file requests from peers. By calling "peer.start()", the FL edge node actively participates in the collaborative model learning process, while leveraging the computational capabilities in edge devices. By using this function, the FL edge node integrates easily into the EdgeFL ecosystem, establishing its role as a contributor to the system of FL.

peer.aggregation_func(): This function is in charge of orchestrating the aggregation process, which is a cornerstone of the EdgeFL system. When triggered, it starts a multidimensional series of operations aimed at increasing the collective intelligence of the FL system. First, the function requests models from fellow FL edge nodes specified by the registration nodes. Following that, it employs the aggregation method, expertly combining these models into a single updated model. The aggregation function facilitates the collaborative nature of FL. It combines the inputs of various peer entities, harmonizing their knowledge to improve the quality and performance of the model. The FL edge node takes an active role in this iterative model aggregation process by calling "peer.aggregation_func()", effectively becoming a contributor within the EdgeFL framework. This not only encourages the FL system's collective intelligence but also improves the overall quality of the model.

peer.unregister_peer(): This function supports the FL edge node's seamless exit from the EdgeFL framework. When invoked, it performs the role of notifying the registration nodes of the approaching unregistration while providing the necessary information, including the FL edge node's hostname. The function, "peer.unregister_peer()", effectively starts the process of removing the FL client node from the list of active participants maintained by the registration nodes. This action ensures the proper management of participants within the EdgeFL framework and allows for efficient resource allocation and coordination among the remaining active peers, adding to the EdgeFL framework's overall resilience and stability.

Data availability

Data will be made available on request.

References

[1] A. L'heureux, K. Grolinger, H.F. Elyamany, M.A. Capretz, Machine learning with big data: Challenges and approaches, IEEE Access 5 (2017) 7776–7797.

[2] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, B. He, A survey on federated learning systems: Vision, hype and reality for data privacy and protection, IEEE Trans. Knowl. Data Eng. (2021).

[3] Z. Li, V. Sharma, S.P. Mohanty, Preserving data privacy via federated learning: Challenges and solutions, IEEE Consum. Electron. Mag. 9 (3) (2020) 8–16.

[4] S. Lu, Y. Yao, W. Shi, Collaborative learning on the edges: A case study on connected vehicles, in: 2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19), 2019.

[5] B. Hu, Y. Gao, L. Liu, H. Ma, Federated region-learning: An edge computing based framework for urban environment sensing, in: 2018 IEEE Global Communications Conference, GLOBECOM, IEEE, 2018, pp. 1–7.

[6] T.S. Brisimi, R. Chen, T. Mela, A. Olshevsky, I.C. Paschalidis, W. Shi, Federated learning of predictive models from federated electronic health records, Int. J. Med. Inform. 112 (2018) 59–67.

[7] H. Zhang, J. Bosch, H.H. Olsson, Real-time end-to-end federated learning: An automotive case study, in: 2021 IEEE 45th Annual Computers, Software, and Applications Conference, COMPSAC, IEEE, 2021, pp. 459–468.

[8] S. Pandya, G. Srivastava, R. Jhaveri, M.R. Babu, S. Bhattacharya, P.K.R. Maddikunta, S. Mastorakis, M.J. Piran, T.R. Gadekallu, Federated learning for smart cities: A comprehensive survey, Sustain. Energy Technol. Assess. 55 (2023) 102987.

[9] A. Rauniyar, D.H. Hagos, D. Jha, J.E. Håkegård, U. Bagci, D.B. Rawat, V. Vlassov, Federated learning for medical applications: A taxonomy, current trends, challenges, and future research directions, IEEE Internet Things J. (2023).

[10] H. Tao, M.Z.A. Bhuiyan, M.A. Rahman, G. Wang, T. Wang, M.M. Ahmed, J. Li, Economic perspective analysis of protecting big data security and privacy, Future Gener. Comput. Syst. 98 (2019) 660–671.

[11] L.E. Lwakatare, A. Raj, J. Bosch, H.H. Olsson, I. Crnkovic, A taxonomy of software engineering challenges for machine learning systems: An empirical investigation, in: International Conference on Agile Software Development, Springer, Cham, 2019, pp. 227–243.

[12] Y. Dai, Z. Chen, J. Li, S. Heinecke, L. Sun, R. Xu, Tackling data heterogeneity in federated learning with class prototypes, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 37, 2023, pp. 7314–7322.

[13] Q. Xia, W. Ye, Z. Tao, J. Wu, Q. Li, A survey of federated learning for edge computing: Research problems and solutions, High-Confid. Comput. 1 (1) (2021) 100008.

[14] M. Aledhari, R. Razzak, R.M. Parizi, F. Saeed, Federated learning: A survey on enabling technologies, protocols, and applications, IEEE Access 8 (2020) 140699–140725.

[15] M. Ye, X. Fang, B. Du, P.C. Yuen, D. Tao, Heterogeneous federated learning: State-of-the-art and research challenges, ACM Comput. Surv. 56 (3) (2023) 1–44.

[16] L. Zhang, X. Lei, Y. Shi, H. Huang, C. Chen, Federated learning with domain generalization, 2021, arXiv preprint arXiv:2111.10487.

[17] D. Gao, H. Wang, X. Guo, L. Wang, G. Gui, W. Wang, Z. Yin, S. Wang, Y. Liu, T. He, Federated learning based on CTC for heterogeneous internet of things, IEEE Internet Things J. (2023).

[18] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H.B. McMahan, et al., Towards federated learning at scale: System design, 2019, arXiv preprint arXiv:1902.01046.

[19] W. Huang, M. Ye, Z. Shi, H. Li, B. Du, Rethinking federated learning with domain shift: A prototype view, in: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR, IEEE, 2023, pp. 16312–16322.

[20] L.U. Khan, W. Saad, Z. Han, E. Hossain, C.S. Hong, Federated learning for internet of things: Recent advances, taxonomy, and open challenges, IEEE Commun. Surv. Tutor. 23 (3) (2021) 1759–1799.

[21] J.C. Jiang, B. Kantarci, S. Oktug, T. Soyata, Federated learning in smart city sensing: Challenges and opportunities, Sensors 20 (21) (2020) 6230.

[22] B. McMahan, E. Moore, D. Ramage, S. Hampson, B.A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, 2017, pp. 1273–1282.

[23] L. Li, Y. Fan, M. Tse, K.-Y. Lin, A review of applications in federated learning, Comput. Ind. Eng. 149 (2020) 106854.

[24] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, Y. Gao, A survey on federated learning, Knowl.-Based Syst. 216 (2021) 106775.

[25] E. Gabrielli, G. Pica, G. Tolomei, A survey on decentralized federated learning, 2023, arXiv preprint arXiv:2308.04604.

[26] G. Liu, C. Wang, X. Ma, Y. Yang, Keep your data locally: Federated-learning-based data privacy preservation in edge computing, IEEE Netw. 35 (2) (2021) 60–66.

[27] TensorFlow, TensorFlow federated, machine learning on decentralized data, 2021.

[28] A. Ziller, A. Trask, A. Lopardo, B. Szymkow, B. Wagner, E. Bluemke, J.-M. Nounahon, J. Passerat-Palmbach, K. Prakash, N. Rose, et al., Pysyft: A library for easy federated learning, Fed. Learn. Syst.: Towards Next-Gener. AI (2021) 111–139.

[29] Y. Liu, T. Fan, T. Chen, Q. Xu, Q. Yang, Fate: An industrial grade platform for collaborative learning with data protection, J. Mach. Learn. Res. 22 (1) (2021) 10320–10325.

[30] S. Caldas, S.M.K. Duddu, P. Wu, T. Li, J. Konečnỳ, H.B. McMahan, V. Smith, A. Talwalkar, Leaf: A benchmark for federated settings, 2018, arXiv preprint arXiv:1812.01097.

[31] Y. Ma, D. Yu, T. Wu, H. Wang, PaddlePaddle: An open-source deep learning platform from industrial practice, Front. Data Domputing 1 (1) (2019) 105–115.

[32] A. Tariq, M.A. Serhani, F. Sallabi, T. Qayyum, E.S. Barka, K.A. Shuaib, Trustworthy federated learning: A survey, 2023, arXiv preprint arXiv:2305.11537.

[33] L. Yuan, Z. Wang, L. Sun, S.Y. Philip, C.G. Brinton, Decentralized federated learning: A survey and perspective, IEEE Internet Things J. (2024).

[34] M. Chahoud, S. Otoum, A. Mourad, On the feasibility of federated learning towards on-demand client deployment at the edge, Inf. Process. Manage. 60 (1) (2023) 103150.

[35] D. Zhang, J.J. Tsai, Machine learning and software engineering, Softw. Qual. J. 11 (2) (2003) 87–119.

[36] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K. El Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, IEEE Trans. Softw. Eng. 28 (8) (2002) 721–734.

[37] D.I. Sjoberg, B. Anda, E. Arisholm, T. Dyba, M. Jorgensen, A. Karahasanovic, E.F. Koren, M. Vokác, Conducting realistic experiments in software engineering, in: Proceedings International Symposium on Empirical Software Engineering, IEEE, 2002, pp. 17–26.

[38] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, et al., Experimentation in Software Engineering, vol. 236, Springer, 2012.

[39] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empir. Softw. Eng. 14 (2009) 131–164.

[40] B. Kitchenham, S. Charters, et al., Guidelines for performing systematic literature reviews in software engineering, 2007.

[41] A. Nilsson, S. Smith, G. Ulm, E. Gustavsson, M. Jirstrand, A performance evaluation of federated learning algorithms, in: Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning, 2018, pp. 1–8.

[42] F. Soppelsa, C. Kaewkasi, Native Docker Clustering with Swarm, Packt Publishing Ltd, 2016.

[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al., Pytorch: An imperative style, high-performance deep learning library, in: Advances in Neural Information Processing Systems, 2019, pp. 8026–8037.

[44] S. Marcel, Y. Rodriguez, Torchvision the machine-vision package of torch, in: Proceedings of the 18th ACM International Conference on Multimedia, 2010, pp. 1485–1488.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[46] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. 25 (2012) 1097–1105.

[47] Y. Gao, M. Kim, S. Abuadbba, Y. Kim, C. Thapa, K. Kim, S.A. Camtepe, H. Kim, S. Nepal, End-to-end evaluation of federated learning and split learning for internet of things, 2020, arXiv preprint arXiv:2003.13376.

[48] R. Kazman, H.-M. Chen, The architecture of complexity revisited: Design primitives for ultra-large-scale systems, 2023.

[49] T. Solanki, B.K. Rai, S. Sharma, Federated learning using tensor flow, in: Federated Learning for IoT Applications, Springer, 2022, pp. 157–167.

[50] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, J. Passerat-Palmbach, A generic framework for privacy preserving deep learning, 2018, arXiv preprint arXiv:1811.04017.

[51] Q. Li, Y. Diao, Q. Chen, B. He, Federated learning on non-iid data silos: An experimental study, in: 2022 IEEE 38th International Conference on Data Engineering, ICDE, IEEE, 2022, pp. 965–978.

[52] P. Kairouz, H.B. McMahan, B. Avent, A. Bellet, M. Bennis, A.N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al., Advances and open problems in federated learning, Found. Trends® Mach. Learn. 14 (1–2) (2021) 1–210.

[53] E.T.M. Beltrán, M.Q. Pérez, P.M.S. Sánchez, S.L. Bernal, G. Bovet, M.G. Pérez, G.M. Pérez, A.H. Celdrán, Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges, IEEE Commun. Surv. Tutor. (2023).

[54] H. Zhu, J. Xu, S. Liu, Y. Jin, Federated learning on non-IID data: A survey, Neurocomputing 465 (2021) 371–390.

[55] Y. Zhou, Y. Shi, H. Zhou, J. Wang, L. Fu, Y. Yang, Toward scalable wireless federated learning: Challenges and solutions, IEEE Internet Things Mag. 6 (4) (2023) 10–16.

[56] P.M. Mammen, Federated learning: Opportunities and challenges, 2021, arXiv preprint arXiv:2101.05428.

[57] J. Pang, Y. Huang, Z. Xie, Q. Han, Z. Cai, Realizing the heterogeneity: A self-organized federated learning framework for IoT, IEEE Internet Things J. 8 (5) (2020) 3088–3098.

[58] C. Xu, Y. Qu, Y. Xiang, L. Gao, Asynchronous federated learning on heterogeneous devices: A survey, Comp. Sci. Rev. 50 (2023) 100595.

[59] W. Huang, M. Ye, Z. Shi, G. Wan, H. Li, B. Du, Q. Yang, Federated learning for generalization, robustness, fairness: A survey and benchmark, IEEE Trans. Pattern Anal. Mach. Intell. (2024).

[60] I. Kholod, E. Yanaki, D. Fomichev, E. Shalugin, E. Novikova, E. Filippov, M. Nordlund, Open-source federated learning frameworks for IoT: A comparative review and analysis, Sensors 21 (1) (2020) 167.

[61] A.L. Maas, R.E. Daly, P.T. Pham, D. Huang, A.Y. Ng, C. Potts, Learning word vectors for sentiment analysis, in: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Portland, Oregon, USA, 2011, pp. 142–150, URL http://www.aclweb.org/anthology/P11-1015.

[62] N.M. Ali, M.M. Abd El Hamid, A. Youssif, Sentiment analysis for movies reviews dataset using deep learning models, Int. J. Data Min. Knowl. Manag. Process. (IJDKP) 9 (2019).

[63] S.M. Qaisar, Sentiment analysis of IMDb movie reviews using long short-term memory, in: 2020 2nd International Conference on Computer and Information Sciences, ICCIS, IEEE, 2020, pp. 1–4.

[64] O. Shahid, S. Pouriyeh, R.M. Parizi, Q.Z. Sheng, G. Srivastava, L. Zhao, Communication efficiency in federated learning: Achievements and challenges, 2021, arXiv preprint arXiv:2107.10996.

[65] K. Bonawitz, Towards federated learning at scale: Syste m design, 2019, arXiv preprint arXiv:1902.01046.

[66] S. AbdulRahman, H. Tout, H. Ould-Slimane, A. Mourad, C. Talhi, M. Guizani, A survey on federated learning: The journey from centralized to distributed on-site learning and beyond, IEEE Internet Things J. 8 (7) (2020) 5476–5497.

[67] X. Li, K. Huang, W. Yang, S. Wang, Z. Zhang, On the convergence of fedavg on non-iid data, 2019, arXiv preprint arXiv:1907.02189.

[68] T. Zhang, L. Gao, C. He, M. Zhang, B. Krishnamachari, A.S. Avestimehr, Federated learning for the internet of things: Applications, challenges, and opportunities, IEEE Internet Things Mag. 5 (1) (2022) 24–29.