Chapter 5

Optimized Container Selection Using Shared Memory Architecture

This chapter presents a novel federated learning system using Docker, enabling memory sharing among interconnected containers. Multiple-leaf devices are linked to each container and contribute to collaborative learning. The iterative process refines the global model continuously, enhancing system performance and resource optimization. The modular architecture promotes scalability and task encapsulation, indicating promising future implications.

5.1 Introduction

Federated learning, though not a recent concept, has garnered significant attention in recent years due to its potential to address data privacy concerns. [**?**] This approach to machine learning involves training data locally on individual devices using a machine learning model residing on the device itself, as opposed to sending the data to a centralized server for training [**?**]. This decentralized approach ensures that sensitive data remains on the user's system, preserving data privacy while still enabling collaborative model training. This paradigm shift in machine learning has profound implications

for safeguarding personal information in an era of increasing data awareness and privacy concerns. In this chapter, we embark on the implementation of a novel federated learning mechanism through Docker, a method that involves establishing connections among multiple Docker containers to facilitate memory sharing within a structured graph framework [**?**]. Each Docker container is intricately linked to five-leaf devices, which, despite their limited processing capabilities, play a pivotal role in our collaborative learning framework. These interconnected containers collaborate to create a global model and transmit updates to this global model to a designated shared memory repository. Subsequently, the containers retrieve these updated models from the shared memory and employ them to train their respective datasets. After this training phase, the modified weight files are uploaded back into the shared memory. Notably, when any container obtains all the weight files, it triggers a signal to other nearby volumes, indicating the commencement of the aggregation process. This container then aggregates the weight files to compute a new model. This iterative process continues until all dataset files have participated in the training, culminating in the generation of an updated global model. This procedure ensures that the new global model is continually refined as all the containers train on their distinct dataset files. One of the key benefits of our proposed approach is the enhanced performance it offers. By connecting containers in a graph structure and enabling memory sharing, we expect to witness substantial improvements in the system's overall performance. Additionally, our approach promises resource optimization by efficiently utilizing shared memory among containers. The structured graph architecture also enhances the scalability of containers, allowing for adaptability to varying workloads [**?**]. Furthermore, our framework adopts a modular and decentralized architecture, where each container represents a specific service, thereby promoting better isolation and encapsulation of tasks within the system [**?**]. These aspects collectively contribute to the significance and innovation of our research endeavor.

5.2 Research Work

In this research endeavor, we propose a sophisticated containerized learning framework that operates through a meticulous cycle of model updates and aggregation, while accommodating three distinct types of devices. Within this framework, Docker containers play a pivotal role, receiving the latest model updates and employing them to train their respective datasets. Following this training phase, the containers contribute to the collective knowledge by uploading their modified weight files into a shared memory repository. What sets our approach apart is the innovative signal mechanism introduced when a container acquires all the weight files it needs. This signal informs nearby containers that the aggregation process has commenced. Subsequently, the container takes charge of aggregating these weight files to compute a new, refined global model. This iterative process perseveres until all dataset files have been fully engaged in the training procedure. The research culminates in the computation of a new global model, a culmination reached only when every container has diligently trained on its unique dataset files. This tailored approach encompasses three distinct device types: the robust containers, brimming with processing power to oversee and execute all tasks; the shared storage repositories, devoid of processing power yet proficient in storing weight files and the global model; and the humble leaf devices, bereft of processing power but entrusted with safeguarding the invaluable dataset files. This research signifies a transformative step towards achieving efficient and privacy-aware federated learning, all within the dynamic realm of containerization.

5.3 Architecture

In configuring the architectural framework for this research, we have adopted a strategic approach to harness the potential of containerization within the context of federated learning. Our framework consists of multiple interconnected containers structured in a

graph formation, enabling shared memory communication among them. Each container is directly linked to leaf devices, collectively contributing to the creation of a new global model through collaborative efforts, as depicted in Figure 5.1.

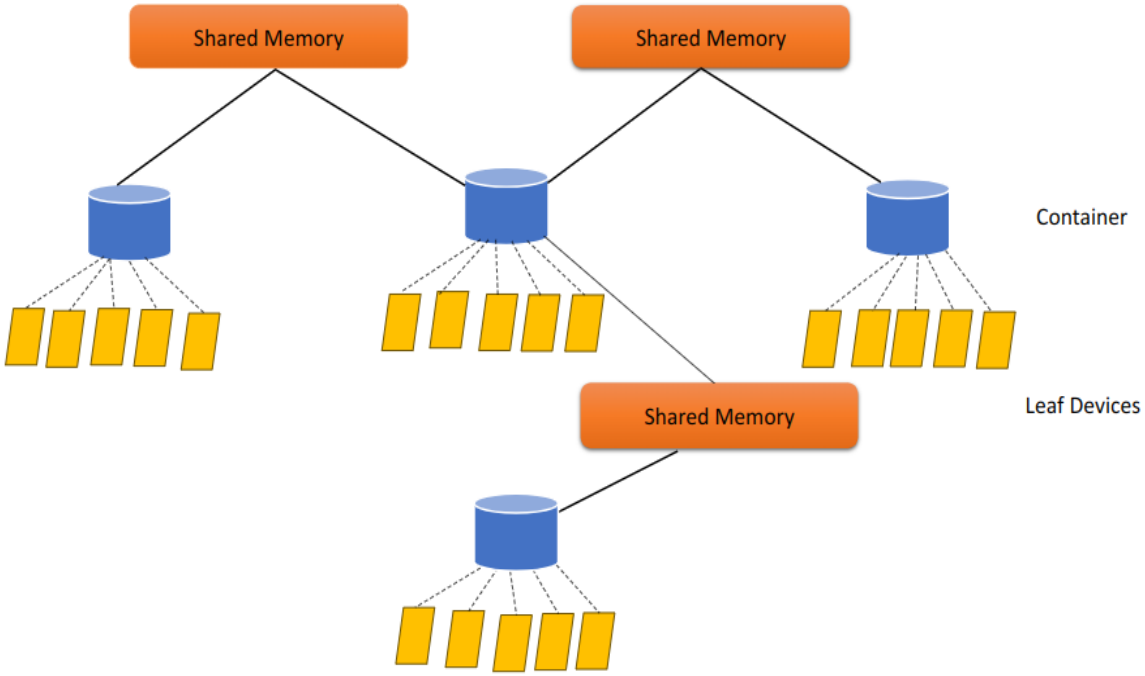


Figure 5.1: An illustration of proposed architecture with leaf devices, containers, and shared memory.

This collaborative model training process aligns with the core principles of federated learning, which ensures data privacy while allowing multiple devices or edge nodes to train a shared model. In our architectural design, data privacy is upheld by keeping data decentralized. Instead of sending data to a central server, the training process occurs locally on each device, ensuring that sensitive data remains on the device itself. Within our framework, each participating container performs local model updates using its own data, following the federated learning concept of local model updates. After these local updates, the container sends only the model update, not the raw data to a central server or aggregator. The aggregator's role is pivotal as it collects and aggregates these updates to create an improved global model, mirroring the process of aggregating model updates in federated learning. This iterative cycle, encompassing

the above steps, repeats for a predefined number of communication rounds, ensuring the continual enhancement of the global model, as per the federated learning principle of iterative training, as illustrated in Figure 5.2. The improved global model is then sent back to the devices, and the cycle continues, allowing the model to learn from the collective knowledge of all devices. Within this architectural blueprint, we have meticulously assigned distinct roles to three key components: the Containers, equipped with the processing power required to efficiently manage and execute tasks; the Shared Memory, serving as a secure storage repository for weight parameter matrices and the global model; and the Leaf Devices, entrusted with the responsibility of safeguarding the essential data files contributed by participants. The architectural design represents a well-structured synergy among these components, poised to redefine the landscape of federated learning within the dynamic realm of containerization. This innovative approach offers both efficiency and scalability, addressing intricate machine learning challenges while staying true to the principles of data privacy and collaborative model training.

The main components of the architecture are as follows:

- **Leaf Devices:** These components serve as the guardians of the dataset files, ensuring their protection and security. With no processing power, their primary responsibility is to safeguard the crucial dataset files, a critical aspect of the data privacy measures in the system.
- **Docker Containers (Processing):** These entities play a crucial role in the system, managing the local model updates with the data at their disposal. Emphasizing the principle of data privacy, the containers ensure that only the model updates, not the raw data, are transmitted. They also facilitate the signalling of the aggregation process, highlighting their active participation in the collaborative model training approach.
- **Shared Memory Repository:** Serving as a vital storage repository, the shared

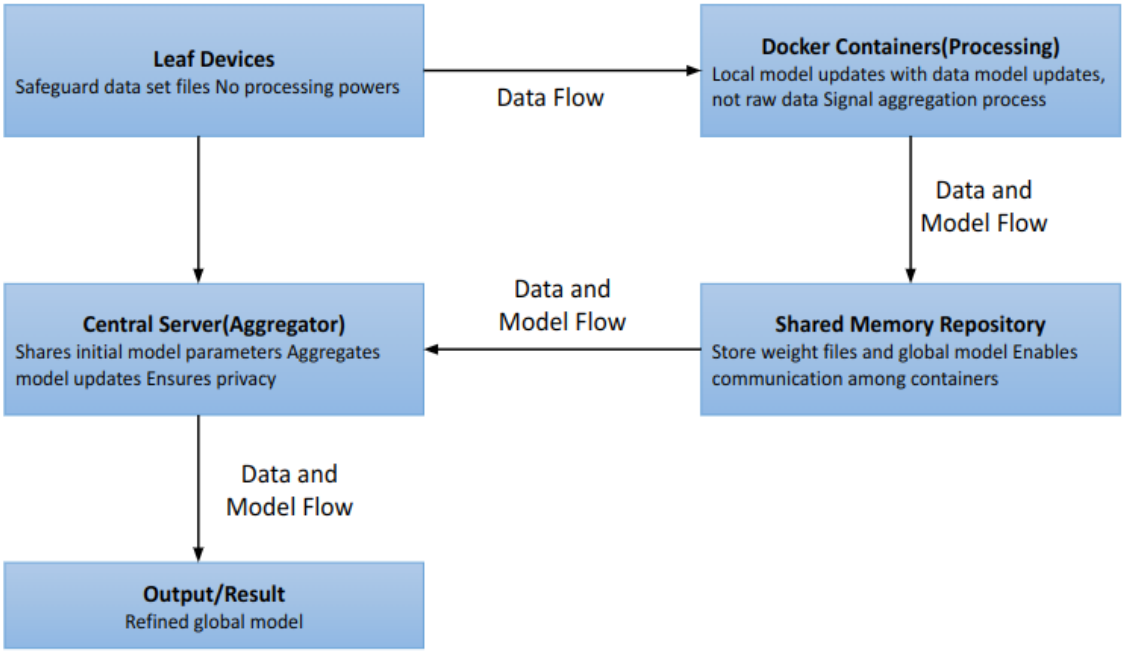


Figure 5.2: Flow diagram of federated learning system architecture with containers and centralized aggregation.

memory repository is tasked with storing weight files and the global model. Facilitating effective communication among the containers, this shared memory component acts as a central hub for data and model exchanges, ensuring seamless coordination and integration of the model updates.

- **Central Server (Aggregator):** The central server assumes the critical role of an aggregator within the system. It shares the initial model parameters and aggregates the model updates from the participating containers. Upholding the imperative of data privacy preservation, the central server ensures that it only receives model updates, not the raw data, thereby safeguarding sensitive information within the system.
- **Outputs:** It represents the culmination of the system's collaborative efforts. The refined global model, the outcome of the comprehensive aggregation and integration of the model updates, embodies the collective wisdom and expertise contributed by the participating entities, ultimately reflecting the successful exe-

cution of the system's data privacy measures and collaborative model training.

5.4 Model Workflow

Federated learning is an innovative machine-learning approach that champions privacy and decentralization while enabling multiple client devices to collaboratively contribute to the training of a shared global model. This progressive process unfolds through several pivotal steps, embodying the essence of privacy-preserving and distributed machine learning.

5.4.1 Model Initialization

The federated learning journey commences with the central server taking the initiative. It shares initial model parameters with all participating client devices. These initial parameters serve as the foundational blueprint for the global model, representing its initial state. Importantly, this stage marks the first instance of the central server's involvement in the model's development, as illustrated in Figure 5.3.

5.4.2 Local Model Training

Once equipped with the initial model parameters, each client device embarks on a journey of individualized model training. Leveraging its local dataset, the client device independently refines the model through training iterations. Crucially, this training process transpires entirely on the client's device, ensuring that sensitive and proprietary data remains securely decentralized and immune to external scrutiny. Upon completing its local training, the client device contributes to the collaborative process by sharing its locally updated model. It's important to emphasize that the client shares model updates derived from its data, never exposing raw, private information.

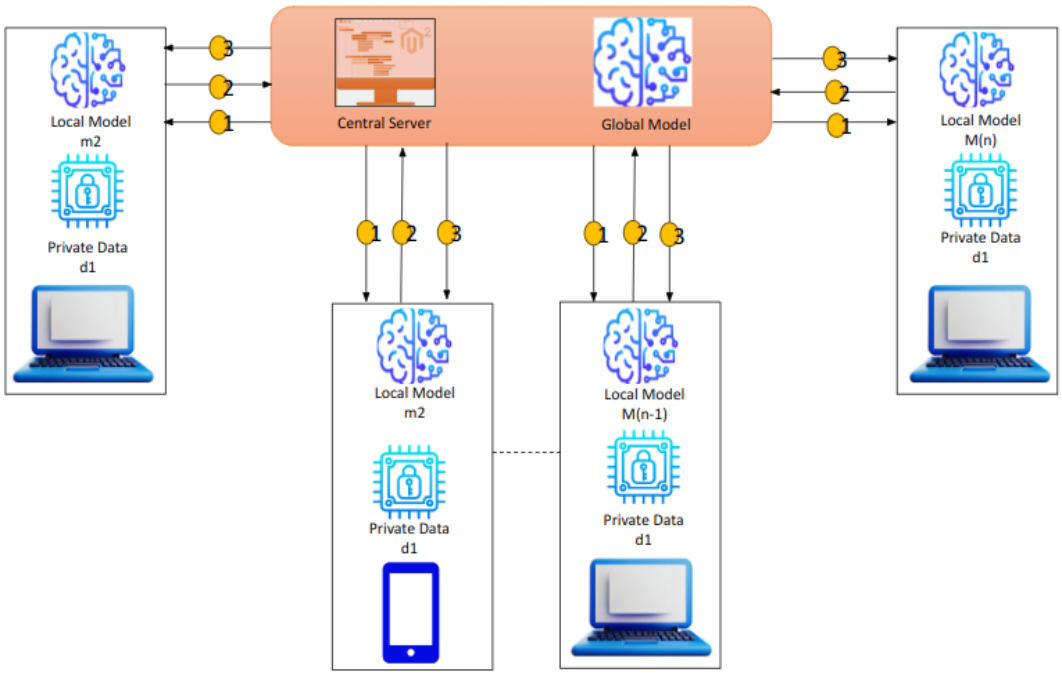


Figure 5.3: Federated learning workflow in this chapter.

5.4.3 Model Aggregation

The central server, now entrusted with a multitude of locally updated models from participating clients, undertakes the role of the model aggregator. Through a sophisticated aggregation process, it systematically combines and integrates the contributed model updates. The outcome of this aggregation is an enhanced global model, meticulously crafted from the collective wisdom and expertise of all participating clients. Notably, the central server only receives model updates and never gains access to the underlying raw data, thereby preserving the fundamental principles of data privacy and security. This iterative process of local model training, model update sharing, and global model aggregation repeats across multiple rounds. As these rounds progress, the global model undergoes continuous refinement and improvement, continually learning from the collective knowledge and diverse data sources represented by the participating clients. Federated learning, with its unique blend of collaboration, decentralization, and privacy protection, represents a potent approach ideally suited for a wide array

of applications across distributed and decentralized environments. In the below flow diagram, the process initiates with the generation of locally updated models, which are formulated independently by each participating device. These locally updated models are then transmitted to the central server, the core entity responsible for the aggregation of these individual model updates. Upon receiving the locally updated models, the central server functions as the model aggregator, orchestrating the systematic combination and integration of the received model updates. This process leads to the creation of an initial global model that encompasses the collective intelligence and knowledge embedded within the individual device updates. The updated global model is continuously refined and improved, demonstrating a commitment to the perpetual enhancement of the system's learning capabilities. This continuous refinement and improvement emphasize the iterative learning and integration process, ensuring that the system remains adaptive and responsive to evolving data trends and patterns.

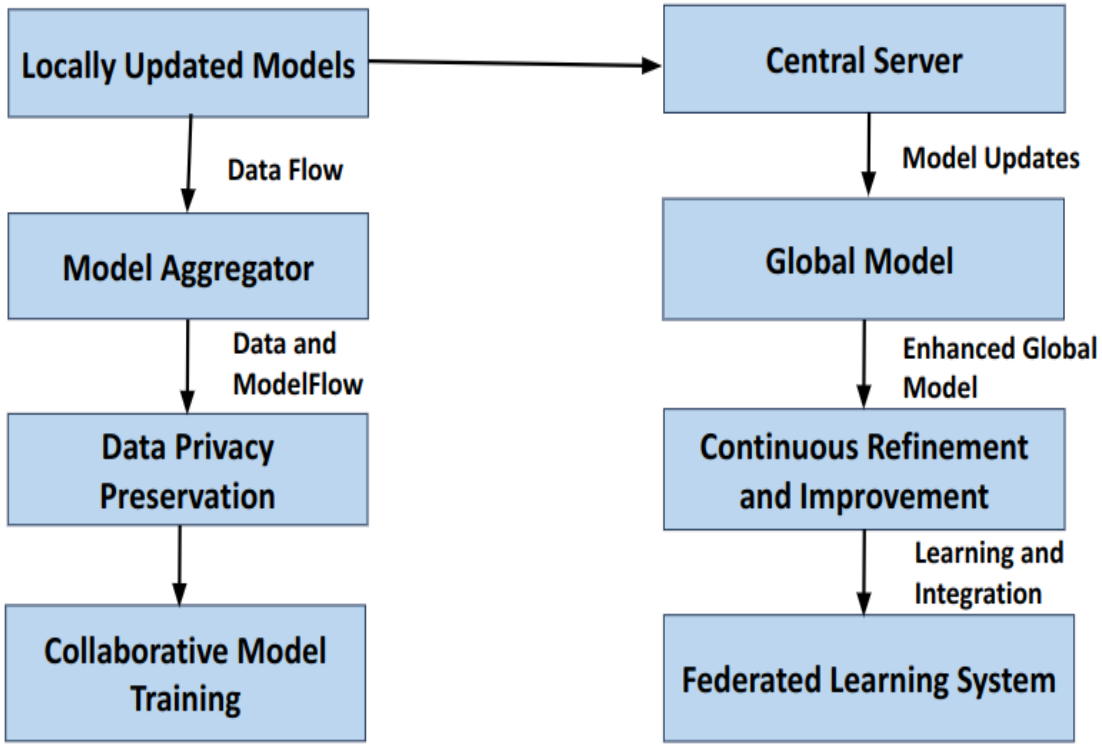


Figure 5.4: Illustration of end-to-end workflow of the proposed approach.

Figure 5.4 depicts the end-to-end workflow of the proposed federated learning based approach. The system's dedication to data privacy preservation is represented in the flow, underscoring the paramount importance of maintaining data privacy while facilitating collaborative model training. The collaborative model training process is a testament to the system's commitment to fostering a federated learning environment that encourages active participation and contribution from all involved devices, promoting a shared learning experience and a cohesive, collective model development approach.

5.5 Methodology

Introducing vital variables and equations crucial to our research that optimizes the framework underlying our architectural design. Let us define the key sets and variables as follows: C represents the set of available containers, M symbolizes the set of available shared memory locations, and P stands for the set of performance metrics.

The objective function can be defined as:

$$\max f(x) = \sum w(p) \cdot \text{performance}(p, x)$$

where x is a binary variable vector representing the selection of containers and their placement in shared memory, $w(p)$ is a weight assigned to each performance metric, and performance(p, x) is a function that calculates the performance of the system based on the selected containers and their placement.

- $C = \{c1, c2, ..., cn\}$ be the set of available containers.
- $M = \{m1, m2, ..., mk\}$ be the set of available shared memory locations.
- $P = \{p1, p2, ..., pm\}$ be the set of performance metrics.
- $x = [x1, x2, ..., xn]$ be a binary variable vector representing the selection of containers, where $xi = 1$ if container ci is selected, and $xi = 0$ otherwise.
- $y = [y_{ij}]$ be a binary variable matrix representing the placement of container ci

in shared memory mj, where $y_{ij} = 1$ if container ci is placed in shared memory mj, and $y_{ij} = 0$ otherwise.

The objective function can be formulated as:

$$\max f(x, y) = \sum_{p \in P} \sum_{i=1}^{n} \sum_{j=1}^{k} w(p) \cdot \text{performance}(p, c_i, m_j) \cdot y_{ij} \cdot x_i$$

where $\text{performance}(p, c_i, m_j)$ is a function that calculates the performance of container c_i in shared memory m_j for the performance metric p.

Constraints

1. Data privacy constraint:

$$\sum_{j=1}^{k} y_{ij} \cdot y_{jj} \leq 0, \text{ for all } i, j \in \{1, 2, ..., n\}$$

2. Shared memory capacity constraint:

$$\sum_{i=1}^{n} \text{size}(c_i) \cdot y_{ij} \leq \text{capacity}(m_j), \text{ for all } j \in \{1, 2, ..., k\}$$

3. Performance constraints:

$$\text{performance}(p, c_i, m_j) \leq \text{threshold}(p), \text{ for all } p \in P, i \in \{1, 2, ..., n\}, \text{ and } j \in \{1, 2, ..., k\}$$

4. Binary variable constraints:

$$x_i \in \{0, 1\}, \text{ for all } i \in \{1, 2, ..., n\}$$

$$y_{ij} \in \{0, 1\}, \text{ for all } i \in \{1, 2, ..., n\} \text{ and } j \in \{1, 2, ..., k\}$$

To derive optimal values for the binary variable vector x and the binary variable matrix y within the specified constraints, we employ linear programming techniques. This

computational approach enables us to systematically address the complex optimization problem that forms the cornerstone of our research, thereby facilitating efficient and effective containerization in the context of federated learning.

In the below flow diagram in Figure 5.5, it is at the outset of the process, that individual devices are responsible for storing their respective local data securely. Each device conducts its own local model updates based on its stored data, ensuring that the sensitive information remains decentralized and isolated within the device itself.

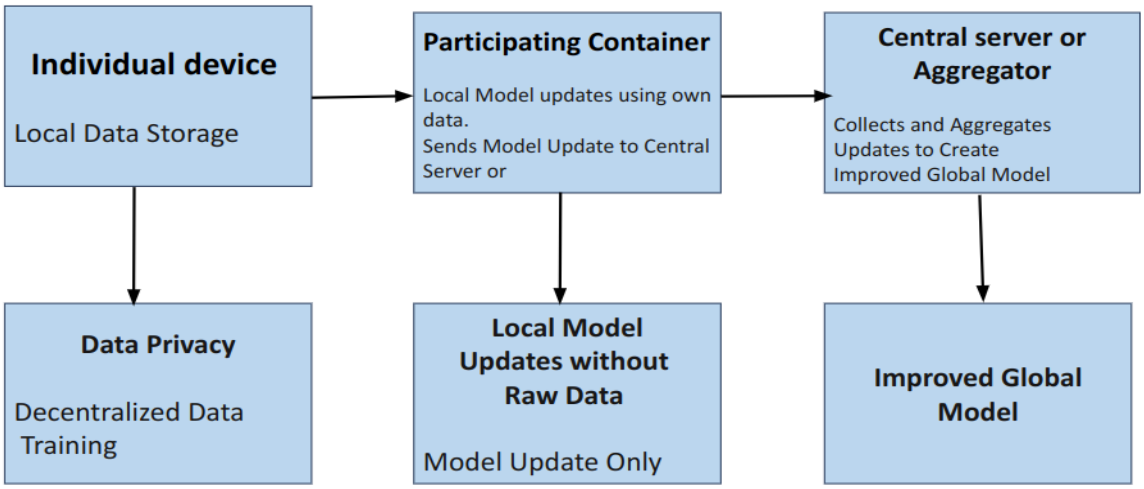


Figure 5.5: Flow diagram of data privacy maintenance in the decentralized federated learning system.

Simultaneously, the participating containers receive these locally updated models from the individual devices. These containers execute their tasks by implementing the received model updates using their own data. They refrain from accessing the raw data of the individual devices, adhering to the principle of data privacy. The containers then transmit the model updates to the central server or aggregator, emphasizing the transmission of the processed updates rather than the raw data. The central server or aggregator plays a pivotal role in the process by collecting and aggregating these model updates received from the participating containers. This aggregation process entails the systematic combination and integration of the individual model updates, ultimately culminating in the creation of an improved global model. It is important to

note that the central server or aggregator does not gain access to the underlying raw data, thus preserving the fundamental principles of data privacy and security. The flow of information and model updates within the system reflects the careful consideration and implementation of privacy-preserving measures, ensuring that data remains decentralized and that only processed model updates are transmitted for aggregation. This approach guarantees that the system operates efficiently while upholding the crucial tenets of data privacy and security.

5.6 Containerized federated learning approach

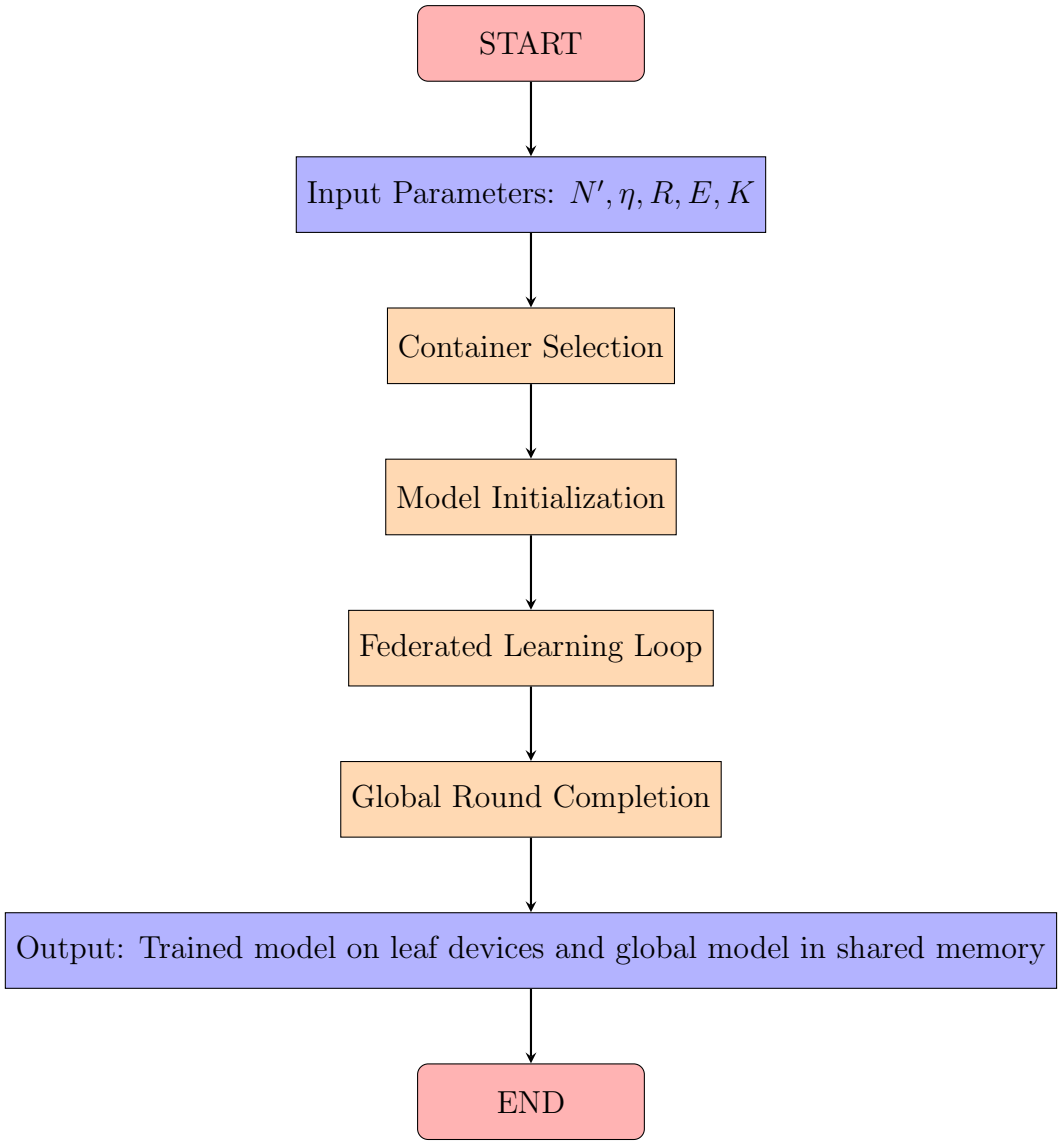
The approach consists of the following parameters as inputs: the number of available containers (N'), learning rate (η), communication rounds (R), epochs (E), and the count of leaf devices (K). The objective is to train a model collaboratively on all leaf devices while storing a global model in shared memory. The process unfolds as follows:

1. Container Selection: We optimally select N containers from the available N', adhering to the graph structure using shared memory S.
2. Model Initialization: The selected containers broadcast a randomly initialized model (M) to all their associated leaf devices.
3. Federated Learning Loop: This loop iterates over communication rounds ($t \leq R$), and for each round:
    - Container Iteration: We consider each container C_j among the selected N containers.
    - Leaf Device Selection: We choose K leaf devices from those available on container C_j (typically set to 5 in our work).
    - Local Training: Each leaf device on C_j trains the model M on its local data samples for E epochs and sends the resulting weight matrix to its corresponding container C_j.
    - Aggregation on Container C_j: Container C_j performs weight aggregation.

- Shared Memory Update: Container C_j also transfers the aggregated weight to the shared memory for global model training and aggregation.

4. Global Aggregation: After local training on all the containers, the received weights are aggregated in the shared memory.
5. Global Round Completion: With the above steps, one global round is completed.

The output of the algorithm includes the trained model on all the leaf devices within each container and the global model residing in the shared memory. This approach enables effective collaboration among containers and leaf devices, maintaining data privacy, and achieving the convergence of a global model.



5.7 Experimental Setup

This section discusses the experimental setup. The experimental setup in this work is similar as Chapter 3. We utilized the Docker swarm as the container manager engine and implemented the approach in Python 3.7.3. Each container is deployed on a machine allocated with limited CPU resources. The simulation is performed on the HP Prodesk desktop with 8-core 3.2 GHz CPU, 24 GB RAM with the installed operating system Linux (Ubuntu 20.04). We followed the steps outlined below to set up the experimental environment. Such an environment makes a shared memory architecture for optimized container selection.

We initiated the process by consolidating all the files in a designated drive, which was subsequently shared with other participants for container creation. In this step, we configured the permissions to execute mode, enabling parallel and independent execution of files, achieved through the 'chmod 777 filename.sh' command. The folder was shared to construct the graph, and commands such as 'sudo python3 filename.py' were executed to create a shared folder and generate a container structure graph. The shared devices were established, facilitating data sharing among them. Docker images were built, and containers were executed using the run command. Additionally, we performed calculations for weight averaging, federated module averaging, and nearby volume checks. This process ensures the execution of a federated learning system with communication between containers, model training, aggregation, and termination conditions.

5.8 Results and discussion

Overlapping region's Data Privacy and Shared Memory Constraints ensure secure shared memory storage in alignment with data privacy. Docker-based memory management efficiently reduces memory usage with increasing tasks. In contrast, non-Docker exhibits

fluctuating memory use, emphasizing Docker's structured memory sharing. Docker enhances efficient memory utilization, while non-Docker shows less consistent growth.

5.8.1 Visualization of Data Privacy Constraints in a System using Logic Diagrams

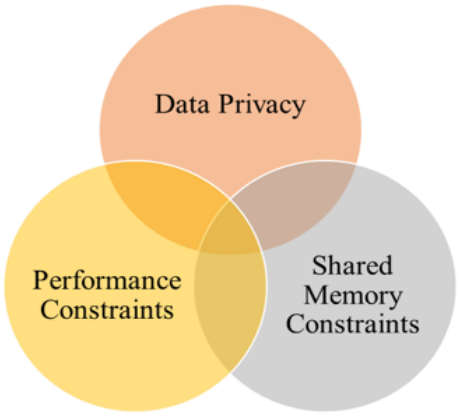


Figure 5.6: Logic diagram of the proposed approach.

Data Privacy and Shared Memory Constraints: The overlap between the "Data Privacy" and "Shared Memory Constraints" regions signifies the integration of measures to ensure that data stored in shared memory remains private and secure, as shown in Figure 5.6. This connection implies that the constraints placed on shared memory usage are designed to align with the overarching goal of maintaining data privacy. This integration allows for the efficient and secure storage of sensitive information within the shared memory infrastructure. Data Privacy and Performance Constraints: The connection between "Data Privacy" and "Performance Constraints" implies that the maintenance of stringent data privacy measures is crucial for the system's optimal performance. By ensuring that data privacy protocols are in place, the system can operate smoothly without the risk of privacy breaches or data leaks, thereby meeting the performance criteria set for the system. Shared Memory Constraints and Performance Constraints: The overlap between "Shared Memory Constraints" and "Performance

Constraints" indicates that the guidelines for managing shared memory resources are directly linked to achieving optimal system performance. Efficient allocation and utilization of shared memory resources contribute to enhancing the overall performance of the system, as it allows for streamlined data access and processing, leading to improved efficiency and speed of operations. In essence, these interrelationships highlight the integrated approach taken in the system design, where data privacy, efficient shared memory utilization, and optimal system performance are all interconnected and dependent on each other for the successful operation of the federated learning framework.

5.8.2 Impact of Number of Tasks on Memory for Loading OS

Our research highlights the potential advantages of the shared memory architecture in optimized container selection. We analyzed the impact of the number of tasks on memory for loading the operating system (OS). Our results demonstrate that within this framework, the Docker-based approach displays an upward linear trend, as shown in Figure 5.7. The Docker environment enables efficient memory sharing and management, leading to a consistent improvement in memory, requiring less memory for loading the OS as the number of tasks increases. This predictable performance increase underscores the importance of containerization and shared memory architecture in optimizing container selection.

Conversely, when assessing the non-Docker approach, we observed an inclining trend with fluctuations. The memory for loading the OS exhibited variance with increasing task numbers, and the overall improvement, while present, was less pronounced compared to the Docker-based approach. This fluctuation in the non-Docker approach underscores the value of structured memory sharing in the shared memory architecture, which contributes to the reliability and predictability of memory utilization.

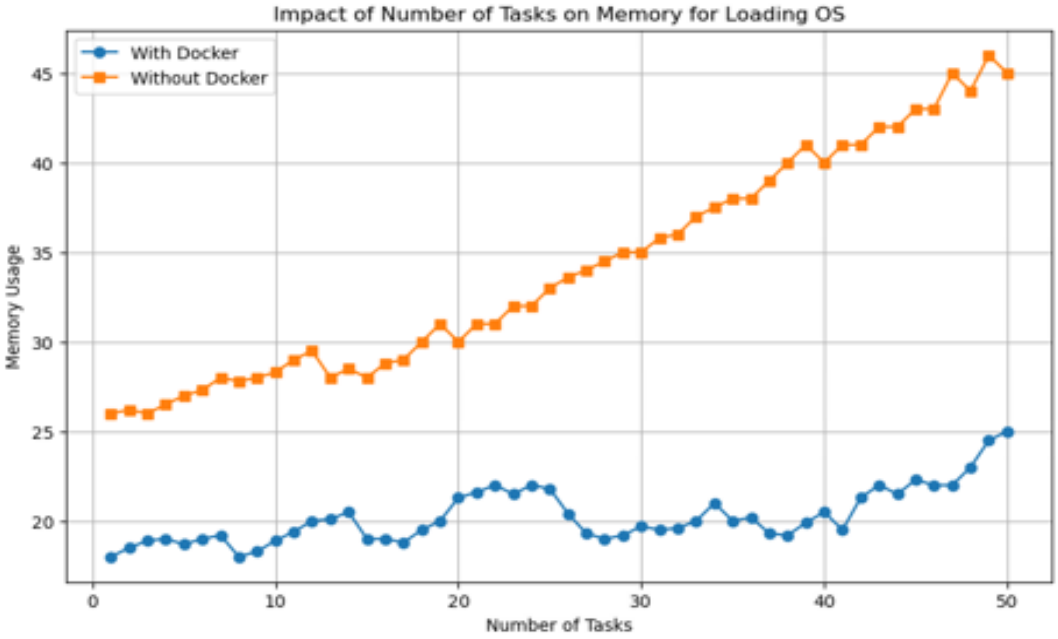


Figure 5.7: Impact of number of tasks on memory for loading operating system.

5.8.3 Impact of Number of Tasks on Memory Utilization

Our study further demonstrates the merits of the shared memory architecture in optimized container selection, especially in the context of memory utilization. In this regard, the Docker-based approach exhibits improvement that is by utilizing less memory with increasing task numbers. This behaviour can be attributed to the efficient memory sharing and management enabled by Docker containers. Conversely, the non-Docker approach, while showing memory utilization, exhibits less pronounced growth. The memory utilization in the non-Docker approach experiences a noticeable increase, indicating potential memory utilization, as shown in Figure 5.8. This inconsistency in memory utilization with non-Docker highlights the advantages of adopting a Docker-based federated learning approach for memory optimization and predictable memory management. The results are visually represented in the accompanying graph.

The graph portrays a compelling trend in validation accuracy as the number of containers increases. Each data point on the graph corresponds to a specific case defined

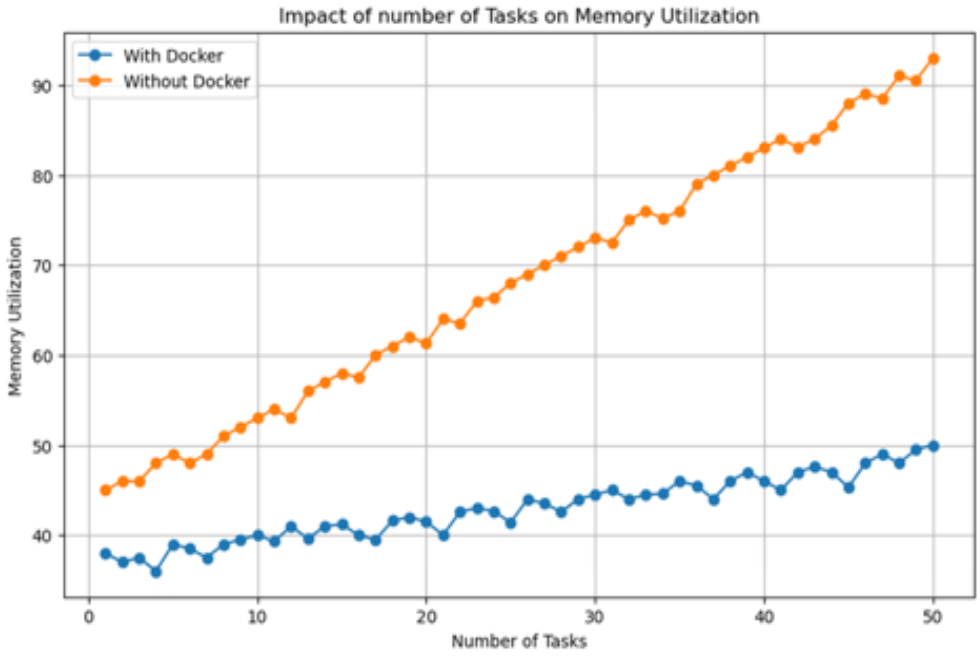


Figure 5.8: Impact of the number of tasks on memory utilization.

by the number of containers, showing a consistent improvement in validation accuracy with an increase in the epoch count. This positive trend persists throughout the experiment, culminating in near-perfect accuracy with 50 containers. The experimental findings suggest that increasing the number of containers has a beneficial impact on the federated learning process. This means that scalability can be achieved without compromising model performance, which holds promise for large-scale federated learning applications, as depicted in Figure 5.9.

These results underscore the significance of containerization technologies, such as Docker, in maintaining compatibility and consistency across diverse platforms. The positive correlation between container count and validation accuracy encourages further exploration of strategies to optimize federated learning setups for improved scalability. It becomes evident that a nuanced approach to scalability is crucial in real-world applications, where both performance and scalability are paramount. The results of our research work focus on “Optimized Container Selection Using Shared Memory Archi-

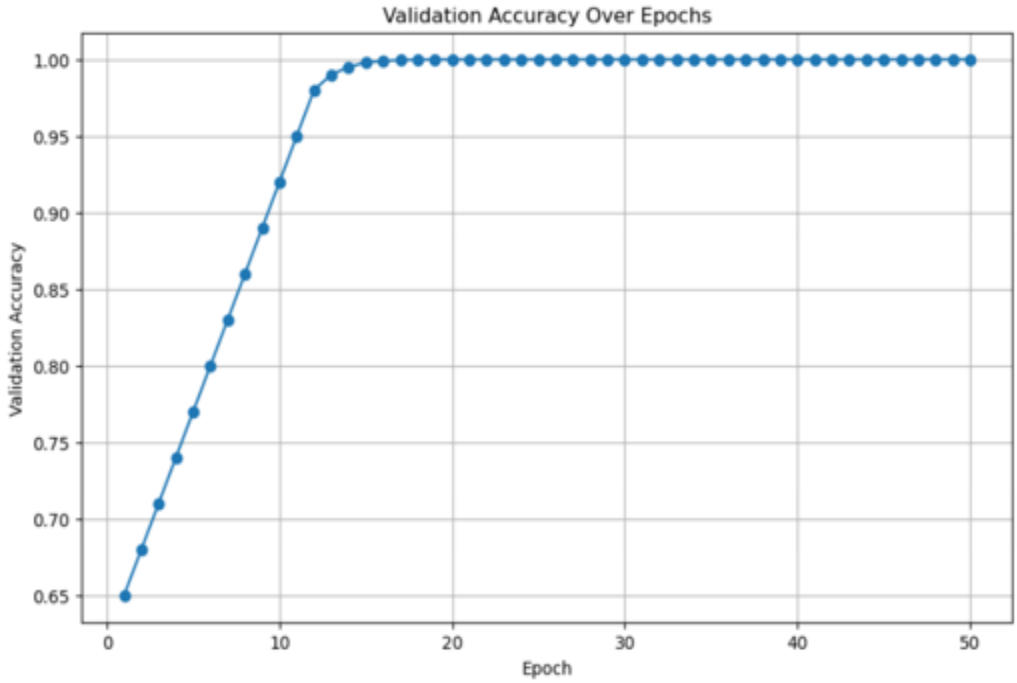


Figure 5.9: Validation accuracy *vs.* epochs graph.

tecture." Our investigation aims to provide insights into the performance and efficiency of this containerized learning framework. We specifically analyze the impact of the number of tasks on memory for loading the operating system and memory utilization to evaluate the effectiveness of our approach.

5.9 Conclusion and discussion

In conclusion, our research underscores the significance of "Optimized Container Selection Using Shared Memory Architecture." The shared memory architecture enhances memory for loading the OS and memory utilization, demonstrating the practical advantages of containerization and structured memory sharing. This approach offers improved performance, resource optimization, and scalability, making it a promising solution for privacy-aware and efficient machine-learning applications in decentralized settings. The results reaffirm the innovation and potential of our containerized learning framework. These findings contribute to the growing body of knowledge in the field

of optimized container selection and further emphasize the relevance of shared memory architecture in achieving superior containerized learning solutions. In this study, we investigated how varying the number of containers impacts the federated learning process. We explored the interplay between the number of containers, training epochs, and validation accuracy.