5-2024

Heterogeneous Federated Learning at Scale

Dmitry Lukyanov
dlukyan@clemson.edu

HETEROGENEOUS FEDERATED LEARNING AT SCALE

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science.
Computer Science

---

by
Dmitry Lukyanov
May 2024

Accepted by:
Carlos Toxtli Hernandez, Committee Chair
Rong Re
Nina Hubig
Mitch Shue

ABSTRACT

Federated learning has emerged as a solution to the challenges faced by traditional centralized machine learning approaches, such as data privacy, security, ownership, and computational bottlenecks. However, federated learning itself introduced new challenges, including system heterogeneity and scalability. Existing federated learning approaches, such as hierarchical and heterogeneous federated learning, address some of these challenges but have limitations in real-world scenarios where multiple issues coexist, particularly in large-scale, heterogeneous environments like mobile applications and IoT devices. This work proposes a new federated learning architecture that combines heterogeneous federated learning and hierarchical federated learning into a unified architecture. The proposed approach aims to address the limitations of existing architectures by building clusters of models based on their types and usage of in-cluster models' weights averaging to create an ensemble of heterogeneous models for further knowledge distillation into student models of different types that are to be distributed into respective clusters to continue training. The created implementation of the proposed architecture showed accuracy comparable with accuracy of FedDF chosen as a baseline heterogeneous federated learning architecture within the same environment with slight advantage in convergence speed in some cases.

ACKNOWLEDGMENTS

I wish to express my gratitude to those whose support was critical in the completion of this thesis. Firstly, I am thankful to my thesis advisor, Dr. Carlos Toxtli Hernandez, for his guidance and support throughout this endeavor. Also, I would like to express my gratitude to my committee members, Dr. Rong Ge, Dr. Nina Hubig and Prof. Mitch Shue, for their valuable input.

Finally, I am grateful to the members of the faculty whose classes I have taken for their contributions to my academic growth and development. Their knowledge and expertise have enriched my understanding of the subject matter.

I would especially like to thank Prof. Carrie Russel for her support and advice during my journey through the program.

TABLE OF CONTENTS

Table of Contents (Continued) Page

LIST OF TABLES

LIST OF FIGURES

LIST OF LISTINGS

CHAPTER ONE

INTRODUCTION

Background and motivation

Machine learning has made significant advancements in recent years in many domains such as computer vision and natural language processing. However, traditional centralized machine learning approaches when all the data is gathered in one storage and used to train one or several models face challenges related to data privacy, security, data ownership, and computational bottlenecks. There are multiple scenarios where data cannot be placed into one silo due to legal or business reasons, or due to physical limitations of communication channels or storage itself. E.g., in many countries, healthcare institutions have quite a limited ability, if any, to share patients' data that significantly affects the ability to use that data for training machine learning models although it could improve the healthcare services quality and patients experience.

Federated learning emerged as a solution to these limitations, enabling collaborative learning across distributed clients while keeping data localized. In a typical federated learning setup, a central server coordinates the training process by aggregating local model updates from participating clients. However, while in theory federated learning provides an ideal solution for distributed settings, in practice it collided with reality and introduced new challenges, such as system heterogeneity as computational resources vary significantly among clients, non-IID data as the data in real scenarios typically is not distributed perfectly equally across clients, communication efficiency as it is required to

take into account how much and how often model data should be transferred among clients and the server, privacy leakage as in some cases it is possible to restore some training data from just changes of trained models, adversarial attacks as even small percentage of hostile clients in a system can poison the training process and models, incentive mechanisms as the contribution among clients can vary significantly, and scalability as in some scenarios the number of clients can be quite large. Thus, with growing connectivity of the world, increasing with each day amount of data, dynamically changing laws in the sphere of artificial intelligence and variety in data handling practices in different countries and companies federated machine learning and consequently its challenges are becoming a more and more important topic in the field of artificial intelligence.

To address these challenges, various federated learning architectures have been proposed. For example, hierarchical federated learning introduced multiple levels of aggregation to improve scalability and handle data heterogeneity, ensemble federated learning combined multiple models to enhance performance and robustness of the system, heterogeneous federated learning incorporated ensemble federated learning and knowledge distillation to support models with different architectures within one federated setup.

Problem definition

However, existing approaches have limitations in real-world scenarios where multiple challenges coexist. In domains like mobile applications or IoT devices, the number of model instances can reach millions or billions, data cannot be gathered in one place due to privacy issues and physical limitations, and models may have different architectures due to hardware heterogeneity or business requirements. In such scenarios

federated learning approaches with multiple clients connected directly to a central server struggle with scalability, while hierarchical approaches solving the scalability issue do not support heterogeneous models.

Consequently, there is a need for a federated learning architecture that combines the benefits of existing approaches while addressing their limitations. The desired architecture should allow different model architectures within a single federated setup, be highly scalable, and ensure coverage of all data during training to avoid situations where some features are underrepresented due to data being non-IID. By developing such an architecture, the aim is to enable effective and efficient federated learning in large-scale, heterogeneous environments.

Solution Approach

The approach chosen to address the challenge is to combine heterogeneous federated learning built with ensemble federated learning and knowledge distillation, and hierarchical federated learning into a unified architecture. Different types of models are trained on different subsets of data allowing to match models' complexity with clients' computational resources. Ensemble federated learning is used to combine different types of the models providing the ability to utilize all the heterogeneous models for inference. Knowledge distillation enables the transfer of knowledge from a heterogeneous ensemble acting as a teacher model to student models, allowing to create new models for each type using all collected knowledge from all types of the models. Hierarchical federated learning introduces multiple levels of aggregation, grouping clients into clusters based on the model types instead of spatial proximity or data features, and performing local aggregation before

sending updates to the central server. It improves scalability by reducing the communication burden on the central server and enables more efficient learning in large-scale federated networks.

Results

To validate the proposed approach, experiments were conducted using the CIFAR-10 dataset. The proposed architecture and several architectures (FedAvg, HierFAVG, and FedDF) used as baseline ones were implemented, trained, and evaluated based on accuracy and convergence speed. While the proposed architecture with heterogeneous models showed lower performance compared to FedAvg and HierFAVG with homogeneous models, its accuracy was comparable to FedDF with heterogeneous models, and in some cases, it converged faster than FedDF on a similar setup.

However, the conducted experiments have some limitations, including sparse model architecture and hyperparameter tuning, the relatively small scale of the emulated distributed system, limiting the application domains to image classification and the use of IID data only that should be addressed in future research along with employing less data-hungry models and evaluating the communication overhead in comparison to other architectures to investigate the potential advantages of the proposed architecture as the achieved during the experimentations results are only slightly and not always better than existing solutions.

CHAPTER TWO

BACKGROUND AND RELATED WORK

Machine Learning and Deep Learning

Machine learning is a field of study that develops algorithms and models to perform tasks by learning from data, instead of being explicitly programmed. Its ability to extract insights and discover patterns from data caused wide adoption across multiple domains. Machine learning techniques are typically split into three main approaches based on the nature of feedback available during the learning process.

Supervised Learning [1] fits a mapping function from input data to target output labels based on labeled training examples. It's applicable in the cases where historical or known data is used to make predictions on unseen data. Supervised learning is widely used in tasks such as email filtering [2], image classification [3], voice recognition [4] or diagnostics [5]. Common algorithms include linear/logistic regression [6][7], decision trees [8], kernel methods [9], ensemble techniques [10], and neural networks [11].

In contrast, Unsupervised Learning [12] is used for discovering patterns and clusters directly from unlabeled data without any supervision. It captures important features in the data and uncover underlying structure through methods like clustering [13], dimensionality reduction [14] and association analysis [15]. Unsupervised learning allows to extract information from raw data that is critical for exploratory analysis, clustering [13], anomaly detection [16] and extracting features for downstream tasks.

Another paradigm, Reinforcement Learning [17], is a paradigm where an agent learns to take optimal actions to maximize a reward signal by interacting with an environment. Unlike supervised learning's static datasets, reinforcement learning agents improve through trial-and-error, making it suitable for systems control and decision-making in dynamic environments. Reinforcement learning achieved significant results in game-playing [18], robotics [19] and recommendation systems [20] with techniques like Q-learning [21], policy gradients [22] and actor-critic methods [23].

Finally, Deep Learning [24], a subfield of machine learning inspired by the brain structure, brought advances into many domains. Deep neural networks, composed of multiple layers of simple units, learn features directly from raw data unlike traditional machine learning algorithms that perform better when the important features are found, extracted and preprocessed in advance. This allows to automate the feature engineering process and to improve performance on highly complex, real-world datasets. Deep learning achieved notable success in areas like computer vision with convolutional neural networks [25], natural language processing with transformer models [26], and generative modeling with variational autoencoders [27] and diffusion models [28]. However, key limitations of deep learning models include the need for large, labeled datasets, poor interpretability, adversarial vulnerabilities and encoding societal biases [29].

However, while the capabilities of machine learning systems have grown rapidly during the last decade, several challenges have to be addressed for real-world deployment.

One challenge is data bottlenecks. Many techniques are data-hungry and require labeled examples which can be costly and impractical to acquire. Data-centric approaches

like transfer learning, active learning, weak supervision, and self-supervision are promising directions.

Another one is interpretability. Complex models often lack transparency, making it difficult to understand the reasoning behind their outputs. It reduces trust, accountability, and adoption in high-stakes domains [32]. Explainable artificial intelligence approach aims to provide insights into these "black box" models.

Robustness is also a concern. Machine learning systems can be vulnerable to distribution shifts [33], adversarial attacks [34] and compounding errors [35]. Improving robustness is crucial for safe usage.

Another issue is that most models excel at capturing correlations but struggle with causal reasoning, limiting their ability to generalize or extrapolate.

A significant challenge is encoding social biases. Since machine learning models learn from data, they can often absorb and integrate societal biases like gender, racial or age discrimination that may be presented in the training datasets [36]. These encoded biases can lead to unfair and discriminatory outcomes when deployed in decision-making systems impacting people's lives. Careful data curation, bias measurement techniques and bias mitigation algorithms are needed to make models fairer and more inclusive.

Lastly, adversarial vulnerabilities. Deep neural networks have been shown to be susceptible to small, carefully crafted perturbations to inputs designed to cause misclassification. These adversarial examples expose blind spots and lack of robust reasoning in models. Adversarial training, defensive distillation and certifiable robustness are areas of research aiming to make models more secure against such attacks.

Federated Machine Learning

However, despite the undeniable advancements in machine learning, the traditional centralized approach has several challenges.

Data privacy is a significant concern in centralized training as it often requires storing all the data in one location. This raises issues of data privacy, especially when dealing with sensitive data like healthcare information or financial records [37]. Data breaches or unauthorized access can have severe consequences for individuals.

Closely related to privacy is the issue of data security. Centralized data storage also presents security vulnerabilities. Malicious actors may attempt to steal, modify, or corrupt the data, compromising the integrity of the trained model, and to do so they need to breach into only one location.

Moreover, data ownership and governance raise questions about control and access. Issues arise regarding who owns the data, who can access it, and how the benefits from having the data are distributed [38].

From a technical perspective, centralized learning can face computational bottlenecks. Training on massive datasets often requires significant computational resources. Centralized servers can become overloaded, leading to bottlenecks, and hindering the training process.

Lastly, centralized approaches can straggle with limited data availability. In some cases, data may be siloed across different organizations or individuals due to privacy regulations such as General Data Protection Regulation in the European Union or

ownership restrictions. Centralized approaches struggle to use this fragmented data for model training.

These challenges highlight the need for alternative approaches that prioritize data privacy, security, and ownership while enabling collaborative learning from distributed data sources. Federated learning [30] was created as a solution to address the limitations of centralized learning. It is a distributed machine learning paradigm where the model training occurs collaboratively across multiple devices or silos while keeping data localized. In a typical naive federated setup, the generalized workflow can be described with the next steps:

1. Global Model Distribution. A central server broadcasts a global model to all participating clients. This initial model can be pre-trained on a general dataset or a smaller set of aggregated data.
2. Local Model Training. Each client downloads the global model and trains it locally on its own private data. This training process updates the model weights based on the local data.
3. Model Update Aggregation. Clients upload only the trained model updates to the central server, not the data itself.
4. Global Model Update. The central server aggregates these model updates from multiple clients using various techniques. The aggregated update is then applied to the global model, improving its overall performance.

5. Iterative Training. This process of model distribution, local training, update aggregation, and global model update is repeated for multiple rounds, iteratively improving the model's performance.

While federated learning can be beneficial in some circumstances, it also brings new challenges.

One of them is system’s heterogeneity as in federated settings the participating devices can have significant system’s heterogeneity in terms of compute capabilities, available memory, network connectivity, energy constraints, and software environments. This system's heterogeneity needs to be taken into account during the architecture design. For example, embedded devices like sensors and IoT gadgets may be very constrained in the sense of computational power compared to cloud servers. Naive federated implementations could lead to stragglers and bottlenecks. Asynchronous and adaptive communication strategies [39], lossy compression techniques [40], and heterogeneity-aware optimization methods [41] may be required.

Another challenge in federated learning is dealing with non-independent and identically distributed (non-IID) data. A key premise of federated learning is using the distributed data across devices to achieve better generalization compared to models trained in isolation. However, the non-IID data distributions resulting from different device usage patterns and biases can lead to data heterogeneity. Naively aggregating model updates from such heterogeneous data can degrade performance or cause convergence issues [31].

Communication efficiency is also a critical consideration in federated learning due to the repeated exchange of models/updates between the server and devices. Efficient communication strategies [42] are critical, especially when dealing with bandwidth constraints and devices with limited connectivity like mobile phones. Methods like gradient compression [43], model quantization [40] and more intelligent update scheduling [41] can help reduce the overall communication overhead.

Privacy leakage is another concern in federated environments as while the core idea of federated learning is to avoid direct sharing of data, care must be taken to prevent information leakage through the parameter updates exchanged during training [44]. Approaches like differential privacy [45] can inject the right amount of noise to model updates to achieve strong privacy guarantees. However, these privacy-preserving methods can impact model performance and add computational overhead, requiring careful analysis of the privacy-accuracy trade-offs.

Federated learning also faces challenges related to adversarial attacks. In open federated settings, there are concerns around the reliability, security and integrity of the end devices participating in the training process. Adversaries could inject malicious updates to degrade the global model, honest devices could drop out due to failures or low connectivity, and hardware faults could lead to corrupted updates [46]. Federated solutions need methods to detect and filter out such poisoned updates, handle device failures gracefully, and provide resilience against a range of attacks like model poisoning, backdoor insertion, and reconstruction attacks.

Incentive mechanisms is another important consideration for federated learning across independent entities like organizations or individuals as having effective incentive structures and monetization models is important to inspire participation [47]. There are concerns that participants with more resources could disproportionately impact the global model. Reputation-based updates weighting, proof-of-stake mechanisms, along with pricing schemes and contractual obligations can influence desired participant behavior.

Lastly, scalability remains a significant challenge in federated learning. Many federated use cases like personalized content recommendations involve extremely large and dynamic populations of participating devices. Handling such a massive scale with potentially millions of unreliable nodes poses scalability challenges. Hierarchical architectures, sharding approaches [48], efficient sampling/sketching [49], and gossiping protocols [50] are some solutions being explored to enable scalable federated learning solutions.

In summary, while federated learning provides an elegant solution, there are multiple challenges that need to be taken into account for real-world deployments.

Hierarchical Federated Learning

As it was mentioned earlier, among the key challenges in federated learning settings there are, especially at large scale with numerous client devices, the communication bottleneck caused by the models' parameters transfer between the central server and client nodes, latency in communications between client nodes and the central server that emerges due to geographical distribution of nodes and aggregation process bottleneck that can be

caused by storage I/O limitations on the central server. To address these limitations and improve scalability, hierarchical federated architectures have been proposed.

In a hierarchical federated setup [51], there are at least two levels of parameter aggregation instead of a single central server. Client nodes are clustered into groups with each group having an intermediate server that first aggregates model updates from its local cluster. The intermediate model updates are then passed up to a global server for final aggregation.

For hierarchical federated learning, the generalized workflow can be described as follows:

1. Global Model Distribution. The global server initializes and broadcasts the global model to a set of intermediate cluster servers/coordinators. These cluster servers can be geo-distributed edge servers or selected participants with higher compute capabilities.
2. Cluster-level Model Distribution. Each cluster server then distributes the global model to the participating client nodes within its local cluster.
3. Local Model Training. Similar to traditional federated learning, each client device trains the model locally using its private data.
4. Cluster-level Aggregation. Clients upload their locally trained model updates to their respective cluster servers. The cluster servers then aggregate these updates from clients within their cluster, using various techniques.
5. Cluster-to-Global Aggregation. The intermediate cluster updates are then sent back to the global server for another round of aggregation across all clusters. Various

algorithms can be employed here as well, accounting for cluster weights, reputations, or data characteristics.

6. Global Model Update. The global server applies the aggregated cluster updates to the global model.
7. Iterative Training. Steps 1 through 6 are repeated for multiple communication rounds until the global model converges or other criteria are met.

Beyond the initial communication efficiency motivation, hierarchical architectures have also been explored for handling additional challenges.

One such challenge is handling data heterogeneity. Intelligent clustering strategies can group statistically similar client data distributions together. The intermediate aggregations help mitigate client drift caused by heterogeneous data distributions - often a key challenge in cross-silo federated settings.

Other advantages of hierarchical architectures are the possibility to implement asynchronous training updates relatively easily [52] and tolerance to node faults or dropouts. The global server can proceed with aggregation from available intermediate clusters without stalling with fewer issues than synchronous centralized approaches.

Recent works [53][54][55] have explored hierarchical federated learning architectures for on-device intelligence and continual learning applications on edge devices like mobile phones and IoT nodes. Intermediate model aggregation at edge servers can help combine real-time learned updates from fast model evolution at the edge with periodic consolidation and updating of global cloud models.

However, while Hierarchical Federated Learning offers better scalability than naive Federated Learning architectures, it still keeps some issues and introduces new challenges.

One challenge is determining the optimal cluster configuration [56][57] which involves factors like data similarity, communication costs, privacy constraints, and fairness. Sub-optimal clustering can lead to poor performance and inefficiency.

Another challenge is intra-cluster concept drift, where the intermediate model aggregations within clusters can cause the cluster-level model to diverge from the global model over time [58]. Finding a balance between frequent global aggregations and preventing intra-cluster representation drift can be a dynamic trade-off.

Privacy and security are also concerns in Hierarchical Federated Learning as the multi-level aggregation exposes more opportunities for potential privacy leakage compared to traditional federated learning. Rigorous privacy-preserving and security mechanisms are required to protect against attacks across the hierarchy.

Additionally, in cross-silo settings, designing effective incentive mechanisms that ensure fair participation, honest contributions, and fair reward sharing across the cluster hierarchy can be more challenging, especially in the presence of dynamic clustering.

Communication-efficiency trade-offs also need to be considered as while Hierarchical Federated Learning can help to reduce overall communication cost, it introduces additional overhead for intra-cluster and inter-cluster communication. Efficient communication strategies and trade-offs between model compression, update frequency, and convergence need to be analyzed for each individual case.

Cluster heterogeneity and fairness are also important considerations, as heterogeneity in cluster sizes, capabilities, and data distributions can lead to fairness concerns and disproportionate influence on the global model [59]. Techniques for fair resource allocation, weighted aggregation, and ensuring equitable contributions are important considerations.

Lastly, Hierarchical Federated Learning primarily focuses on aggregating local updates to a shared global model, assuming all clients train compatible models. However, in real-world scenarios, clients may have distinct data characteristics, system's constraints, or application requirements demanding diverse model architectures or hyperparameters. Enforcing strict global model consistency across all levels of the hierarchy can lead to suboptimal performance for clients with unique model requirements and hinder the ability to use specialized local models.

In summary, Hierarchical Federated Learning offers a promising approach to address scalability challenges in large-scale federated learning settings. By introducing multiple levels of aggregation and clustering strategies, it can help mitigate issues like client drift and fault tolerance. However, it also introduces new challenges related to optimal cluster configuration, intra-cluster concept drift, privacy risks, incentive misalignments, communication-efficiency trade-offs, cluster heterogeneity, fairness, and model heterogeneity. Addressing these challenges requires careful design considerations and novel techniques that balance the benefits of hierarchical architectures with the unique requirements and constraints of specific federated learning scenarios.

Ensemble Federated Learning

Still, one of key challenges of hierarchical federated learning is inefficiency with a heterogeneous system. To address the issue, several other concepts should be considered, and the first one is ensemble learning [60] that combines multiple base models to improve prediction accuracy and robustness compared to using a single model. The key idea behind ensemble learning is that the combination of diverse models can lead to better generalization performance by leveraging the strengths of individual models and mitigating their weaknesses. Ensemble learning has been widely used in various domains and has achieved state-of-the-art performance in many machine learning tasks. The success of ensemble learning can be attributed to its ability to reduce overfitting, improve generalization, and handle complex data distributions, which is a combination that is critical for federated learning.

Ensemble federated learning [61] is an approach that combines the principles of federated learning and ensemble learning to improve the performance and robustness of federated learning systems. The main idea behind ensemble federated learning is to leverage the benefits of ensemble learning in the federated setting by training multiple diverse models on distributed data and combining their predictions.

There are several motivations for usage of ensemble techniques in federated learning.

First, ensemble learning can help improve the generalization performance of federated learning models by reducing overfitting and taking into account diverse patterns in the decentralized data [62].

Second, federated learning often involves non-IID (non-independently and identically distributed) data across clients, which can lead to performance degradation. Ensemble methods can mitigate the impact of data heterogeneity by combining models trained on different data distributions.

Third, federated learning systems are vulnerable to adversarial attacks, such as data poisoning or model update manipulations. Ensemble techniques can enhance the robustness of federated learning by reducing the influence of individual malicious clients or models.

In ensemble federated learning, a central server distributes ensemble models to clients, who train and update them locally using ensemble techniques. Clients send only model updates to the server, which aggregates them to improve the global ensemble while preserving privacy. This process is repeated iteratively until the desired performance is achieved, and the final ensemble is used for inference on new data. In a typical ensemble setup, the generalized workflow can be described with the next steps:

1. Global Model Distribution. A central server initializes several global models to form an ensemble. These models can be diverse in architecture, initialized randomly, or pre-trained on various datasets to ensure diversity. The server distributes the ensemble of models to all participating clients. Depending on the strategy, clients may receive all models or a subset to reduce computational and communication overhead.

2. Local Model Training. Each client receives one or more models from the ensemble. Locally, clients train each model on their private data. Post-training, clients generate updates for each model they trained.
3. Model Update Aggregation. Clients send their model updates back to the central server. The central server aggregates updates for each model separately.
4. Global Ensemble Update. The updated models are integrated into the global ensemble. Integration techniques can vary, including simple averaging, weighted voting, or meta-learning models that learn how to best combine the ensemble's outputs.
5. Iterative Training. The process of distributing the updated ensemble, local training on clients, updating aggregation, and global ensemble updating repeats for multiple rounds. With each iteration, the ensemble models are trained, improving the overall predictive performance and robustness against diverse data distributions.

While ensemble federated learning is showing good results, there are several challenges.

One challenge is ensemble weight optimization, particularly in the presence of data heterogeneity and client variability. Designing efficient and adaptive weight optimization methods that can handle the federated learning setting can be case-by-case and not a trivial problem.

Another challenge is data heterogeneity and non-IID data. Although ensemble federated learning can help mitigate the impact of data heterogeneity to some extent by

combining models trained on different local datasets, it does not completely solve the problem of non-IID data across clients. The performance of ensemble models can still be affected by significant differences in data distributions, and specialized techniques for handling non-IID data may be necessary.

Communication bottlenecks are also a concern in ensemble federated learning as the learning process can suffer from communication bottlenecks as the number of clients increases, due to the need to exchange model updates between the clients and the server. Ensemble federated learning does not inherently solve the scalability issue and can even worsen the situation due to multiple models needing to be trained on each client.

Another one is models' heterogeneity. While Ensemble Federated Learning in theory allows the use of heterogeneous models, heterogeneity is quite limited as it typically requires each client to train a subset of models.

Lastly, ensemble federated learning increases the workload for each client as each client should train a set of models. This increased workload can be problematic for clients without powerful hardware, limiting their ability to effectively participate in the federated learning process.

Knowledge Distillation

The second concept that is used to address the system heterogeneity problem is knowledge distillation [63] that is used for transferring the knowledge from a large and complex model (teacher) to a smaller and simpler model (student), without directly sharing the raw data between the models. The key idea is to train the student model to mimic the behavior or outputs of the teacher model, by minimizing the divergence between their

predictions, while also fitting the student model to the local data on each client in the case of federated setting. Knowledge distillation in federated learning is motivated by the fact that the global model learned by federated learning may be too large or complex to be deployed to resource-constrained devices, such as mobile phones, IoT sensors, or wearables. In addition to the deployment benefits, there are several other reasons why knowledge distillation is gaining traction in machine learning.

First, knowledge distillation can lead to improved performance for smaller models. By using the knowledge of a well-performing teacher model, even a smaller student model can achieve competitive accuracy on the target task [64].

Second, knowledge distillation can significantly reduce the training time and computational resources required for the student model compared to training it from scratch [65]. This is especially valuable when dealing with limited training data for the target task.

Third, knowledge distillation can potentially improve the robustness of the student model by helping it learn from the teacher's knowledge, which might encompass a wider range of patterns or scenarios compared to the limited target data [63].

The typical knowledge distillation workflow involves the following steps:

1. Teacher Model Pre-training. A complex model (teacher) is trained on a potentially large dataset relevant to the source task. This teacher model is assumed to capture rich and effective knowledge for the task.

2. Student Model Selection. A smaller and less complex model (student) is chosen for the target task. This model architecture might be specifically designed for deployment on resource-constrained devices.
3. Joint Loss Optimization. During the student model training process, two main loss functions are typically optimized:
    - Target Task Loss. This loss measures the student model's performance on the target task using labeled data.
    - Distillation Loss. This loss encourages the student model to mimic the behavior of the teacher model. This can be achieved by comparing the student's predictions with the teacher's predictions (soft targets).
4. Knowledge Transfer and Model Improvement. Through the combined optimization of both loss functions, the student model gradually learns not only from the target data but also from the distilled knowledge of the teacher model, leading to improved performance.

While knowledge distillation is a useful technique for transferring knowledge from a large and complex model to a smaller and simpler model, it also has several problems and limitations that need to be considered.

One of the main challenges of knowledge distillation is the capacity gap between the teacher model and the student model. If the student model is too small or simple compared to the teacher model, it may not be able to fully capture and retain the knowledge distilled from the teacher model, leading to a significant performance gap between the two

models [66]. On the other hand, if the student model is too large or complex, it may not provide significant advantages in terms of model compression, eliminating the purpose of knowledge distillation.

Another problem of knowledge distillation is the potential inconsistency between the knowledge distilled from the teacher model and the true knowledge required for the target task. This can happen when the teacher model is not well-suited for the target task, or when the distillation process introduces noise or bias into the transferred knowledge. In such cases, the student model may learn to mimic the behavior of the teacher model but fail to generalize well to new data or situations.

The success of knowledge distillation heavily depends on the quality and quantity of the data used for distillation. If the data is noisy, biased, or insufficient, the distilled knowledge may be inaccurate or incomplete, leading to poor performance of the student model. In federated learning scenarios, this problem can be aggravated by the data heterogeneity and imbalance across different clients, making it difficult to ensure consistent and reliable knowledge distillation.

Computational overhead is another consideration in knowledge distillation. While knowledge distillation can help to reduce the computational cost of large models, the distillation process itself can be computationally expensive, especially when dealing with complex models and large datasets.

Additionally, the performance of knowledge distillation can be sensitive to the choice of hyperparameters, such as the temperature scaling factor, the distillation loss weight, or the learning rate. Finding the optimal hyperparameters for a given task and

dataset can be challenging and time-consuming, especially in federated learning scenarios where the data and models are distributed across multiple clients.

Lastly, knowledge distillation can make the resulting student model less interpretable and explainable, as the distilled knowledge may not have a clear correspondence to the original features or concepts of the task. This can be problematic in applications where model interpretability is important, such as healthcare or finance, where the decisions made by the model need to be transparent and justifiable.

In summary, even with all its caveats knowledge distillation is a powerful technique for transferring knowledge from large and complex models to smaller and simpler models, which is particularly useful in federated learning scenarios where the global model may be too large or complex to be deployed on resource-constrained devices. However, to implement knowledge distillation it is required to define a loss function that would measure the difference in outputs between a teacher model and a student model.

Kullback-Leibler divergence

Kullback-Leibler (KL) divergence [67] is a measure that quantifies how much one probability distribution diverges from a reference probability distribution.

$$D_{KL}(P(x) \,||\, Q(x)) = \sum P(x) \ln(P(x) / Q(x))$$

It is widely used in statistics, data science and machine learning as it can be applied in anomaly and fraud detection, model and data monitoring, model selection, clustering,

and many other domains. However, there are several limitations and properties that should be taken into account:

- KL divergence is asymmetric, so in many cases $D_{KL}(P \parallel Q) \neq D_{KL}(Q \parallel P)$
- KL divergence is not a distance and cannot be used to define a metric space. Consequently, it cannot be used to measure how far apart distributions from each other
- KL divergence is non-negative, equals to zero only if $P(x) = Q(x)$ for each x and is undefined if $Q(x) = 0$

Due to its properties, Kullback-Leibler divergence was used for knowledge distillation as KL divergence allows to measure a difference in distributions between teacher model output logits and student model output logits. Therefore, KL divergence is utilized as a loss function with weighted inputs from a teacher and a student model forcing the student model to mimic the teacher up to some degree. This weighted approach also allows to have and balance different objectives for the teacher and the student models. Additionally, to provide the student model with richer information and transfer nuances in teacher's predictions revealing more information about the teacher's uncertainty and the relative likelihood of non-maximum classes, the distribution of those teacher model output logits can be smoothed by applying a temperature scaling factor.

In summary, KL divergence is a crucial component of knowledge distillation, where it quantifies and helps minimize the difference between the teacher and student models' predictions.

Heterogeneous Federated Learning

Federated learning has emerged as a promising approach for collaborative machine learning across distributed clients without compromising data privacy. By allowing the clients to train models locally on their own data and only share the model updates with a central server, federated learning can mitigate the risks of data breaches and privacy violations. However, naive federated learning frameworks assume that all clients have similar data distributions and model architectures. This assumption is often violated in practice due to the inherent diversity of the clients.

One aspect of this diversity is data heterogeneity, where clients may have different data distributions, such as different feature spaces, label spaces, or data volumes. This can lead to significant variations in the local models trained by each client, making it difficult to aggregate them into a global model that performs well for all clients.

Another aspect is model heterogeneity, where clients may have different computational resources and system constraints, such as different memory sizes, processing speeds, or energy budgets. This can lead to the need for different model architectures or hyperparameters for each client, making it challenging to maintain a consistent global model across all clients.

Lastly, objective heterogeneity can arise when clients may have different learning objectives or tasks, such as different target domains, cost functions, or fairness criteria. This can lead to conflicting or incompatible goals among the clients, making it hard to find a global model that satisfies all clients.

These heterogeneities can significantly degrade the performance and convergence of traditional federated learning algorithms, leading to suboptimal models, slow convergence, or even divergence. Moreover, they can also introduce fairness and robustness issues, where some clients may benefit more from the global model than others, or the global model may be biased towards certain clients or data distributions.

To address these challenges, the field of heterogeneous federated learning [68] has emerged, where a central server coordinates a learning process across clients with diverse computational resources, data distributions, and differing model architectures. The objective is to optimize a global model or a set of personalized models that perform well across all clients, accommodating the inherent diversity in the federated network. The typical workflow for heterogeneous federated learning can be described through the following steps.

1. Central Models Distribution. The central server prepares a central model, or a set of models tailored to the heterogeneity in the client devices and data. This could involve varying model architectures designed to be compatible with the computational capabilities of different clients. The server distributes the central model(-s) to all participating clients.
2. Local Model Training. Each client receives a model that best fits their local environment. Clients train the received model on their local datasets.
3. Model Update Aggregation. After local training, clients prepare and send their model updates back to the central server. Depending on the chosen approach, the server might distill the knowledge from an ensemble of the client models into the

central model and then distill it from the central model to the models with architectures that corresponds to the clients' model architectures or distill it directly from the ensemble of the client models into the models with architectures that corresponds to the clients' model architectures.

4. Iterative Training. The process of model distribution, local training, update contribution and server-side heterogeneity handling is repeated iteratively for multiple rounds.

While Heterogeneous Federated Learning combines advantages of Ensemble Federated Learning and Knowledge Distillation, it also inherits many of their problems, such as the need for ensemble weight optimization, a limited ability to handle data heterogeneity and non-IID distributions, teacher-student capacity gap, knowledge inconsistency, data quality and quantity for knowledge distillation, computational overhead, knowledge distillation hyperparameter sensitivity, and potential deterioration in model interpretability. In addition, Heterogeneous Federated Learning keeps the traditional for non-hierarchical potential scalability issue.

CHAPTER THREE

PROBLEM STATEMENT

As we can see, there are multiple approaches to federated machine learning addressing different issues. A naive federated learning approach solves the problem of model training on distributed data in a setting where it is not possible to access all the data from one place. Hierarchical federated learning allows to scale federated learning almost indefinitely by introducing multiple layers of intermediate aggregators. Ensemble federated learning lets us improve the performance by adding ensembles into the federated environment. And heterogeneous federated learning combines ensemble federated learning and knowledge distillation to create an architecture where it is possible to utilize models with different architectures in the same federated system.

However, while the mentioned approaches address separate issues, many real-world systems typically meet multiple issues at the same time. For example, it may be sufficient to use naive federated learning approach in the environment with several or even dozens of clients, but in the world of mobile and IoT devices this approach is becoming virtually unfeasible as we would need to aggregate billions of models on each aggregation round, and for that we need to communicate with billions of clients that is limited by the network throughput and store all the models in the central storage that is limited by its capacity and I/O. One of solutions for this problem is to utilize for aggregation only a random subset of the client models on each round, but taking into account the potential scale of the problem in many cases an acceptable subset would include just a fraction of a percent of all clients that can lead to slow convergence, a lower achievable performance

and underrepresentation of some data during the training. Hierarchical federated learning solves this problem, but by its nature it demands each single client model to have the same architecture. This demand can significantly affect models in an environment with huge differences in client devices' performance and make federated learning virtually impossible in some cases such as changing models' architecture in the mobile applications during application updates as those updates in most cases initiated by users and are not guaranteed that leads to the situation where different groups of devices can have dozens of different versions of the application installed. If different versions contain different model architectures, it makes the setup incompatible with hierarchical federated learning approach. This particular problem is solved by heterogeneous and ensemble federated learning, but as well as a naive federated learning approach, neither heterogeneous federated learning nor ensemble federated learning is capable of scaling without taking a hit on convergence speed and the maximum achievable performance.

Thus, we can clearly see the case where the existing approaches to federated learning are potentially suboptimal - domains such as mobile applications or IoT / network devices where the count goes into millions or billions of models' instances, the data cannot be shared, and models should have different architectures due to devices hardware heterogeneity, a business cycle or environment limitations. If we combine the approaches into one table considering this case, we will see it distinctly.

	Scalability	Heterogeneity
Naive FL	no	no

Hierarchical FL	yes	no
Ensemble FL	no	limited
Heterogeneous FL	no	yes

Table 1.1. Architectures feature comparison

Therefore, our task is to try to compose an architecture that would allow different models' architectures within one federated setup, be virtually indefinitely scalable and cover all the data during training to avoid the situation where some data is randomly underrepresented during the training.

CHAPTER FOUR

PROPOSED ARCHITECTURE AND BASELINE MODELS

As the main task of the new architecture is to resolve the conflict between heterogeneity and scalability in the existing approaches, three architectures were chosen as baseline ones.

FedAvg

The first baseline architecture, FedAvg introduced in 2017 [69], is the most fundamental and well-known approach in federated learning that serves as a commonly used benchmark for evaluating the effectiveness of more complex architectures in the sense of accuracy, convergence speed, and communication overhead. Despite its simplicity, FedAvg has achieved good performance in many applications and has been widely adopted. Comparing against FedAvg allows for a clear assessment of whether the new architecture brings significant improvements over this basic approach.

For this work a set of clients was created with appointing each client to a separate subset of the data. At each round a random subset of clients were trained on the respective data subsets, after which the trained clients were collected on a central server and averaged. The resulting model was spread among all clients for further training. The algorithm can be written as follows.

```
Server executes:
      initialize model M
      for each round do
            K <- (random subset of clients)
            for each client k in K do
                  M’ <- ClientUpdate(k)
            M <- (average weights across M’)

ClientUpdate(k):
      receive model Mk
      for each local epoch do
            M’ <- (train Mk on the local dataset)
      send M’ to server
```

Listing 1.1. FedAvg algorithm

HierFAVG

As one of two factors in the addressed issue is scalability, the second baseline architecture should represent hierarchical federated learning as it allows to scale a system virtually indefinitely. While there are a lot of variations in this approach that address different aspects such as decreasing latency across cellular networks by M.S.H. Abad et al. [70], improving performance on non-IID data via automated hierarchical clustering by Cristopher Briggs et al. [57], decreasing network workload with models quantization by Lumin Lui et al. [71] or with cluster formation in respect to data distribution by YongHeng Deng et al. [72], increasing the convergence speed via cluster formation based on the computational resources similarity and asynchronous communications by Zhiyuan Wang et al. [73], increasing privacy with adding random noise by Lu shi et al. [74], and many

others, the architecture that represents hierarchical federated learning in its purest form and introduced the concept per se is HierFAVG created in 2020 by Lumin Liu et al. [51]. As the goal is to compare the proposed architecture with baseline ones, HierFAVG was chosen as it reflects the fundamental principles of all hierarchical federated learning architectures and does not contain additional factors that can interfere with fairness of the comparison.

For this work the HierFAVG was implemented according to the description in the original paper with one difference - in the original paper clients were randomly assigned to edge-servers and correspondingly clusters at each round while in this baseline implementation they were assigned statically at the beginning of training and stayed within the same clusters. There are two reasons behind this decision. The first one is that typically one of factors for assigning is a spatial proximity to minimize the latency in communications within clusters, and on a truly large scale when clients are spread across countries, continents, or the globe random reassignment of clients among clusters will potentially increase the latency. The second reason is keeping the differences between baseline architectures and the proposed architecture minimal as the proposed architecture contains heterogeneous models that cannot be randomly joined into clusters due to their incompatibilities. Thus, a set of clients was created with appointing each client to a separate subset of data. All clients were assigned randomly to several clusters with an edge-server within each cluster and one cloud server. At each edge round, client models were trained on the local data and were collected by the cluster edge-server for averaging and spreading a produced model within cluster clients. After several edge rounds, edge-servers send

produced models to the cloud server for averaging and spreading back to edge-servers and consequently within respective clusters. The algorithm can be written as follows.

```
Server executes:
      initialize model M
      for each cloud round do
            for each edge server e in E do
                  for each edge round do
                        for each client k in K do
                              M' <- ClientUpdate(k)
                        M_e <- (average weights across all M' within e)
            M <- (average weights across all M_e)

ClientUpdate(k):
      receive model M_k
      for each local epoch do
            M' <- (train M_k on the local dataset)
      send M' to server
```

Listing 2.1. HierFAVG algorithm

FedDF

The second factor in the addressed issue is heterogeneity, and one of the first approaches to heterogeneity in federated learning is ensemble federated learning. However, ensemble federated learning implies each client training the full set of models and addresses not the differences in client devices' variability in available computational resources, but the accuracy of models' predictions due to ensembling. The first architecture

that truly focused on heterogeneity of models itself and incorporated knowledge distillation allowing adjustment of the system to variety in devices and bridge the gap between different models was FedDF introduced in 2020 by Tao Lin et al. [75]. Some other works were introduced addressing specific aspects of the architecture such as replacing knowledge distillation to small local models with knowledge distillation to one large model by Yae Jee Cho et al. [76] or reducing network workload via shifting knowledge distillation from a cloud server to local clients by Sohei Itihara et al. [77], but in the fundamental sense they were knowledge distillation-based variations of FedDF. FedMD [78] and Cronus [79], both are knowledge distillation-based architectures for heterogeneous federated learning and published before FedDF, were rejected as baseline candidates as they require knowledge distillation not only on a public dataset, ut also on all local ones that can lead to unpredictable consequences in performance and makes the results highly dependent on the local data. As the goal is to compare the proposed architecture with baseline ones, FedDF was chosen as it reflects the core principles of heterogeneous federated architecture and does not include additional factors that can affect the fairness of the comparison.

However, in the original work the authors mentioned the ability of the architecture to handle heterogeneous models, but virtually homogeneous setups only were tested and evaluated. For this work FedDF was implemented with heterogeneous models as the main purpose of it is to create a baseline for heterogeneity. At each round a random subset of clients was chosen in a way that the subset would include all types of the models the same number of times to provide an equal representation for better fairness in evaluating. As the next step, that subset was combined into an ensemble that was used to train new models -

one per type - with a separate dataset via knowledge distillation. The resulting models were spread across corresponding clients for further training. The algorithm can be written as follows.

```
Server executes:
      initialize models M of types T
      for each round do
            K <- (random subset of clients)
            for each client k in K do
                  M' <- ClientUpdate(k)
            for each model type t in T do
                  M_t <- KnowledgeDistillation(t, all M')
            (update M with M_t)

ClientUpdate(k):
      receive model M_k
      for each local epoch do
            M' <- (train M_k on the local dataset)
      send M' to server

KnowledgeDistillation(t, all M'):
      for each local epoch do
            M_t <- (train M_t with inference of all M')
      return M_t
```

Listing 3.1. FedDF algorithm

Proposed architecture

To address the issue, the proposed architecture should combine advantages of both heterogeneous and hierarchical federated learning in order to negate their respective disadvantages. To do so, for providing scalability hierarchical multi-layer structure is taken as a basis as it allows to scale the system varying the number of clusters and layers. But, unlike the typical approach where clusters in the hierarchical structure are created based on spatial proximity to minimize the latency in communications within each cluster, the clusters are composed based on model types that allows to combine all models within each cluster seamlessly. To enable heterogeneity, combined models from the clusters are used as an ensemble for knowledge distillation into new models, one per existing in the system model type, that are spread within corresponding clusters.

It allows us to create an infinitely scalable heterogeneous system. However, it contains two potential flaws that should be addressed carefully during each implementation. The first one is potential increased communication overhead overall in comparison to non-hierarchical heterogeneous federated architectures as a representation of heterogeneous federated learning due to an increased number of elements within the system and accordingly an increased number of communications among them. The second flaw is potential increased latency in the communications within clusters in comparison to hierarchical non-heterogeneous federated architectures as the primary factor for grouping clients into clusters is not spatial proximity but a model type. The third flaw is virtually unavoidable in the most cases and is larger energy consumption within the system overall in comparison to non-hierarchical heterogeneous federated architectures based on selecting

random subset of clients as in the case of the proposed architecture all clients should spend energy on training in contract to training of only subsets of clients.

Thus, a set of clients was created with appointing each client to a separate subset of data. All clients were assigned based on the model type to several clusters with an edge-server within each cluster and one cloud server. At each edge round, client models were trained on the local data and were collected by the cluster edge-server for averaging and spreading a produced model within cluster clients. After several edge rounds, edge-servers send produced models to the cloud server where those were combined into an ensemble that was used to train new models - one per type - with a separate dataset via knowledge distillation. The resulting models were spread across corresponding clusters for further training. The algorithm can be written as follows.

```
Server executes:
      initialize models M of types T
      for each cloud round do
            for each edge server e in E do
                  for each edge round do
                        for each client k in K do
                              M' <- ClientUpdate(k)
                        M_e <- (average weights across all M' within e)
            for each model type t in T do
                  M_t <- KnowledgeDistillation(t, all M_e)
            (update M with M_t)

ClientUpdate(k):
      receive model M_k
```

```
      for each local epoch do
            M’ <- (train Mk on the local dataset)
      send M’ to server

KnowledgeDistillation(t, all M’):
      for each local epoch do
            Mt <- (train Mt with inference of all M’)
      return Mt
```

Listing 4.1. Proposed architecture algorithm

CHAPTER FIVE

IMPLEMENTATION AND EXPERIMENTAL RESULTS

As it was mentioned in chapter 4, many details of the reference implementations in the original works for the chosen baseline models have not been disclosed, and those that have been disclosed vary among models. In conjunction with the variety of other aspects such as data splitting, data preprocessing and model architectures it makes it next to impossible to find a definitive overlap between the experiments in the original works for the baseline models or reproduce them in the limited timeline as they heavily rely on hyperparameter tuning within unknown search spaces and on unknown seeds. We can see it clearly from the pivot tables.

	FedAvg	HierFAVG	FedDF
datasets	MNIST CIFAR-10 custom text dataset	MNIST CIFAR-10	CIFAR-10 CIFAR-100 ImageNet AG News SST2
data preprocessing	crop to 24x24 horizontal flips contrast adjusting brightness adjusting whitening adjusting	normalization random crop padding horizontal flips	n/a
non-IID	no yes	yes	yes
non-IID details	n/a	partially	n/a
data split	n/a	n/a	partially

batch size	10 50 100%	20	n/a
local epochs	1 5 20 25 50 100 200	5 6 10 15 25 30 50 60	20 40
edge epochs	-/-	1 2 4 5 10	-/-
clients	100	50	20
participation rate	1 client 0.1 0.2 0.5 1.0	-/-	0.2 0.4 0.8
edges	-/-	10	-/-
edge assignment	-/-	dynamic	-/-
models	custom MLP custom CNN custom LSTM	custom CNN	ResNet-8 ResNet-20 ResNet-32 ShuffleNetV2 DistilBERT
seed	n/a	n/a	n/a
optimizer	n/a	SGD	SGD Adam
scheduler	n/a	0.992 / epoch 0.995 / epoch	1

learning rate	n/a	0.01 0.1	0.00001 0.05 0.1
weight decoy rate	n/a	n/a	1
KD dataset	-/-	-/-	CIFAR-100 ImageNet BigGAN-generated
KD batch	-/-	-/-	128
KD optimizer	-/-	-/-	Adam
KD learning rate	-/-	-/-	0.001
KD function	-/-	-/-	Kullback-Leibler
target metric	epochs	time local energy	accuracy epochs

Table 2.1. Reference architectures' details

In addition, in some cases where the original implementation was available it was designed for a cluster structure that is incompatible with the experimental setup. For example, the reference implementation of FedDF is created to be executed on a Kubernetes cluster with MPI interactions between pods, which would require reimplementing a significant part of it to adapt to the available infrastructure.

Thus, in order to maximize the integrity of the setup across the baseline architectures and the proposed architecture by standardization of data handling, data preprocessing, results evaluation and some aspects of model training, baseline architectures were implemented from scratch. By doing so, it was possible to minimize the impact of variations in the experimental setup or external dependencies. While some hyperparameter tuning has been executed in order to maximize the performance of the models, limitations

in the timeline and computational resources constrained the thoroughness of the tuning. Therefore, it is not expected to achieve the maximum possible performance and is expected to achieve the results lower than the shared in the original works. However, as the main goal is to compare the proposed architecture with the baseline models, the relative results are more important in our case than absolute numbers.

Dataset

As some of baseline architectures were evaluated on image classification and text classification tasks with variety of dataset in the original works, the next factors were taken into considerations:

- number of samples in the dataset
- number of classes in the dataset
- feature variability
- acceptance

due to the following reasons.

As the main reason for the proposed architecture is scalability, it is important to have enough clients that leads to splitting the dataset into multiple pieces. Thus, it's becoming important to have the ratio "samples per class" as large as possible to be able to scale the experimental setup.

At the same time, it's important to have a decent diversity in classes that would require models to develop robust feature extraction capabilities, ensuring that successful approaches are genuinely effective in recognizing a wide range of patterns.

Another factor is the number of samples itself as while it is important to have enough samples for scalability, out of practical reasons it is better to find a balance between the scalability potential and the training time that will increase with increasing the number of samples.

Assessing different machine learning architectures using datasets with complex, high-dimensional features allows us to directly compare their abilities to process and organize data. However, when the tasks involve additional feature variability, it adds extra layers of complexity. This can make it harder to see how changes in the architecture affect the model's performance.

Finally, if there are no specific requirements to a dataset, it’s important to utilize ones that are widely accepted by other researchers within the field for the sake of reproducibility, comparability, and data quality.

In the following table we can see the most popular datasets for image and text classification.

Text classification tasks can introduce additional complexities compared to image classification. These complexities include handling variable-length sequences, dealing with text preprocessing steps like tokenization and lemmatization, and potentially word embeddings. While these complexities are manageable, they introduce additional variability. Thus, out of this it was decided to limit the comparison by image classification only that excluded IMDb [80], 20 Newsgroups [81], Reuters-21578 [82], AG News [83], DBpedia [84] and Yahoo! Answers datasets.

Out of datasets for image classification LSUN [85] and MS Coco [86] were excluded as each of them contains images with complex content that can be classified in multiple ways that can lead to mistraining the models and therefore require additional effort for preventing it.

Out of the rest datasets CIFAR-100 [87] was excluded as it contains only 600 samples per class that makes it less convenient for scaling the system that is the main point of the experimental setup. For the same reason, in combination with a significantly larger number of samples that makes every experiment noticeably longer, ImageNet dataset was excluded.

Thus, for the experimental setup for all architectures and models CIFAR-10 [87] was used.

Dataset	Samples	Classes	Samples / class	Feature variability
CIFAR-10	60000	10	6000	low
CIFAR-100	60000	100	600	low
ImageNet	1431167	1000	1431	low
MS Coco	328000	80	4100	high
LSUN	59000000+	10 / 20	varies	high
IMDb	50000	2	25000	high
20 Newsgroups	18000	20	900	high
Reuters-21578	10369	118	87	high
AG News	1000000+	4	250000+	high
DBpedia	630000	14	45000	high

Yahoo! Answers	1460000	10	146000	high

Table 3.1. Datasets' feature comparison

Out of IID and non-IID data distributions for the experimental setup IID distribution was chosen due to the following reasons.

Using IID data distribution provides a controlled and balanced setting for evaluating the performance of different architectures. It helps to isolate the impact of the architectures themselves and minimize the influence of data heterogeneity on the performance results.

IID data distribution serves as a baseline and a starting point for evaluating federated learning architectures. It represents an ideal scenario where the data is homogeneously distributed across clients, and the architectures can be assessed under optimal conditions. By comparing different architectures on IID data, we can establish a benchmark for their performance that later can be used for experiments with non-IID data.

As usage of a completely unrelated dataset showed unsatisfactory results for knowledge distillation, the test subset of CIFAR-10 containing 1000 images per class was randomly split into 300 images for final validation, 560 images for knowledge distillation training and 140 images for knowledge distillation testing per class. The training subset of CIFAR-10 was randomly split into 18 partitions each of which contained 222 images for local training and 55 images for local testing per class.

As the main goal was not to achieve the maximum possible performance, but to compare achieved performance among selected and constructed architectures, the data

preprocessing was limited by standard for CIFAR-10 normalization where the mean = [0.4914, 0.4822, 0.4465] and the standard deviation = [0.2023, 0.1994, 0.2010].

Models

In order to include the heterogeneity factor within the FedDF architecture and the proposed architecture, it can be critical to select an appropriate number of different models. This choice can significantly impact the experimental setup and the ability to simulate real-world scenarios accurately.

On one hand, increasing the number of models naturally leads to a higher number of participating clients in the architectures with random client choice at each round such as FedDF, as each model requires at least one corresponding client. In federated learning settings, it is desirable to keep the participation ratio relatively low to imitate real-world scenarios, where only a subset of clients may be available or willing to participate in each round of training.

On the other hand, having too few models and clients may not sufficiently represent the federated nature of the setting. Federated learning is designed to use the collective knowledge and data from multiple clients while saving their privacy. If the number of clients is too small, it may not capture the diversity and heterogeneity present in real-world federated environments. Additionally, a limited number of models may not provide enough variety in terms of model architectures, capacities, and learning capabilities, which is a key aspect of heterogeneous federated learning.

In this case, three different models have been chosen as a reasonable compromise. This choice allows for a sufficient level of heterogeneity in terms of model architectures

while keeping the ratio of participating clients manageable. While three clients out of eighteen that is equal to ~0.16 participating rate is suboptimal in the sense of modeling a real-world high-scale scenario, the dataset size imposes restrictions on the maximal number of clients that limits the minimal participation rate.

For the choosing specific models for the federated setting, one of the most important factors was required computational resources as it was crucial to iterate through hyperparameter tuning and experiments with limited computational resources and within a limited timeline. An additional factor was modeling a potential diversity in models due to variety in computational resources available for different clients. As a result, the three chosen models are a simple custom CNN, ResNet-8 [88] and DenseNet [89] that are often used for image classification, quite effective and at the same time noticeably different in the required computational resources.

The custom CNN is composed of three convolutional blocks each of which contains a convolutional layer, ReLU as an activation function, a pooling layer and batch normalization, and a sequential block with two fully connected layers, ReLU as an activation function and batch normalization.

The used variation of ResNet-8 is composed of a convolutional layer, batch normalization and ReLU as an activation function for the convolutional base, three residual layers with each layer doubling the number of filters and reducing the spatial dimensions of the feature maps through strided convolutions for downsampling, and average pooling. Outside of hyperparameters, an optimizer and a scheduler, the architecture is identical to the one that was used in the reference implementation of FedDF.

The implementation of DenseNet is composed of a simple convolutional layer followed by 3 dense blocks with multiple layers containing batch normalization, ReLU as an activation function, a convolutional layer and a dropout layer each with transitional layers composed of batch normalization, ReLU as an activation function, a convolutional layer and a dropout layer. Finally, global average pooling is applied to obtain a fixed-size representation of the feature maps, followed by a fully connected layer for classification. Outside of hyperparameters, an optimizer and a scheduler, the architecture is identical to the one that was used in the reference implementation of FedDF.

The used variation of MobileNetV2 [90] is composed of an initial convolutional layer, batch normalization and ReLU as an activation function, ten block each of which contains from two to three convolutional layers and batch normalizations, and the final section with average pooling, and a fully connected layer. Outside of hyperparameters, an optimizer and a scheduler, the architecture is identical to the one that was used in the reference implementation of FedDF.

Hyperparameters

Hyperparameter tuning via random grid search has been conducted separately for the proposed architecture with heterogeneous models, FedDF with heterogeneous models, for HierFAVG and FedAvg with each mode and for knowledge distillation. The optimal parameters and the respective results of the experiments are listed below in the corresponding sections of the chapter.

Training

For each heterogeneous architecture and for each model in homogeneous architectures three experiments were conducted with different seeds. The results were averaged across all three seeds within each heterogeneous architecture and each model in homogeneous architectures to minimize a potential impact of seeds on the comparison.

As the architectures contain from 2 to 3 layers that consist of clients, edge-services, and cloud-services, to provide a fair comparison, the number of epochs was calculated based on the total number of local epochs for the lowest level of architectures that includes clients reflecting organization or devices. Each architecture was trained for 200 local epochs in order to compare the speed of convergence and the achieved accuracy.

FedAvg

FedAvg architecture was trained with a custom CNN, ResNet-8 and DenseNet with the parameters listed in the table 4. The listed parameters for the reference implementation correspond to the best achieved in the paper result. For each of 18 partitions a separate client with a model was created. At each round random 3 clients were chosen for training during the round. Each model of the chosen three ones was trained for 3 local epochs after which models were shared to a central server that averaged their weights and distributed a new produced model back to all clients. The process continued until the number of local epochs reached 200.

	reference	experiment (CNN)	experiment (ResNet-8)	experiment (DenseNet)
batch size	50	64	64	64
local epochs	5	3	3	3
clients	100	18	18	18
participation rate	0.1	0.17	0.17	0.17
models	custom CNN	custom CNN	ResNet-8	DenseNet
optimizer	n/a	Adam	Adam	Adam
scheduler	n/a	0.11 / 2 epochs	0.99 / epoch	1 / 1 epoch
learning rate	n/a	0.0007	0.001	0.0007
weight decoy rate	n/a	0.001	0.001	0.001
target metric	epochs	accuracy convergence	accuracy convergence	accuracy convergence

Table 4.1. FedAvg training parameters

The achieved accuracy within 200 local epochs is reflected in table 5 and figures 1, 2, and 3.

	CNN	ResNet-8	DenseNet	average
Accuracy	75.31%	68.36%	68.72%	70.8%

Table 4.2. FedAvg accuracy

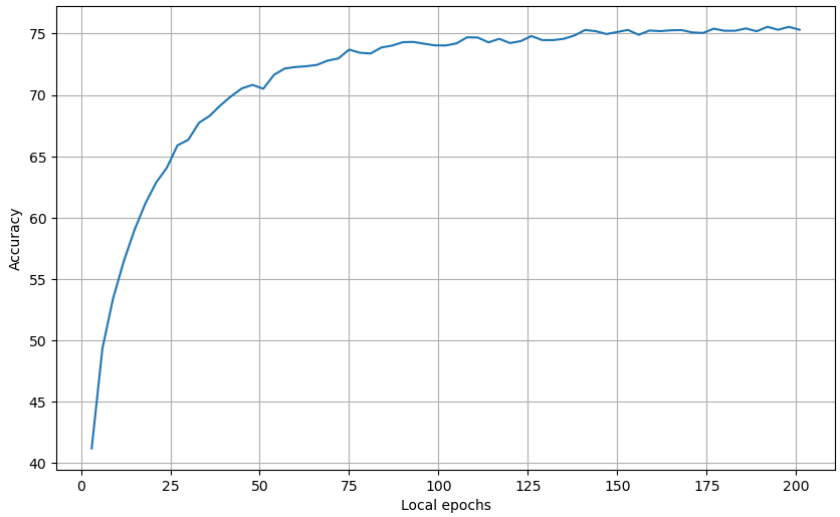

Figure 1.1. FedAvg with CNN accuracy

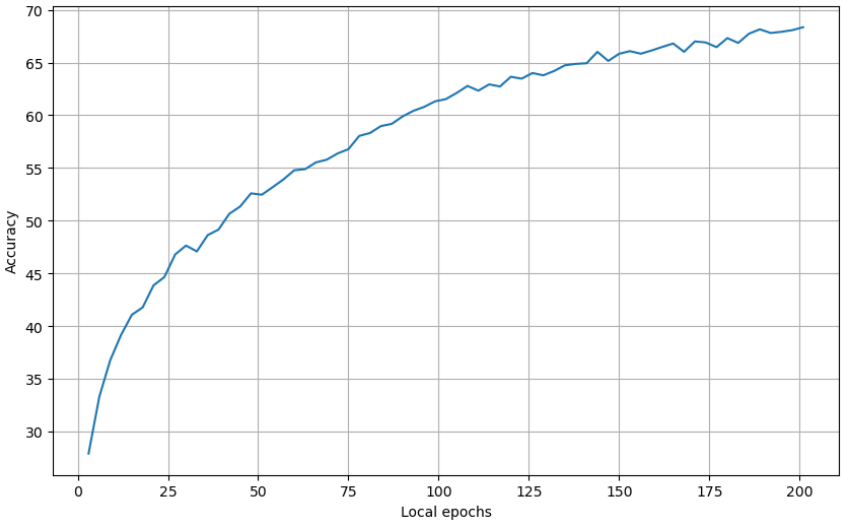

Figure 1.2. FedAvg with ResNet-8 accuracy

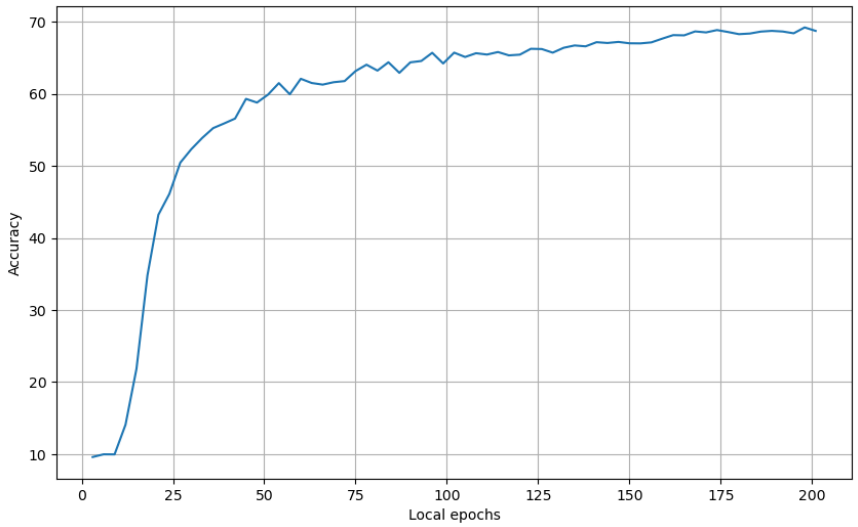

Figure 1.3. FedAvg with DenseNet accuracy

HierFAVG

HierFAVG architecture was trained with a custom CNN, ResNet-8 and DenseNet with the parameters listed in the table 6. The listed parameters for the reference implementation correspond to the best achieved in the paper result. For each of 18 partitions a separate client with a model was created. All the clients were splitted into 3 clusters each of which contained 6 clients. Each cluster contained an edge server responsible for averaging weights within the respective cluster. The cloud server was appointed for averaging the models produced at the edge servers. At each client the model was trained for 3 local epochs. After that trained models were sent to respective edge servers where they were averaged within the cluster, and the produced model was sent back to clients within the same cluster. After 3 rounds at the edge servers, the produced at the edge servers were sent to the cloud server that averaged the received models and sent it

back to the edge servers that in its turn sent it back to clients within the respective clusters. The process continued until the number of local epochs reached 200.

	reference (CNN)	experiment (CNN)	experiment (ResNet-8)	experiment (DenseNet)
batch size	20	64	64	64
local epochs	5	3	3	3
edge epochs	10	3	3	3
clients	50	18	18	18
edges	5	3	3	3
edge assignment	dynamic	constant	constant	constant
optimizer	SGD	Adam	Adam	Adam
scheduler	0.992 / epoch	0.11 / 2 epochs	0.99 / epoch	0.11 / 2 epochs
learning rate	0.1	0.0007	0.001	0.0007
weight decoy rate	n/a	0.001	0.001	0.001
target metric	time local energy	accuracy convergence	accuracy convergence	accuracy convergence

Table 5.1. HierFAVG training parameters

The achieved accuracy within 200 local epochs is reflected in table 7 and figures 4, 5, and 6.

	CNN	ResNet-8	DenseNet	average
Accuracy	75.78%	70.07%	67.4%	71.08%

Table 5.2. HierFAVG accuracy

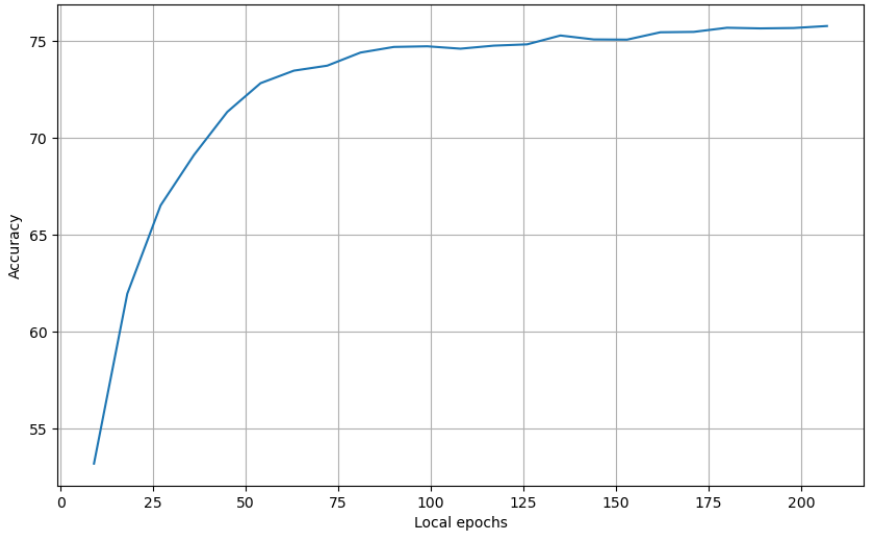

Figure 2.1. HierFAVG with CNN accuracy

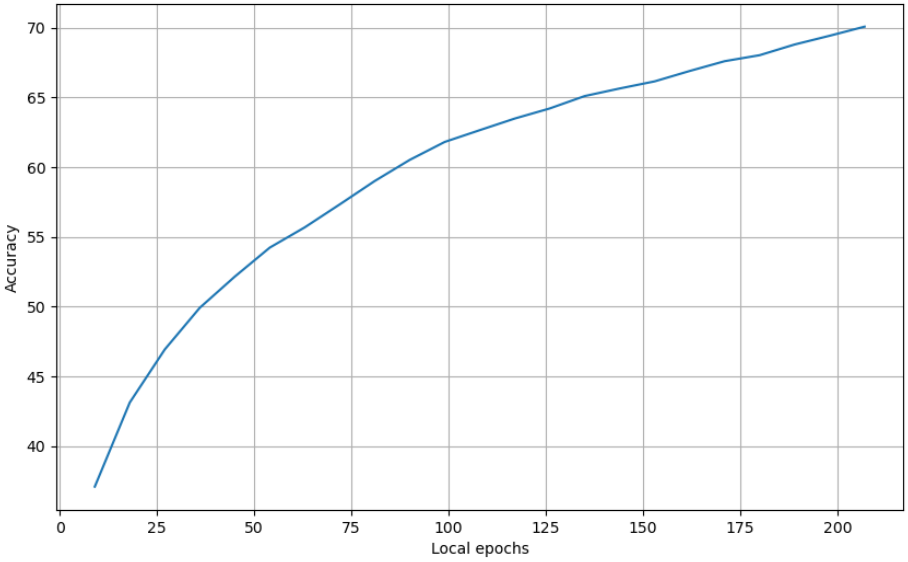

Figure 2.2. HierFAVG with ResNet-8 accuracy

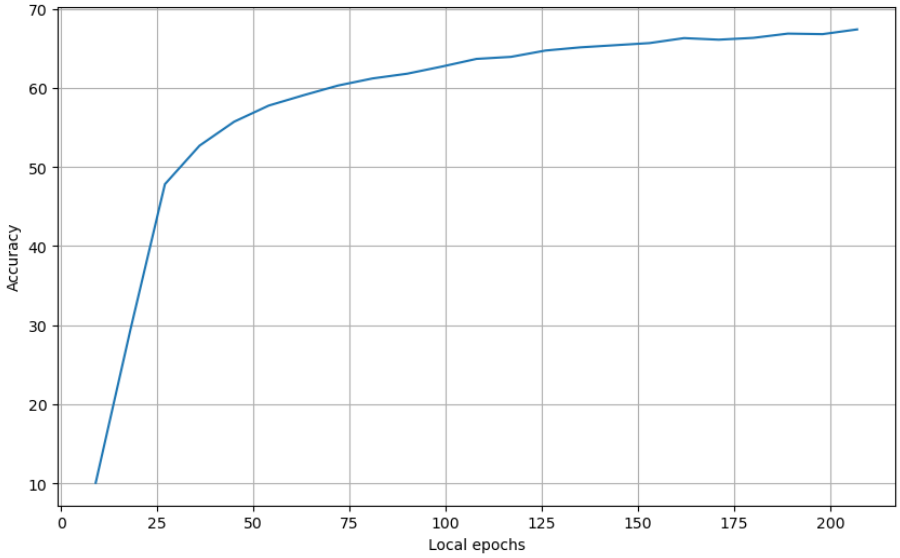

Figure 2.3. HierFAVG with DenseNet accuracy

FedDF

FedDF architecture was trained with a custom CNN, ResNet-8 and DenseNet with the parameters listed in the table 8. The listed parameters for the reference implementation correspond to the best achieved in the paper result for the heterogeneous setup. For each of 18 partitions a separate client with a model was created. At each round, 3 clients were chosen in a way that every time all models were represented equally among the chosen clients. Each model of the chosen three ones was trained for 5 local epochs after which models were sent to a central server. The central server combined the received models into an ensemble and used a knowledge transfer data subset to train 3 new models, each of CNN, ResNet-8 and DenseNet, based on averaged outputs of the ensemble, outputs of the training model and Kullback-Leibler divergence. After that the produced models were sent back to all clients. The process continued until the number of local epochs reached 200.

	reference	experiment		
		CNN	ResNet-8	DenseNet
batch size	n/a	64	64	64
local epochs	80	5	5	5
clients	21	18		
participation rate	0.4	0.17		
models	ResNet-20 ResNet-32 ShuffleNetV2	custom CNN	ResNet-8	DenseNet
optimizer	n/a	Adam	Adam	Adam
scheduler	n/a	0.11 / 2 epochs	0.99 / epoch	0.11 / 2 epochs
learning rate	0.1	0.0007	0.001	0.0007
weight decoy rate	n/a	0.001	0.001	0.001
KD dataset	CIFAR-100	CIFAR-10	CIFAR-10	CIFAR-10
KD batch	128	64	64	64
KD optimizer	Adam	Adam	Adam	Adam
KD learning rate	0.001	0.0007	0.001	0.0007
KD function	Kullback-Leibler	Kullback-Leibler	Kullback-Leibler	Kullback-Leibler
target metric	accuracy epochs	accuracy convergence	accuracy convergence	accuracy convergence

Table 6.1. FedDF with DenseNet training parameters

The achieved accuracy within 200 local epochs is reflected in table 9 and figure 7.

	CNN	ResNet-8	DenseNet	average
Accuracy	67.57%	67.97%	62.7%	66.08%

Table 6.2. FedDF with DenseNet accuracy

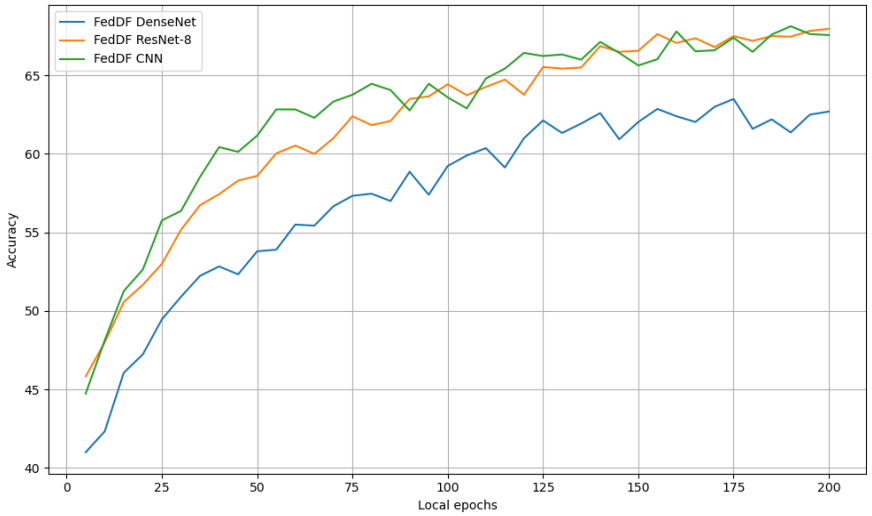

Figure 3.1. FedDF with DenseNet accuracy

Proposed architecture

The proposed architecture was trained with a custom CNN, ResNet-8 and DenseNet with the parameters listed in table 10. For each of 18 partitions a separate client with a model was created, 6 clients per each model type. All the clients were split into 3 equal clusters grouped by the model type. Each cluster contained an edge server responsible for averaging weights within the respective cluster. A cloud server was appointed for knowledge transfer from the models produced by edge servers into new models. At each client the model was trained for 2 local epochs. After that trained models were sent to respective edge servers where they were averaged within the cluster, and the produced

model was sent back to clients within the same cluster. After 5 rounds at the edge servers, the produced at the edge servers were sent to the cloud server. The cloud server combined the received models into an ensemble and used a knowledge transfer data subset to train 3 new models, each of CNN, ResNet-8 and DenseNet, based on averaged outputs of the ensemble, outputs of the training model and Kullback-Leibler divergence. After that the produced models were sent back to the respective edge servers that in turn sent it back to clients within the respective clusters. The process continued until the number of local epochs reached 200.

	experiment		
	CNN	ResNet-8	DenseNet
batch size	64	64	64
local epochs	2	2	2
edge epochs	5	5	5
clients	18	18	18
edges	3	3	3
edge assignment	constant	constant	constant
optimizer	Adam	Adam	Adam
scheduler	0.11 / 2 epochs	0.99 / epoch	0.11 / 2 epochs
learning rate	0.0007	0.001	0.0007
weight decoy rate	0.001	0.001	0.001
KD dataset	CIFAR-10	CIFAR-10	CIFAR-10

KD batch	64	64	64
KD epochs	10	10	10
KD optimizer	Adam	Adam	Adam
KD learning rate	0.0007	0.001	0.0007
KD function	Kullback-Leibler	Kullback-Leibler	Kullback-Leibler
target metric	accuracy	accuracy	accuracy

Table 7.1. Proposed architecture with DenseNet training parameters

The achieved accuracy within 200 local epochs is reflected in table 11 and figures 8, 9, 10 and 11.

	CNN	ResNet-8	DenseNet	average
Accuracy	70.37%	66.07%	59%	65.17%

Table 7.2. Proposed architecture with DenseNet accuracy

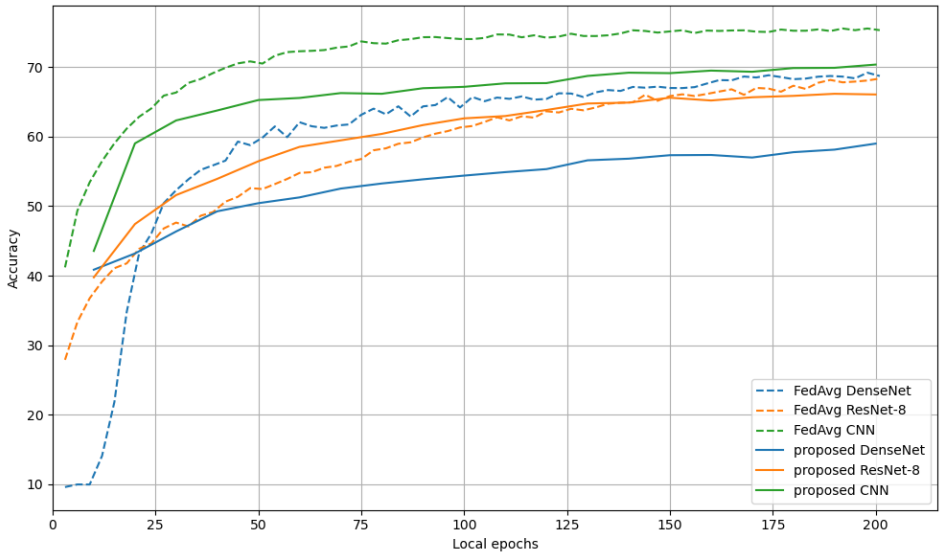

Figure 4.1. Proposed architecture with DenseNet accuracy in respect to FedAvg

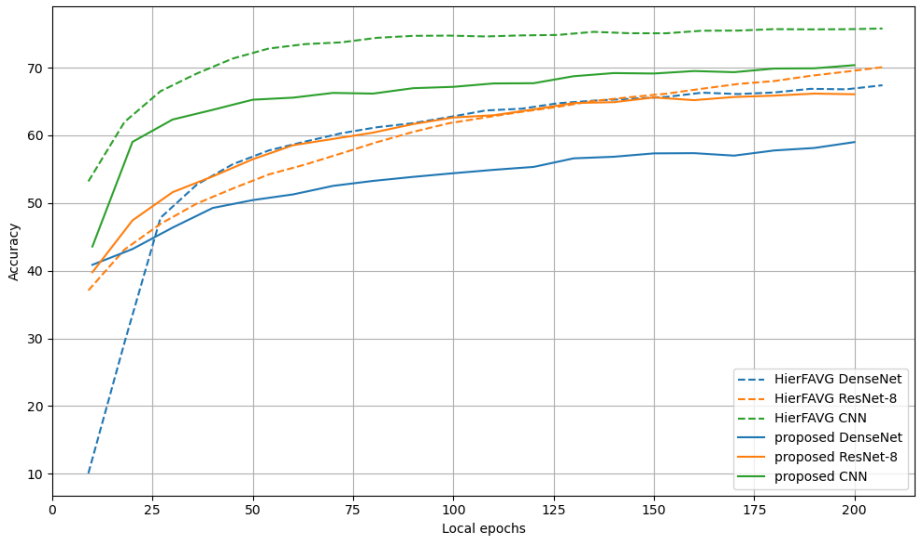

Figure 4.2. Proposed architecture with DenseNet accuracy in respect to HierFAVG

As it is clearly visible from the figures 8 and 9, the implementation of the proposed architecture is noticeably lagging behind more simple and homogeneous architectures

without knowledge distillation for a custom CNN and DenseNet and is slightly lagging behind for ResNet-8.

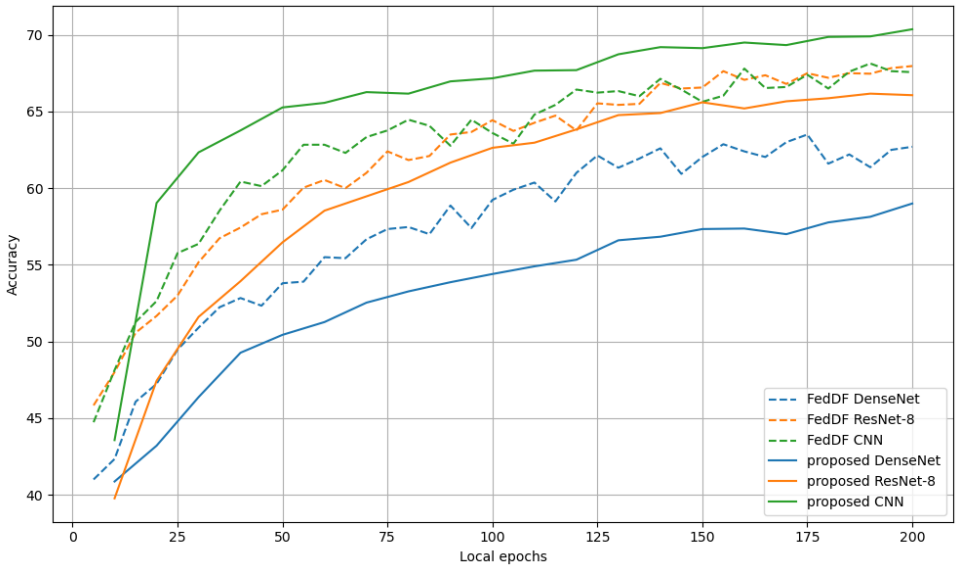

Figure 4.3. Proposed architecture with DenseNet accuracy in respect to FedDF

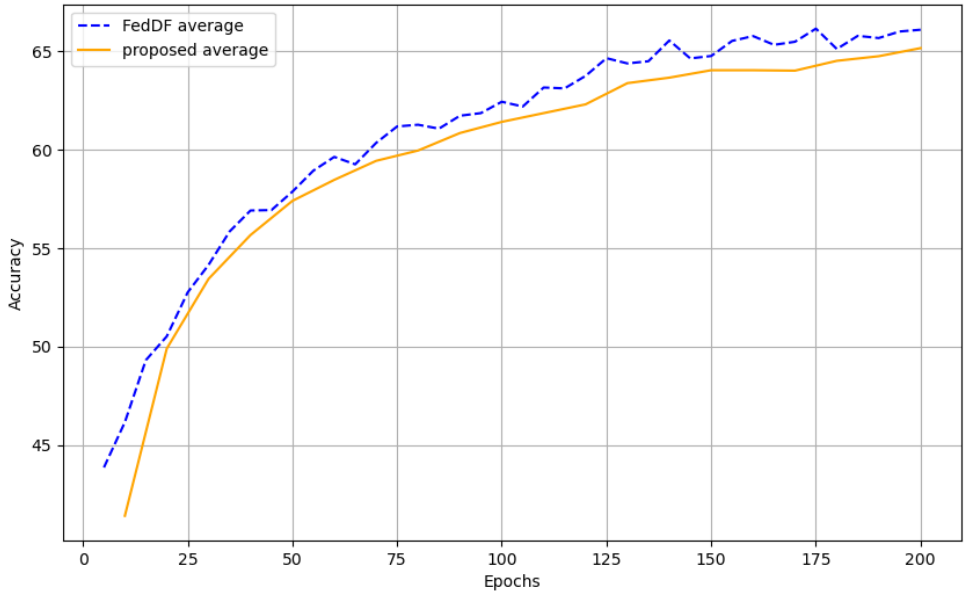

Figure 4.4. Proposed architecture with DenseNet average accuracy in respect to FedDF

However, when the proposed architecture is compared with FedDF, another architecture with knowledge transfer, the situation is not so clear. There is still a gap between models in the proposed architecture and FedDF for DenseNet models and ResNet-8 instances are quite close to each other, but a custom CNN is the proposed architecture clearly performs better than in FedDF. After averaging the accuracy across different models in the proposed architecture and FedDF it is visible that on average the proposed architecture is quite close to FedDF in terms of accuracy and convergence speed.

The possible reasons of the worse performance are the following:

- In FedDF architecture at each round the knowledge is extracted from a random and every time different subset of clients which leads to more dynamic data coverage and less biased models while in the proposed architecture the clusters are assigned permanently which can lead to worse or at least slower generalization even with sharing the knowledge among clusters via knowledge distillation. Potentially, it can be compensated with dynamic cluster reassignment during training, but in some real-world scenarios it may be not feasible.
- The participation rate for clients in the experiments is ~0.17 which can be high enough for random client selections in FedDF to cover most of the data quite fast. In the setups with a larger scale, such as million clients, where an expected participation rate can drop lower than 0.01 we can expect a faster convergence rate for the proposed architecture than for FedDF as the data coverage with a low participation rate can also be low in comparison to the proposed architecture that

covers all the data due to its structure, especially if the data varies significantly among clients.

Additional experiments

As it is clearly visible from the results of the experiments, the DenseNet is noticeable behind other models in the term of accuracy in FedDF and the proposed architecture that can affect the relative comparison. To decrease this effect another series of experiments was conducted for FedDF and the proposed architecture after replacing DenseNet with MobileNetV2.

FedDF architecture was trained with a custom CNN, ResNet-8 and MobileNetV2 with the parameters listed in table 12.

	reference	experiment		
		CNN	ResNet-8	MobileNetV2
batch size	n/a	64	64	64
local epochs	80	5	5	5
clients	21	6	6	6
participation rate	0.4	0.17		
models	ResNet-20 ResNet-32 ShuffleNetV2	custom CNN	ResNet-8	MobileNetV2
optimizer	n/a	Adam	Adam	Adam
scheduler	n/a	0.11 / 2 epochs	0.99 / epoch	0.11 / 2 epochs

learning rate	0.1	0.0007	0.001	0.0007
weight decoy rate	n/a	0.001	0.001	0.001
KD dataset	CIFAR-100	CIFAR-10	CIFAR-10	CIFAR-10
KD batch	128	64	64	64
KD optimizer	Adam	Adam	Adam	Adam
KD learning rate	0.001	0.0007	0.001	0.0007
KD function	Kullback-Leibler	Kullback-Leibler	Kullback-Leibler	Kullback-Leibler
target metric	accuracy epochs	accuracy convergence	accuracy convergence	accuracy convergence

Table 8.1. FedDF with MobileNetV2 training parameters

The achieved accuracy within 200 local epochs is reflected in table 13 and figure 12.

	CNN	ResNet-8	MobileNetV2	average
Accuracy	67.3%	67.9%	63.5%	66.23%

Table 8.2. FedDF with MobileNetV2 accuracy

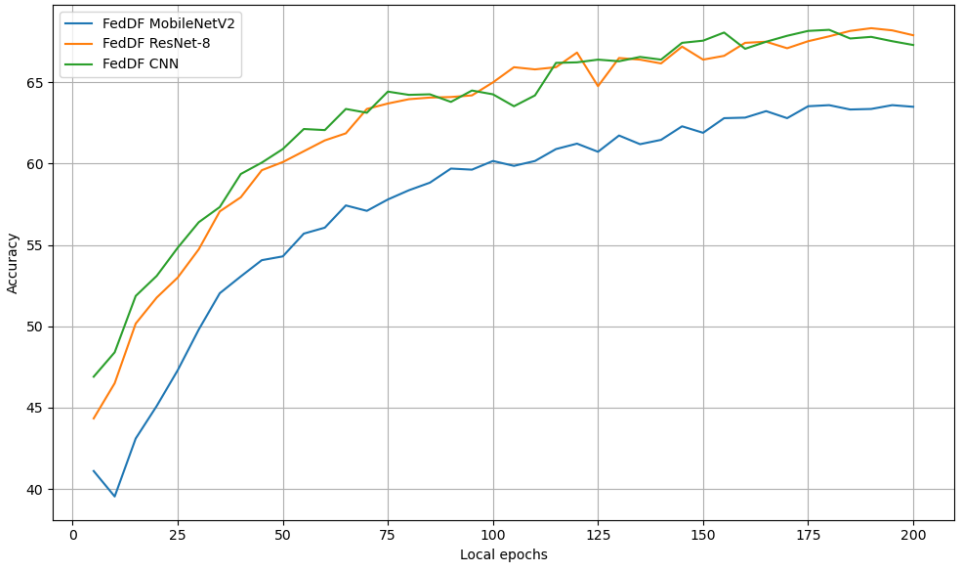

Figure 5.1. FedDF with MobileNetV2 accuracy

The proposed architecture was trained with a custom CNN, ResNet-8 and MobileNetV2 with the parameters listed in table 14.

	experiment		
	CNN	ResNet-8	MobileNetV2
batch size	64	64	64
local epochs	2	2	2
edge epochs	5	5	5
clients	18	18	18
edges	3	3	3
edge assignment	constant	constant	constant
optimizer	Adam	Adam	Adam

scheduler	0.11 / 2 epochs	0.99 / epoch	0.11 / 2 epochs
learning rate	0.0007	0.001	0.0007
weight decoy rate	0.001	0.001	0.001
KD dataset	CIFAR-10	CIFAR-10	CIFAR-10
KD batch	64	64	64
KD epochs	10	10	10
KD optimizer	Adam	Adam	Adam
KD learning rate	0.0007	0.001	0.0007
KD function	Kullback-Leibler	Kullback-Leibler	Kullback-Leibler
target metric	accuracy	accuracy	accuracy

Table 8.3. Proposed architecture with MobileNetV2 training parameters

The achieved accuracy within 200 local epochs is reflected in table 15 and figures 13 and 14.

	CNN	ResNet-8	MobileNetV2	average
Accuracy	69.83%	67.87%	61.13%	66.28%

Table 8.4. Proposed architecture with MobileNetV2 accuracy

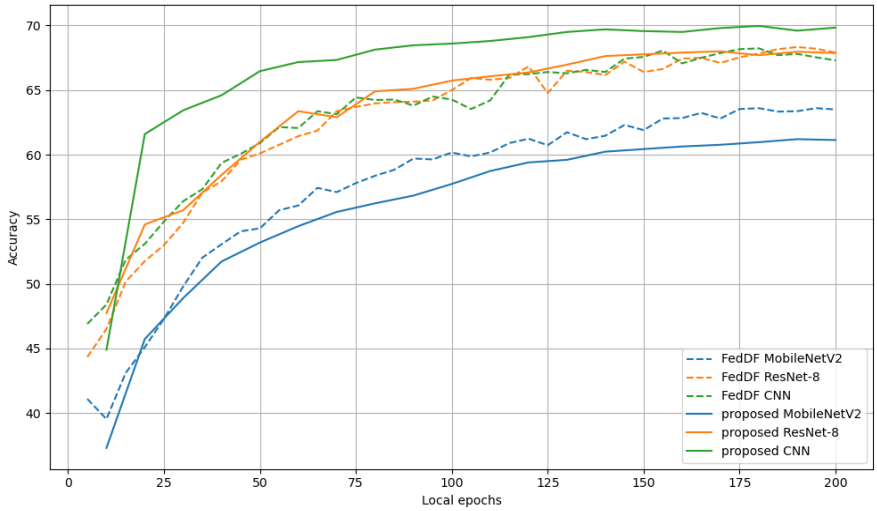

Figure 5.2. Proposed architecture with MobileNetV2 accuracy in respect to FedDF

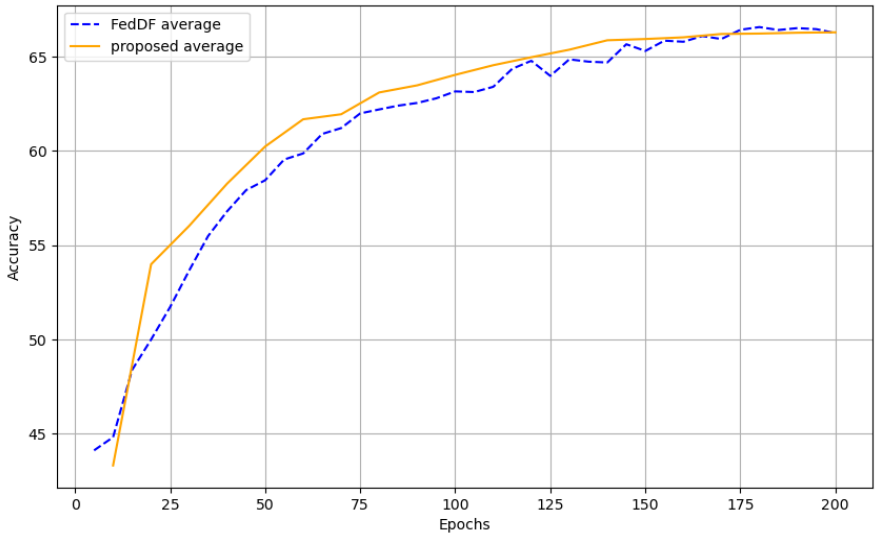

Figure 5.3. Proposed architecture with MobileNetV2 average accuracy in respect to FedDF

While MobileNetV2 models are still lagging behind the rest of them, their performance is better than the performance of DenseNet implementation used in the experiments. The replacement led to reduction of the gaps between the same types of models within FedDF and the proposed architecture and equalizing the accuracy of these

two architectures at 200 local epochs. However, while the final accuracy at 200 local epochs is equal between architectures, the proposed architecture converges slightly faster. It allows us to speculate that with a) increasing an average accuracy, b) decreasing the difference between different models' performance, c) decreasing the clients participation rate for FedDF, the gap in convergence speed between architectures will grow in favor of the proposed architecture.

CHAPTER SIX

CONCLUSIONS AND FUTURE WORK

While various approaches to federated machine learning address specific challenges, many real-world scenarios present a combination of issues that existing methods struggle to fully resolve. For example, in domains like mobile applications or IoT/network devices, where the number of model instances can reach millions or billions, data cannot be shared, and models should have different architectures due to hardware heterogeneity, business cycles, or environment limitations, the existing approaches may be suboptimal. Naive and ensemble federated learning approaches struggle with scalability, hierarchical federated learning does not support heterogeneous models, and heterogeneous federated learning, while supporting different architectures, lacks scalability. Therefore, there is a need for a new architecture that combines the benefits of these approaches, allowing different model architectures within a single federated setup, being virtually infinitely scalable, and covering all data during training to avoid underrepresentation of certain subsets of data.

The proposed architecture combines such approaches as ensemble federated learning, knowledge distillation and hierarchical federated learning into one structure where models are grouped into clusters based not on spatial proximity or data features, but on model types. In combination with knowledge distillation from an ensemble of different models it allows to create a system where models of different types can coexist and learn from each other. At the same time, employment of principles of hierarchical federated learning allows the structure to scale virtually indefinitely.

In this work several architectures such as FedAvg, HierFAVG and FedDF were implemented to create a baseline for a new architecture. CIFAR-10 used as a dataset for the experiments was split into a test and a train parts. The test part was split into test and knowledge transfer parts where the test part was used exclusively for evaluating the models' performance and the knowledge transfer was used exclusively for training new models via knowledge distillation. The train part was split into 18 partitions that represented different clients or devices. After that the splitted data was used to train baseline architectures and the proposed one for 200 local epochs in total per architecture with 3 distinctive seeds. All measurements were averaged across seeds and evaluated based on the achieved accuracy and the convergence speed.

While in the conducted experiments the proposed architecture with heterogeneous models showed less performance than FedAvg and HierFAVG with homogeneous models, its accuracy is comparable with FedDF with heterogeneous models and in some cases the proposed architecture converges faster than FedDF on a similar setup. It allows to assume optimistically that with increasing the scale of the system and consequent decreasing the client participation rate the gap in convergence speed between the proposed architecture and architectures that allow having heterogeneous models within the same system by extracting knowledge from a limited subset of clients will grow in favor to the proposed architecture. It would allow to a) improve scalability, b) improve data coverage in systems with heterogeneous models.

Based on that work it is possible to claim that in at least some setups the proposed architecture is at least not worse than FedDF in terms of accuracy and better in terms of

convergence speed. However, additional research with extensive hyperparameter tuning is needed in order to validate this statement on a larger scale.

However, the work has some limitations, many of which are caused by limited available computational resources and by a limited timeline. For example, the authors of FedAvg, the simplest architecture out of four implemented in this work, conducted more than 2000 experiments to find the optimal model architecture and hyperparameters. Taking into account a significantly larger computational cost of experiments in this work, it is expected to have models not reaching the SOTA performance. As a result, while in the given setup the proposed architecture showed decent results, the situation can change after more thorough model architecture and hyperparameter tuning.

Another major limitation of this work is the scale of the implemented system. The key point of the proposed architecture is a potential ability to handle cases where the number of clients reached millions better than non-hierarchical architectures with heterogeneous models. However, emulating such cases requires datasets of the corresponding scale that can be quite hard to obtain outside of some private entities in the industry. As a result, the experiments were conducted on quite a small dataset that led to inability to create a scenario that would be close to the intended use of the proposed architecture. Thus, on a large scale it can show better results. Another potential approach would be usage of less data-hungry or pre-trained models to decrease the required for model training number of samples per client.

One more limitation of the work is restricting the experiment's domain by image classification only in order to narrow the scope down. Employment of other domains such

as text classification in the experiments can give a more holistic view on the architecture applicability.

In the real-world situations the data is virtually never distributed independently and identically across all the clients. However, in this work the experimental environment was constructed with the data distributed evenly to create a baseline potential future research. Usage of non-IID data on a large scale can show different relative results for the proposed and baseline architectures.

In some cases, in the real-world clients are permanently assigned to clusters, in some others they migrate between clusters and in some they can be reassigned randomly at each training round. In this work the only tested scenario is permanent assignment. Thus, the effectiveness of reassignment of clients between clusters in comparison to permanent assignment and the degree of dependency between the percentage of the reassigned clients and reassignment scheme, and the performance of the architecture is still an open question.

Another limitation of the experiments was employment of only one data splitting and only one cluster configuration. While all architectures were tested in virtually identical environments with some unavoidable differences conditioned by the differences in the architectures, different cluster configuration could produce different results so it would be useful to investigate potential tendencies in the proposed architecture performance affected by the cluster configurations, and usage of different data splitting with averaging results across them could give more precise evaluation of the performance.

Summarizing, the potential directions of future research would be

- more extensive and thorough model architecture and hyperparameter tuning in order to compare architecture at the level of their respective maximum performance
- exploring the relative performance in a close to real-world scenario with significantly bigger number of clients with a corresponding level of data splitting granularity
- employment of less data-hungry models with more granular data splitting on order to compensate the available size of the datasets
- exploring the proposed architecture applicability within the domains besides image classification
- experiments with non-IID data that can lead to better in comparison to at least some baselines architecture performance of the proposed architecture due to potentially better data coverage
- exploring different cluster configurations in order to identify the most and the least applicable scenarios for the proposed architecture applicability
- investigating an effect of clients reassignment between clusters on the proposed architecture performance
- evaluation of the overhead in overall communications conditioned by the hierarchical nature of the proposed architecture in comparison to architectures with choosing a random subset of clients each round and identification of scenarios where that overhead would be acceptable

Additionally, three trade-offs of the proposed architecture should be mentioned. The first one is potentially increased communication cost of the system overall in

comparison to non-hierarchical heterogeneous federated architectures that stems from increased number of entities in the system and increased number of communications among them. The second one is potentially increased latency within clusters in comparison to non-heterogeneous hierarchical federated architectures that emerges due to the need to group clients into clusters based on model types instead of spatial proximity. As a result, each production system should be carefully analyzed in order to find the optimal architecture and the optimal trade-offs. The third trade-off is virtually always increased energy consumption in the system overall in comparison to architectures based on random client subset selection for training at each round as in the proposed architecture each client should participate in training at each round.

The future of federated learning lies in the development of solutions that effectively address the challenges of heterogeneity and scalability. The proposed architecture, combining ensemble federated learning, knowledge distillation, and hierarchical federated learning, while the conducted experiments didn't show a significant advantage in the given environment, still can represent a promising direction for scalable heterogeneous systems. As research progresses, advancements in clustering techniques, knowledge distillation methods, and aggregation strategies will optimize the performance, efficiency, and robustness of federated learning systems.

REFERENCES

[1] Hastie, T., Tibshirani, R., Friedman, J., Hastie, T., Tibshirani, R., & Friedman, J. (2009). Overview of supervised learning. *The elements of statistical learning: Data mining, inference, and prediction*, 9-41.

[2] Dada, E. G., Bassi, J. S., Chiroma, H., Adetunmbi, A. O., & Ajibuwa, O. E. (2019). Machine learning for email spam filtering: review, approaches and open research problems. *Heliyon*, *5*(6).

[3] Singh, A., & Singh, P. (2020). Image classification: a survey. *Journal of Informatics Electrical and Electronics Engineering (JIEEE)*, *1*(2), 1-9.

[4] Tandel, N. H., Prajapati, H. B., & Dabhi, V. K. (2020, March). Voice recognition and voice comparison using machine learning techniques: A survey. In *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)* (pp. 459-465). IEEE.

[5] Kaur, S., Singla, J., Nkenyereye, L., Jha, S., Prashar, D., Joshi, G. P., ... & Islam, S. R. (2020). Medical diagnostic systems using artificial intelligence (ai) algorithms: Principles and perspectives. *IEEE Access*, *8*, 228049-228069.

[6] Montgomery, D. C., Peck, E. A., & Vining, G. G. (2021). *Introduction to linear regression analysis*. John Wiley & Sons.

[7] Berkson, J. (1944). Application of the logistic function to bio-assay. *Journal of the American statistical association*, *39*(227), 357-365.

[8] Song, Y. Y., & Ying, L. U. (2015). Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, *27*(2), 130.

[9] Hofmann, T., Schölkopf, B., & Smola, A. J. (2008). Kernel methods in machine learning.

[10] Dong, X., Yu, Z., Cao, W., Shi, Y., & Ma, Q. (2020). A survey on ensemble learning. *Frontiers of Computer Science*, *14*, 241-258.

[11] Dongare, A. D., Kharde, R. R., & Kachare, A. D. (2012). Introduction to artificial neural network. *International Journal of Engineering and Innovative Technology (IJEIT)*, *2*(1), 189-194.

[12] Celebi, M. E., & Aydin, K. (Eds.). (2016). *Unsupervised learning algorithms* (Vol. 9, p. 103). Cham: Springer.

[13] Saxena, A., Prasad, M., Gupta, A., Bharill, N., Patel, O. P., Tiwari, A., ... & Lin, C. T. (2017). A review of clustering techniques and developments. *Neurocomputing*, *267*, 664-681.

[14] Sorzano, C. O. S., Vargas, J., & Montano, A. P. (2014). A survey of dimensionality reduction techniques. *arXiv preprint arXiv:1403.2877*.

[15] Ceglar, A., & Roddick, J. F. (2006). Association mining. *ACM Computing Surveys (CSUR)*, *38*(2), 5-es.

[16] Samariya, D., & Thakkar, A. (2023). A comprehensive survey of anomaly detection algorithms. *Annals of Data Science*, *10*(3), 829-850.

[17] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[18] Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., ... & Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

[19] Brunke, L., Greeff, M., Hall, A. W., Yuan, Z., Zhou, S., Panerati, J., & Schoellig, A. P. (2022). Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, *5*, 411-444.

[20] Afsar, M. M., Crump, T., & Far, B. (2022). Reinforcement learning based recommender systems: A survey. *ACM Computing Surveys*, *55*(7), 1-38.

[21] Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*, 279-292.

[22] Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, *12*.

[23] Grondman, I., Busoniu, L., Lopes, G. A., & Babuska, R. (2012). A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, part C (applications and reviews)*, *42*(6), 1291-1307.

[24] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444.

[25] Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017, August). Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)* (pp. 1-6). Ieee.

[26] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Rush, A. M. (2020, October). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations* (pp. 38-45).

[27] Kingma, D. P., & Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, *12*(4), 307-392.

[28] Yang, L., Zhang, Z., Song, Y., Hong, S., Xu, R., Zhao, Y., ... & Yang, M. H. (2023). Diffusion models: A comprehensive survey of methods and applications. *ACM Computing Surveys*, *56*(4), 1-39.

[29] Zohuri, B., & Moghaddam, M. (2020). Deep learning limitations and flaws. *Mod. Approaches Mater. Sci*, *2*, 241-250.

[30] Zhang, C., Xie, Y., Bai, H., Yu, B., Li, W., & Gao, Y. (2021). A survey on federated learning. *Knowledge-Based Systems*, *216*, 106775.

[31] Zhao, Y., Li, M., Lai, L., Suda, N., Civin, D., & Chandra, V. (2018). Federated learning with non-iid data. *arXiv preprint arXiv:1806.00582*.

[32] Linardatos, P., Papastefanopoulos, V., & Kotsiantis, S. (2020). Explainable ai: A review of machine learning interpretability methods. *Entropy*, *23*(1), 18.

[33] Quiñonero-Candela, J., Sugiyama, M., Schwaighofer, A., & Lawrence, N. D. (Eds.). (2008). *Dataset shift in machine learning*. Mit Press.

[34] de Mello, F. L. (2020). A survey on machine learning adversarial attacks. *Journal of Information Security and Cryptography (Enigma)*, *7*(1), 1-7.

[35] Lambert, N., Pister, K., & Calandra, R. (2022). Investigating compounding prediction errors in learned dynamics models. *arXiv preprint arXiv:2203.09637*.

[36] Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., & Galstyan, A. (2021). A survey on bias and fairness in machine learning. *ACM computing surveys (CSUR)*, *54*(6), 1-35.

[37] Liu, B., Ding, M., Shaham, S., Rahayu, W., Farokhi, F., & Lin, Z. (2021). When machine learning meets privacy: A survey and outlook. *ACM Computing Surveys (CSUR)*, *54*(2), 1-36.

[38] Asswad, J., & Marx Gómez, J. (2021). Data ownership: a survey. *Information*, *12*(11), 465.

[39] Wang, J., & Joshi, G. (2019). Adaptive communication strategies to achieve the best error-runtime trade-off in local-update SGD. *Proceedings of Machine Learning and Systems*, *1*, 212-229.

[40] Underwood, R., Calhoun, J. C., Di, S., & Cappello, F. (2024). Understanding The Effectiveness of Lossy Compression in Machine Learning Training Sets. *arXiv preprint arXiv:2403.15953*.

[41] Narayanan, D., Santhanam, K., Kazhamiaka, F., Phanishayee, A., & Zaharia, M. (2020). {Heterogeneity-Aware} cluster scheduling policies for deep learning workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)* (pp. 481-498).

[42] Sattler, F., Wiedemann, S., Müller, K. R., & Samek, W. (2019). Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems*, *31*(9), 3400-3413.

[43] Lin, Y., Han, S., Mao, H., Wang, Y., & Dally, W. J. (2017). Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887*.

[44] Yin, X., Zhu, Y., & Hu, J. (2021). A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions. *ACM Computing Surveys (CSUR)*, *54*(6), 1-36.

[45] Wei, K., Li, J., Ding, M., Ma, C., Yang, H. H., Farokhi, F., ... & Poor, H. V. (2020). Federated learning with differential privacy: Algorithms and performance analysis. *IEEE transactions on information forensics and security*, *15*, 3454-3469.

[46] Bhagoji, A. N., Chakraborty, S., Mittal, P., & Calo, S. (2019, May). Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning* (pp. 634-643). PMLR.

[47] Zhan, Y., Zhang, J., Hong, Z., Wu, L., Li, P., & Guo, S. (2021). A survey of incentive mechanism design for federated learning. *IEEE Transactions on Emerging Topics in Computing*, *10*(2), 1035-1044.

[48] Madill, E., Nguyen, B., Leung, C. K., & Rouhani, S. (2022, May). ScaleSFL: a sharding solution for blockchain-based federated learning. In *Proceedings of the Fourth ACM International Symposium on Blockchain and Secure Critical Infrastructure* (pp. 95-106).

[49] Chen, W., Horvath, S., & Richtarik, P. (2020). Optimal client sampling for federated learning. *arXiv preprint arXiv:2010.13723*.

[50] Hu, C., Jiang, J., & Wang, Z. (2019). Decentralized federated learning: A segmented gossip approach. *arXiv preprint arXiv:1908.07782*.

[51] Liu, L., Zhang, J., Song, S. H., & Letaief, K. B. (2020, June). Client-edge-cloud hierarchical federated learning. In *ICC 2020-2020 IEEE international conference on communications (ICC)* (pp. 1-6). IEEE.

[52] Nguyen, J., Malik, K., Zhan, H., Yousefpour, A., Rabbat, M., Malek, M., & Huba, D. (2022, May). Federated learning with buffered asynchronous aggregation. In *International Conference on Artificial Intelligence and Statistics* (pp. 3581-3607). PMLR.

[53] Mills, J., Hu, J., & Min, G. (2019). Communication-efficient federated learning for wireless edge intelligence in IoT. *IEEE Internet of Things Journal*, *7*(7), 5986-5994.

[54] Akter, M., Moustafa, N., Lynar, T., & Razzak, I. (2022). Edge intelligence: Federated learning-based privacy protection framework for smart healthcare systems. *IEEE Journal of Biomedical and Health Informatics*, *26*(12), 5805-5816.

[55] Wang, X., Han, Y., Wang, C., Zhao, Q., Chen, X., & Chen, M. (2019). In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *Ieee Network*, *33*(5), 156-165.

[56] Sattler, F., Müller, K. R., & Samek, W. (2020). Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE transactions on neural networks and learning systems*, *32*(8), 3710-3722.

[57] Briggs, C., Fan, Z., & Andras, P. (2020, July). Federated learning with hierarchical clustering of local updates to improve training on non-IID data. In *2020 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-9). IEEE.

[58] Jothimurugesan, E., Hsieh, K., Wang, J., Joshi, G., & Gibbons, P. B. (2023, April). Federated learning under distributed concept drift. In *International Conference on Artificial Intelligence and Statistics* (pp. 5834-5853). PMLR.

[59] Cui, Y., Cao, K., Zhou, J., & Wei, T. (2022). Optimizing training efficiency and cost of hierarchical federated learning in heterogeneous mobile-edge cloud computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[60] Dong, X., Yu, Z., Cao, W., Shi, Y., & Ma, Q. (2020). A survey on ensemble learning. *Frontiers of Computer Science*, *14*, 241-258.

[61] Shi, N., Lai, F., Kontar, R. A., & Chowdhury, M. (2021). Fed-ensemble: Improving generalization through model ensembling in federated learning. *arXiv preprint arXiv:2107.10663*.

[62] Shi, N., Lai, F., Al Kontar, R., & Chowdhury, M. (2023). : Ensemble Models in Federated Learning for Improved Generalization and Uncertainty Quantification. *IEEE Transactions on Automation Science and Engineering*.

[63] Gou, J., Yu, B., Maybank, S. J., & Tao, D. (2021). Knowledge distillation: A survey. *International Journal of Computer Vision*, *129*(6), 1789-1819.

[64] Hong, Y. W., Leu, J. S., Faisal, M., & Prakosa, S. W. (2022). Analysis of model compression using knowledge distillation. *IEEE Access*, *10*, 85095-85105.

[65] Yim, J., Joo, D., Bae, J., & Kim, J. (2017). A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4133-4141).

[66] Zhu, Y., & Wang, Y. (2021). Student customized knowledge distillation: Bridging the gap between student and teacher. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 5057-5066).

[67] Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, *22*(1), 79-86.

[68] Gao, D., Yao, X., & Yang, Q. (2022). A survey on heterogeneous federated learning. *arXiv preprint arXiv:2210.04505*.

[69] McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A. (2017, April). Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics* (pp. 1273-1282). PMLR.

[70] Abad, M. S. H., Ozfatura, E., Gunduz, D., & Ercetin, O. (2020, May). Hierarchical federated learning across heterogeneous cellular networks. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 8866-8870). IEEE.

[71] Liu, L., Zhang, J., Song, S., & Letaief, K. B. (2022). Hierarchical federated learning with quantization: Convergence analysis and system design. *IEEE Transactions on Wireless Communications*, *22*(1), 2-18.

[72] Deng, Y., Lyu, F., Ren, J., Zhang, Y., Zhou, Y., Zhang, Y., & Yang, Y. (2021, July). SHARE: Shaping data distribution at edge for communication-efficient hierarchical federated learning. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)* (pp. 24-34). IEEE.

[73] Wang, Z., Xu, H., Liu, J., Xu, Y., Huang, H., & Zhao, Y. (2022). Accelerating federated learning with cluster construction and hierarchical aggregation. *IEEE Transactions on Mobile Computing*.

[74] Shi, L., Shu, J., Zhang, W., & Liu, Y. (2021, December). HFL-DP: Hierarchical federated learning with differential privacy. In *2021 IEEE Global Communications Conference (GLOBECOM)* (pp. 1-7). IEEE.

[75] Lin, T., Kong, L., Stich, S. U., & Jaggi, M. (2020). Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems*, *33*, 2351-2363.

[76] Cho, Y. J., Manoel, A., Joshi, G., Sim, R., & Dimitriadis, D. (2022). Heterogeneous ensemble knowledge transfer for training large models in federated learning. *arXiv preprint arXiv:2204.12703*.

[77] Itahara, S., Nishio, T., Koda, Y., Morikura, M., & Yamamoto, K. (2021). Distillation-based semi-supervised federated learning for communication-efficient collaborative training with non-iid private data. *IEEE Transactions on Mobile Computing*, *22*(1), 191-205.

[78] Li, D., & Wang, J. (2019). Fedmd: Heterogeneous federated learning via model distillation. *arXiv preprint arXiv:1910.03581*.

[79] Chang, H., Shejwalkar, V., Shokri, R., & Houmansadr, A. (2019). Cronus: Robust and heterogeneous collaborative learning with black-box knowledge transfer. *arXiv preprint arXiv:1912.11279*.

[80] Maas, Andrew L. and Daly, Raymond E. and Pham, Peter T. and Huang, Dan and Ng, Andrew Y. and Potts, Christopher. (2011). Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* (pp. 142-150). Association for Computational Linguistics.

[81] Mitchell,T. (1999). Twenty Newsgroups. UCI Machine Learning Repository

[82] Lewis,David. (1997). Reuters-21578 Text Categorization Collection. UCI Machine Learning Repository.

[83] Gulli, A. (2004). http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html

[84] Hasibi, Faegheh and Nikolaev, Fedor and Xiong, Chenyan and Balog, Krisztian and Bratsberg, Svein Erik and Kotov, Alexander and Callan, Jamie. (2017). DBpedia-Entity V2: A Test Collection for Entity Search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '17) (pp. 1265-1268). ACM

[85] Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., & Xiao, J. (2015). Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*.

[86] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13* (pp. 740-755). Springer International Publishing.

[87] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

[88] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[89] Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700-4708).

[90] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4510-4520).