



University of
Zurich^{UZH}

A Solution for Decentralized Federated Multi-Task Learning

Nicolas Kohler
Zurich, Switzerland
Student ID: 16-712-887

Supervisor: Alberto Huertas and Chao Feng
Date of Submission: October 11, 2024

Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich, **11. October 2024**



Signature of student

Abstract

Die Kombination von Federated Learning (FL) und Multi-Task Learning (MTL) wurde für verschiedene Arten von heterogenen Clients angewendet. Einflussreiche Ansätze behandelten hierbei oftmals jeden Client als Task, was den Begriff Federated Multi-Task Learning (FMTL) ins Leben rief. Jedoch haben sich die meisten Ansätze auf das non-IDD Problem fokussiert, was dazu führte, dass Clients ähnliche Task Typen bearbeiteten (z.B. wenn alle Clients Bilderkennung betreiben), obwohl sie als einzelne Tasks im FMTL Kontext modelliert sind. Aktuelle Forschung hat sich diesem Thema in der Form von Heterogenous FMTL angenommen. Diese Studien haben sich jedoch mehrheitlich auf Centralized FMTL fokussiert, wo ein zentraler Server die Federation leitet. Dies öffnet eine Forschungslücke für Heterogeneous Decentralized FMTL, welches ohne eine zentrale Autorität auskommt. Diese Arbeit will diese Lücke adressieren und beginnt mit einer Bestandsaufnahme der existierenden Literatur in den Bereichen Centralized und Decentralized FMTL. Anschließend entwickelt die Arbeit ein Framework, welches fähig ist, heterogene Clients in einem dezentralisierten federalisierten Umfeld zu adressieren. Dafür lässt sich die Arbeit von MTL inspirieren und teilt die Modelle in private und aggregierbare Layer ein. Ähnliche Clients werden in Gruppen zusammengefasst, welche einen Gruppenführer stellen. Dieser führt konfliktvermeidende gruppenübergreifende Aggregationen durch. Die Arbeit simuliert zahlreiche Federationen, welche zeigen, dass das entwickelte Framework fähig ist, die verglichenen Aggregationsweisen in einem föderierten Umfeld mit Klassenlabel Heterogenität und Task Heterogenität zu übertreffen.

The combination of Federated Learning (FL) and Multi-Task Learning (MTL) has been applied to address various client heterogeneity challenges. To do this, influential approaches often treated clients as separate tasks, giving rise to Federated Multi-Task Learning (FMTL). However, much of the research has predominantly focused on addressing the non-IID problem, leading to clients solving similar task types (e.g. all clients are performing image classification), even when modelled as distinct tasks in an FMTL framework. Recent research has investigated this issue in the form of Heterogeneous FMTL. But these studies primarily concentrated on Centralized FMTL, where a central server oversees the federation. This leaves a gap for Heterogeneous Decentralized FMTL, where no central authority exists. The thesis sets out to address this gap by examining existing literature in the realm of Centralized and Decentralized FMTL. It then develops a framework capable of addressing clients with heterogeneous tasks in a decentralized federated environment. It does this by taking inspiration from MTL directly and splitting the models into private and aggregated layers. Similar clients form groups, with a group leader performing a conflict-averse cross-group aggregation. By simulating various federations, the thesis found that the framework is capable of outperforming the compared aggregation schemes in federated settings with class label and task heterogeneity.

Acknowledgments

I want to thank my supervisors, Alberto Huertas and Chao Feng. Their feedback during our bi-weekly meetings was very valuable and helped guide the thesis' direction. Additionally, I want to thank Dominik Kohler, Silvan Kohler and Catherine Lau for reviewing the grammar of my thesis prior to its submission. Lastly, I want to extend my thanks to Prof. Dr. Burkhard Stiller, the responsible supervising professor.

Contents

Declaration of Independence	i
Abstract	iii
Acknowledgments	v
1 Introduction	1
1.1 Motivation	1
1.2 Description of Work	2
1.2.1 Availability of Source Code and Trained Models	3
1.3 Thesis Outline	3
2 Background	5
2.1 Multi-Task Learning	5
2.1.1 Optimization in MTL	6
2.1.2 Modelling MTL	8
2.1.3 Leveraging Task Relationships	10
2.2 Federated Learning	11
2.2.1 Centralized vs Decentralized FL	12
2.2.2 Aggregation and Regularization	12
2.2.3 Heterogeneous Clients	13
2.3 A Brief Introduction to Federated Multi-Task Learning	15
2.3.1 A Primer in FMTL	15
2.3.2 Heterogeneity in FMTL	16
2.3.3 FMTL Categorization	16

3	Related Work	19
3.1	Methodology	19
3.2	Centralized Federated Multi-Task Learning	20
3.2.1	Convex Optimization	21
3.2.2	Model Convergence	22
3.2.3	Layer Privatization	23
3.2.4	Grouping	24
3.2.5	Knowledge Distillation	26
3.3	Decentralized Federated Multi-Task Learning	27
3.3.1	Regularization	27
3.3.2	Model Aggregation	28
3.4	Heterogeneous Centralized Federated Multi-Task Learning	30
3.4.1	Indirect Solutions	30
3.4.2	Direct Solutions	31
3.5	The Research Gap: HDFMTL	32
4	Architecture	35
4.1	Motivation	35
4.2	Framework	36
4.2.1	Intra-task Aggregation	37
4.2.2	Cross-task Aggregation	38
4.2.3	Additional Considerations	39
4.2.4	Comparison to existing work	40

5	Implementation	43
5.1	Overview of Code Repository	43
5.1.1	Configs Folder	44
5.1.2	Src Folder	45
5.2	Aggregation Schemes	47
5.2.1	No Aggregation (None-None)	49
5.2.2	Intra-Task Aggregation with FedPer (Fedper-None)	49
5.2.3	Intra-Task Aggregation with FedPer and Cross-Task Aggregation with FedPer (Fedper-Fedper)	49
5.2.4	Intra-Task Aggregation with FedPer and Cross-Task Aggregation with HCA (FedPer-HCA)	49
5.3	Available User Interactions	51
5.3.1	Run	51
5.3.2	Plot	51
5.4	Reproducibility	51
5.5	Limitations	52
5.5.1	Decentralization	52
5.5.2	Leader Selection, Faults, Stragglers and Late Joiners	52
6	Evaluation	55
6.1	Setup	55
6.1.1	Examined Metrics	56
6.1.2	Class Label Heterogeneity Setup	57
6.1.3	Task Heterogeneity Setup	59
6.2	Impact of Aggregation Frequency on the Framework	61
6.2.1	Aggregation Frequency and Class Label Heterogeneity	61
6.2.2	Aggregation Frequency and Task Heterogeneity	62
6.3	Impact of the Privatization Degree on the Framework	64
6.3.1	Degree of Privatization and Class Label Heterogeneity	66

6.3.2	Degree of Privatization and Task Heterogeneity	66
6.4	Evaluation of Different Aggregation Schemes	69
6.4.1	Aggregation Scheme and Class Label Heterogeneity	70
6.4.2	Aggregation Scheme and Task Heterogeneity	71
6.5	Discussion	74
6.5.1	Challenges	74
6.5.2	Main Experimental Findings	76
7	Summary and Conclusions	79
7.1	Summary	79
7.2	Limitations and Potential Future Work	80
	Bibliography	83
	Abbreviations	95
	Glossary	97
	List of Figures	98
	List of Tables	101
A	Installation Guidelines	105
B	Additional Plots	107

Chapter 1

Introduction

Federated Learning (FL) has gathered a lot of research interest in the last few years. The aim of FL is to train Machine Learning (ML) models on a set of clients, by sporadically exchanging model updates with each other via a central server [1]. Since this approach does not require the exchange of raw data samples, it is a viable strategy when privacy and data protection is key, such as in the field of medicine [2]–[5]. Decentralized Federated Learning (DFL) takes this a step further and foregoes the central server completely, building a decentralized federated environment and removing a potential single point of failure [6].

One way to approach this setting is by taking inspiration from Multi-Task Learning (MTL). The general idea of MTL is to train on different but somewhat related tasks simultaneously, while using sophisticated mechanisms to let tasks influence each other [7]. The aim is, that the presence of related tasks positively impacts the model performance [8], by leading to better generalization, less over-fitting and addressing the data sparsity problem [9]. While MTL is usually conducted on one machine, it can be extended into a federated environment. [10] developed one of the first and most influential studies to use MTL techniques in FL, and popularized the term Federated Multi-Task Learning (FMTL). They considered each client of the federation to be its own task, essentially making FMTL a sub problem of Personalized Federated Learning (PFL) [11]. Since then various studies have been conducted on Centralized FMTL (CFMTL) and more recently Decentralized FMTL (DFMTL).

1.1 Motivation

Most existing research in CFMTL and DFMTL aims to tackle the issue of training data that is non-independently and non-identically distributed (Non-IID) among clients, a well-known challenge in FL [12]. Examples of non-IID data in FL include attribute skew (differences in feature distribution), label skew (differences in label distribution), and temporal skew (time-based distribution changes), each posing distinct challenges for model training across clients [13]. However, most studies fail to consider more profound heterogeneous clients [14]. For instance, it is usually required that every client is aware of all

potential class labels even in non-IID settings. However, in non-IID settings, clients might not encounter all potential class labels in their data. This is especially true if a client has no interest in learning to predict certain types of labels. Additionally, sharing class labels amongst clients might add to the myriad of potential privacy risks [15]. This problem can potentially be severe, as it can lead to a client joining a federation with a model that holds a different amount of output features. In this scenario, techniques like federated averaging [16] will fail, as they usually require model homogeneity [17]. By extending this thought, one can see that clients might not only differ in their class labels, but in their tasks overall [11]. One client might solve an image classification problem, while another is focused on point detection. This, again, leads to potentially different models and adds complexity. In addition, a major problem in such a setting is that different tasks result in different model parameters, leading to the problem of conflicting and dominant gradients [18]. While FMTL, by its name, implies that it deals with such heterogeneous tasks, most FMTL research is still focused on conventional non-IID problems. That is also true for the paper [10], which is one of the most influential in this field. While [10] considered each client to be their own task, their goal was still to address non-IID problems and not heterogeneous task objectives between clients. As such, it is crucial that research in the field of FMTL considers such heterogeneous clients, making it more aligned with the general idea of MTL. Some solutions for dealing with very Heterogeneous clients have been proposed [11], [14], [19], but they seem to focus on heterogeneous CFMTL (HCFMTL). This leaves a gap for Heterogeneous DFMTL (HDFMTL).

1.2 Description of Work

To investigate HDFMTL the thesis establishes various definitions that guide an in-depth literature review. The focus of the literature review is to explore existing research in CFMTL, DFMTL as well as in HCFMTL and HDFMTL. Based on this, a framework is proposed to address these heterogeneity concerns in a DFMTL environment. The framework divides the clients into different groups, based on what tasks the clients solve. After a predefined amount of epochs, the clients exchange their backbones (i.e. non privatized layers) within the task group and perform federated averaging on their backbones, a technique discussed by [20]. Afterwards, the leader of such a task group exchanges the averaged backbone with leaders of other task groups, and performs a simplified version of the hyper conflict-averse aggregation scheme, proposed by FedHCA² [11]. This allows to minimize impacts of conflicting and dominating gradients. The thesis describes various experiments which it ran on two different datasets. The CIFAR-10 dataset [21] is used to study heterogeneous class labels, while the CelebA dataset [22] is utilized to model a heterogeneous task settings. Before comparing the proposed framework to other aggregation schemes, the thesis discusses two preliminary experiments which it conducted. First, it analyzes the impact of aggregation round frequency on the proposed framework and finds that more frequent aggregations improve model performance. Second, it explores various levels of layer privatization and discovers that keeping most layers shared is advantageous in the class label heterogeneity setup. In contrast, increasing the number of privatized layers slightly improves performance in task heterogeneity scenarios. Finally, the thesis compares the proposed framework with other aggregation schemes, demonstrating that it

outperforms them in both the modeled class label and task heterogeneity scenarios.

The main contributions can be summarized as follows: It presents a detailed literature review of current FMTL and DFMTL research with a focus on heterogeneous clients. It develops and implements a framework to handle clients with heterogeneous class labels and tasks. Finally, it evaluates the framework by analyzing the effects of aggregation round frequency, layer privatization and compares its performance against other aggregation schemes.

1.2.1 Availability of Source Code and Trained Models

The source code as well as all trained models and corresponding figures are publicly available under the following two links:

Code Repository:

<https://github.com/nicolas1a2b/asfdfmtl>

Models and Figures:

https://drive.google.com/drive/folders/1Au6neZziuD0q4_pd3T8q06-kMceEvVpK?usp=sharing

Please note that the availability of the code and trained models associated with this thesis cannot be guaranteed indefinitely.

1.3 Thesis Outline

The thesis is structured into various chapters. Chapter 2 builds a theoretical basis by covering MTL, FL and the combination of the two in FMTL. This is followed by the literature review in Chapter 3, that studies CFMTL, DFMTL and HCFMTL. It also outlines the research gap, that is HDFMTL. Afterwards, Chapter 4 describes the developed framework, for which the implementation is described in Chapter 5. Chapter 6 introduces the conducted experiments evaluates the framework and summarizes its main findings. It also lists challenges faced and how they were addressed. Finally, the last Chapter 7, provides a concise summary of its contributions, highlights its limitations, and suggests potential areas for future research.

Chapter 2

Background

Developing a solution for DFMTL requires a profound understanding of both MTL and FL. As such, this chapter focuses on providing an introduction into these two distinct topics. Finally the concepts will be brought together to establish a definition of DFMTL.

2.1 Multi-Task Learning

MTL is an emerging and already extensive sub-field of ML, in which the main goal is to train a model on multiple tasks simultaneously. This is in contrast to "standard" single-task ML, where a model is trained to perform well on one specific task. As such, a major benefit of MTL is its capability to leverage data and information from related tasks, which might be beneficial to the model's overall performance [8]. This makes MTL a great solution to the data sparsity problem and can lead to better generalization and lower overfitting [9].

In MTL, one can distinguish between conventional MTL and heterogeneous MTL. Conventional MTL generally involves tasks of the same nature, such as all being classification or regression tasks, which enables effective shared feature learning [23]. In contrast, heterogeneous MTL deals with a mix of task types, like classification, detection and regression tasks trained simultaneously, adding complexity to the shared representation learning [23], [24].

Research in the area of MTL is extensive, with new techniques and architectures constantly emerging. This inspired recent surveys to develop taxonomies to capture the wide range of techniques in MTL. [25] distinguished between architectures belonging to the fields of computer vision, natural language processing, reinforcement learning as well as some more unique architectures that can be learnable, conditional or belong to multiple modalities. [26] distinguished between encoder and decoder focused MTL, where encoder focused MTL is sharing information on the encoder level, while decoder-focused MTL extends this to the decoder level. The survey of [18], seems to be the most recent one and provides a very in-depth and comprehensive review of MTL, covering the traditional, deep learning and foundational eras. [18] covered similar topics as [25]. They group the existing MTL

methods into regularization, relationship learning, feature propagation, optimization and pre-training.

These surveys are used heavily throughout this section to provide a general overview of the MTL field. To do this, the section is grouped into three subsections. The first elaborates on optimization and its challenges in the context of MTL. The second covers general approaches and MTL architectures. Lastly, the third describes how relationships between tasks can be leveraged, for instance by grouping tasks together. It is important to note that, this background section does not aim to provide an all encompassing overview of MTL. Instead, it focuses on introducing the concepts required to follow the later chapters. For an in-depth review of MTL, the two surveys provide a good read [18], [25].

2.1.1 Optimization in MTL

In ML, the derivative (gradient) of the loss function, with respect to the model parameters, is used to update the model parameters in a direction that minimizes the loss. The general update rule of the model parameters with respect to the loss and learning rate is given in equation 2.1.

$$W \leftarrow W - \eta \nabla L(W) \quad (2.1)$$

This fundamental technique is also applied in MTL. However, MTL aims to train models on multiple tasks simultaneously, necessitating the combination of individual task-specific loss functions into a single aggregated loss function. This results in a problem, that can be framed as a weighted linear combination of loss functions of different tasks [18]. This is shown in equation 2.2.

$$\min_W \mathcal{L}_{\text{total}}(W) = \sum_{t=1}^T \alpha^{(t)} \mathcal{L}^{(t)}(W) \quad (2.2)$$

Here, $\alpha^{(t)}$ represents the task weights (i.e. preferences), W represents the model parameters and $\mathcal{L}^{(t)}$ represents the loss functions. However, choosing the correct $\alpha^{(t)}$ poses a great challenge, especially when considering dominant and conflicting gradients. These issues cause negative transfer, a situation, in which training multiple tasks together adversely affects the learning process, rather than improving it [25]. [18] provided a great overview of the problem, which is visualized in Figure 2.1. They state that the dominating gradient issue occurs, when the gradients of certain task losses have significantly larger norms than others, causing the update direction to be biased towards these tasks with larger gradients. In contrast, the conflicting gradients issue arises, when progress in one task degrades the performance of another task. Prominent solutions to these problems are loss weighting techniques and direct the modulation/correction of gradients [18], [25].

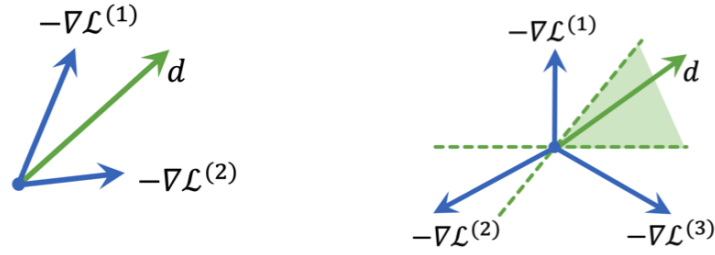


Figure 2.1: Dominant and Conflicting Gradients. Illustration by [18]

Loss Weighting

As hinted, loss weighting is one potential solution to the problem of conflicting and dominating gradients. While both surveys [25] and [18], discussed loss weighting, [18] focused specifically on addressing dominating and conflicting gradients. To provide a general idea of how loss weighting can be used to address these problems, the thesis picked three examples:

The most well known approach is GradNorm [27]. It ensures that each task’s gradients have the appropriate magnitude by averaging the gradient norm of a task and its relative learning progress [18]. Contrary to this, DWA [28] focuses on applying weighting by learning speed [25]. It adjusts task loss weights dynamically based on the relative training speed of each task, assigning higher weights to tasks that are learning slower, to ensure balanced training progress. Lastly, identified by [25] and [18], uncertainty weighting [29] aims to model the network probabilistically and tries to create a loss function, that gives more weight to tasks with less uncertainty.

Gradient Modification

Compared to loss weighting, one can also directly modify the gradients. Similar to the research of [25] and [18], the thesis distinguishes between three main modification techniques: Adversarial gradient modulation, gradient replacement and gradient correction.

Adversarial gradient modulation aims to align the gradients of related tasks in MTL by using an adversarial loss term [25]. This term encourages the gradients from different tasks to be similar. However, [25] stated that, while the idea makes sense, there are questions about its theoretical basis. Gradients of related tasks may not always need to be similar, and they can change during training.

Gradient Replacement focuses on replacing conflicting with non-conflicting gradients. An example mentioned by [25] is gradient episodic memory [30]. In that technique the task’s gradient is replaced, if it conflicts with previously learned tasks by solving a quadratic optimization problem to find a non-conflicting gradient. Another example was proposed in the framework PCGrad [31]. There, conflicting gradients are projected into the orthogonal subspace to mitigate conflicts, ensuring that the new gradient direction does not conflict with other task gradients.

Gradient correction, as defined by [18], is a technique that modifies the existing gradients to reduce conflicts, without replacing them entirely. Based on the research of [25] and [18], a few examples that apply this technique can be mentioned: GradDrop [32]: Focuses on the signs of gradients and uses probabilistic masking to retain only gradients with consistent signs. CAGrad [33]: Solves an optimization problem to find a new gradient direction that minimizes conflicts while staying close to the original total gradient. Recon [34]: Analyzes gradients layer-wise to identify and adjust parameters that may cause conflicts, converting certain parameters to task-specific ones if necessary.

2.1.2 Modelling MTL

To better give an idea of how an MTL framework looks, the thesis will provide an overview of some common techniques to approach and model a MTL problem. While historically, MTL has been divided into hard and soft parameter sharing [8], these days more elaborate techniques exist. This motivated recent surveys, such as [25], [18] and [26] to investigate the broader field of MTL. As such, the thesis first introduces hard and soft parameter sharing, followed by some additional important techniques, discussed by the aforementioned surveys. It is worth noting that overlaps between these techniques can exist. An overview of these concepts can be found in Table 2.1.

Table 2.1: Architectural MTL Concepts, Based on [25] and [18]

Concept	Description
Hard Parameter Sharing	Using a common backbone to extract features for multiple tasks.
Soft Parameter Sharing	Regularizing separate networks to ensure parameter similarity.
Cross-Talk	Feeding the output of a network's layer directly into parallel layers of other networks during training.
Cascading	Modeling tasks hierarchies, where the output of one model is fed into a subsequent model.
Knowledge Distillation	Training a smaller model to replicate a larger model's behavior for similar performance with fewer resources by either employing feature-level distillation or response-level distillation.

Hard Parameter Sharing

Probably the most intuitive approach is hard parameter sharing. In it, the network is divided into shared layers and task specific heads. As such, the initial model parameters are shared for multiple tasks, with each task keeping a model specific head. This approach has also been called "Shared Trunk" [25]. Given the general idea of a shared feature extractor followed by task specific heads, it can be seen as encoder-focused [26]. [25] listed several examples for this approach in the realm of image processing, such as

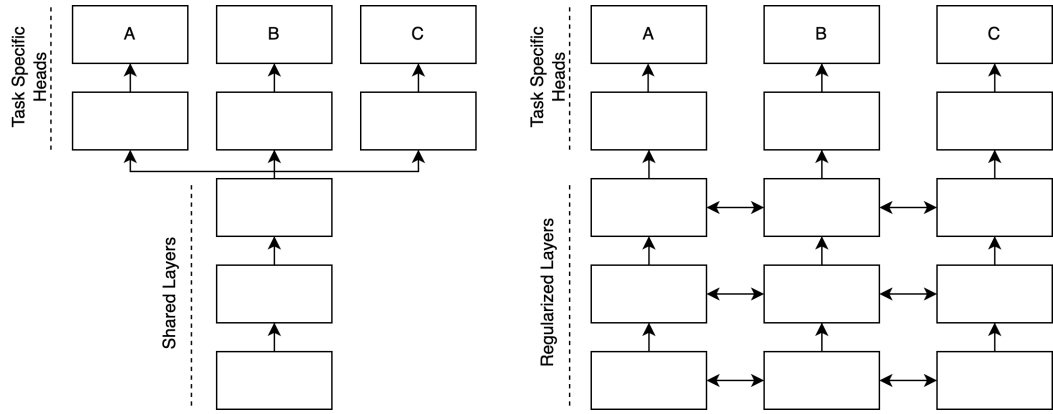


Figure 2.2: Depiction of Hard and Soft Parameter Sharing Inspired by [8]

the Tasks-Constrained Deep Convolutional Network [35], which was used for facial landmark detection. Multi-task Network Cascades [36], are seen as a variation of the shared trunk approach, where the output of each task-specific branch is fed into the input of the subsequent task-specific branch, creating the "cascade". This cascading will be discussed in a bit more detail in one of the following subsections. Multi-gate Mixture-of-Experts [37] is another architectural variation mentioned. In it, the network has multiple shared trunks, with each task-specific output head receiving inputs as a weighted combination of outputs from these trunks. A depiction of hard parameter sharing can be seen in Figure 2.2.

Soft Parameter Sharing

Another well known technique is soft parameter sharing. Here each task has its own model parameters. Knowledge is shared by regularizing models, so that the parameters are similar. As such, regularization is an important concept, especially in soft parameter sharing [25]. It encourages similarity by adding a penalisation term to the training objective that minimizes the distance between the parameters of different task models. [18] provided an in-depth summary of existing regularization techniques. [8] and [26] provided a great overview of how soft and hard parameter sharing differ. Soft parameter sharing, is depicted in Figure 2.2.

Cross-Talk

Cross-talk, as defined by [25], seems more related to soft parameter sharing. Similar to soft parameter sharing, cross-talk involves having distinct networks for each task. However, while in traditional soft parameter sharing the link between the networks is usually based on regularization, cross-talk is more intertwined, as outputs from one layer can be fed to the parallel layers directly during training [25]. This technique was hinted at by [18] as well, but, they refer to it as feature fusion, which covers a broader spectrum of approaches. Regardless, both, [25] and [18], listed the Cross Stitch Network [38], the Sluice Network [39] and the Neural Discriminative Dimensionality Reduction CNN [40] as examples.

Cascading

Compared to the previously introduced approaches cascading has the potential to take task hierarchies into account [25]. Cascading is especially beneficial in NLP, as tasks can be cascaded based on their hierarchy [25]. For instance, one can first conduct part of speech tagging, followed by a question answering system. [18] identified the following examples: Multi-task Network Cascades [36], Joint Many-Task Model [41], Deep Cascade Multi-Task Learning [42] and Hierarchical Multi-Task Learning [43].

Knowledge Distillation

Lastly, an important technique that warrants some close inspection is knowledge distillation. In MTL, knowledge distillation is a technique, where a teacher model guides a student model by transferring meaningful knowledge, such as soft labels. It involves first training the teacher model on multiple tasks to become an expert, and then training the student model to replicate the teacher’s behavior [25]. [18] distinguished between two different levels of knowledge distillation in MTL: Feature level distillation and response-level distillation.

Feature-level distillation involves transferring knowledge from task-specific networks to a multi-task network by aligning and integrating their features. This technique leverages the specialized capabilities of individual networks to enhance the performance of a unified model, capable of handling multiple tasks. Two notable methods in this context are KD4MTL [44] and OKD-MTL [45]. For instance, in KD4MTL [44], task-specific networks are trained offline, meaning each network is trained independently on its respective task. That means that separate networks might be trained for image classification and object detection. Once these task-specific networks are trained, their learned features are used to train a multi-task network. During the training of the multi-task network, a loss function is employed to minimize the distance between the features generated by the task-specific networks and those from the multi-task network.

In response-level distillation, approaches such as MuST [46] train specialized teacher models for specific tasks and transfer their knowledge to a general-purpose student model through pseudo-labeling. Pseudo-labeling is a technique where a model trained on a labeled dataset is used to generate labels for an unlabeled dataset. These generated labels, known as pseudo-labels, are then used to train the model further, as if they were true labels.

2.1.3 Leveraging Task Relationships

Having discussed how an MTL problem can be modelled, it is also important to consider how relationships between tasks can be leveraged to aid MTL. This is sometimes also referred to as representation learning, where the goal is to develop a representation of task relationships. This involves clustering tasks based on their similarity and using these learned relationships to enhance the learning process [18], [25].

[25] investigated multiple methods that group tasks based on whether negative transfer is likely to occur or not. In simpler terms, that means that similar tasks are grouped together while very different tasks are trained separately. [18] also investigates task grouping and divides them into horizontal cluster and hierarchical trees. Works listed by [25] include [47]–[51].

[47] and [48] explored task grouping in natural language processing, showing that auxiliary tasks with high entropy label distributions improved main task performance. Later, [49] continued this research and found consistent improvement in computer vision tasks. Another approach is selective sharing [50]. It addresses task grouping by clustering tasks based on gradient similarity during training. Lastly, [51] provided an empirical study on grouping.

Another way to group tasks is by making use of graph based methods. These methods are widely used in data mining and ML due to their ability to effectively represent objects and their interactions [18]. Graph neural networks are able to model relationships among nodes within graphs, by treating tasks as nodes and relationships between tasks as edges. [18] mentioned MultiKernel [52] as an example. It applies MTL to a series of classification tasks with predefined hierarchical relations, typical in biological problems. It constructs a tree reflecting these relations, with tasks as leaf nodes and their biological domains as non-leaf nodes. A more recent example is ML-GCN [53]. This graph convolutional model is designed for multi-label image recognition. It achieves this by training a CNN for image representations and a graph convolutional network for label representations, which are then combined to make multi-label predictions.

2.2 Federated Learning

ML usually requires a lot of centrally located data. However, data is not always stored at one entity. It might be spread across multiple organisations, each with their own application and privacy concerns. [54] referred to this phenomena as "data islands". Data between such islands can not be shared, due to various regulations such as the European general data protection regulation (GDPR) [55]. A common example to illustrate this problem and motivate the use of federated learning can be found in the medical field [2]–[5].

Medical institutions might want to employ ML to conduct medical imaging. However, the performance of such ML models will be limited by the amount of available patient data one institution holds. This data is especially limited, as sharing such highly sensitive data among institutions can be in violation with privacy and data protection regulations, like the previously mentioned GDPR [55]. FL has emerged as a solution to this problem and gained a lot of research interest. FL describes a paradigm that enables multiple nodes to train one or many models, while keeping their training data local and private [56]. Given this definition, organizations can leverage the knowledge of other institutions to further improve their own models. However, one has to define how these nodes are connected, how they communicate, how they propagate model updates across the federation, and how they combine information. There exist two main networking approaches to build such a

federation, centralized and decentralized [6]. Centralized and decentralized federations usually apply techniques like regularization, model aggregation or a combination of the two [57].

2.2.1 Centralized vs Decentralized FL

As mentioned, FL can be distinguished into two main approaches: Centralized FL (CFL) and Decentralized FL (DFL). In CFL, a central server manages communication, coordination, and aggregation for all clients [58]. Clients train on local data and send their model parameters to the server, which aggregates them into a global model and passes it back to the clients, which apply it [59]. While CFL requires a central server to collect and aggregate parameters of the local models, DFL pushes the aggregation step towards the nodes themselves [60]. As such, DFL does not require a central server. In fact, in DFL the clients manage communications with other nodes and parameter aggregation independently, resulting in a fully autonomous network [6]. Some frameworks fall between the two approaches and are sometimes referred to as Semi-DFL. In Semi-DFL architectures, the server role periodically changes from one client to another, based on some predefined criteria [6]. Figure 2.3 created by [6] depicts these three different architectures.

CFL comes with certain limitations like single points of failure [61], trust dependencies, and server bottlenecks [6]. DFL addresses these limitations. It improves fault tolerance, distributes trust among federation nodes, reduces single points of attacks and mitigates bottlenecks by distributing communication and workload evenly across nodes [6]. Additionally, DFL enables self-scaling federations, efficient resource utilization, and prevents congestion, making it more adaptable and resilient than CFL [6].

However, DFL also has some apparent drawbacks. For one, DFL may experience increased communication overhead, which requires optimizing communication protocols, client selection, and trust mechanisms [6]. The lack of proper management via a central server also introduces challenges and can lead to the confusion of clients [58]. Lastly, proper client selection seems especially important, given that DFL faces an increased risk exposure to potentially malicious nodes, caused by the lack of a central regulatory authority [58]. These malicious clients might harm the federation by mounting model inversion attacks, allowing the reconstruction of private data [59].

2.2.2 Aggregation and Regularization

While there are many different techniques on how information in a federated setting can be leveraged, two are very often used: Aggregation and regularization. Aggregation combines model updates from multiple clients to create an aggregated model [57]. It involves techniques like federated averaging (e.g. FedAvg [16]), which averages the complete model and FedPer [20], which only aggregates part of a model. More advanced techniques include weighted averaging and attentive aggregation [57]. Compared to this, regularization techniques add constraints to either local client objectives or aggregation formulas [57]. Notable examples include FedProx [62], which introduces proximal terms to regularize

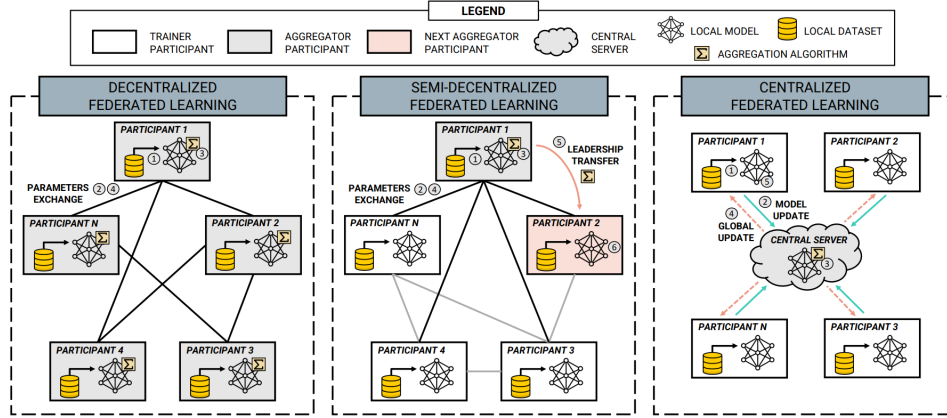


Figure 2.3: CFL, DFL and Semi-DFL as Depicted by [6]

local training and FedU [63], which uses laplacian regularization. Aggregation and regularization are not exclusive, meaning that some studies combine the two techniques. It is important to keep these terms in mind, as they will be often referenced in the literature review, in combination with additional examples.

2.2.3 Heterogeneous Clients

Clients within a federated setting can be similar, but do not have to be. As such, heterogeneous clients pose a major challenge in FL [58]. Many research papers have discussed such heterogeneous clients, with [64] being one of the most holistic reviews. According to [64], there exist five main sources of heterogeneity: Data heterogeneity, model heterogeneity, task heterogeneity, communication heterogeneity and device heterogeneity. Table 2.2 depicts these types.

Table 2.2: Heterogeneous Clients as Identified by [64]

Heterogeneity Type	Description
Data Heterogeneity	Differences in distribution, label, feature, quantity, and quality skews
Model Heterogeneity	Variations in model architecture and optimization algorithms
Task Heterogeneity	Differences in objectives or tasks
Communication Heterogeneity	Discrepancies in network bandwidth, latency, and connection stability
Device Heterogeneity	Variability in computing power, storage, and energy supply

Data Heterogeneity

Data heterogeneity is one of the main challenges of FL, and is sometimes also referred to as a statistical heterogeneity [10]. [64] provided an in-depth analysis of data heterogeneity and differentiated between distribution, label, feature, quantity, and quality skew. Distribution skew describes the general skew of data held by clients and can easily be modelled via the multivariate probability dirichlet distribution [65]. Label skew is described as a skew in which clients only have data of a subset of all classes. This could already be regarded as a light task heterogeneity. Feature skew refers to a skew in data features. For this, [64] gave an example with images of dogs. The feature skew arises as some clients see proper pictures, while others see painting or cartoons. Lastly, the quantity and quality skews are concerned about the skew in the amount of data per client and the quality of said data.

Model Heterogeneity

[64] described two main forms of model heterogeneity: Heterogeneous model structures and heterogeneous optimization algorithms.

The structure of models solving the same task can vary, as some clients may be more resource-constrained than others. This might result in resource-constrained devices, like IoT devices, training shallower models compared to more powerful servers [64]. In addition, models may differ in the optimization algorithms they use. Some devices might use a lightweight optimizer like stochastic gradient descent, while more powerful machines might opt for ADAM [64].

Task Heterogeneity

While the focus of the previous section was on model heterogeneity triggered by devices with differing resource constraints, model heterogeneity can also be triggered by clients solving different tasks. Once again the thesis refers to [64], as they provide a great overview of different task heterogeneity levels. The first level of task heterogeneity still assumes model homogeneity. The task heterogeneity is solely triggered by clients using different class labels, and as such, is referred to as class label heterogeneity. However, this class label heterogeneity can potentially extend to model heterogeneity when the amount of classes to predict differ, leading to differences in the final fully connected layer of neural networks. Lastly, task heterogeneity can also take an extreme form, in which the tasks are so different, that they require fundamentally different models. For instance one client might be solving a regression task while another is focused on classification. Given that task heterogeneity, often implies model heterogeneity, techniques like MTL, knowledge distillation, transfer learning and similarity calculations can be applied [64].

Communication Heterogeneity and Device Heterogeneity

Communication heterogeneity and device heterogeneity, identified and discussed by [64], are bundled together, as they are somewhat similar. Communication heterogeneity refers to differences in devices' network bandwidth, latency, and connection stability, while device heterogeneity is caused by variations in computing power, storage, and energy supply [64]. As such, high-performance devices with high bandwidth and low latency can handle complex models quickly, while low-performance devices may become bottlenecks and stragglers, impacting and potentially delaying the learning process. As previously discussed, device and to some extent communication heterogeneity, can be seen as a trigger for model heterogeneity.

2.3 A Brief Introduction to Federated Multi-Task Learning

Having discussed FL and MTL separately, it is now time to combine the two concepts into FMTL. This section will provide an overview of important concepts and establish definitions that will guide the literature review.

2.3.1 A Primer in FMTL

[10] was one of the first papers to use MTL to address the client data heterogeneity challenge. By doing that, they established and shaped the notion of Federated Multi-Task Learning (FMTL). Their approach was motivated by the observation, that each node generates their own, but potentially similar, data distribution. As such, they decided to have each node train a local single task model, while capturing the relationship with other nodes with a MTL Framework. They defined that learning one local model is considered a task. This means that the learning of multiple models on multiple nodes at the same time translates to MTL [63]. Since then, many studies such as [56], [63], [66]–[71] have made use of this definition, which essentially results in FMTL being a branch of Personalized Federated Learning (PFL) [72], [73]. The goal in PFL is not to learn global model, but train a personalized model for each client [74]. Note that, the aforementioned definition of FMTL, where each client is considered a task, does not necessarily mean that a client is only solving one task. Recent studies [11], [75] started investigating clients, which solve multiple tasks simultaneously, effectively solving a MTL problem locally, while modelling the federation as a MTL problem.

Like FL, FMTL can be conducted in a centralized, decentralized or semi-decentralized setting. The benefits and drawbacks of each approach are similar to what has been discussed in in 2.2.1. Generally, Centralized FMTL (CFMTL) has seen a tremendous amount of research, while Decentralized FMTL (DFMTL), remains relatively niche.

2.3.2 Heterogeneity in FMTL

As was discussed earlier, heterogeneous clients are one of the major challenges in FL. [10] was one of the first papers to use MTL to address the client heterogeneity challenge. However, they and many other such as [56], [66], [67], [69] only considered client data heterogeneity [72]. Since clients are referred to as tasks, [10] described data heterogeneity under the umbrella term of task heterogeneity. As such, some papers explicitly consider data heterogeneity, but implicitly assume model and task homogeneity. However, in real life FL settings this assumption does not always hold. [76] stated that, especially federations containing cross-silo participants (clients belonging to different organizations), have to deal with model, task and training process heterogeneity. As such, it makes sense to consider different heterogeneity types for FL clients, as outlined in Section 2.2.3. Doing this, will make it more in-line with the notion of conventional and heterogeneous MTL introduced in Section 2.1.

Inspired by the above and research like [11], [14], [76], the thesis extends these heterogeneity consideration to FMTL, leading to "conventional" FMTL and heterogeneous FMTL. For the purpose of this thesis, conventional FMTL refers to research that handle clients with the same overarching task. Research focusing on heterogeneous data distribution (i.e. non-iid) still falls under this term, as clients might still solve the same general task. Examples that are covered by this definition of conventional FMTL include the influential MOCHA [10] and OFMTL [67] frameworks. Heterogeneous FMTL refers to research that properly investigates different task settings. Existing work in that area are NTKFedAvg [19] and FedHCA² [11]. Similar to heterogeneous MTL this includes clients with different task objectives. For instance, some clients might focus on face attribute classification while others perform face landmark point detection [77].

2.3.3 FMTL Categorization

Having talked about different heterogeneity types and their importance for CFMTL and DFMTL, it is crucial to establish proper definitions in the context of this thesis. This enables the thesis to ensure a consistent vocabulary for the later sections and allows to structure the literature review. The following four definitions combine the two main federated networking types with heterogeneous and non-heterogeneous settings.

Definition 1 - Centralized Federated Multi-Task Learning: *In Centralized Federated Multi-Task Learning (CFMTL) a centralized node is responsible for the coordination and/or aggregation of network nodes and their respective models. Each node in the network learns a model based on its own data, while taking its relationship with other nodes into account. While heterogeneous data between clients can exist, the task objectives are the same.*

Definition 2 - Heterogeneous Centralized Federated Multi-Task Learning: *In Heterogeneous Centralized Federated Multi-Task Learning (HCFMTL) a centralized node is responsible for the coordination and/or aggregation of network nodes and their respective models. Each node in the network learns a model based on its own data, while taking its relationship with other nodes into account. The task objectives among clients differ.*

Definition 3 - Decentralized Federated Multi-Task Learning: *In Decentralized Federated Multi-Task Learning (DFMTL) nodes communicate directly with each other, without relying on a central server to conduct coordination and/or aggregation. Each node in the network learns a model based on its own data, while taking its relationship with other nodes into account. While heterogeneous data between clients can exist, the task objectives are the same.*

Definition 4 - Heterogeneous Decentralized Federated Multi-Task Learning: *In Heterogeneous Decentralized Federated Multi-Task Learning (HDFMTL) nodes communicate directly with each other, without relying on a central server to conduct coordination and/or aggregation. Each node in the network learns a model based on its own data, while taking its relationship with other nodes into account. The task objectives among clients differ.*

Chapter 3

Related Work

The goal of this chapter is to provide a current overview of FMTL research. First, the methodology applied to identify related work is described in 3.1. The thesis then reviews a sample of existing CFMTL literature in Section 3.2. This is followed up by a more complete review of DFMTL and HFMTL, in Section 3.3 and 3.4 respectively. Lastly, the research gap is discussed in combination with HDFMTL in Section 3.5.

Please note, that this chapter uses the names of the proposed frameworks whenever possible. If a framework name could not be identified the thesis abbreviates the title. This ensures a consistent structure in the literature review.

3.1 Methodology

As a starting point Google Scholar [78] was consulted. To get a broad overview of the field, it was decided to search for all work that contains the term "Federated Multi-Task Learning" in its title. This resulted in a huge amount of papers (>100). Based on their title and abstract, it was decided whether or not they are relevant for the thesis. Relevant papers were categorized according to the four established definitions in 2.3.3. This comes with a caveat: It is not always straight forward to put the research in its respective category. This is especially true for differentiating between CFMTL and HCFMTL.

As per the established definition, work belonging to CFMTL might address data heterogeneity challenges such as non-IID data distributions, but fails to take different task or model architectures into account. The notion that one client shall be considered a task, as introduced by [10], makes the distinction between CFMTL and HCFMTL challenging. That is, because some research mixes the terms task heterogeneity and data heterogeneity. For them, different data distributions automatically translate to different tasks, violating the established definition of HCFMTL in Section 2.3.3.

As such, the very broad approach is error prone and might miss important work. Hence, two additional databases with specific key terms were consulted. For this, IEEE Explore [79] and Arxiv [80] were chosen. To properly cover more specific aspects relevant to the

thesis, the following key terms were applied to the abstract in combination with FMTL: decentralized, heterogeneous and knowledge distillation. Table 3.1 provides an overview of the specific search terms. Again the identified research was categorized. Lastly, during the literature review process, some additional work was identified by following citations etc. A lot of research falls within the scope of CFMTL or focus on clients implementing a Multi-Task model, rather than treating the federation as a Multi-Task problem. As such, these works are not the primary focus of this thesis and were not reviewed in detail. Nevertheless, a selection of influential studies within the CFMTL domain was properly reviewed and is highlighted in Section 3.2 to showcase various applied techniques. For the other categories, which are more important to the goal of this thesis, all identified research was properly reviewed and is discussed in Section 3.3, 3.4 and 3.5. Figure 3.2 provides an overview of the literature reviewed in detail.

Table 3.1: Database and Search Queries

Database	Search Query
Google Scholar [78]	"allintitle: Federated Multi-Task Learning"
IEEE [79]	("Document Title":Federated Multi-Task Learning) AND ("Abstract":Decentralized)
	("Document Title":Federated Multi-Task Learning) AND ("Abstract":Heterogeneous)
	("Document Title":Federated Multi-Task Learning) AND ("Abstract":Knowledge Distillation)
Arxiv [80]	order: -announced_date_first; size: 50; include_cross_list: True; terms: AND title=Federated Multi-Task Learning; AND abstract=Decentralized
	order: -announced_date_first; size: 50; include_cross_list: True; terms: AND title=Federated Multi-Task Learning; AND abstract=Heterogeneous
	order: -announced_date_first; size: 50; include_cross_list: True; terms: AND title=Federated Multi-Task Learning; AND abstract=Knowledge Distillation

Table 3.2: Reviewed Research

Category	Amount	Studies
CFMTL	29 +	[10], [67], [81], [69], [82], [83], [84], [70], [85], [86] [20], [87] [88], [71], [89], [66], [90], [91], [92], [93], [94], [95], [96], [97], [98], [99] [100], [101], [102]
DFMTL	8	[103], [104], [63], [105] [68], [106], [56], [107]
HCFMTL	10	[11], [14], [19], [76], [108], [109], [110], [111], [112], [113]
HDFMTL	0 ?	Apparent Research Gap

3.2 Centralized Federated Multi-Task Learning

CFMTL was first introduced in 2017 [10]. Since then, a plethora of frameworks for various use cases have emerged. By reviewing relevant papers and by making use of

existing surveys [64], [114], it was possible to identify some characteristics. Early research was mainly focused on solving convex problems, not suitable for modern non-convex deep learning approaches. This was addressed in later work, which extended CFMTL to non-convex approaches. Some research focused on aggregating the full client models or introducing regularization terms. Others took inspiration from traditional MTL and divided the models into shared and private layers. Another very common approach is to create groups or clusters of similar clients. Lastly, some took inspiration from knowledge distillation. However, whether or not this should be considered FMTL is debatable.

In the following, the thesis discusses the above mentioned approaches in more detail. However, CFMTL is not the main focus of the thesis. Hence, the thesis lists and describes exemplary work only. This mainly includes work that treats each client as a task (following the notion of [10]), instead of clients holding full MTL models. Table 3.3 summarizes the exemplary work per the above themes. It should be noted that themes are non-exclusive. For instance, a research can focus on model grouping but still apply layer privatization. As such research might be listed as relevant to multiple themes.

Table 3.3: Centralized Federated Multi-Task Learning

Theme	Exemplary Work
Convex Optimization	MOCHA [10], OFMTL [67], FKMTL [81], RAMOCHA [69],
Model Convergence	FMTLJD [82], FML [83], Ditto [84], FATHOM [70], PMTL [85], OAFMTL [86]
Layer Privatization	FedPer [20], DPFedMTL [87], FedRep [88], VIRTUAL [71], HOTA-FedGradNorm [89], MTFL [66], FedMat [90]
Model Grouping	HFedMTL [91], RHFedMTL [92], HeurFedAMP [93], CFL [94], FairCFL [95], FEDAC [96], CFMTL [97], MUFL [98], PFedSA [99]
Knowledge Distillation	EFDLS [100], FedICT [101], DaFKD [102]

3.2.1 Convex Optimization

In convex optimization problems the objective function, as well as the set of solutions that satisfy the problem’s constraint, are convex [115]. An important property of convex functions is that no local minimum can exist outside of a global minimum [116]. This makes solving convex problems robust, as it is assured that a local minimum is also a global minimum, something that is not guaranteed in non-convex deep learning methods. For more information on convex optimization this thesis recommends the books ”Convex Optimization” [115] and ”Convex Analysis and Optimization ”[116]. Early research in FMTL focused primarily on convex optimization problems [67], [69], [81], with most building upon the very influential MOCHA [10] framework.

One of the first major contributions towards the field of FMTL, was titled Multi-Objective COCOA for Heterogenous Agent networks [10], often simply called **MOCHA**. The proposed framework extended the COCOA [117] framework into the federated setting. The goal is to learn multiple related and personalized models on each device, to address the

statistical and systems challenges of FL, as defined by [10]. MOCHA is often referenced as the work which popularized the term FMTL, by treating each client as a task. The personalized nature of MOCHA and most FMTL techniques is the reason why FMTL is often seen as a sub-field of PFL [56]. In MOCHA the relationship between the different nodes is represented by a shared matrix, enabling knowledge transfer between the nodes. Each node solves a local optimization problem, based on its personal data. This local optimization problem includes a regularization term that depends on the shared matrix, thus forcing the model to take relationships between nodes into account. Once complete, the local updates are sent to the central server where the updates are aggregated to refine the shared matrix. This shared matrix is then sent back to the local nodes, where it is used in the next iteration. MOCHA inspired many other studies.

The **OFMTL** [67] algorithm also focuses on learning personalized models for individual clients. However, it extends the focus of MOCHA on situations where new devices continuously join the system. In **FKMTL** [81] the model’s parameters and the relationships between tasks are optimized using kernel methods, allowing the federated learning system to capture more complex patterns in the distributed data. However, the paper is solely theoretical and no experiments are run. **RAMOCHA** [69] addresses the challenges of efficient AI model training in 6G networks. Based on MOCHA, the authors developed a method, which optimizes FL under resource constraints. Similar to MOCHA, the approach makes use of a primal-dual optimization technique to manage multiple tasks across distributed devices, without requiring raw data upload, ensuring privacy and efficiency. They incorporated dynamic tuning of learning parameters to balance resource usage and model accuracy. Conducted experiments highlight effectiveness of the proposed method in reducing resource costs.

3.2.2 Model Convergence

Recent research started focusing on non convex problems. Frameworks that adapt the full client model by regularizing towards a global model or replacing the local model completely, are discussed under the model convergence umbrella. It is important to note, that the frameworks discussed in the following subsections are also focused on non convex problems. However, they introduce more sophisticated methods to model the FMTL environment.

The privacy policies in medical imaging hinders the effective training of deep learning models, as large datasets are needed. As such, the **FMTLJD** [82] framework combines MTL with FL in a non convex setting. Compared to this **FML** [83] is an example of a framework, that enables the collaborative training of a generalized model, while allowing clients to create personalized models tailored to their specific needs.

Other frameworks do not keep the private and global models fully separate. **DITTO** [84] aims to ensure uniform performance across devices and resists data and model poisoning attacks. To do this, DITTO enables each device to maintain a personalized (local) model while contributing to a global model that aggregates model parameters from all devices. The method uses bi-level optimization, where each device solves its local objective while being regularized towards the global model. The regularization term is added to the clients

local loss function and penalizes deviations of the local model from the global model. Similar to this **FATHOM** [70] uses a hierarchical attention mechanism to learn feature representations with a regularization technique to ensure that local models remain close to the global model. **PMTL** [85] can be cited as another example of such a regularization approach. The general idea of the PMTL framework is to use a form of relaxed differential privacy, to learn personalized models for each task, while adding noise during global aggregation to ensure privacy. As such, each client trains its local model on private data and sends their model updates to a central server. The server then adds noise to these updates, to ensure the model updates are obscured. Once the models have been aggregated, the server shares the noisy model back with the clients, which add a regularization term to their objective function to regularize towards the noisy model.

One unique, but very interesting framework that can be counted towards this category, is **OA FMTL** [86]. Its goal is to reduce the communication overhead in FL environments. To achieve this, the framework relies on over-the-air computing, a technique that allows the computation of mathematical functions by exploiting the signal superposition of wireless multiple access channels [118].

3.2.3 Layer Privatization

Frameworks that apply layer privatization, only aggregate a subset of the model layers. By keeping some layers private, the clients can better tailor their models towards their local data distribution. Some studies keep the final layers private [20], [87]–[89], while others allow the intermediate layers to be kept private [66].

One major contribution is **FedPer** [20]. FedPer divides models into two distinct parts: Base layers and personalization layers. This structure is reminiscent of the, in Section 2.1 discussed, hard parameter sharing technique. The base layers of the individual clients are replaced in each communication round by the aggregated global base layers. In contrast, the personalization layers are unique to each device and are trained locally, using only the data available on that specific device. This approach allows the model to effectively capture and adapt to individual user characteristics [20]. FedPer outperforms FedAvg, particularly in scenarios characterized by high statistical heterogeneity [20]. **DPFedMTL** [87] is similar to FedPer, as it also aggregates and distributes the non-private layers across clients for replacement. However, it goes a step further and enables effective parameter transfer, while ensuring differential privacy to safeguard client-level gradients. **FedRep** [88] is also similar to FedPer. FedRep first keeps the shared layers frozen and updates only the head. Once the head is sufficiently updated, they freeze it and unfreeze the base layers to update them accordingly. The base layers are then sent to the server where they are aggregated and distributed to the next batch of clients. Given the similarity to FedPer, they compared FedPer’s performance to FedRep and observed a higher test accuracy on the CIFAR-10 dataset.

Other methods apply more distinct techniques. **VIRTUAL** [71] keeps some parameters private while others are shared. The server and clients use bayesian inference to update the respective models. Clients initialize their models with server-shared parameters, train on private data, update parameters using variational inference, and send changes to the

server, which aggregates and returns updated shared parameters. Another interesting approach is **HOTA-FedGradNorm** [89], as it is similar to the previously discussed OA FMTL. While HOTA-FedGradNorm also makes use of a common network having distinct personalized heads, it enhances communication efficiency between clients and the server by using over-the-air aggregation. A unique part of the HOTA-FedGradNorm algorithm is, that it dynamically adjusts weights based on channel conditions. Their conducted experiments show that this leads to faster training speeds and greater robustness against negative channel effects, like bandwidth-limited fading.

Instead of keeping the final layers private **MTFL** [66] introduces private batch normalization layers with a distributed ADAM optimization technique. Batch normalization [119] is a technique used in deep neural networks to stabilize and accelerate the training process. It does this by normalizing the inputs to each layer, which reduces the internal covariate shift. The authors of MTFL stated that private batch normalization layers provide two key benefits: They customize each model to fit the client's unique data and improve data privacy. Privacy is increased since with private batch normalization layers, less information about the client's data distribution is revealed. The MTFL framework can be summarized as follows: After local training, the models, excluding the batch normalization layers, are aggregated by the server. The resulting global model is distributed back to the clients. These clients then replace their local non-private layers with the updated global model and the process repeats. Their experiments show that by using MTFL, few communication rounds are required to reach a client accuracy target. Another framework that shares intermediate layers is **FedMat** [90]. The FedMAT framework introduces a Federated Multi-Task Attention approach for human activity recognition. The private layers, including attention modules [120] and General Recurrent Units [121] for feature extraction, are kept on the clients and are not shared with the server. The attention mechanism within these private layers allows each client to focus on the most relevant features from the shared layers.

3.2.4 Grouping

Many frameworks make use of static [91], [92] or dynamic [93]–[99] grouping, to ensure that similar clients have a higher impact on the individual models. This can address the problem of dominant and conflicting gradients, thus increasing model performance. More on dominant and conflicting gradients as well as on grouping in MTL can be found in the sections 2.1.1 and 2.1.3.

In its simplest form, the grouping can be predefined. For instance, **HFedMTL** [91] takes inspiration from the inherent cloud-BS-terminal structure in cellular networks. HFedMTL focuses on establishing a hierarchical structure for FMTL that reduces communication costs and improves model performance and convergence. Terminals are grouped under base stations, and base stations are interconnected through the cloud server. Each group collaboratively works on a task, and the models from within the same group are taken into account during the training process. The training process includes a regularization step, to align each local model with those of other clients within the same group. This ensures personalization and enforces that the clients align with the overreaching task. A primal-dual optimization method is used, splitting the global problem into local sub-problems

which are solved individually. **RHFedMTL** [92] builds upon the HFedMTL framework by adding a resource-awareness component. As such, it dynamically adjusts the number of iterations for updates to balance learning accuracy with resource consumption. However, this approach, like HFedMTL, assumes that the grouping is known and constructed before training starts, making it unusable for more dynamic scenarios.

Other approaches rely on more dynamic grouping techniques, such as the cosine similarity. A prominent example is **HeurFedAMP** [93], an extension to FedAMP. In it the server aggregates models by combining the parameters of similar clients, using a weighted approach based on cosine similarity. Each client’s model parameters are compared with those of other clients, to determine how similar they are. Similar clients receive higher weights. These weights are used to combine model parameters, creating a personalized similarity model for each client. This model blends the client’s own model with those of similar clients. The client then updates its local model by refining it based on this combined model and its own data. It does this by adding a regularization term to the objective function, that penalizes the difference between the local model and the aggregated similarity model. This process is repeated iteratively, allowing clients to gradually refine their models by incorporating relevant information from other similar clients. Similar to HeurFedAMP, **CFL** [94] and **FairCFL** [95] use the cosine similarity between clients to create client groups that share similar data distributions. These groups of clients can be considered separated tasks and are training a model together. On top of the cosine similarity, **FEDAC** [96] also considers the degree of the loss reduction. Clients are grouped based on the angular distance between their gradient updates, with those having the most distinct data distributions being separated into different clusters. This process groups clients with similar data, allowing for better and more effective model training for each group.

CFMTL [97] compares the performance of the cosine similarity with other metrics, such as Manhattan (L1), Euclidean (L2) and Equal Weighting. The authors found, that the L2 distance generally provides the best or comparable performance in terms of convergence speed and accuracy, except in highly non-IID settings. There cosine similarity performs better. While cosine similarity performs poorly in IID settings, it works great in non-IID environments, particularly with significant data heterogeneity. The equal weighting metric performs reasonably well but is not the best in any non-IID setting. They concluded that L2 is typically the most reliable metric, but cosine similarity may be preferable in scenarios with highly non-IID data distributions.

Lastly, two other interesting approaches are **MUFL** [98] and **PFedSA** [99]. MUFL addresses the issue of resource overload from simultaneous FL activities on edge devices, known as multi-tenant FL. It optimizes this process with two strategies: Consolidating similar tasks into a single model and splitting tasks based on what the authors call: affinity scores. These scores measure how a client training a task affects another client’s loss. This approach reduces energy consumption by efficiently grouping and training tasks. PFedSA can be seen as an extension to the idea of private batch normalization layers of the previously introduced MTFM [66]. In PFedSA, clients are first clustered based on the similarity of their model parameters, with private layers, such as the discussed Batch Normalization layers. Similar to MTFM, these personalized layers are shared exclusively among clients within the same group. This helps to balance the personalization and generalization of the model. The method further considers the degree of similarity between clients during the

aggregation of model parameters, which ensures that clients with greater similarity have a stronger influence on the aggregated model. Similarities are calculated using the cosine similarity. Compared to the similar but non grouping based MTL approach, PFedSA reaches higher accuracies on image datasets such as MNIST, EMNIST and CIFAR-10.

3.2.5 Knowledge Distillation

As discussed in Section 2.1.2, knowledge distillation trains a simple model (student) to replicate the behaviour of a more complex model (teacher). The goal is to generate a small model that maintains a performance close to the teacher’s model. Many approaches that apply knowledge distillation have only little in common with the standard FMTL techniques. Regardless, three exemplary works are introduced below.

The idea of **EFDLS** [100] is to address users, that handle different time series classification tasks. In the framework, each user has both a student and a teacher model with identical architectures. Knowledge distillation occurs locally at each client, where the teacher model transfers knowledge from its hidden layers to the student model hidden layers. After this, only the student model’s hidden layer weights are uploaded to the server. The server uses these weights to find the most similar models across users, using the least square distance and pairs them accordingly. The matched weights are then sent back to the users, where they are used to update the teacher models. This updated teacher model further improves the student model in the next training round.

Compared to EFDLS, knowledge distillation in **FedICT** [101] occurs both at the client side, during local training, and at the server side during global aggregation and adjustment. It has two main parts: Federated prior knowledge distillation and local knowledge adjustment. The first helps each client’s model to better match its own data by using prior knowledge. The second corrects global model updates to reduce client differences. Instead of sharing full model parameters, the framework exchanges only reduced knowledge (i.e. features and logits) between clients and the server. The goal of this is to allow for different model architectures. The benefit of this two-way distillation process is that it customizes local models for specific tasks while keeping the global model adaptable.

Lastly, **DaFKD** [102] introduces a domain discriminator for each client, which evaluates, how well a given distillation sample aligns with the client’s data. This discriminator assigns importance weights to each client’s contributions, ensuring that only the most relevant knowledge is aggregated during the distillation process. A global generator produces pseudo-data samples used in this process, helping to simulate a more representative global data distribution. The method also shares parameters between the domain discriminator and the client’s classification model to boost performance and cut down on communication costs.

3.3 Decentralized Federated Multi-Task Learning

All the previously discussed frameworks have one thing in common: They require a central server for coordination and or aggregation tasks. However, as discussed earlier, relying on a central server is not always feasible and implies privacy/data protection considerations. So far, only a small subset of studies have tackled DFMTL. The identified research can broadly be categorized into studies that focus mainly on regularization and studies that rely more on aggregation, terms that have been introduced in 2.2.2. Table 3.4 provides an overview of the DFMTL literature.

Table 3.4: Decentralized Federated Multi-Task Learning

Theme	Work
Regularization	DCLPMN [103], FDJLPMCG [104], dFedU [63], NFMTL [105], SpreadGNN [68]
Aggregation	SECMTL [106], D-FEDEM [56], DAG [107]

3.3.1 Regularization

As described earlier, regularization adds constraints to either local client objectives or aggregation formulas [20]. This also applies in the case of DFMTL.

One of the earliest DFMTL frameworks is **DCLPMN** [103]. While the authors of DCLPMN framed it as a PFL framework, they also noted that it is reminiscent of MTL. DCLPMN lets the nodes communicate in a peer-to-peer fashion directly with other connected neighbouring nodes, thus giving importance to the network graph itself, as it reflects the similarity between nodes. To do this, the authors proposed two approaches, one related to model propagation and one related to collaborative learning. The main difference between these two approaches is that model propagation trains the model on local data and then regularizes it over the network, while collaborative learning trains and smooths models at the same time. Regardless of the approach, the framework relies on local clocks, asynchronous gossip communication and random wake-ups.

Like in DCLPMN, the graph in **FDJLPMCG** [104] represents similarities between clients. FDJLPMCG jointly learns personalized models, as well as the collaboration graph. While the graph is updated iteratively, it has to remain sparse to limit communication costs. Model training is conducted with a greedy boosting method allowing incremental updates based on the current collaboration graph. The approach alternates between model training and graph updates. However, the authors did not mention how often that alternation shall take place. For training a model, a random node wakes up, which is similar to [103]. Once awake, the node performs a conditional gradient descent based on the last models received from its neighbors. Once complete, the user sends its updated model to its neighborhood. To update the collaboration graph, nodes need to communicate beyond their neighbors, to discover new relevant neighbors. Similar to the model training, a random node wakes up. It then conducts peer sampling to select a small subset of other nodes. The node then requests the current models, loss value and degree from the sampled nodes. Once received,

these values are used to calculate the gradient of the objective function with respect to the weights for connections between the node and sampled nodes. These weights represent the similarity between the tasks and thus allow the collaboration graph to learn relationships. Once complete, the newly updated weights are sent to each associated node of the sample and a new node wakes up, repeating the process. Compared to dFedu [63], which is discussed next, this approach has the benefit that the graph is dynamic.

dFedU [63] aims to capture the relationship between local models using laplacian regularization. This regularization technique ensures that nodes that are close to each other have similar values by penalizing those who don't. dFedU models the relationship between the nodes as a connected (but not necessarily fully connected) graph. This graph, including the weights of connections, seems to remain fixed. This results in the necessity to predefine the relationship between clients. However, that means that the relationships between connections can not be changed like in [103], nor can new connections beyond direct neighbors be considered like in [104]. Once the graph is fixed, their main goal is to leverage the defined relationships between models for MTL. dFedU does this by minimizing the sum of all local losses, combined with laplacian regularization. The algorithm works as follows: In every round all clients compute a mini batch gradient descent for n times. dFedU does this by computing the local loss based on local data. After the local updates are complete and the models updated, each client sends its updated model to its neighbors. Each node then uses the received models and its own, to perform the laplacian regularizations step, to increase the smoothness between the two models on the nodes. This approach seems to work well when node relationships are well-defined. However, it assumes a fixed graph, requires exchanging the entire model, does not handle stragglers, and seems to demand the same model architecture across all nodes.

Compared to dFedu **NFMTL** [105] iteratively refines the model weights and creates a clustering structure based on the network's feedback. Instead of laplacian regularization NFMTL makes use of generalized total variation minimization. This also works as a regularizer and ensures smooth transitions in model weights across similar datasets within a cluster, while allowing for distinct models across different clusters.

SpreadGNN [68] is a DFMTL framework for training Graph Neural Networks (GNNs) on molecular data. It uses a decentralized version of stochastic gradient descent to update local models at each client. Each client only communicates model updates with its neighbors. This is achieved, through a mechanism called periodic averaging, where clients exchange and average their local model updates at specific intervals. This approach allows the framework to be scalable. Another interesting part of this framework, is its ability to handle partial labels. Molecular data often has multiple labels per molecule, such as toxicity, interaction properties, or biological activity [68]. SpreadGNN introduces a method, to effectively manage partial labels across clients, by leveraging a DFMTL.

3.3.2 Model Aggregation

Research listed here is focused on implementing decentralized aggregation techniques, while also incorporating more sophisticated methods such as layer privatization and clustering.

Like some of previously discussed layer privatization approaches, **SECMTL** [106] also keeps some layers of the networks private. In the paper, the network calls these parts "slices". These slices allow for partial model averaging and enable the sharing of knowledge between peers, while preserving task-specific specializations. An interesting feature, is that the authors tested different "averaging levels", with 0 sharing corresponding to local training and 100 referring to standard federated averaging. They found, that the best performance is achieved with an averaging level of 80. The paper does not go into much detail about the communication scheme used but a gossip-based system seems implied. This is a drawback as the communication scheme is of great importance, especially when considering idle time and stragglers, something that is investigated by **DFLAPS**[122]. They introduced an asynchronous parameter-sharing mechanism, which allows each node to send and receive model updates independently, without having to wait for synchronization with other nodes. This approach minimizes idle times and enhances overall training efficiency.

Compared to the previous two frameworks **D-FEDEM** [56] is quite unique. It assumes, that each client's data is a mixture of several shared distributions, which capture common patterns and relationships across clients. This mixture model allows capturing complex relationships among clients' data, facilitating better personalization. The proposed method, called FedEM, uses an Expectation-Maximization (EM) algorithm to jointly learn shared component models and personalized mixture weights for each client. In addition to a centralized version, they also proposed a decentralized version called D-FedEM. The decentralized version works similar to the centralized one. In it, each client first estimates its own mixture weights for each underlying component model, using its local data. After obtaining these mixture weights, clients update their local estimates of the shared component models. Clients perform a decentralized aggregation step by communicating their updated models to their neighbors. Each client then calculates a weighted average of its own parameters, along with those received from its neighbors, to refine its local component models. This approach seems to make sure that, after several rounds of local updates and communication with neighbors, the clients' estimates align to the same set of parameters, even without a central coordinator.

Probably the most interesting framework of this subsection is **DAG** [107], as it uses a Directed Acyclic Graph (DAG) to manage model updates. Each node in the DAG represents a model update, and the edges represent the relationship between updates. These relationships can be thought as approvals to updates. The training process for each client involves four steps. First, a biased random walk selects two tips from the DAG. Next, the selected models are averaged. Afterwards, the resulting model is refined through training on local data. Finally, if the training results in an improved model, the new model gets added to the DAG. The biased random walk is guided by a weighting mechanism that assigns higher probability to models that perform well on the client's local data. Over time, this process leads to the formation of implicit groups in the DAG, where the idea is that clients with similar data characteristics share model updates.

3.4 Heterogeneous Centralized Federated Multi-Task Learning

The papers discussed so far mainly focused on data or statistical heterogeneity between clients. As such, they generally assumed that each client solves the same or a very similar task. However, to make FMTL more applicable, heterogeneous tasks should be taken into consideration as well. It is important to note, that the thesis is studying frameworks that address heterogeneity among clients. As such frameworks that address heterogeneity within a client (i.e a client has a MTL model that has heterogeneous tasks), are not considered.

Heterogeneous tasks often come with the added challenge of conflicting and dominating gradients, as described in Section 2.1. Most work in the space of HFMTL suggest approaches to deal with these problems either directly or indirectly.

Table 3.5: Heterogeneous Federated Multi-Task Learning

Theme	Work
Indirect Solutions	CoFed [76], EFHIGFMTL [108], FedLPS [109], AFL [110], MATFL [111], MAS [112], FedMSplit [113]
Direct Solutions	FedBone [14], FedHCA ² [11], NTKFedAVG [19]

3.4.1 Indirect Solutions

Indirect solutions to the problem of conflicting and dominant gradients include data augmentation techniques, model pruning and grouping.

Examples of frameworks that rely heavily on data augmentation techniques are **EFHIGFMTL** [108] and **CoFed** [76]. EFHIGFMTL is a framework for FMTL that leverages feature distribution heterogeneity across different clients and tasks to improve model generalization. The authors proposed an augmented dataset approach and a novel client weighting method to utilize this heterogeneity to train models. CoFed also relies on augmented data, more specifically on pseudo-labeling, like the previously discussed DaFKD [102]

Network pruning is a technique to reduce the size of a neural network by removing certain elements that contribute little to the network’s performance [123]. An example of this is the **FedLPS** [109] framework. It is designed to handle both homogeneous scenarios and heterogeneous scenarios. While technically not described as an FMTL approach, it is still an interesting way to deal with heterogeneous clients. The authors implemented a layer-wise pruning mechanism that focuses on pruning entire layers, rather than individual neurons or intra-layer connections. It is stated that the method allows for effective pruning and reduces complexity. In homogeneous settings, their clients randomly prune layers before sending them to the server. In heterogeneous settings, clients selectively prune and upload specific layers based on their computational capabilities. **AFL** [110] discusses the negative correlation issue that arises from task heterogeneity, where different tasks might have conflicting optimization directions. To mitigate this, AFL utilizes a sparse

sharing approach, allowing for partial parameter sharing, only among highly related tasks. The system uses iterative pruning and task-specific mask layers to minimize interference, which, according to them, is particularly beneficial in scenarios where task relationships vary significantly.

Lastly, several grouping based approaches exist. The aim of **MATFL** [111] is to define a framework that can handle both data and task heterogeneity. As such clients should be able to collaborate, even if they are solving different tasks (ex. segmentation vs classification) This is achieved by relying on dynamic grouping and adaptive aggregation. During each communication round the server collects all local updates and calculates the normalized cosine-similarity of each weight update at each layer. Each node then accumulates these similarities to define the most similar other nodes to form a group with which the node would like to aggregate updates. During the aggregation more similar nodes within the group will have a higher impact than less similar ones, with nodes outside the group having no impact at all. A completely different approach compared to MAT FL and other approaches is employed by **MAS** [112]. The general idea is to merge the federated models into one big Multi-Task Model, located on the server. Once merged, the model is trained and affinity scores between the tasks is calculated. Once complete the complete multi-task model is split into smaller models with similar affinities. These smaller models are then further trained on the local nodes. Lastly, the **FedMSplit** [113] framework is designed to support FL across clients with varying combinations of active sensors (modalities). It leverages a dynamic graph structure to capture correlations between client models. In this graph clients are represented as vertices and their relationships are visualized as edges. The framework tackles the problem of clients having different types of sensors and solving different tasks, which makes it hard to combine their models. It does this by dividing models into small components and using a multi-armed bandit algorithm [124], [125].

3.4.2 Direct Solutions

Compared to the previously introduced indirect solutions, the direct solutions acknowledge and address the problem of conflicting and dominant gradients head on. There is of course some overlap to the indirect solutions, as the direct solutions might also apply grouping mechanisms etc.

One example of such an approach is **FedBone** [14]. Its goal is to enable large scale HFMTL on devices with limited computational resources. To achieve this, FedBone applies split learning, which splits its model into two parts: A large scale general model located on the central sever and task specific models on the nodes. The general model on the server is responsible for resource intensive computations as well as extracting high level features. The models on the clients are task specific and less resource intensive. Fedbone also addresses the problem of gradient conflicts. They do this by introducing gradient projection aggregation, which rescales gradients based on historical gradients to resolve conflicts. Lastly, FedBone also introduced a task adaption module. By using deformable convolutions [126] and self attention [127], the module is able to capture task specific features improving task specific performance. Fedbone works as follows: The clients perform local emebdding and send these embeddings to the server. The server processes

these embeddings to extract high level features which are then sent back to the individual nodes. The clients then process them in their task specific heads, marking the end of the forward process. Once complete, the clients compute the task specific losses and perform a partial backpropagation step on their heads. The intermediate gradients are then sent to the sever which completes the backward propagation on the general model by relying on gradient projection aggregation.

FedHCA² [11] takes inspiration from FedBone but takes it a step further. Its goal is to provide a framework that enables aggregation across clients with varying numbers and types of tasks. This completely foregoes the notion of one node per task as one node itself might be a multi-task network. In FedHCA² clients use an encoder-decoder architecture, in which the encoder is shared and decoders are task specific. Comparing this to FedBone, the encoder could be seen as the general model, which in FedBone resides on the sever. However, contrary to FedBone, the network is not split, meaning that the nodes hold the complete networks and the server is mainly responsible for model aggregation. Similar to FedBone FedHCA² also tries to tackle the problem of conflicting gradients. It does so by introducing the Hyper Conflict-Averse Aggregation (HCA) scheme for encoders and Hyper Cross Attention Aggregation for decoders, respectively. Additionally, it also makes use of Hyper Aggregation Weights to foster personalized updates. FedHCA² is a major inspiration for proposed framework in 4.2, as such the relevant parts of it will be explained in more detail there.

The authors of **NTKFedAVG** [19] noted that the dynamic grouping mechanism of MAT FL and the split learning approach of FedBone requires the server to manage individual models for each client or task. According to them, this increases the risk of privacy breaches and requires great storage and computational capacities on the server side. As such, their approach only requires one model residing on the server. To this end, they introduce NTKFedavg, a variation of the classic FedAvg. In their approach each node first linearizes their model around the server model using neural tangent kernels. Once complete, the nodes train their models and compute local task vectors. These task vectors are then aggregated by the server and used to updated the global model, which then is returned to the nodes for the next round.

3.5 The Research Gap: HDFMTL

During the literature review no work was found that would properly fit into the category of HDFMTL. It is possible that some relevant research was missed or wrongly classified. But even then, the research in that area has to be very limited. As otherwise, chance would have it, that at least one paper would have been identified accordingly.

While some work properly discusses and proposes solutions to address heterogeneous clients, that differ in more than just their data distributions, as reviewed in Section 3.4, they fail to consider decentralized frameworks. Of those, some could probably be adapted into a decentralized setting easily. For instance, [19] only requires one model on the server. Such a scenario could be translated well into Semi-DFL with rotating leaders. Extending this thought, a simple way to translate this into fully DFL, would be to have each client

holding the global model, in addition to its client model and apply techniques reviewed in the Section 3.3. Other work would not translate well into a decentralized environment, such as [14]. There, the framework relies heavily on outsourcing computation to the server. The work discussed in 3.3 focuses specifically on DFMTL. But while some work does seem to support class label heterogeneity [68], most do not address task heterogeneity beyond simply dealing with the non-IID problem.

This lack of research in HDFMTL, motivates the development of a framework. This framework, which takes inspiration from existing work in the realm of CFMTL and HCFMTL, is introduced in the next chapter.

Chapter 4

Architecture

This chapter introduces a framework to deal with specific heterogeneity challenges in DFMTL. It does so, by first motivating the problem, followed by a general overview of the framework. Afterwards, the thesis dives deeper into the proposed aggregation algorithm. Following this, some additional considerations, mainly with regards to challenges found in decentralized environments are discussed. Lastly, the thesis elaborates on how the proposed framework differs from existing works.

4.1 Motivation

As was discussed in the literature review, there exists already a lot of work focusing on non-iid settings in CFMTL and DFMTL. Therefore, most of these studies seem to focus on settings where the overall task, like image classification, is the same among all participants. While this approach allows each participant to have the same overall model architecture and allows to study non-iid settings, it fails to address situations where clients make use of similar but slightly different architectures. Such similar architectures might employ the same backbone but require different final layers or output heads. For example, in a non-iid scenario, some clients may only encounter a subset of all potential class labels in their data. As such, these clients, might only have an output head for the labels present in their own data distribution. This is because in FL the goal is to learn a model without sharing data between clients, resulting in clients not knowing all the potential class labels and thus having different output heads. However, this seems to be often overlooked and most research simply assumes that every client is aware of all potential class labels, resulting in each client having the same output heads. Hence, one of the goals of this thesis is to address this scenario, in which some clients only see a subset of all class labels. As part of this thesis, this scenario is called heterogeneous class labels. A scenario heavily inspired by FedPer [20]. Some more recent research addressed clients that are actually solving different tasks, like object detection and instance segmentation. Thus, this more complex second scenario is referred to as heterogeneous tasks. In this setting, it is assumed that parameter updates differ more significantly. This significant

difference makes simply averaging parameters not feasible and requires a more nuanced approach to limit the problem of conflicting gradients.

Both heterogeneous class labels and heterogeneous tasks imply at least a difference in the final layers of the models. While having completely different model architectures would be powerful, the feasibility of such an approach is questionable and would most likely introduce extreme complexity. As such, keeping a common backbone architecture with task specific heads, like reviewed in Section 3.2.3, seems to be a decent approach. Especially if compared with traditional MTL, given that the architecture is very reminiscent of hard and soft parameter sharing, depending on how the backbone is shared among clients.

Thus, the thesis aims to develop a framework where heterogeneous class labels and heterogeneous tasks can be modeled through DFMTL. In the developed framework, clients shall utilize a common backbone architecture but maintain distinct final layers.

4.2 Framework

To explain the proposed framework this section will follow Figure 4.1. The framework takes inspirations from already existing grouping approaches [91]–[99]. In them, clients with similar data distributions, or in this case, tasks are grouped together. Therefore, before any training or aggregation can start, such groups have to be defined. In the example shown in Figure 4.1, there are two task groups with 3 similar clients each. Note that within a task group each client sees different data samples, but their model architectures are the same. Going across task groups, model architectures can differ. More precisely, the final layers of a model differ, while the underlying backbone architecture is consistent. This is also depicted in Figure 4.1 with different head nodes. Once the task groups are defined and clients are assigned, local training can begin, marking the first step of the framework. This, and the following steps are depicted with Diamonds in Figure 4.1. Each client of every group trains for a specified amount of epochs locally. Once the number of epochs are completed, the intra-task aggregation round starts, triggering the second step of the framework. In it, the clients exchange their backbone parameters within their respective group and perform averaging on the collected backbones. This is similar to the approaches discussed in the layer privatization Section 3.2.3, with the most influential example being FedPer [20]. Afterwards, the cross-task communication round is triggered, resulting in the third step of the framework. Now each group leader exchanges its backbone delta (the difference between the intra-group averaged backbone and the backbone at the start of the current local training round), with all other leaders of different groups. Once the exchange is complete, the group leader conducts a simplified version of Hyper Conflict Averse Aggregation (HCA), proposed by FedHCA² [11]. This new backbone is then exchanged in step 4 with all members of the corresponding groups, which update their backbones accordingly. Once the cross-task communication is complete a new round can start, beginning again at step 1. The framework is also depicted as pseudo-code in Algorithm 1. The combination of ideas from FedPer [20] and FedHCA² [11] motivated the framework name: FedPer-HCA.



Figure 4.1: Proposed Framework

The following sections provide a more detailed discussion of the intra-task and cross-task aggregation mechanisms. Subsequently, the thesis will address additional considerations, including leader selection, communication schemes, faults, stragglers, and late joiners. Note that these three topics are not included in the pseudo-code to maintain clarity and readability.

4.2.1 Intra-task Aggregation

The aggregation process within a task group is similar to FedPer [20]. After completing the local training round and exchanging all backbone parameters, each client averages the collected backbones and updates its own backbone accordingly. For a client the intra-task aggregation update can be expressed as follows:

$$\bar{\theta}^B = \frac{1}{N} \sum_{i=1}^N \theta_i^B \quad (4.1)$$

Here θ^B are the backbone parameters and N represents the amount of clients per task group. Such a simple federated averaging method has proven to work well, even if clients have different final layers [20]. Therefore, with increasing task differences such an approach is expected to reach its limits. This is why the thesis also makes use of HCA aggregation [11]. At this point it is important to note, that FedHCA² [11] also implement some form of

grouping and inter-group exchange, at least in their source code. However, their focus is on the centralized setting. Additionally, FedHCA² incorporates more techniques beyond just HCA.

4.2.2 Cross-task Aggregation

Cross-task aggregation is more challenging. The model backbone parameters can differ significantly, since they are focusing on different tasks. This problem results in dominant and conflicting gradients, which are described in detail in Section 2.1.1. As reviewed in Section 3.5, a few papers are addressing this issue. The most promising is FedHCA² [11]. As such, the cross-task aggregation is a simplified implementation of the algorithm proposed by FedHCA² [11]. They define and proof the following theorem:

”Given clients with a shared encoder and multiple task-specific decoders structure, the gradient descent in the shared encoder of Multi-Task Learning is equivalent to averaging parameter aggregation in Federated Learning, adding an extra term $\nabla_{\theta^{(0)}} \langle \hat{g}_i^{(p)}, \hat{g}_j^{(q)} \rangle$ that maximizes the inner product of gradients $\hat{g}_i^{(p)}$ and $\hat{g}_j^{(q)}$ between all pairs of tasks i and j in each iteration p and q .” [11]

They acknowledge that the the inner product of gradients is a measurement of accordance, meaning that maximizing the inner product corresponds to reducing the conflict of gradients. This is important as conflicting gradients can ruin a parameter update. Following their description and formulas HCA, in the context of this thesis, can be explained as follows:

The overall goal is to find an optimal aggregated update $\tilde{U}$ for the backbone parameters θ^B of the group leaders during each communication round. This update minimizes conflicts between group leader updates, while optimizing the main objective of finding $\tilde{U}$. The optimization problem defined by [11] is as follows:

$$\max_{\tilde{U}} \min_i \langle \Delta\theta_i^B, \tilde{U} \rangle \quad \text{s.t.} \|\tilde{U} - \Delta\bar{\theta}^B\| \leq c\|\Delta\bar{\theta}^B\| \quad (4.2)$$

Here $\Delta\bar{\theta}^B = \frac{1}{N} \sum_{i=1}^N \Delta\theta_i^B$ is the average update from all group leaders, and $c \in [0, 1)$ controls the update’s deviation from the average. The average update of a group leader $\Delta\theta_i^B$ represents the difference between the previous communication round backbone parameters and the current round backbone parameters. In the context of this thesis, this translates to the backbone at the start and end (after intra-task aggregation) of a local training round. The constraints $\|\tilde{U} - \Delta\bar{\theta}^B\| \leq c\|\Delta\bar{\theta}^B\|$ ensure that the final update $\tilde{U}$ stays close to the average update $\Delta\bar{\theta}^B$ while allowing some flexibility (controlled by the hyper-parameter $c \in [0, 1)$). To solve the above, [11] use lagrangian optimization, which they simplify to:

$$\min_w F(w) = U_w^\top \Delta\bar{\theta}^B + \sqrt{\phi}\|U_w\| \quad \text{where} \quad U_w = \frac{1}{N} \sum_{i=1}^N w_i \Delta\theta_i^B, \quad \phi = c^2 \|\Delta\bar{\theta}^B\|^2 \quad (4.3)$$

[11] conclude that the final aggregated update is:

$$\tilde{U} = \Delta \bar{\theta}^B + \frac{\sqrt{\phi}}{\|U_w\|} U_w \quad (4.4)$$

This final update balances client updates, while minimizing conflicts and keeping the update close to the average direction. The final update $\tilde{U}$ is then added to each group leader’s backbone. Following this, the group leader shares its new backbone with all members of its task group, which update their backbone accordingly.

4.2.3 Additional Considerations

So far, the framework left out some important considerations. For instance, the existence of a leader was assumed. It was also not specified, whether an asynchronous or synchronous communication scheme is employed. Lastly, the problem of faulty clients, stragglers or late joiners was purposefully ignored as well. The reasons for this are that implementing such schemes is not the primary focus of this thesis, as it requires a proper decentralized setup across multiple computers. Therefore, while these issues may be considered somewhat secondary, the thesis still addresses them in the following sections and suggests potential solutions for each. However, these proposed solutions are merely conceptual and are not intended as definitive or robust implementations. As a result, they are not considered in the implementation discussed in Section 5.

Leader Selection

Since the framework requires task group leaders, it is necessary to define how they are selected. During the development of the framework, it was decided to implement rotating leaders, similar to the approach used in Semi-DFL [6]. This approach appears fair, as it distributes the additional computing requirements of being a leader across different group members. During the initial setup, one group member is randomly chosen as the leader. After each cross-task group aggregation round, the current leader randomly selects a new leader and notifies the respective client. The leader then informs all other task group leaders, who, in turn, relay this information to the new client leaders designated for their own groups.

Communication Scheme

The thesis proposes a communication scheme, that does not require local clocks nor makes use of random wake-ups like [103]. The communication mechanism can be considered a mix between synchronous and asynchronous. It’s somewhat asynchronous, as clients broadcast [58] their backbone parameters as soon as they are done with local training. Yet, they still need to wait for other clients to share their backbone parameters in order to perform the aggregation, making the process somewhat synchronous. Additionally, at the

beginning of each communication round, a synchronization entity is required to provide a start signal. Overall, the proposed approach operates as follows:

Once a client completes its local training round, it sends its backbone to all its neighbors. Upon receiving the backbone, each neighbor sends an "OK" response to indicate that they are still active. If no "OK" response is received, the client removes that neighbor from its list, marking it as "inactive". After a certain period, the client checks in again. If it still only receives an "OK," the neighbor is dropped and considered "delayed". This step is necessary to prevent a very slow neighbor from delaying the entire federation. Once a client has received all backbones from neighbors that are neither "delayed" nor "inactive", it proceeds with the aggregation. If the client is the leader, it sends the aggregated backbone to all other leaders. Here the process is the same as within a group. A receiving client leader has to send an "OK" back, if it is still active. If after some time no backbone is received, the leader follows up. If another "OK" is sent, the leader gets dropped and marked as "delayed". Like within a group, "inactive" task group leader get dropped as well. Once a new communication round starts, "delayed" neighbours and leaders are reset, meaning that they are treated like regular neighbours again. "Inactive" Clients are not reset and remain disregarded. If an "inactive" client somehow becomes active again (ex: through rebooting), it has to undergo the same process like a late joiner. This process is described in the next subsection.

Late Joiners, Stragglers and Faults

If a client decides to join the federation at a later stage or was previously "inactive" and wishes to become active again, it must register with the current group leader. To do this, it simply sends a message to the leader. After the current aggregation round is complete, the leader shares the current group backbone with the newly joined client. The client can then proceed with training like all other clients.

Another issue is the presence of faulty clients. Fortunately, with the communication scheme described above, these faults are easily handled. A faulty client will not be able to send back an "OK" message, causing it to be marked as "inactive". Similarly, stragglers (clients that are significantly slower [128]) will be marked as "delayed" since they are identified when clients check in after a predefined time period has elapsed. One thing that bears mentioning but is not discussed, is the risk of malicious clients, that want to actively send bad model updates [129]. For this, the thesis refers to already existing propositions such as [129] and [130].

4.2.4 Comparison to existing work

As mentioned, a major inspiration for this framework was the FedHCA² [11] framework. This is particularly true for the HCA algorithm. However, FedHCA² still relies on a central server. As such, one major distinction is that the proposed framework of this thesis models the problem in a decentralized manner. Also, FedHCA² implements additional sophisticated methods such as Hyper-weights and Encoder Attention Aggregation. The

thesis decided against implementing such elaborate methods to keep the framework simple, as every added complexity could result in more computing time needed on the client or client leader. Keeping the complexity and thus competing power reasonable on edge devices is crucial, as they are often confronted with resource constraints. Another major inspiration for the proposed framework can be found in the realm of layer privatization as described in 3.2. Examples like FedPer [20], DPFedMTL [87] and FedRep [88] worked as a baseline. Especially FedPer was crucial, as the intra-task aggregation followed their approach closely. This is why the intra-task aggregation approach is referred to as FedPer in this thesis. However, Fedper, DPFedMTL and FedRep did not consider DFL, nor did they address the problem of conflicting and dominant gradients, which is especially prevalent across different task types. Lastly, many FL approaches, be it CFMTL, DFMTL or HCFMTL have implemented grouping mechanisms. It is noteworthy that FedHCA² also employs grouping and inter-group aggregation, further underscoring its influence on the proposed solution. Some even implemented dynamic grouping, based on metrics such as the Euclidean L2 norm or cosine similarity. Examples include HeurFedAMP [93], CFL [94], FairCFL [95] and FEDAC [96]. The proposed framework refrains from introducing dynamic grouping but acknowledges it as an area for further improvement.

Algorithm 1 Client Pseudocode

```

1: Input: List of connected intra-task group clients  $C$ , list of connected task group
   Leaders  $L$ , number of epochs  $epochs$ , number of communication rounds  $rounds$ , leader
   boolean  $isLeader$ 
2: procedure MAIN
3:   for  $r = 1$  to  $rounds$  do
4:     for  $e = 1$  to  $epochs$  do ▷ Local Training
5:        $\theta \leftarrow \text{Train}$ 
6:     end for
7:     for  $c$  in  $C$  do ▷ Intra Task Communication
8:       Send  $\theta^B$  to  $c$ 
9:       Receive  $\theta_c^B$  from  $c$ 
10:    end for
11:     $\theta^B \leftarrow \text{Perform equation 4.1 with } \theta^B \text{ and all } \theta_c^B$  ▷ Intra Task Aggregation
12:    if  $isLeader$  then
13:       $\Delta\bar{\theta}^B \leftarrow \theta^B - \theta^{B-1}$ 
14:      for  $l$  in  $L$  do ▷ Cross Task Communication
15:        Send  $\Delta\bar{\theta}^B$  to  $l$ 
16:        Receive  $\Delta\bar{\theta}_l^B$  from  $l$ 
17:      end for
18:       $\tilde{U} \leftarrow \text{Perform equation 4.4 with } \Delta\bar{\theta}^B \text{ and all } \Delta\bar{\theta}_l^B$  ▷ Cross Task Update
19:       $\theta^B \leftarrow \theta^B + \tilde{U}$ 
20:      for  $c$  in  $C$  do ▷ Intra Task Communication
21:        Send  $\theta^B$  to  $c$ 
22:      end for
23:    else
24:      Wait until  $\theta_l^B$  is received from Leader
25:       $\theta^B \leftarrow \theta_l^B$  ▷ Cross Task Aggregation
26:    end if
27:  end for
28: end procedure

```

Chapter 5

Implementation

Implementing the proposed framework is challenging due to its decentralized nature. Although decentralization is simulated where possible, the framework operates on a single machine. As a result, instead of implementing a fully decentralized environment, the implementation primarily focuses on evaluating the proposed aggregation algorithm and benchmarking it against other aggregation methods. The reasons for not adopting a fully decentralized setup are discussed in detail in the final section of this chapter.

This chapter first provides a general overview of the code repository. Afterwards, it focuses on how the specific aggregation schemes were implemented, followed by a discussion about the two main functionalities that a user can trigger: `run` and `plot`. The thesis then addresses how reproducibility was ensured, followed by a discussion of the thesis' limitations, primarily focusing on decentralization aspects. The link to the source code is provided in Section 1.2.1.

This section does not aim to provide a comprehensive explanation of the source code, covering each function in detail. Instead, it focuses on main functionalities and should be used in conjunction with the source code itself. The source code includes a `ReadMe`, `docstrings` and `comments` to clarify specific functions.

5.1 Overview of Code Repository

The repository is structured into six main folders: *Checkpoints*, *configs*, *data*, *results*, *figures* and *src*. These six folders, and the sub folders of *src*, as well as the *run.py* and *plot.py* files, are depicted in Figure 5.1. Please note that the folders *checkpoints* *data* and *results* are dynamically created upon code execution, which is why they are visualized in blue.

The primary purpose of each folder is as follows: Intermediate model checkpoints for the clients are stored in the *checkpoints* folder during training. Upon completion, the clients, along with their models, metrics and used configs, are saved in the *results* folder. If subsequently figures are created, they will also be saved there. The *data* folder serves as

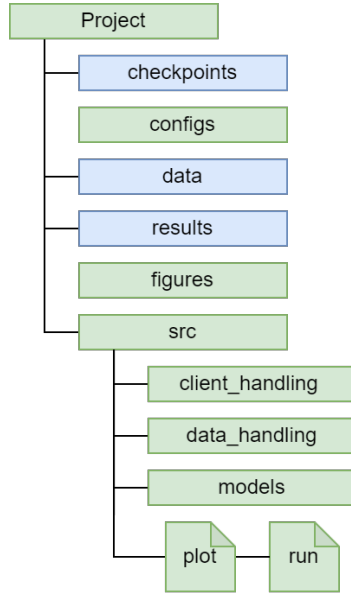


Figure 5.1: Overview of the Code Repository

a central hub for storing datasets, while the *src* folder is responsible for organizing and storing all source code files. The repository also includes some of the figures generated as part of the thesis’ experiments in the *figure* folder. The reason why not all figures and generated models were saved in the repository, is GitHub’s file size limitation. To access all generated models and figures that were generated as part of the experiments discussed in Section 6, please see the linked folder in Section 1.2.1.

Crucial parts of the implementation are located in the *configs* and *src* folders. Thus, the thesis will spend some time explaining their features. Please note that this description offers a high-level overview of the general functionality and purpose of these code segments. It is intended to serve as a starting point for those looking to explore the source code in greater detail.

5.1.1 Configs Folder

The *config* folder contains configuration files for each experiment, defined using YAML [131]. Each configuration file defines the title, tasks, number of clients per task, number of aggregation rounds, seed, number of local training epochs per round, dataset, the save location and the HCA hyperparameter. This parameter controls the impact of the hyper conflict-averse aggregation. Additionally, the config file holds parameter to limit training to a fraction of the dataset, which is highly beneficial for quickly testing code changes. Significantly, a config file also determines the algorithms for inter-task and cross-task aggregation, which allows the definition of the various aggregation schemes, introduced later in this chapter. Another crucial piece of information specified in a configuration file is the amount of backbone layers. This parameter determines how the model should be divided into backbone and head components. Thus it defines the level of privatization, which will be explained further in Section 5.1.2. The benefit of using YAML-based configuration files

is that they can be easily understood and modified. It also benefits the reproducibility of the experiments, as the configuration file, including the used seed, automatically gets saved into the respective result folder.

5.1.2 Src Folder

Located within the *src* folder are the sub folders *client_handling*, *data_handling* and *models*, as well as the *run.py* and *plot.py* files. The folders *client_handling*, *data_handling* and *models* are covered in this subsection, while the *run.py* and *plot.py* files are discussed in the later Section 5.3.

Client Handling

The client handling folder contains the Python scripts *client_manager.py*, *hca.py*, *client.py* and *seed.py*. The functions within *client_manager.py* are used by *run.py* to create and instantiate federation clients. The structure and functionality of a client are set within the *Client* class, which is defined in *client.py*. At the time of instantiation, each client is assigned an id, task type, title and a PyTorch [132] model. The instantiation also passes information about the extent of layer privatization and the required data loaders. The *Client* class defines functionalities such as checkpoint management, backbone extraction, backbone averaging, training and testing. Additionally, it can also trigger the hyper conflict-averse aggregation, which is defined in the *hca.py* script. The *hca.py* script is a simplified implementation of the hyper conflict averse-aggregation scheme proposed by [11], and reuses a lot of their code. The implementation of HCA is explained in more detail in Section 5.2.4.

Data Handling

All files responsible for downloading and preprocessing datasets are located in the data *data_handling* folder. Like the *client_handling* folder, it also contains a manager file called *data_manager.py*. This script is responsible for instantiating the corresponding Data class, as each dataset has a dedicated class. Once instantiated, the dataset object loads the dataset and applies various image transformation techniques to increase the data diversity. It then splits the train and test dataset into the corresponding tasks, ensuring that the data samples between the tasks do not overlap. Afterwards, it divides the task samples into client subsets, which are eventually split further into train and validation sets, as is standard in training neural networks. This approach ensures that each client has access to its own training and validation data only, without any overlap with other clients. All clients use the same test set within a task group to make the testing fair. Eventually, the data object creates PyTorch based data loaders [132]. Using data loaders is a common technique when using PyTorch. They provide an efficient way to iterate through datasets and make handling large datasets easier.

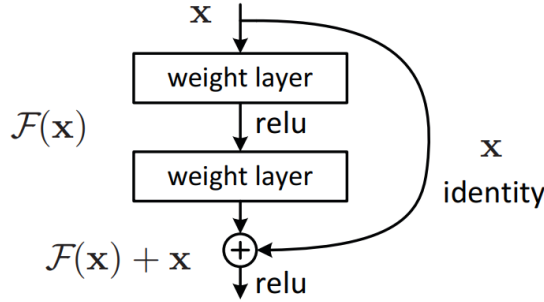


Figure 5.2: Residual Block as Depicted by [133]

Models

Probably the most interesting folder is the last sub-folder of *src*, *models*. It contains an abstract model class, from which all specific models inherit. This enforces a common structure among models, making sure that each model at least implements its own *forward()*, *train()*, *validate()* and *test()* function. In addition to that, it defines a method to create the backbones and heads for each client and task. This function, called *_create_backbone_and_head()*, takes the backbone layers definition of the config file and splits the model into a backbone and a head. This modular approach allows using the defined amount of backbone layers in the config file, enabling experiments with differing backbone/head splits.

The thesis uses three primary model types, each inheriting from the aforementioned parent class: a single-label classification model, a multi-label classification model, and a landmark detection model. All models are based on ResNet18 [133]. Residual Networks (ResNets), as introduced by [133] are a deep learning architecture that addresses the problem of vanishing gradients, a problem especially prevalent when training deep networks. The number 18 refers to the total number of layers. ResNet18 is considered a smaller model, making it well-suited for relatively simple tasks, like the ones examined in later sections of this thesis. This is important as deeper networks might overfit on simple tasks [134]. The core innovation of ResNets is the residual block, depicted in 5.2. Residual blocks learn residual functions, representing the difference between the desired output and the input x , rather than learning the entire mapping from scratch [133]. [133] showed that this eventually leads to the following simple formulation: $F(x) + x$. To model this form, the authors used feedforward neural networks and shortcut connections. These shortcut connections are identity mappings and thus do not introduce additional computational complexity, which is essential not to degrade performance [133].

The thesis uses the torchvision [135] implementation of ResNet18, together with the torch.nn module [132] to create these models. However, one must remember, that the ResNet18 implementation of PyTorch is divided into layer groups, which confusingly are called layer1-layer4. Each of those layer groups contains two residual blocks with two convolutional layers. This results in 16 layers, meaning there are still two missing. These are the initial convolutional layer and the final fully connected layer. Note that, other

model specifics, such as batch normalization layers, are not counted towards the layer count.

Listing 5.1 presents a code snippet, depicting how the model is split into a privatized head and common backbone. The thesis conducted experiments with various degrees of privatization. These experiments are described in Section 6.3. The most promising configuration is depicted in the mentioned code snippet and represents the standard degree of layer privatization used in Chapter 6.

Listing 5.1: Simplified Code Snippet Visualizing the Backbone/Head Layer Implementation

```
import torchvision
import torch.nn as nn
model=torchvision.models.resnet18(pretrained=False)
backbone = nn.Sequential(
    model.conv1,
    model.bn1, # Batch normalization
    model.relu, # Activation function to introduce non-linearity
    model.maxpool, # Reduces spatial dimensions
    model.layer1,
    model.layer2,
    model.layer3,
)

head = nn.Sequential(
    model.layer4,
    model.avgpool, # Reduces spatial dimensions
    nn.Flatten(), # Converts feature map into 1D tensor
    nn.Linear(in_features=model.fc.in_features, out_features=num_classes
              , bias=True)
)
```

5.2 Aggregation Schemes

To evaluate the performance of the proposed framework, the code has to support different aggregation techniques. As such, the configuration file can specify four different aggregation schemes. These include: No aggregation among clients (None-None), Intra-Task Aggregation with FedPer (FedPer-None), Intra-Task Aggregation with FedPer and Cross-Task Aggregation with FedPer (FedPer-Fedper), and lastly, the proposed approach: Intra-Task Aggregation with FedPer and Cross-Task Aggregation with HCA (FedPer-HCA). Figure 5.3 provides an overview of these 4 aggregation schemes. Once the models have completed their local training round, the code checks the aggregation scheme defined in the configuration file and continues accordingly.

In the following, the thesis gives a rough overview of how these four schemes were implemented. For a more detailed explanation, please refer to the repository directly, as the methods have comments explicitly explaining their intended purpose and functionality.

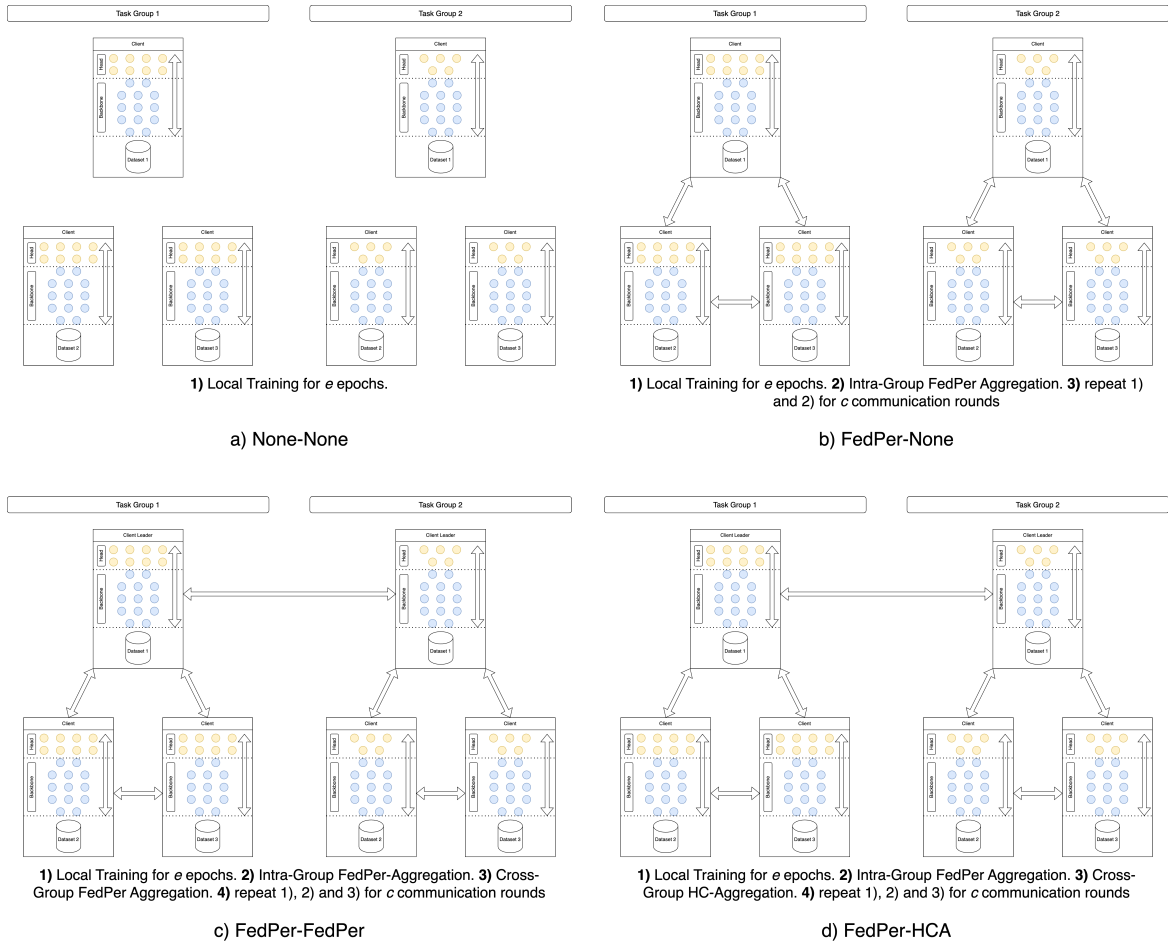


Figure 5.3: Implemented Aggregation Schemes

5.2.1 No Aggregation (None-None)

Implementing no aggregation at all is very straightforward. Once the clients have trained for the specified number of epochs per aggregation round, the code simply jumps to the next aggregation round, without inter or cross-task aggregation.

5.2.2 Intra-Task Aggregation with FedPer (Fedper-None)

Intra-task aggregation with FedPer requires the extraction and averaging of backbones within a task group. Upon completion of a local training round, the clients exchange their backbones with each other. The code does not build a fully decentralized environment to model this communication, as this work was out of scope. Instead, it is assumed that the clients receive the responsible backbones, without having to deal with stragglers, faults or malicious nodes. In addition, for the purpose of the experiment, the client leader is pre-selected and does not change. As changing the client leader would only increase code complexity while still being unable to properly model leader selection, since the code only runs on one machine. Eventually, once the client has received the backbones of all their neighbours, they average their personal backbone with all the received backbones. In code, this is done by calling the `_intra_task_backbone_averaging()` function within the `run.py` script. This function iterates through all clients and generates a list of the neighbour backbones. These are then averaged by calling the client's `average_backbone()` function. Lastly the current backbone of each client is replaced by the the newly averaged backbone.

5.2.3 Intra-Task Aggregation with FedPer and Cross-Task Aggregation with FedPer (Fedper-Fedper)

This aggregation scheme starts the same as the above-described FedPer-None approach. However, once all the clients have their backbones averaged within their respective task group, the leader shares the newly averaged backbone with the other task group leaders. Each leader then averages its backbone with the one it has received. In the code, the leader is designated as the first object in the list of clients per task, making the leader selection straightforward. The code then iterates through all clients of the tasks and replaces their backbones with the newly computed cross-task group average.

5.2.4 Intra-Task Aggregation with FedPer and Cross-Task Aggregation with HCA (FedPer-HCA)

This is the proposed aggregation scheme of this thesis inspired by [11]. As such, it warrants a close inspection.

Like the previous two aggregation mechanisms, FedPer-HCA also uses FedPer to aggregate tasks within the task group. However, once the leader has received the averaged group-based current and previous backbones, it starts the HCA mechanism. To do this, the

leader client calls the *conflict_averse()* function and passes the current and past averaged backbones of itself and the other leader as arguments. In addition to these backbones, it must pass the alpha parameter, controlling the extent of the hyper conflict averse aggregation scheme. The *conflict_averse()* function of the leader client then reroutes the code to the *conflict_averse()* function located in the *hca.py* script, which handles everything related to the conflict-averse mechanism. The code in *hca.py* was created by [11] and has only seen minor modifications as part of this thesis.

Once the *conflict_averse()* function within *hca* gets called, the parameter dictionaries, names, layers and shapes are extracted from the current and past backbones. These parameter dictionaries, together with the layers, are flattened. Flattening is a technique to transfer parameters into a single vector, allowing easier manipulation and comparison of model parameters across models. In addition, the code calls *get_delta_dict_list()* to get the difference of parameters between the current and previous averaged backbones. This essentially calculates the update of one communication round prior to using *hca*. These deltas are then used with the alpha parameter to call the *get_ca_delta()* function. This function represents the core implementation of *hca*.

The function *get_ca_delta()* calculates the aggregated conflict-averse update *delta*, based on the passed flattened parameter updates *flatten_delta_list*, using an optimization approach that considers the conflict-averse parameter. Therefore, it represents an implementation of the mathematical formula by [11], shown in 4.2.2. It starts by stacking the parameter changes into a matrix called *grads*. Each column within that matrix represents a different flattened update. Based on this, a gram matrix [136] is computed. This matrix captures the relationships and conflicts between these changes. The parameter *alpha* influences how strongly the function should penalize conflicting updates, affecting the overall smoothness of the optimization. Additionally, an objective function *objfn* is defined. This function incorporates *alpha* into a combination of linear and square root terms. This minimizes conflicts while maintaining effective updates. The optimization uses *scipy.optimize.minimize* [137] to find the optimal set of weights *ww* that minimize the above objective, reflecting the contribution of each gradient to the final update. After obtaining the weights, a weighted sum of the gradients *gw* is computed, and a scaling factor *lmbda*, dependent on *alpha*, is applied to adjust the aggregated update. The final update is calculated based on the *rescale* and *alpha* parameters, where an increase in alpha reduces the magnitude of the final update. To summarize: The *get_ca_delta()* function returns the conflict-averse update, with alpha managing how aggressively conflicting gradients should be addressed.

Once the conflict-averse update is calculated and applied to each averaged task group backbone, these updated task group backbones replace the backbones of the clients within their respective groups. It is important to note that this process results in distinct backbones for each group, as the backbones are not averaged across groups but are instead adjusted slightly in a specific direction with the hyper conflict-averse update.

5.3 Available User Interactions

When using the implementation, one can run a federation or plot the results of runs. Examples of these commands, as well as instructions on how to install the implementation can be found by following the link to the source code provided in Section 1.2.1.

5.3.1 Run

In order to run a federation a user needs to reference the configuration files and pass them as an argument. This will trigger the *run.py* file. Based on the provided configs, the code will, if necessary, download the required data. Following this, the dataset will be preprocessed and the dataset specific dataloaders will be generated. These dataloaders, together with information about the experiment and the applicable model, are used to instantiate the specified amount of clients and tasks, by passing this information to the *client_manager.py* and *client.py*. Currently the setup only supports two different task types per federation, but this could easily be extended in the future.

Once the clients are generated, the training starts. The code iterates through the tasks and respective clients and trains them for the specified amount of epochs. Once a training round is complete, the aggregation round starts. Potential aggregation approaches are the above-introduced setups: None-None, Fedper-None, Fedper-Fedper, and Fedper-HCA. After successful aggregation, the models resume training. Once all aggregation rounds are completed, the final model is tested and saved to the *results* folder.

5.3.2 Plot

A user can generate plots and tables by starting *plot.py* and passing the results that shall be visualized. The script first loads the results specified and then iterates through them, visualizing each. A user can also define whether cross-configuration plots shall be generated. These plots compare different setups, like different aggregation schemes or levels of layer privatization. The plots show the average of different experiments, making comparison among them more accessible. Again, example commands for plotting and table creation can be found by following the link to the source code provided in Section 1.2.1.

5.4 Reproducibility

A significant challenge when training neural networks is reproducibility. Given the inherent randomness in training models, one has to try to limit potential sources of non-deterministic behaviour. The first step towards reproducible code can be made by relying on a package management system. For this Conda [138] was chosen. Conda is an open-source tool for package and environment management. It allows users to easily install,

update, and manage software packages. More importantly it allows the creation of virtual environments. Such an environment was defined for this implementation, enabling a user-friendly setup experience and comparable computing environments.

While using a virtual environment is great, it does not address the randomness introduced by functions prevalent in neural network training. A lot of potential non deterministic behaviour can be addressed but it is stated that full reproducibility is not always possible with PyTorch, especially between CPU and GPU executions, even when identical seeds are used [139]. As running the experiments on a CPU would not be feasible anyway, the thesis focuses on GPU reproducibility. Every experiment has its own seed that controls various sources of randomness, such as numpy, torch and Python’s random functions. In addition to that, the thesis followed the PyTorch Guideline on reproducibility [139] and forced PyTorch to use deterministic algorithms. This forces the PyTorch library to use operations that produce the same result upon code reruns, regardless of the hardware or execution environment.

5.5 Limitations

The main objective of the implementation is to test the proposed frameworks performance against other aggregation mechanisms. Given the time constraints, it was not possible to implement the framework fully decentralized, nor was it possible to cover all potential challenges, such as faults and stragglers.

5.5.1 Decentralization

Implementing the proposed framework to simulate a decentralized behaviour is non-trivial. While the research group at the University of Zurich uses a decentralized platform for FL, extending the platform to incorporate the framework was not feasible. This is because the platform was not built with MTL in mind. Extending it to manage clients with different datasets, models and aggregation mechanisms, has been deemed too time-consuming, given the thesis time constraint of 6 months. Alternatively, the thesis investigated the feasibility of leveraging multi-threading. The idea was to assign each client its own thread, allowing parallelization. However, this approach severely limits the number of simulated clients, as one quickly reaches the hardware capacity. Therefore, it was decided to focus on the performance of the proposed framework, as a proper decentralized implementation would only be warranted if the framework shows promising results anyway.

5.5.2 Leader Selection, Faults, Stragglers and Late Joiners

The topics of leader selection, faults, stragglers and late joiners were discussed in the previous Section 4.2.3. Potential solutions to these problems have been proposed. However, implementing and investigating these challenges requires a proper decentralized setup, which, as explained above, was not feasible. As such, leader selection, faults, stragglers

and late joiners are not implemented. Instead, the framework chooses the first client of a task group as a leader without changing the leader after initial setup.

Chapter 6

Evaluation

Two heterogeneity types were investigated to evaluate the performance of the proposed architecture: class label heterogeneity and task heterogeneity. For both of these heterogeneity types, a specific dataset was chosen. After initial tinkering and intermediate results, it was decided to run three experiments for each heterogeneity type and consequently dataset.

The thesis first compared different aggregation frequencies within the proposed FedPer-HCA framework and found that aggregating every second training round was optimal. This finding was then used in subsequent experiments. Next, it explored various levels of layer privatization. The backbone/head split, which was eventually chosen, has already been depicted and discussed in Listing 5.1 and Section 5.1.2, respectively. Note that, after the backbone head split was selected, the first experiment had to be rerun with the appropriate split. Finally, the most exciting experiment was the evaluation of different aggregation schemes. This experiment compared the proposed FedPer-HCA framework with the previously discussed None-None, FedPer-None and FedPer-FedPer aggregation mechanisms. A visualization of these schemes can be found in the previous chapter in Figure 5.3. The experiments and their goals are depicted in Table 6.1.

In the following, the thesis will explain the experiment setups. Afterwards, the experiments will be presented and discussed independently, while considering both class label and task heterogeneity. Finally, the chapter concludes with a broader discussion of challenges faced and main takeaways. Additional plots can be found in Appendix B and on the Google Drive referenced in Section 1.2.1. The code for the experiments is linked in the same Section 1.2.1.

6.1 Setup

As discussed, the experiments focus on class label and task heterogeneity. To assess how client performance is impacted by these heterogeneity types, the thesis will present a range of evaluation metrics. Once the metrics have been established, the thesis will describe class label and task heterogeneity separately. It will motivate both and describe the chosen dataset, federation structure and model specifics.

Table 6.1: Experimental Evaluation of the Proposed Framework

Experiment	Goal
Aggregation Frequency	Determine a suitable aggregation frequency for the proposed framework.
Degree of Privatization	Determine a suitable backbone/head split for the proposed framework.
Aggregation Scheme	Use the above experimental findings to compare the proposed framework with other aggregation schemes.

6.1.1 Examined Metrics

The experiments will demonstrate model performance on four widespread metrics: loss, precision, recall and f1. While they influence each other, each of them shows different aspects of model performance.

The loss quantifies the difference between the network’s predictions and the target values. It is crucial for training a model, as the learning process aims to minimize this loss. It does so by adjusting the network’s weights through backpropagation and optimization algorithms (e.g., Gradient Descent). Many different loss functions exist which are chosen depending on the network’s task. Well known loss functions are the Mean Squared Error (MSE) and the Cross-Entropy loss. The MSE calculates the average of the squared differences between predicted and actual values, making it suitable for regression problems. Compared to this, the Cross-Entropy loss is often used for classification tasks, as it measures the difference between two probability distributions. Generally, the lower the loss, the better the network.

However, the loss metric might not always accurately represent the performance of a network. This is especially true in situations where the data is imbalanced. Consider a dataset with 100 images, of which 90 are dogs and 10 are cats. If the model is capable of predicting dogs very well, it will result in a low loss, even if all cats are wrongly classified. This issue can be addressed with the metrics precision, recall and f1.

Let us continue with the above example, describing the metrics from the ”cat” perspective. The cat precision shows how many images classified as cats, are actually cats. Hence, a high cat precision shows that the model is usually correct when classifying an image as a cat. Contrary to this, the cat recall measures how many images containing a cat, were correctly classified as such. Therefore, a high recall means the model only missed a few cat images. Let’s say the model classifies almost all of the 100 images as cats. That means that the recall is very high, since it probably caught most cat images. However, the precision must be very low, since the dataset only contains 10 cats, resulting in many wrongly classified images. These two metrics can be combined into the f1 score. This metric provides a unified score that shows how well the model performs while considering both metrics.

In the following experiments the precision, recall and f1 are calculated using torchmetrics

[140] with the "micro" averaging strategy. That means that the metrics are computed across all classes. Referring to the example above, the metrics are calculated by using the total counts of true positives, false positives, and false negatives of the dog and cat classes combined, resulting in a single precision, recall and f1. Note that the test metrics do not extend across task groups in the following experiments. That is because the class labels (in the class label heterogeneity setup) and the tasks (in the task heterogeneity setup) differ among task groups. As such, each task group has its own loss, precision, recall and f1.

6.1.2 Class Label Heterogeneity Setup

As discussed in 2.2.3, class label heterogeneity refers to a federated environment in which clients train a model on different subsets of class labels [64]. This environment can be modelled by assigning each client the same model architecture with an identical number of output features. However, given that each client only encounters a subset of all possible labels, such an approach seems counter-intuitive and might negatively impact learning performance. Also, it requires that each client is aware of all possible labels, potentially leading to a privacy risk. To address this, the model has to be split into a head and backbone, as described in previous sections of this thesis. Clients that encounter the same set of class labels are grouped together to form a task group. Consequently for the three experimental setups where aggregation is necessary, only the backbone is shared among clients. Although sharing the entire model within a group is possible, it was avoided to maintain consistency in the experimental setup.

Setting up the class label heterogeneity experiment involves selecting an appropriate dataset, defining the federation structure, and choosing a suitable model architecture. The implementation of this setup can be found in the repository linked in Section 1.2.1.

Dataset

The CIFAR-10 [21] dataset was chosen to create such a federated environment. The goal when using it for image classification tasks is to classify each image into one of the 10 different classes of the dataset. Some exemplary research in the realm of FL that makes use of the CIFAR-10 dataset are the previously introduced FedEM [56], FedU [63] and MTFL [66] frameworks. The dataset holds 60000 32x32 color images, with 6000 images per class. The relatively low complexity of the task and small image size makes using shallower models suitable, resulting in less computing time. Important to note is that the classes are mutually exclusive, meaning there is no overlap between classes. Figure 6.1 by [141] gives an overview of the existing classes accompanied by example images.

Federation

Two task groups were created to model a federated environment with class label heterogeneity: animals and objects. As such, the clients of the animal group only see images

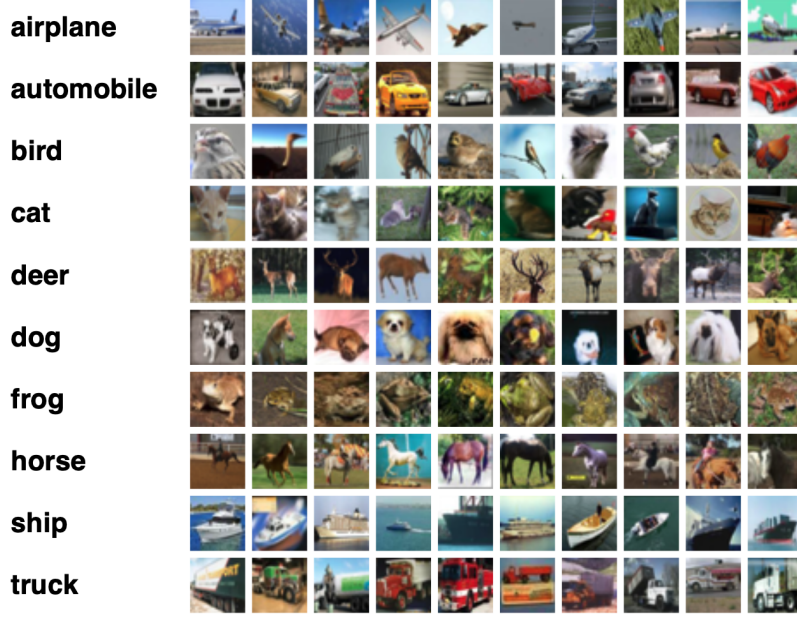


Figure 6.1: Cifar-10 Dataset as Visualized by [141]

with the labels bird, cat, deer, dog, frog and horse. Conversely, the object group clients only see images labelled airplane, automobile, ship and truck. The federation itself holds 6 clients, with both of the groups holding three clients each. The dataset is split into a train and test set of 50'000 and 10'000 images respectively. To support this setup, the train and test datasets are further divided into dataset containing either animal or object images. All clients of the same group receive the same test dataset. The train splits are evenly distributed to the clients according to their group, with no overlap among clients. The training data for each client is further split into a training and validation set with a ratio of 0.2. This approach ensures that each client has its own unique training and validation data while guaranteeing that all clients within the same group use the same test set. Since each client only sees a subset of all data samples, the risk of overfitting is high. Hence, various image transformation techniques were applied, such as Randomized Cropping, Randomized Flipping and Color Jittering. Generally, all clients are trained for a total of 30 epochs, with an aggregation round after every second epoch. This is true for all models, except the ones taking part in the aggregation frequency experiments. The results of these experiments motivated the choice of two local training epochs per aggregation round. The aggregation frequency experiments are presented and discussed in Section 6.2.1.

Models

Each client uses parts of a non-pretrained Resnet18 [133] model as backbone, as described in Section 5.1.2. Since the object group clients encounter four labels and the animal group clients train on six different labels, the shape of the final output tensor does not match, making a backbone/head split necessary. At the very least, the last linear layer of the

respective model needs to be tailored to each specific task group because each group has a different number of output classes. In other words, since the output dimensions vary between groups, a shared final layer does not work. The experiments in Section 6.3.1 inspired the following backbone head split. Each single-label classification model creates a head with the final block of ResNet18 and a final output layer. The exact implementation, including additional model specifics like pooling, is described and visualized in the previous Section 5.1.2. Each model focused on class label heterogeneity uses Cross Entropy Loss and Stochastic Gradient Descent with a learning rate of 0.01, momentum of 0.9 and weight decay of $1e-3$. Please see the provided source code for the complete implementation.

6.1.3 Task Heterogeneity Setup

Compared to label heterogeneity, task heterogeneity is more complex. In label heterogeneity, all clients, regardless of their group, work on tasks of the same type [64]. However, this is not true for scenarios involving task heterogeneity. For instance, instead of all clients solving tasks of the same type, some might conduct image classification while others handle object detection. When tasks differ to this extent, implementing at least task-specific final layers becomes essential. Furthermore, they may require distinct loss functions and optimizers, adding complexity to cross-task aggregation. As a result, dividing the models into an aggregatable backbone and task-specific heads is one approach to facilitate an aggregation-based solution. More information about task heterogeneity is provided in Section 2.2.3.

Like the class label heterogeneity experiment, the task heterogeneity experiment also involves selecting an appropriate dataset, defining the federation, and choosing a suitable model architecture. Like for the class label heterogeneity setup, the code is available and linked in Section 1.2.1.

Dataset

A suitable dataset for task heterogeneity must support more than one task setting. Therefore, the previously used CIFAR-10 dataset is not applicable. In addition to accommodating multiple tasks, the dataset should still offer relatively simple tasks and use reasonably small image sizes. This ensures that the federation can be run and tested on standard GPU hardware, making the experiments easily reproducible at home. If results are promising, scaling to more complex tasks can be done at a later stage with potentially more powerful hardware, or by simply committing more computing time. As such, common large-scale MTL datasets like COCO [142] or NYU Depth V2 [143] are not suitable. Eventually, the thesis decided on the CelebA [22] dataset. It consists of more than 200'000 face images belonging to more than 10'000 different celebrities. An example image of the CelebA dataset is depicted in 6.2. Each image comes with up to 40 binary attributes and five landmark locations, making the dataset suitable for a wide range of tasks such as face attribute recognition, face detection, landmark localization, face editing and more [22]. While not as well known as the CIFAR-10 dataset, it has gathered attention, especially for experiments focusing on facial features [144], [145]. The thesis decided to use face

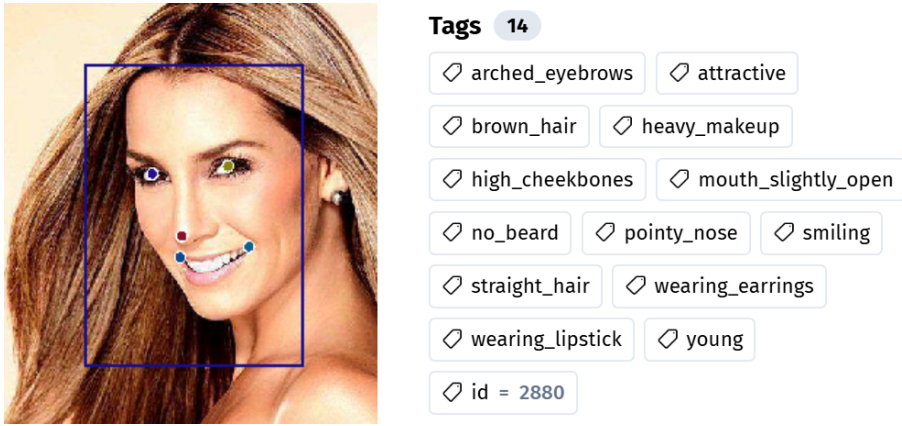


Figure 6.2: An Image of the CelebA Dataset with Labels as Depicted by [22]

landmark detection and face attribute classification to model heterogeneous tasks, similar to a previous study by [77]. This modelling results in two distinct task groups. One group of tasks are training to classify face attributes, while the others perform face landmark detection.

Federation

As mentioned, the thesis utilizes face landmark detection and face attribute classification to represent two distinct task groups, thereby capturing task heterogeneity. The face landmark detection task involves predicting key points in an image, such as the nose, eyes, and mouth edges. In contrast, the multi-label classification task assigns one or more of 40 possible face attributes to an image. Similar to the class label heterogeneity experiments, six clients are defined, each belonging to one of the two groups. To save computing time and resources, only 40 percent of the dataset was used for training. Each task group receives half of the reduced dataset and distributes it evenly among its clients, who then split their training data into training and validation sets with a ratio of 0.2. Again, all clients of the same task group have the same test dataset. Randomized Rotation and Horizontal Flips are introduced to combat overfitting. Like the setups with class label heterogeneity, all clients are trained for a total of 30 epochs, with an aggregation round after every second epoch. As previously discussed, this is true for all models except the ones taking part in the aggregation frequency experiments presented in Section 6.2.2.

Models

The face landmark detection and the multi-label classification task make use of non-pretrained ResNet18 models, like in the class label heterogeneity setup. Again, please see the Section 5.1.2 for an explanation of ResNets. The models are split into a common backbone and task-specific heads. The head holds the final output layer and the last block of ResNet18, as defined by torchvision [135]. The experiments in Section 6.3.2 motivated a split where the privatized head contains the final torchvision block of ResNet18 and a

final output layer. Please see listing 5.1 and Section 5.1.2 for a detailed depiction and explanation of this implementation. Compared to the previous experiment, the tasks differ more significantly. Face attribute detection is a multi-label classification task, while face landmark detection translates to a regression task. These differences require the use of different hyperparameters. Therefore, the multi-label classification models use a Binary Cross Entropy Loss and an AdamW optimizer with a low learning rate of $5e-5$ and a weight decay of $5e-3$. The facial landmark detection models use a standard Mean Squared Error Loss as well as an Adam optimizer with a learning rate of $5e-4$.

6.2 Impact of Aggregation Frequency on the Framework

The first experiment focused on the frequency of aggregation rounds. Hence, the goal was to determine how many local epochs a model should train in the FedPer-HCA framework to reach good performance. To examine this, the thesis investigated class label heterogeneity and task heterogeneity separately. For each heterogeneity type the thesis trained three federations. Each federation holds two task groups as defined in Section 6.1. The first federation performed the FedPer-HCA aggregation scheme after every second local training round. The second and third did so after every third and fifth epoch. This resulted in the training of 18 models per heterogeneity type, resulting in 36 models total. That is because there exist three clients per task group, two task groups per federation and three federations per heterogeneity setup, of which there exist two. All models were trained on a Nvidia GeForce 3080. Overall training time of this experiment added up to approximately five hours.

6.2.1 Aggregation Frequency and Class Label Heterogeneity

Figures 6.3 and 6.4 depict the averaged test metrics for the CIFAR-10 animal and objective classification clients respectively. The best value per column is indicated with opening and closing brackets. Looking at Figure 6.3, one can observe that the most frequent aggregation scheme produces the best test values in all metrics for the animal classification task group. Interestingly, this is not the case for the object classification task group, visualized in Figure 6.4. However, the values in the object classification task group with two epochs per aggregation round don't differ much from the ones with three epochs per aggregation round. One can also see that the recall, precision and f1 score metrics are relatively close to each other. This was expected as the dataset is distributed evenly, meaning clients have the same amount of data samples per class. While the thesis also observes that the overall performance of the animal classification task is significantly better, this topic is postponed to the main analysis in Section 6.4.

A potential answer to the question of why worse performance with two epochs per aggregation round was observed for object classification, can be found in its training dynamics, depicted in Figure 6.6. While the validation loss and f1 score seem to perform better overall, compared to less frequent aggregation schemes, they don't do so consistently. There are various small drops in performance during training that, result in it falling

Configuration	Loss	Precision	Recall	F1
CIFAR-10_FedPer_HCA_2epr	[0.673]	[0.779]	[0.768]	[0.769]
CIFAR-10_FedPer_HCA_3epr	0.786	0.764	0.732	0.729
CIFAR-10_FedPer_HCA_5epr	0.801	0.743	0.727	0.717

Figure 6.3: Average Client Test Metrics for Animal Classification with Different Epochs per Aggregation Round

Configuration	Loss	Precision	Recall	F1
CIFAR-10_FedPer_HCA_2epr	0.37	0.888	0.878	0.877
CIFAR-10_FedPer_HCA_3epr	[0.333]	[0.895]	[0.891]	[0.891]
CIFAR-10_FedPer_HCA_5epr	0.383	0.884	0.874	0.874

Figure 6.4: Average Client Test Metrics for Object Classification with Different Epochs per Aggregation Round

behind less frequent aggregation schemes. The slight decline beginning at epoch 25 is particularly problematic, as it is likely the main reason why the final test performance of aggregating every second round lags behind the less frequent scheme. Early stopping would be a potential solution to this problem. But if early stopping would be applied, the animal classification task group could be impacted in unforeseen ways, so the thesis decided against implementing such a mechanism and leaves that open for future research. Figure 6.5 paints a similar picture for the animal classification validation loss and f1 score, where the frequent scheme with two local training epochs significantly outperforms the less frequent ones by a wider margin.

The thesis concludes that an increase in the aggregation frequency has the potential to increase the performance of the FedPer-HCA framework in this experimental setup. As such, it was decided to conduct all remaining class label heterogeneity experiments of this thesis, with two epochs per aggregation round. The setups slight decrease in performance in the object classification group was weighted less than the relatively significant performance gain observed in the animal classification group.

6.2.2 Aggregation Frequency and Task Heterogeneity

Similar to the aggregation frequency experiments conducted with class label heterogeneity on the CIFAR-10 dataset, the impact of aggregation frequency in scenarios with heterogeneous tasks also shows mixed results. Figure 6.7, depicting the test metrics of the face attribute classification task, shows a performance increase with more frequent aggregations. Interestingly, one can observe a stark difference between the precision, recall and f1 metrics. The low recall makes sense. Given that it is a multi-label classification task with 40 labels, it seems challenging to predict them all confidently. Surprising to the author is the relatively high precision, indicating that if a model predicts an attribute, it is often correct. This precision lifts up the whole f1.

The training dynamics of the face landmark classification task in Figure 6.9 provided some interesting insights. The f1 clearly illustrates the advantage that aggregating after every second training round has over the less frequent schemes. Moving on to the second

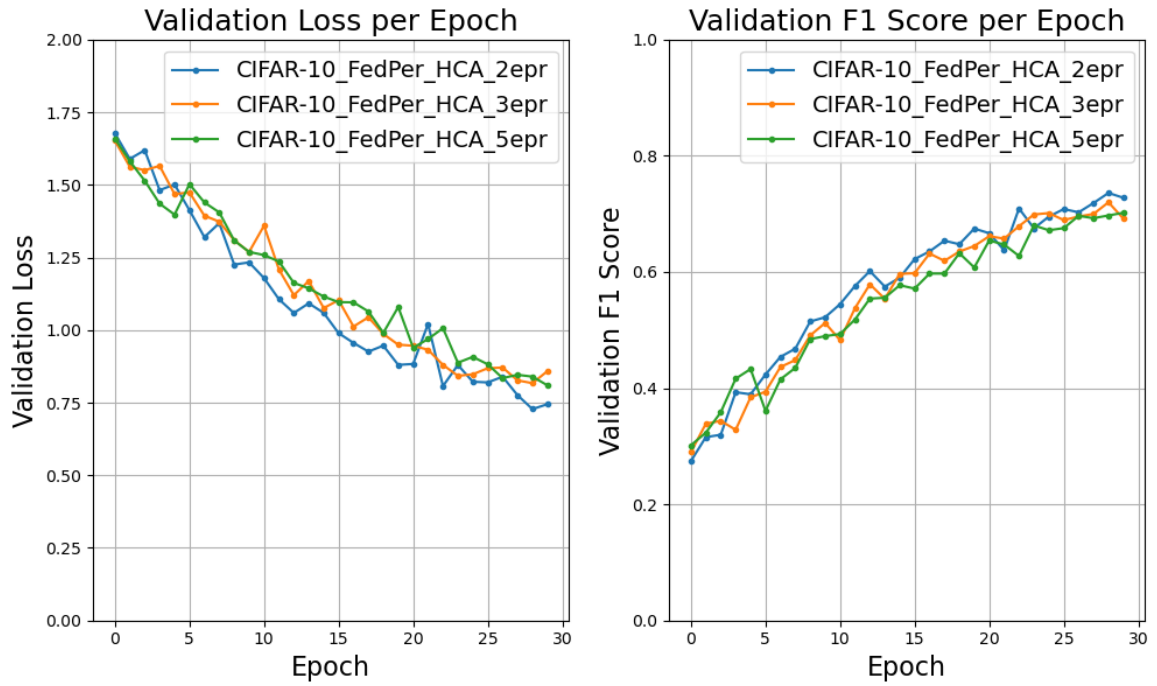


Figure 6.5: Average Client Validation Metrics for Animal Classification with Different Epochs per Aggregation Round

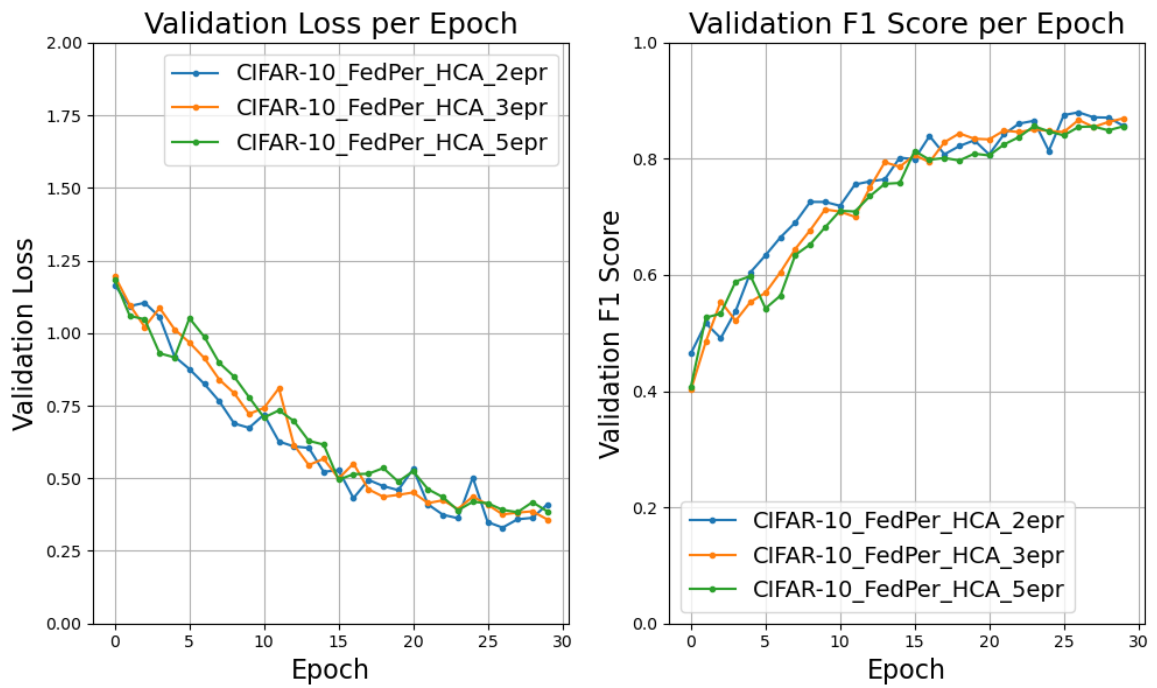


Figure 6.6: Average Client Validation Metrics for Object Classification with Different Epochs per Aggregation Round

task, landmark detection, the best performance is notably achieved with three aggregation rounds per epoch. The reason for this remains unanswered. Note that Figure 6.8 does not depict precision, recall and f1 metrics. That is because they do not directly apply to the implemented regression task. It is also worth noting that, the mean squared error loss is shown since it is a regression task. Hence, there is a big difference in values compared to the loss of the face attribute classification task. Investigations of the resulting face landmark detection loss plot did not provide any additional insights. That is why it is not shown here. It, among all other plots, are accessible via the links provided in Section 1.2.1.

In conclusion, the thesis opted for two local training epochs per aggregation round, similar to the class label heterogeneity setting. Although the performance gains in the face attribute classification task group were minor, they were sufficient to justify the slight performance decrease in the face landmark detection task group.

6.3 Impact of the Privatization Degree on the Framework

In addition to variations in aggregation rounds, explored in the previous section, the number of privatized layers (i.e. non-aggregated layers) can also vary. As discussed in Section 2.3.1, FMTL is often a subfield of PFL. This also applies to this thesis. As such, privatizing the model layers by only aggregating a subset of the backbone seems intuitive to increase privatization. However, if privatization is pushed too far, the "knowledge" of other task groups will have little to no positive effect on the model. This experiment was motivated by [20]. They found that having at least one block as part of the privatized head in a Resnet34 does improve client performance. The thesis wanted to investigate if that is also true for ResNet18 in combination with the proposed FedPer-HCA framework.

The thesis examined the effect of three different degrees of layer privatization on the proposed framework. Note that this work follows the terminology of torchvision's [135] ResNet18 implementation, described in 5.1.2. As such, a layer refers to a group of two residual blocks, containing two convolutional layers each.

The first degree of layer privatization only privatizes the head, meaning the final fully connected layer. As such, the thesis refers to this setup as fully shared backbone, which translates to no reduction of the backbone layers (BL-0). The second degree reduces the backbone layer groups by one (BL-1) and adds it to the privatized head. Finally, the third privatization degree reduces the backbone layer groups by two (BL-2) and adds them to the privatized head. The BL-1 setup is depicted in the code snippet 5.1 in Section 5.1.2. Note that the BL-0 section, moves the *model.layer4* out of the head and into the top of the backbone. Conversely, the BL-2 model setup moves the *model.layer3* into the privatization head.

Given these three different layer privatization setups, the thesis ran six federations with the proposed framework, three on each heterogeneity type. Again, this resulted in the training of 36 clients. That is, because there exist three clients per task group, two task groups per federation and three federations per heterogeneity setup, of which there exist

Configuration	Loss	Precision	Recall	F1
CelebA_FedPer_HCA_2epr	[0.231]	[0.741]	[0.545]	[0.605]
CelebA_FedPer_HCA_3epr	0.233	0.735	0.542	0.598
CelebA_FedPer_HCA_5epr	0.237	0.733	0.522	0.579

Figure 6.7: Average Client Test Metrics for Face Attribute Classification with Different Epochs per Aggregation Round

result	Loss
CelebA_FedPer_HCA_2epr	5.299
CelebA_FedPer_HCA_3epr	[5.224]
CelebA_FedPer_HCA_5epr	5.762

Figure 6.8: Average Client Test Metrics for Face Landmark Detection with Different Epochs per Aggregation Round

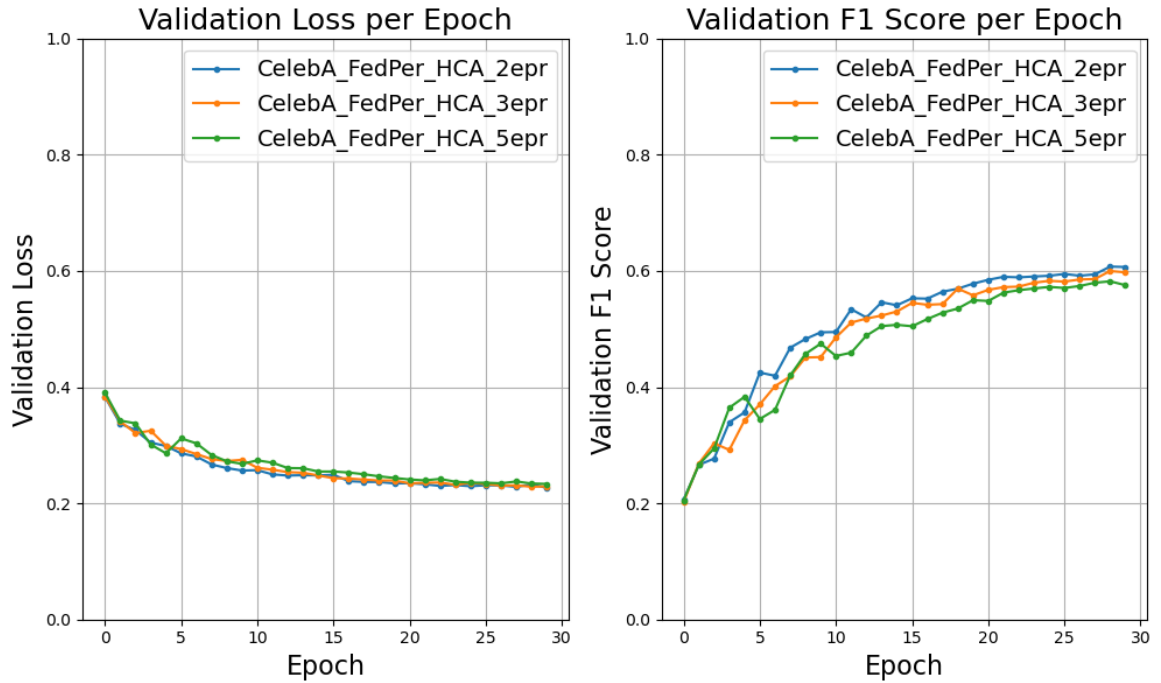


Figure 6.9: Average Client Validation Metrics for Face Attribute Classification with Different Epochs per Aggregation Round

Configuration	Loss	Precision	Recall	F1
CIFAR-10_FedPer_HCA_BL-0	0.706	[0.786]	0.759	0.758
CIFAR-10_Fedper_HCA_BL-1	[0.673]	0.779	[0.768]	[0.769]
CIFAR-10_FedPer_HCA_BL-2	0.81	0.739	0.72	0.716

Figure 6.10: Average Client Test Metrics for Animal Classification with Different Degrees of Privatizations

Configuration	Loss	Precision	Recall	F1
CIFAR-10_FedPer_HCA_BL-0	[0.313]	[0.9]	[0.895]	[0.895]
CIFAR-10_Fedper_HCA_BL-1	0.37	0.888	0.878	0.877
CIFAR-10_FedPer_HCA_BL-2	0.371	0.881	0.876	0.875

Figure 6.11: Average Client Test Metrics for Object Classification with Different Degrees of Privatization

two. Training was conducted on a Nvidia GeForce 3080 and took approximately five hours.

6.3.1 Degree of Privatization and Class Label Heterogeneity

Consider the Figures 6.10 and 6.11 depicting the averaged client test metrics per privatization degree. One can observe that the animal classification clients perform the best on BL-1, albeit, with a reduced precision compared to BL-0. For object classification, the results are painting a different picture. There BL-0 outperforms BL-1 and BL-2. Generally, performance suffers slightly with an increase in privatization, which becomes apparent when examining the Figures 6.12 and 6.13. There, BL-1 tends to be positioned between BL-0, which performs better and BL-2 which performs worse. This indicates that by reducing privatization and consequently increasing parameter aggregation, the models perform better with the proposed FedPer-HCA framework. This is good news! Since it hints at the benefits of using the proposed FedPer-HCA framework over local training, as local training would translate to complete privatization.

Given these results, it would seem logical to chose BL-0 for all subsequent experiments. But, the thesis made the conscious decision against this. This has two reasons. For one, the results of BL-0 and BL-1 are somewhat similar, with BL-1 actually outperforming BL-0 in some metrics. Therefore, picking one over the other only makes a minor difference. Second, and that is crucial, with reduced parameter aggregation, the data that has to be transferred through a network and the computational complexity of the aggregation steps are reduced. While this is of no concern for the simulated federation, it might be for real FL environments. Hence, the minor decrease in model performance was accepted to gain this hypothetical reduction of data transfer and computational complexity.

6.3.2 Degree of Privatization and Task Heterogeneity

By examining the Figure 6.14, which shows the averaged client test metrics per privatization degree, one can see that BL-1 outperforms BL-0 and BL-2 on all metrics except

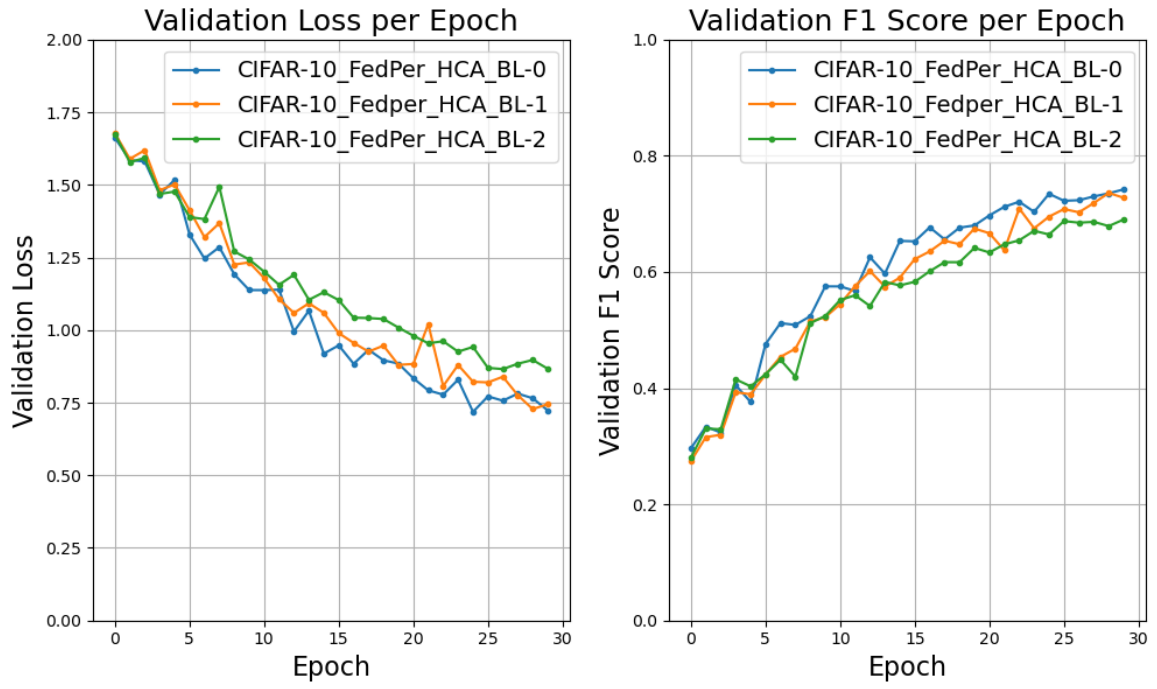


Figure 6.12: Average Client Validation Metrics for Animal Classification with Different Degrees of Privatization

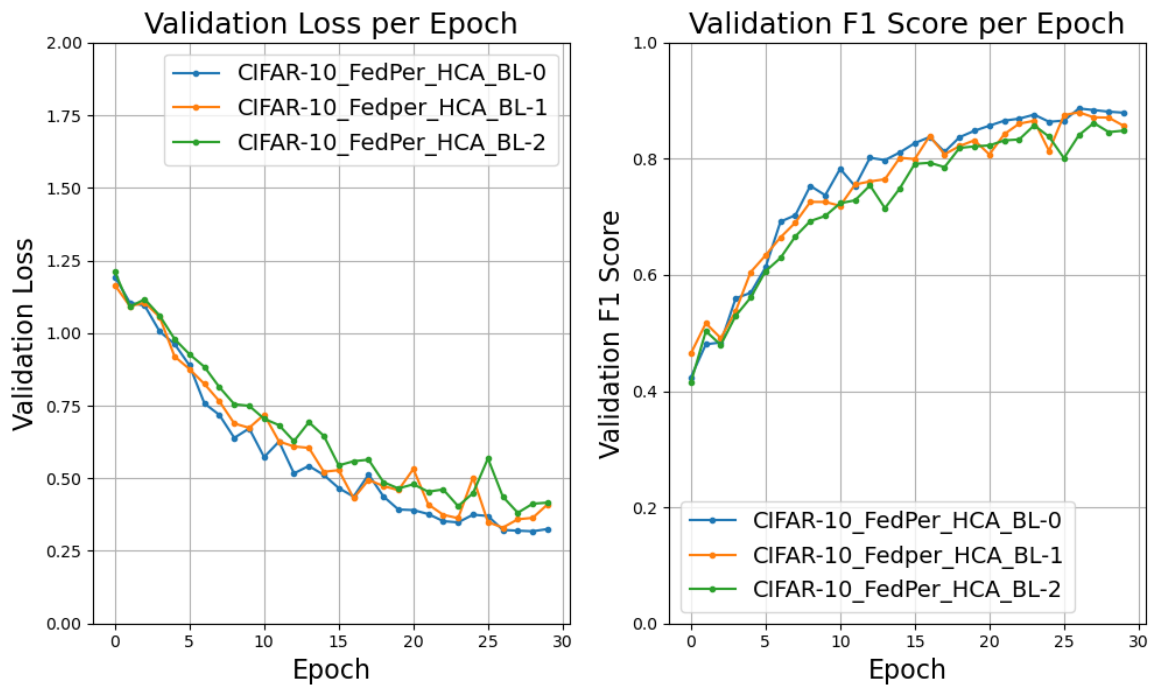


Figure 6.13: Average Client Validation Metrics for Object Classification with Different Degrees of Privatization

Configuration	Loss	Precision	Recall	F1
CelebA_FedPer_HCA_BL-0	[0.23]	0.738	0.541	0.599
CelebA_FedPer_HCA_BL-1	0.231	[0.741]	[0.545]	[0.605]
CelebA_FedPer_HCA_BL-2	0.236	0.734	0.533	0.592

Figure 6.14: Average Client Test Metrics for Face Attribute Classification with Different Degrees of Privatization

result	Loss
CelebA_FedPer_HCA_BL-0	[5.285]
CelebA_FedPer_HCA_BL-1	5.299
CelebA_FedPer_HCA_BL-2	5.829

Figure 6.15: Average Client Test Metrics for Face Landmark Detection with Different Degrees of Privatization

the loss. While confusing at first, one has to keep in mind that the CelebA dataset used for the face attribute classification task, does not have its attributes evenly distributed. In such a scenario, the loss is not that reliable.

A very noteworthy effect can be observed when looking at Figure 6.16. BL-0 suffers a slight drop in performance after the first aggregation round and a considerable drop in performance after the second. BL-1 and BL-2 are not affected by these initial drops. Intuitively, this makes sense. BL-0 dealt with heterogeneous tasks while sharing almost a complete model. Such a scenario was expected to suffer some performance hits. That is because the models might differ so heavily that even HCA can not appropriately compute a conflict-averse update. To be more precise, it can not with the standard alpha hyperparameter of 0.4. Thus, an interesting follow up experiment would be to change the alpha parameter and measure its impact on performance. The thesis assumes that an increase in alpha would reduce the significance of that drop, given that with an increase in alpha the applied update is reduced, as described in 5.2.4. While the loss value of BL-0 on the landmark detection task is lower than that of BL-1, as depicted in Figure 6.15, the lead is relatively small. Sadly, the training dynamics of the landmark detection tasks do not provide any additional insights and are thus not plotted here. Again the plot is available by following the link presented in 1.2.1.

To summarize the above findings, the thesis finds that aggregating almost the complete model with this setup is not appropriate. BL-0 has only a small lead over BL-1 when considering the test loss values. But, it suffers a huge drop in performance after the second aggregation round, leading to delayed model quality. Finally, it also encounters worse performance on f1, precision and recall, when compared to BL-1. All of these observations lead the thesis to take the BL-1 setup for the other experiments. Note, the previous experiments regarding aggregation frequency were rerun with the BL-1 setup, as these experiments initially used BL-0. However, this is already accounted for in this thesis. As the experiments in Section 6.2 were already presented with BL-1.

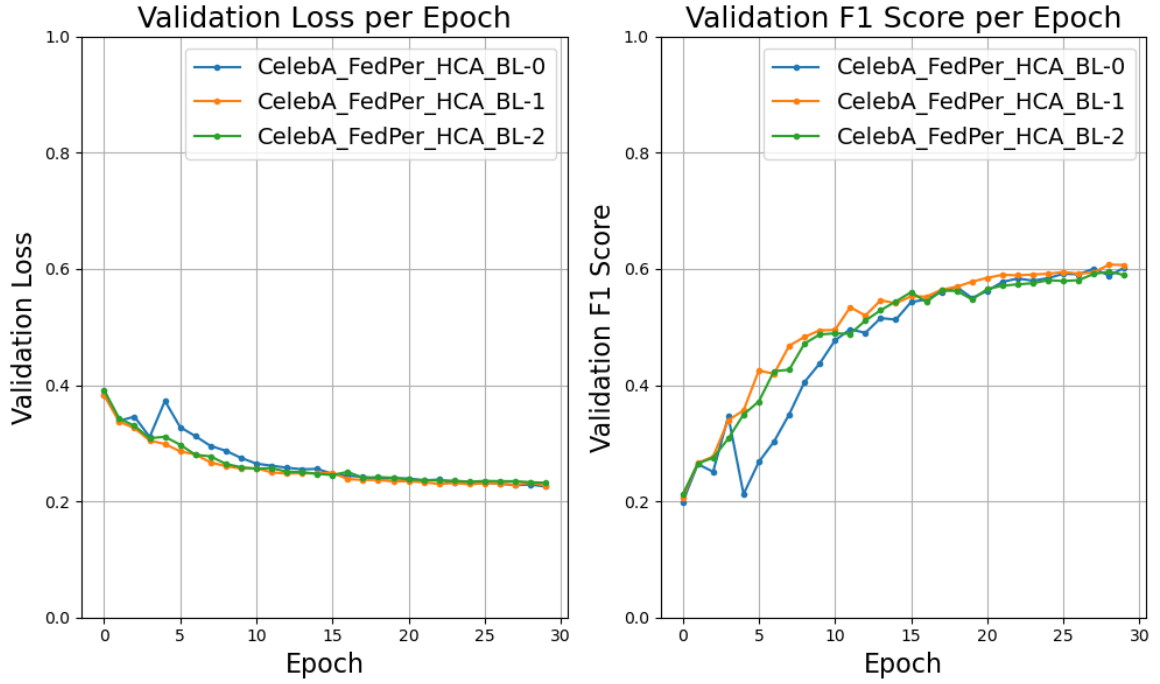


Figure 6.16: Average Client Validation Metrics for Face Attribute Classification with Different Degrees of Privatization

6.4 Evaluation of Different Aggregation Schemes

Having determined an appropriate aggregation frequency and the degree of layer privatization, the proposed framework is ready to be compared with other aggregation schemes. The thesis acknowledges that the other schemes did not have the benefit of going through aggregation frequency and layer privatization testing. Thus, they will use the same amount of privatized layers and aggregation frequencies as the the proposed FedPer-HCA framework. While this makes the results comparable, it also might mean that the compared aggregation schemes might perform better with different degrees of privatization or aggregation frequencies. Regardless, every scheme needs to be evaluated under the same conditions to have a fair comparison.

Both heterogeneity setups were run with the four different aggregation schemes. As previously introduced, these include the proposed FedPer-HCA, None-None, FedPer-None and FedPer-FedPer. These schemes have been discussed at length in Section 5.2 and will not be explained again here. For information about the two examined heterogeneity types, their datasets, federation setups, models and hyperparameters, please see Section 6.1.2 and 6.1.3 respectively. Each aggregation scheme was run once on the class label heterogeneity setup and once on the task heterogeneity setup. This resulted in running eight federations, with in total 48 trained clients. That is, because there exist three clients per task group, two task groups per federation and four federations per heterogeneity setup, of which there exist two. Aggregation was conducted after every second local training epoch, except in None-None, where no aggregation occurs. Layers were privatized according to the previously discussed BL-1 scheme. Again, training was conducted on a Nvidia

Configuration	Loss	Precision	Recall	F1
CIFAR-10_None_None	0.938	0.689	0.675	0.669
CIFAR-10_Fedper_None	0.953	0.723	0.68	0.679
CIFAR-10_Fedper_Fedper	0.853	0.73	0.7	0.698
CIFAR-10_Fedper_HCA	[0.673]	[0.779]	[0.768]	[0.769]

Figure 6.17: Average Client Test Metrics for Animal Classification with Different Aggregation Schemes

Configuration	Loss	Precision	Recall	F1
CIFAR-10_None_None	0.464	0.846	0.84	0.84
CIFAR-10_Fedper_None	0.409	0.867	0.855	0.856
CIFAR-10_Fedper_Fedper	[0.346]	0.879	0.875	0.875
CIFAR-10_Fedper_HCA	0.37	[0.888]	[0.878]	[0.877]

Figure 6.18: Average Client Test Metrics for Object Classification with Different Aggregation Schemes

GeForce 3080, but given the increase in total client count, the computing time increased to approximately 8 hours.

The main goal of this experiment is to evaluate whether clients that leverage knowledge across task groups perform better on their own task. The hope is that the proposed FedPer-HCA framework outperforms other aggregation schemes.

6.4.1 Aggregation Scheme and Class Label Heterogeneity

The Figures 6.17 and 6.18 show the test metrics after 15 aggregation rounds with a total of 30 epochs. Note that the depicted values are the averages of the three involved clients per experiment, and the precision, recall and f1 were calculated as described in 6.1.1. As always, the best value per column is indicated with opening and closing brackets. Looking at Figure 6.17, depicting the different aggregation schemes on the animal classification task, it is evident that the proposed framework outperforms the other schemes across all metrics significantly. The results are also promising when taking Figure 6.18 into account, which shows the same metrics but on the object classification task. While the proposed FedPer-HCA has the lead in precision, recall and f1 score, the loss is lower for the FedPer-FedPer scheme. Overall, the performance gain of FedPer-HCA on object classification is relatively minor compared to the performance gain on animal classification. precision recall and f1 all hold similar values within a task group, which is expected in this setup, since the class labels are evenly distributed within a task group. Training locally (None-None), without any aggregation mechanism, provides the worst results overall, in both animal and object classification. Aggregation via backbone averaging within a task group (FedPer-None) performs significantly better than local training. Again, this is true for animal and object classification. Excluding the loss value, FedPer-FedPer performs similarly to FedPer-None in the case of animal classification and almost reaches the values of FedPer-HCA in object classification.

To better understand the training dynamics and to visualize the superiority of the proposed framework in this experiment, one can also look at the gathered validation metrics. These are depicted in Figure 6.19 and 6.20. As for the test metrics, the plotted validation loss and f1 represent the average of the three clients per aggregation scheme and task. Looking at the two Figures 6.19 and 6.20, it is evident that the suggested framework outperforms the other three experimental setups in classifying animals and objects. However, in animal and object classification it takes about five epochs until the proposed FedPer-HCA framework can clearly surpass the other aggregation schemes. Interestingly, the performance of FedPer-HCA on the object classification task seems to degrade after 25 epochs. This is indicated by an increasing loss value and decreasing f1. However, since this is not the case for the animal classification task, early stopping would be difficult to justify at this stage, as this might be a temporary performance decline. Figure 6.20 might also explain the small test loss performance lead of FedPer-FedPer. Over most epochs, FedPer-FedPer performed significantly worse than FedPer-HCA. The erratic jump at the last epoch pushed it slightly below FedPer-HCA. As such, by excluding the last epoch, the thesis concludes that FedPer-HCA outperforms FedPer-FedPer also in terms of loss.

Another thing that warrants attention is the general better performance of clients conducting object classification. Recall, that animal classification clients are tasked with differentiating between six different class labels, while object classification clients only need to consider four. Having to distinguish between fewer labels, results in an easier task, which is reflected in the overall better performance of clients performing object classification.

Overall, the Figures 6.17, 6.18, 6.19 and 6.20 demonstrate the proposed framework’s superior performance compared to the schemes None-None, FedPer-None and FedPer-FedPer in the case of the studied class label heterogeneity.

6.4.2 Aggregation Scheme and Task Heterogeneity

Similar to the class label heterogeneity results, the Figures 6.21 and 6.22 show the averaged client results after 30 epochs with 15 aggregation rounds on the test dataset for all four aggregation schemes. As mentioned in the other experiments, the precision, recall and f1 metrics were not calculated for the landmark detection task. Again, the best value for each column is indicated with opening and closing brackets. The Figures 6.21 and 6.22 show that the proposed FedPer-HCA framework achieves the best results across all metrics and both task groups. However, the performance gap to the other schemes is smaller than it was in the class label heterogeneity setup. Regardless, the fact that there is a performance increase, albeit a small one, is a success! This shows that despite different tasks (multi-label classification vs regression), both task groups can increase their respective model performance by using the proposed framework!

Interestingly, pure local training performs the worst, like in the class label heterogeneity setup. This is especially true for the landmark detection task. One reason for this could be overfitting, as the introduction of aggregation shocks could work as some form of regularization. But, there is a counterargument. If regularization would be the reason, the proposed FedPer-HCA framework would probably not outperform FedPer-FedPer.

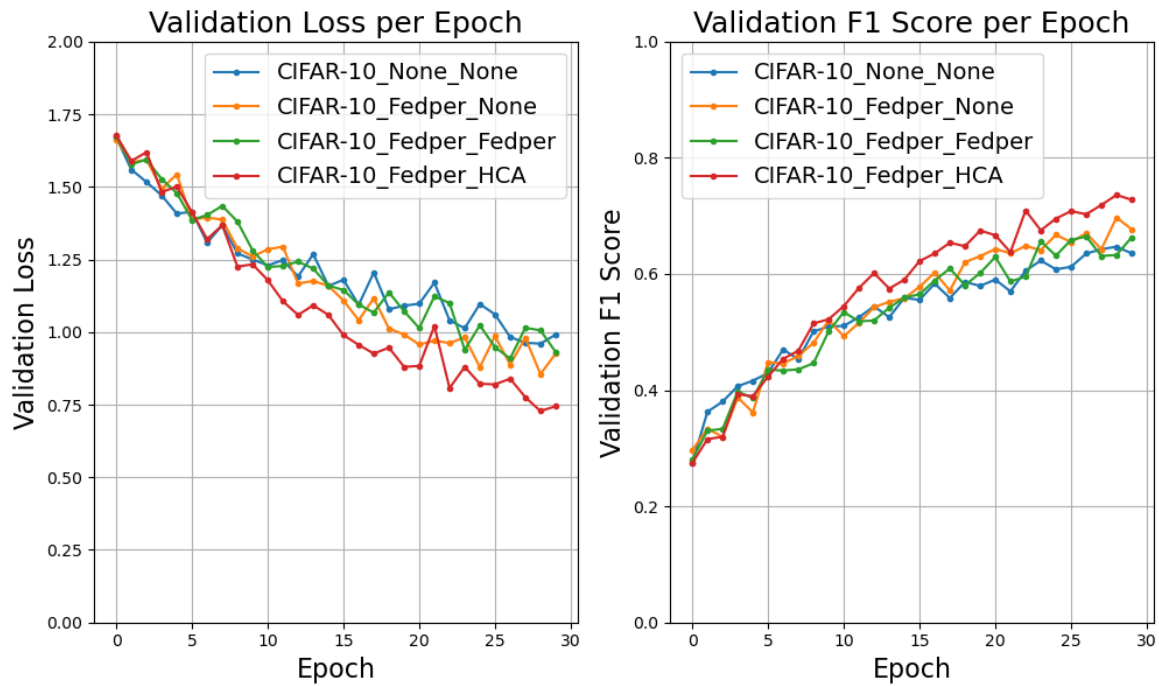


Figure 6.19: Average Client Validation Metrics for Animal Classification with Different Aggregation Schemes

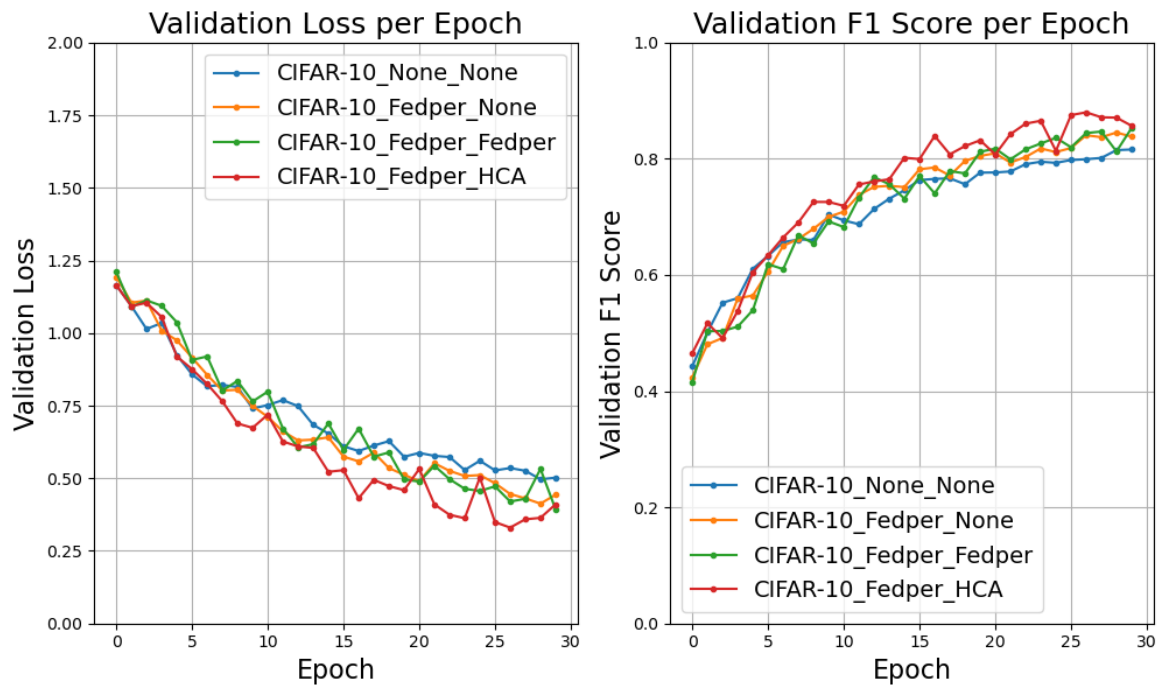


Figure 6.20: Average Client Validation Metrics for Object Classification with Different Aggregation Schemes

Configuration	Loss	Precision	Recall	F1
CelebA_None_None	0.244	0.722	0.511	0.57
CelebA_FedPer_None	0.236	0.733	0.526	0.585
CelebA_FedPer_FedPer	0.239	0.726	0.513	0.571
CelebA_FedPer_HCA	[0.231]	[0.741]	[0.545]	[0.605]

Figure 6.21: Average Client Test Metrics for Face Attributed Classification with Different Aggregation Schemes

result	Loss
CelebA_None_None	6.054
CelebA_FedPer_None	5.392
CelebA_FedPer_FedPer	5.325
CelebA_FedPer_HCA	[5.299]

Figure 6.22: Average Client Test Metrics for Face Landmark Detection with Different Aggregation Schemes

That is because FedPer-FedPer simply averages shared layers, while the FedPer-HCA introduces "smoother" updates, as it only nudges the backbones into a direction, when conducting cross-task aggregation. As such, the thesis can not provide a conclusive answer as to why local training performs so much worse on the face landmark detection task.

Like the previous experiment, investigating the validation metrics captured during training sheds light on the training dynamics. Figure 6.23 illustrates the progression of the validation loss and f1 score for the multi-label classification task. In contrast, plotting the single loss metric for the landmark detection task does not offer any additional insights and is therefore omitted. Still, it is accessible on the GitHub repository. Focusing on the multi-label classification task, depicted in Figure 6.23, one can see the struggle of FedPer-FedPer. It receives a significant hit in performance after the second aggregation round and does not manage to catch up until training concludes. As shown in Figure 6.21, it barely overtakes local training in terms of test metrics. This might be caused by its comparatively hard cross-task aggregation technique, compared to the soft nudge introduced by FedPer-HCA. One can also compare the performance of FedPer-FedPer across class label heterogeneity and task heterogeneity. With class label heterogeneity FedPer-FedPer does not experience an extreme shock to its model performance. This aligns with the introduced theory of conflicting and dominating gradients in Section 2.1.1. In the class label heterogeneity setup, where both task groups perform single-label classification, the tasks are more related, leading to only slight differences in gradients. However, in the task heterogeneity setup discussed here, the gradients are expected to vary significantly, causing aforementioned conflicts and negatively affecting model performance. Compared to FedPer-FedPer, the proposed FedPer-HCA addresses these conflicts with its conflict-averse aggregation scheme. That is likely why FedPer-HCA does not experience such a shock and even outperforms the other studied schemes, despite performing cross-task aggregation.

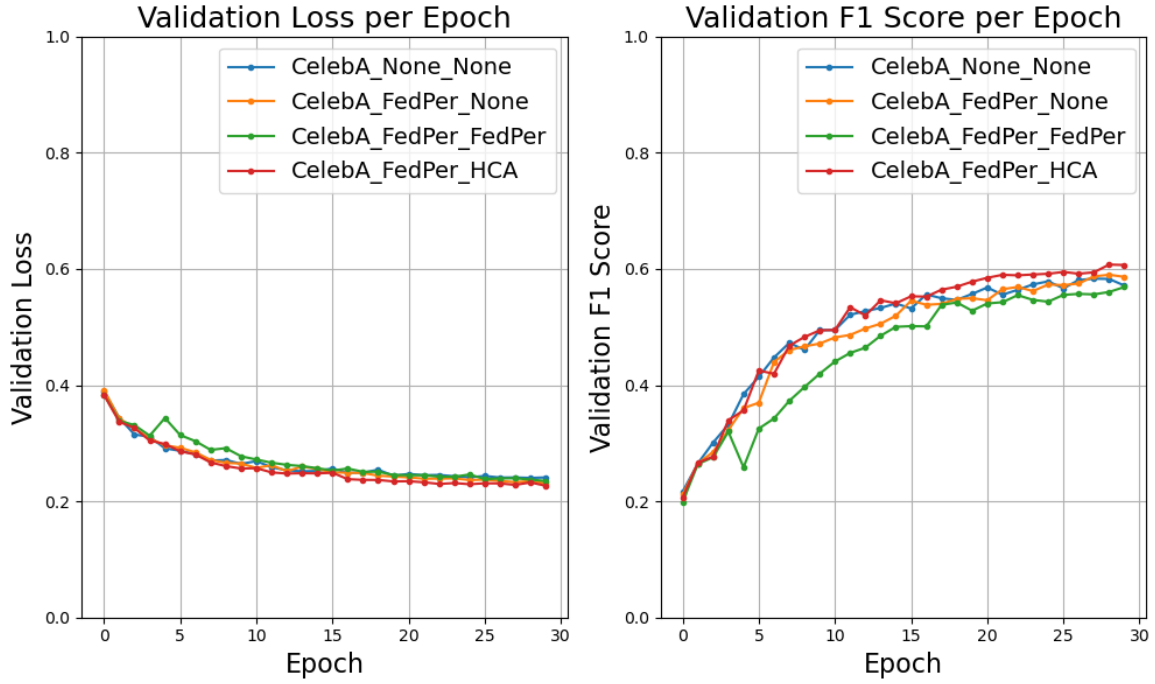


Figure 6.23: Average Client Validation Metrics for Face Attributed Classification with Different Aggregation Schemes

6.5 Discussion

The study has thus far presented results that demonstrate the superiority of the proposed FedPer-HCA framework over the other aggregation schemes examined. However, before outlining the key takeaways from these experiments, it is crucial to highlight the challenges encountered and acknowledge the study's limitations.

6.5.1 Challenges

Simulating the described FL environment was more demanding than training a model on one machine. That is, because the data had to be distributed among clients, reducing the amount of data samples each client encounters and thus increasing the risk of overfitting. Accounting for class label and task heterogeneity between clients added to this challenge. Lastly, the thesis also had to tackle the mystery of suddenly appearing Nan values, which quickly spread through the federation, rendering the models useless. Addressing all these issues was a major time sink, as they required rerunning experiments over and over again to test out different configurations. To do these struggles justice, the following two subsections will be dedicated to hyperparameter selection and handling NaN values.

Hyperparameter Selection

Before the thesis started to train models on the proposed framework, a decent baseline had to be created. As such, the first task was to create a federation that does not overfit when training locally. This was relatively easy for the single-label classification task on the CIFAR-10 dataset. For this task, a PyTorch Lightning article [146] provided a good starting point and motivated using ResNet18. Weight decay and learning rate only had to be adjusted slightly to achieve reasonable performance. After only a few test runs these adjustments together with some image transformation techniques yielded a satisfying result. Getting decent performance without overfitting on the CelebA dataset was more challenging. The thesis did not find a good baseline and had to start adjusting hyperparameters from scratch. As such, the current performance of the CelebA clients could be improved further.

Another hyperparameter that bears mentioning is the hyper conflict averse α , used for the proposed FedPer-HCA framework. After some initial testing it was decided to use the standard value used by [11]. Eventually, the thesis started to investigate the different aggregation schemes properly. During this step the problem of NaN values was first encountered.

Problem of NaN values

NaN values were a major challenge, especially when using the schemes FedPer-HCA and FedPer-FedPer. The emergence of NaN values in a single client infected the whole federation after only one aggregation round. This is because aggregating a NaN value with a non-NaN value resulted in a NaN value. Interestingly, while some emerged during training, others first showed during validation. This section will illustrate the problem and show how the thesis addressed it. However, the thesis did so in a non-rigorous manner, as it was not the main focus.

Many factors can impact the likelihood of NaN values during training. The most straightforward factor is the presence of such values in the dataset [147]. However, this was not the reason, as the datasets do not contain NaN values. Another potential cause are excessive gradients. This can be observed if the loss in the training log starts increasing dramatically [148]. Once the loss becomes too large, values can no longer be represented with floating point numbers, resulting in an overflow and NaN values [148]. Solutions to fix this include decreasing the learning rate [147] or implementing gradient clipping [148]. In fact, the thesis observed that after a few rounds of cross-task group aggregation, the loss started exploding, especially when clients only process a very small amount of data. Following the potential suggestions mentioned above, the thesis tuned the learning rate and implemented gradient clipping [149]. This effectively addressed the issue of NaN values during training.

Handling the NaN values that emerged during validation was more challenging. This scenario seems counter-intuitive, since the validation step itself is not focused on updating model parameters. This issue was especially prominent when clients only saw a

small dataset and conducted experiments with cross-task aggregation schemes like FedPer-FedPer and FedPer-HCA. Looking at existing research, the thesis found that batch normalization layers [119] have been identified to cause adverse effects in FL [150], [151]. Batch normalization is a common technique to stabilize and accelerate training by normalizing inputs with the mean and variance of the previous layer’s outputs, reducing internal covariate shift and maintaining running statistics throughout training [151]. These running statistics are used during evaluation, ensuring that the model makes reliable predictions [151]. This works well for non-federated settings, where these batch statistics are consistent across the entire dataset [151]. However, in FL the local statistical parameters and their gradients differ significantly, which can lead to inconsistencies [150], especially when data distributions of clients differ [151]. As such, this seems in line with what the thesis observed, as the issue seems to be mainly caused by cross-task aggregation schemes like FedPer-FedPer and FedPer-HCA. The data distributions among task groups differ significantly in the class label heterogeneity case and for task heterogeneity contrasting gradients can be expected.

While the most straightforward approach is to remove these batch normalization layers, some studies have proposed more elaborate solutions, such as [151] and [150]. FBN [151] ensures that batch normalization during training is consistent by preserving data distribution and providing running statistics that accurately reflect global statistics. Compared to this, FedTAN [150] ensures performance across various data distributions using iterative layer-wise parameter aggregation. Hence, FedTan is somewhat similar to the backbone head split used in this thesis. An approach discussed in the literature review is MTFL [66]. Following layer privatization, they decided to keep the batch normalization layers private, enabling the model to better fit the client’s data distribution and increasing privacy.

The thesis observed that the issue decreased with an increase in epochs between aggregation rounds. This indicated that the problem could be a lack of updates to the running statistics, as these statistics are used during validation [151]. Given that these batch statistics are updated for every batch [119], the idea of the thesis was to decrease the batch size, thus triggering more frequent updates. Surprisingly, this worked to some degree. Eventually, this allowed to decrease the epochs between aggregation rounds to two. However, it did not allow for one epoch per aggregation round. That is also why the thesis did not investigate that scenario in Section 6.2. To conclude, while the thesis managed to address the issue of NaN values for its specific use case, it does not claim to have found a proper solution nor does it claim to have studied the existing literature on that topic well enough. It also did not approach the issue in a rigorous manner, so no hard evidence for the observed NaN behaviour can be demonstrated.

6.5.2 Main Experimental Findings

To conclude the evaluation chapter, the thesis presents four main findings:

Hyperparameter Selection and NaN Values Demand More Research: As previously discussed, hyperparameter selection and addressing NaN values are major challenges. Although hyperparameter selection is a common practice in ML, the issue of NaN values appears to be more pronounced in FL and is often caused by batch normalization layers

[150], [151]. The thesis managed to address the issue in the context of its experiment but would like to motivate further research in that area.

Degree of Layer Privatization Can Impact the Proposed Framework’s Performance Significantly: In line with existing research [20], the thesis found that privatizing model layers impacts networks performance. The thesis observed this in the case of ResNet18. In the class label heterogeneity setup, performance declined as privatization decreased. This suggests that update directions are relatively similar, so increased sharing is beneficial. In contrast, aggregating almost the entire model caused a significant performance drop in the initial epochs when confronted with the heterogeneous task setting. In this scenario, increasing the number of privatized layers slightly improved performance.

More Frequent Aggregation Rounds Benefit The Proposed Framework: The experiments on aggregation frequency demonstrated that the performance of the framework improves as the number of aggregation rounds increases.

The Proposed Framework Outperformed the Compared Aggregation Schemes in Class Label Heterogeneity and Task Heterogeneity: The conducted experiments show that the proposed framework outperforms the compared aggregation schemes in both the class label heterogeneity and task heterogeneity setup. The performance boost compared to the FedPer-FedPer scheme shows that the hyper conflict-averse aggregation of FedPer-HCA efficiently addresses heterogeneity challenges and outperforms cross-task group backbone averaging.

Chapter 7

Summary and Conclusions

Overall, the thesis was successful in the development and implementation of a framework that addresses heterogeneous clients in the dimensions of class label heterogeneity and task heterogeneity. The conducted experiments showed the superiority of the proposed framework compared to the three aggregation schemes: None-None, FedPer-None and FedPer-FedPer. However, limitations and potential improvements remain which, together with a short summary, will be discussed in this final chapter. Please refer to Section 1.2.1 for accessing the source code, trained models and additional plots.

7.1 Summary

FL can come in many shapes and forms. Traditional CFL relies on a central authority to manage the federation, whereas DFL operates in a fully decentralized setting, where clients communicate directly with each other. Both, CFL and DFL, face the challenge of heterogeneous clients. One approach to deal with heterogeneous clients is FMTL. However, most existing research addresses solely the problem of non-IID data. But, clients might have more profound differences. One client in a federation might solve an image classification task, while another is focused on a regression task. This type of heterogeneity is known as task heterogeneity. While some recent studies emerged that tackle this problem from an FMTL perspective, they predominantly make use of CFL. The goal of this thesis was to explore solutions for such heterogeneous clients and to develop and to evaluate a DFMTL framework.

It did this by first providing and exploring relevant foundational topics related to the purpose of this thesis. This was done in Chapter 2. Once the theoretical foundations were established, the thesis moved on to an in-depth literature review of existing work in CFMTL, DFMTL and HCFMTL. This review was conducted in Chapter 3. It found that most work in FMTL falls under CFMTL. Although some studies address DMFTL or HCFMTL, the combination of the two, defined as HDFMTL, appears to be quite rare. Having gained a profound understanding of existing approaches, the thesis moved on to create its own framework in Chapter 4. The proposed framework divides tasks into task

groups and performs two types of aggregation. First, the backbones (i.e. non-privatized layers) of clients of the same task group are aggregated with an approach inspired by FedPer [20]. Following this, task group leaders, aggregate their backbone with other task group leaders, following a simplified version of the proposed FedHCA² framework [11]. Eventually, the newly aggregated backbones are distributed back by the leaders to their group members. The primary motivation of this approach was to reduce conflicting and dominating gradients, which are especially likely if clients perform different tasks. The implementation of this proposed framework was described in Chapter 5. There, thesis showed how the aggregation scheme was implemented. It also focused on providing insights into challenges and limitations of the current implementation, especially with regards to decentralization.

In Chapter 6, the thesis conducted various experiments on class label heterogeneity and task heterogeneity setups. It examined the impact of aggregation round frequency on the framework and found, that with more frequent aggregations the model performance increased. It also investigated different degrees of layer privatization. Here it discovered that keeping most layers non-private in the class label heterogeneity setup was beneficial. Contrary to this, increasing the privatized layer count slightly increased the performance of task heterogeneity setups. Eventually, the thesis compared the proposed framework with other aggregation schemes. The results showed that proposed framework outperformed the compared aggregation schemes on class label heterogeneity and task heterogeneity.

7.2 Limitations and Potential Future Work

The current implementation of the proposed framework, although outperforming the compared schemes, has some limitations that present opportunities for future research.

The most evident limitation is that the implementation was not conducted in a fully decentralized environment. Instead, it was executed on a single machine, focusing on evaluating the proposed framework. However, the frameworks design allows for a decentralized implementation. Therefore, a potential next step would be to implement the framework in a properly decentralized manner. This would include the analysis of stragglers, faults and malicious clients. While doing so, future work should also incorporate a rigorous approach to analyze the occurrence and mitigation of NaN values.

The framework itself can also be improved further. For example, the current framework requires predefined task group definitions. Future approaches could consider utilizing methods such as cosine similarity or L2 distance to dynamically create groups [97]. Also, since the experiments have shown that layer privatization impacts framework performance, one could dynamically adjust the number of privatized layers based on task similarity to further refine clients.

Another limitation arises from the aggregation schemes used for comparison. While it is valid to compare the framework against pure local training and one of its main inspirations, FedPer [20], there is room for additional comparisons. In particular, a comparison with its second major inspiration, FedHCA² [11], would be beneficial. However, the thesis assumes

that the proposed framework may not achieve the same performance as FedHCA², as the latter incorporates more sophisticated methods. Additionally, such a comparison would be somewhat flawed, given that FedHCA² relies on a central server, whereas the proposed framework does not. Regardless, given the initial success of the proposed framework further comparisons are warranted. This is particularly relevant, since some experiments showed only minor improvements. Replicating these experiments under different settings and on additional datasets and tasks would provide a more precise and comprehensive understanding.

Bibliography

- [1] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, “Advances and open problems in federated learning”, *Foundations and trends® in machine learning*, vol. 14, no. 1–2, pp. 1–210, 2021.
- [2] B. Pfitzner, N. Steckhan, and B. Arnrich, “Federated learning in a medical context: A systematic literature review”, *ACM Transactions on Internet Technology (TOIT)*, vol. 21, no. 2, pp. 1–31, 2021.
- [3] N. Rieke, J. Hancox, W. Li, *et al.*, “The future of digital health with federated learning”, *NPJ digital medicine*, vol. 3, no. 1, pp. 1–7, 2020.
- [4] M. J. Sheller, B. Edwards, G. A. Reina, *et al.*, “Federated learning in medicine: Facilitating multi-institutional collaborations without sharing patient data”, *Scientific reports*, vol. 10, no. 1, p. 12 598, 2020.
- [5] A. Chowdhury, H. Kassem, N. Padoy, R. Umeton, and A. Karargyris, “A review of medical federated learning: Applications in oncology and cancer research”, in *International MICCAI Brainlesion Workshop*, Springer, 2021, pp. 3–24.
- [6] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, *et al.*, “Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges”, *IEEE Communications Surveys & Tutorials*, 2023.
- [7] R. Caruana, “Multitask learning”, *Machine learning*, vol. 28, pp. 41–75, 1997.
- [8] S. Ruder, “An overview of multi-task learning in deep neural networks”, *arXiv preprint arXiv:1706.05098*, 2017.
- [9] Y. Zhang and Q. Yang, “A survey on multi-task learning”, *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 12, pp. 5586–5609, 2021.
- [10] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, “Federated multi-task learning”, *Advances in neural information processing systems*, vol. 30, 2017.
- [11] Y. Lu, S. Huang, Y. Yang, S. Sirejiding, Y. Ding, and H. Lu, “Fedhca2: Towards hetero-client federated multi-task learning”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 5599–5609.
- [12] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, “Federated learning with non-iid data”, *arXiv preprint arXiv:1806.00582*, 2018.
- [13] H. Zhu, J. Xu, S. Liu, and Y. Jin, *Federated learning on non-iid data: A survey*, 2021. arXiv: 2106.06843 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2106.06843>.

- [14] Y. Chen, T. Zhang, X. Jiang, Q. Chen, C. Gao, and W. Huang, “Fedbone: Towards large-scale federated multi-task learning”, *arXiv preprint arXiv:2306.17465*, 2023.
- [15] X. Yin, Y. Zhu, and J. Hu, “A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions”, *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–36, 2021.
- [16] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data”, in *Artificial intelligence and statistics*, PMLR, 2017, pp. 1273–1282.
- [17] W. Huang, M. Ye, and B. Du, “Learn from others and be yourself in heterogeneous federated learning”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 10 143–10 153.
- [18] J. Yu, Y. Dai, X. Liu, *et al.*, “Unleashing the power of multi-task learning: A comprehensive survey spanning traditional, deep, and pretrained foundation model eras”, *arXiv preprint arXiv:2404.18961*, 2024.
- [19] A. Muhamed, M. Mankikar, and V. Smith, “Fed up with complexity: Simplifying many-task federated learning with ntkfedavg”, in *Privacy Regulation and Protection in Machine Learning*.
- [20] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, “Federated learning with personalization layers”, *arXiv preprint arXiv:1912.00818*, 2019.
- [21] A. Krizhevsky, G. Hinton, *et al.*, “Learning multiple layers of features from tiny images”, 2009.
- [22] Z. Liu, P. Luo, X. Wang, and X. Tang, “Deep learning face attributes in the wild”, in *Proceedings of International Conference on Computer Vision (ICCV)*, Dec. 2015.
- [23] R. Aoki, F. Tung, and G. L. Oliveira, “Heterogeneous multi-task learning with expert diversity”, *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 19, no. 6, pp. 3093–3102, 2022. DOI: 10.1109/TCBB.2022.3175456.
- [24] D. Kollias, V. Sharmanska, and S. Zafeiriou, *Distribution matching for heterogeneous multi-task learning: A large-scale face study*, 2021. arXiv: 2105 . 03790 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/2105.03790>.
- [25] M. Crawshaw, “Multi-task learning with deep neural networks: A survey”, *arXiv preprint arXiv:2009.09796*, 2020.
- [26] S. Vandenhende, S. Georgoulis, W. Van Gansbeke, M. Proesmans, D. Dai, and L. Van Gool, “Multi-task learning for dense prediction tasks: A survey”, *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, pp. 3614–3633, 2021.
- [27] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, *Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks*, 2018. arXiv: 1711.02257 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/1711.02257>.
- [28] S. Liu, E. Johns, and A. J. Davison, *End-to-end multi-task learning with attention*, 2019. arXiv: 1803.10704 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/1803.10704>.

- [29] A. Kendall, Y. Gal, and R. Cipolla, “Multi-task learning using uncertainty to weigh losses for scene geometry and semantics”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7482–7491.
- [30] D. Lopez-Paz and M. Ranzato, “Gradient episodic memory for continual learning”, *Advances in neural information processing systems*, vol. 30, 2017.
- [31] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, “Gradient surgery for multi-task learning”, *Advances in Neural Information Processing Systems*, vol. 33, pp. 5824–5836, 2020.
- [32] Z. Chen, J. Ngiam, Y. Huang, *et al.*, “Just pick a sign: Optimizing deep multitask models with gradient sign dropout”, *Advances in Neural Information Processing Systems*, vol. 33, pp. 2039–2050, 2020.
- [33] B. Liu, X. Liu, X. Jin, P. Stone, and Q. Liu, “Conflict-averse gradient descent for multi-task learning”, *Advances in Neural Information Processing Systems*, vol. 34, pp. 18 878–18 890, 2021.
- [34] G. Shi, Q. Li, W. Zhang, J. Chen, and X.-M. Wu, “Recon: Reducing conflicting gradients from the root for multi-task learning”, *arXiv preprint arXiv:2302.11289*, 2023.
- [35] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, “Facial landmark detection by deep multi-task learning”, in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part VI 13*, Springer, 2014, pp. 94–108.
- [36] J. Dai, K. He, and J. Sun, “Instance-aware semantic segmentation via multi-task network cascades”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3150–3158.
- [37] J. Ma, Z. Zhao, X. Yi, J. Chen, L. Hong, and E. H. Chi, “Modeling task relationships in multi-task learning with multi-gate mixture-of-experts”, in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, 2018, pp. 1930–1939.
- [38] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, “Cross-stitch networks for multi-task learning”, in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 3994–4003.
- [39] S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard, “Latent multi-task architecture learning”, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 4822–4829.
- [40] Y. Gao, J. Ma, M. Zhao, W. Liu, and A. L. Yuille, “Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction”, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 3205–3214.
- [41] K. Hashimoto, C. Xiong, Y. Tsuruoka, and R. Socher, “A joint many-task model: Growing a neural network for multiple nlp tasks”, *arXiv preprint arXiv:1611.01587*, 2016.

- [42] Y. Gong, X. Luo, Y. Zhu, *et al.*, “Deep cascade multi-task learning for slot filling in online shopping assistant”, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 6465–6472.
- [43] V. Sanh, T. Wolf, and S. Ruder, “A hierarchical multi-task approach for learning embeddings from semantic tasks”, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, 2019, pp. 6949–6956.
- [44] W.-H. Li and H. Bilen, “Knowledge distillation for multi-task learning”, in *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part VI 16*, Springer, 2020, pp. 163–176.
- [45] G. M. Jacob, V. Agarwal, and B. Stenger, “Online knowledge distillation for multi-task learning”, in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023, pp. 2359–2368.
- [46] G. Ghiasi, B. Zoph, E. D. Cubuk, Q. V. Le, and T.-Y. Lin, “Multi-task self-training for learning general representations”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 8856–8865.
- [47] H. M. Alonso and B. Plank, “When is multitask learning effective? semantic sequence prediction under varying data conditions”, *arXiv preprint arXiv:1612.02251*, 2016.
- [48] J. Bingel and A. Søgaard, “Identifying beneficial task relations for multi-task learning in deep neural networks”, *arXiv preprint arXiv:1702.08303*, 2017.
- [49] C. Doersch and A. Zisserman, “Multi-task self-supervised visual learning”, in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2051–2060.
- [50] G. Strezoski, N. van Noord, and M. Worring, “Learning task relatedness in multi-task learning for images in context”, in *Proceedings of the 2019 on international conference on multimedia retrieval*, 2019, pp. 78–86.
- [51] T. Standley, A. Zamir, D. Chen, L. Guibas, J. Malik, and S. Savarese, “Which tasks should be learned together in multi-task learning?”, in *International conference on machine learning*, PMLR, 2020, pp. 9120–9132.
- [52] C. Widmer, J. Leiva, Y. Altun, and G. Rätsch, “Leveraging sequence classification by taxonomy-based multitask learning”, in *Research in Computational Molecular Biology: 14th Annual International Conference, RECOMB 2010, Lisbon, Portugal, April 25-28, 2010. Proceedings 14*, Springer, 2010, pp. 522–534.
- [53] Z.-M. Chen, X.-S. Wei, P. Wang, and Y. Guo, “Multi-label image recognition with graph convolutional networks”, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 5177–5186.
- [54] Q. Li, Z. Wen, Z. Wu, *et al.*, “A survey on federated learning systems: Vision, hype and reality for data privacy and protection”, *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3347–3366, 2021.
- [55] G. D. P. R. GDPR, “General data protection regulation”, *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC*, 2016.

- [56] O. Marfoq, G. Neglia, A. Bellet, L. Kameni, and R. Vidal, “Federated multi-task learning under a mixture of distributions”, *Advances in Neural Information Processing Systems*, vol. 34, pp. 15 434–15 447, 2021.
- [57] S. Ji, Y. Tan, T. Saravirta, *et al.*, “Emerging trends in federated learning: From model fusion to federated x learning”, *International Journal of Machine Learning and Cybernetics*, pp. 1–22, 2024.
- [58] L. Yuan, Z. Wang, L. Sun, P. S. Yu, and C. G. Brinton, *Decentralized federated learning: A survey and perspective*, 2024. arXiv: 2306.01603 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2306.01603>.
- [59] E. Hallaji, R. Razavi-Far, M. Saif, B. Wang, and Q. Yang, “Decentralized federated learning: A survey on security and privacy”, *IEEE Transactions on Big Data*, vol. 10, no. 2, pp. 194–213, Apr. 2024, ISSN: 2372-2096. DOI: 10.1109/tbdata.2024.3362191. [Online]. Available: <http://dx.doi.org/10.1109/TBDATA.2024.3362191>.
- [60] B. Liu, N. Lv, Y. Guo, and Y. Li, “Recent advances on federated learning: A systematic survey”, *Neurocomputing*, p. 128 019, 2024.
- [61] E. Gabrielli, G. Pica, and G. Tolomei, *A survey on decentralized federated learning*, 2023. arXiv: 2308.04604 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2308.04604>.
- [62] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, “Federated optimization in heterogeneous networks”, *Proceedings of Machine learning and systems*, vol. 2, pp. 429–450, 2020.
- [63] C. T. Dinh, T. T. Vu, N. H. Tran, M. N. Dao, and H. Zhang, “Fedu: A unified framework for federated multi-task learning with laplacian regularization”, *arXiv preprint arXiv:2102.07148*, vol. 400, 2021.
- [64] C. Chen, T. Liao, X. Deng, Z. Wu, S. Huang, and Z. Zheng, “Advances in robust federated learning: Heterogeneity considerations”, *arXiv preprint arXiv:2405.09839*, 2024.
- [65] K. W. Ng, G.-L. Tian, and M.-L. Tang, “Dirichlet and related distributions: Theory, methods and applications”, 2011.
- [66] J. Mills, J. Hu, and G. Min, “Multi-task federated learning for personalised deep neural networks in edge computing”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 3, pp. 630–641, 2021.
- [67] R. Li, F. Ma, W. Jiang, and J. Gao, “Online federated multitask learning”, in *2019 IEEE International Conference on Big Data (Big Data)*, IEEE, 2019, pp. 215–220.
- [68] C. He, E. Ceyani, K. Balasubramanian, M. Annavaram, and S. Avestimehr, “Spreadgnn: Decentralized multi-task federated learning for graph neural networks on molecular data”, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 36, 2022, pp. 6865–6873.
- [69] R. Li, W. Liang, C. Peng, X. An, Z. Zhao, and H. Zhang, “Network ai management & orchestration: A federated multi-task learning case”, in *2021 IEEE Globecom Workshops (GC Wkshps)*, IEEE, 2021, pp. 1–6.

- [70] Y. Chen, Y. Ning, Z. Chai, and H. Rangwala, “Federated multi-task learning with hierarchical attention for sensor data analytics”, in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–8.
- [71] L. Corinzia, A. Beuret, and J. M. Buhmann, “Variational federated multi-task learning”, *arXiv preprint arXiv:1906.06268*, 2019.
- [72] L. Yao, D. Gao, Z. Wang, *et al.*, “A benchmark for federated hetero-task learning”, *arXiv preprint arXiv:2206.03436*, 2022.
- [73] A. Z. Tan, H. Yu, L. Cui, and Q. Yang, “Towards personalized federated learning”, *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 12, pp. 9587–9603, 2023. DOI: 10.1109/TNNLS.2022.3160699.
- [74] A. A. Khan, A. F. Khan, H. Ali, and A. Anwar, *Personalized federated learning techniques: Empirical analysis*, 2024. arXiv: 2409.06805 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2409.06805>.
- [75] W. Zhuang, Y. Wen, L. Lyu, and S. Zhang, “Mas: Towards resource-efficient federated multiple-task learning”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2023, pp. 23 414–23 424.
- [76] X. Cao, Z. Li, G. Sun, H. Yu, and M. Guizani, “Cross-silo heterogeneous model federated multitask learning”, *Knowledge-Based Systems*, vol. 265, p. 110 347, 2023.
- [77] M. Mortaheb, C. Vahapoglu, and S. Ulukus, “Fedgradnorm: Personalized federated gradient-normalized multi-task learning”, in *2022 IEEE 23rd International Workshop on Signal Processing Advances in Wireless Communication (SPAWC)*, IEEE, 2022, pp. 1–5.
- [78] Google Scholar, *Google scholar*, 2024. [Online]. Available: <https://scholar.google.com/>.
- [79] IEEE Xplore, *Ieee xplore digital library*, 2024. [Online]. Available: <https://ieeexplore.ieee.org/>.
- [80] arXiv, *Arxiv e-print archive*, 2024. [Online]. Available: <https://arxiv.org/>.
- [81] S. Caldas, V. Smith, and A. Talwalkar, “Federated kernelized multi-task learning”, in *Proc. SysML Conf*, 2018, pp. 1–3.
- [82] Z.-A. Huang, Y. Hu, R. Liu, *et al.*, “Federated multi-task learning for joint diagnosis of multiple mental disorders on mri scans”, *IEEE Transactions on Biomedical Engineering*, vol. 70, no. 4, pp. 1137–1149, 2022.
- [83] T. Shen, J. Zhang, X. Jia, *et al.*, “Federated mutual learning”, *arXiv preprint arXiv:2006.16765*, 2020.
- [84] T. Li, S. Hu, A. Beirami, and V. Smith, “Ditto: Fair and robust federated learning through personalization”, in *International conference on machine learning*, PMLR, 2021, pp. 6357–6368.
- [85] S. Hu, Z. S. Wu, and V. Smith, “Private multi-task learning: Formulation and applications to federated learning”, *arXiv preprint arXiv:2108.12978*, 2021.

- [86] H. Ma, X. Yuan, Z. Ding, D. Fan, and J. Fang, “Over-the-air federated multi-task learning via model sparsification, random compression, and turbo compressed sensing”, *IEEE Transactions on Wireless Communications*, vol. 22, no. 7, pp. 4974–4988, 2022.
- [87] H. Wu, C. Chen, and L. Wang, “A theoretical perspective on differentially private federated multi-task learning”, *arXiv preprint arXiv:2011.07179*, 2020.
- [88] L. Collins, H. Hassani, A. Mokhtari, and S. Shakkottai, “Exploiting shared representations for personalized federated learning”, in *International conference on machine learning*, PMLR, 2021, pp. 2089–2099.
- [89] C. Vahapoglu, M. Mortaheb, and S. Ulukus, “Hierarchical over-the-air fedgrad-norm”, in *2022 56th Asilomar Conference on Signals, Systems, and Computers*, IEEE, 2022, pp. 754–758.
- [90] Q. Shen, H. Feng, R. Song, S. Teso, F. Giunchiglia, H. Xu, *et al.*, “Federated multi-task attention for cross-individual human activity recognition”, in *IJCAI*, IJCAI, 2022, pp. 3423–3429.
- [91] X. Yi, R. Li, C. Peng, J. Wu, and Z. Zhao, “Hfedmtl: Hierarchical federated multi-task learning”, in *2022 IEEE 33rd Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, IEEE, 2022, pp. 1–6.
- [92] X. Yi, R. Li, C. Peng, F. Wang, J. Wu, and Z. Zhao, “Rhfedmtl: Resource-aware hierarchical federated multi-task learning”, *IEEE Internet of Things Journal*, 2024.
- [93] Y. Huang, L. Chu, Z. Zhou, *et al.*, “Personalized cross-silo federated learning on non-iid data”, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 7865–7873.
- [94] F. Sattler, K.-R. Müller, and W. Samek, “Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints”, *IEEE transactions on neural networks and learning systems*, vol. 32, no. 8, pp. 3710–3722, 2020.
- [95] A. M. Albaseer, M. Abdallah, A. Al-Fuqaha, A. M. Seid, A. Erbad, and O. A. Dobre, “Fair selection of edge nodes to participate in clustered federated multitask learning”, *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1502–1516, 2023.
- [96] H.-Y. Hsu, K. H. Keoy, J.-R. Chen, H.-C. Chao, and C.-F. Lai, “Personalized federated learning algorithm with adaptive clustering for non-iid iot data incorporating multi-task learning and neural network model characteristics”, *Sensors*, vol. 23, no. 22, p. 9016, 2023.
- [97] Y. Xiao, J. Shu, X. Jia, and H. Huang, “Clustered federated multi-task learning with non-iid data”, in *2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS)*, IEEE, 2021, pp. 50–57.
- [98] W. Zhuang, Y. Wen, and S. Zhang, “Smart multi-tenant federated learning”, *arXiv preprint arXiv:2207.04202*, 2022.
- [99] C. Ye, H. Zheng, Z. Hu, and M. Zheng, “Pfedsa: Personalized federated multi-task learning via similarity awareness”, in *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, 2023, pp. 480–488.

- [100] H. Xing, Z. Xiao, R. Qu, Z. Zhu, and B. Zhao, “An efficient federated distillation learning system for multitask time series classification”, *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–12, 2022.
- [101] Z. Wu, S. Sun, Y. Wang, *et al.*, “Fedict: Federated multi-task distillation for multi-access edge computing”, *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 6, pp. 1107–1121, 2023.
- [102] H. Wang, Y. Li, W. Xu, R. Li, Y. Zhan, and Z. Zeng, “Dafkd: Domain-aware federated knowledge distillation”, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 20 412–20 421.
- [103] P. Vanhaesebrouck, A. Bellet, and M. Tommasi, “Decentralized collaborative learning of personalized models over networks”, *arXiv preprint arXiv:1610.05202*, 2016.
- [104] V. Zantedeschi, A. Bellet, and M. Tommasi, “Fully decentralized joint learning of personalized models and collaboration graphs”, in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 864–874.
- [105] Y. SarcheshmehPour, Y. Tian, L. Zhang, and A. Jung, “Networked federated multi-task learning”, *Authorea Preprints*, 2023.
- [106] A. Bouchra Pilet, D. Frey, and F. Tairani, “Simple, efficient and convenient decentralized multi-task learning for neural networks”, in *Advances in Intelligent Data Analysis XIX: 19th International Symposium on Intelligent Data Analysis, IDA 2021, Porto, Portugal, April 26–28, 2021, Proceedings 19*, Springer, 2021, pp. 37–49.
- [107] J. Beilharz, B. Pfitzner, R. Schmid, P. Geppert, B. Arnrich, and A. Polze, “Implicit model specialization through dag-based decentralized federated learning”, in *Proceedings of the 22nd International Middleware Conference*, 2021, pp. 310–322.
- [108] R. Liu, J. Yang, and C. Shen, “Exploiting feature heterogeneity for improved generalization in federated multi-task learning”, in *2023 IEEE International Symposium on Information Theory (ISIT)*, IEEE, 2023, pp. 180–185.
- [109] Y. Jia, X. Zhang, A. Beheshti, and W. Dou, “Fedlps: Heterogeneous federated learning for multiple tasks with local parameter sharing”, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 12 848–12 856.
- [110] C. Zhao, Z. Gao, Q. Wang, K. Xiao, and Z. Mo, “Afl: An adaptively federated multitask learning for model sharing in industrial iot”, *IEEE Internet of Things Journal*, vol. 9, no. 18, pp. 17 080–17 088, 2022. DOI: 10.1109/JIOT.2021.3125989.
- [111] R. Cai, X. Chen, S. Liu, *et al.*, “Many-task federated learning: A new problem setting and a simple baseline”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2023, pp. 5037–5045.
- [112] W. Zhuang, Y. Wen, L. Lyu, and S. Zhang, “Mas: Towards resource-efficient federated multiple-task learning”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2023, pp. 23 414–23 424.
- [113] J. Chen and A. Zhang, “Fedmsplit: Correlation-adaptive federated multi-task learning across multimodal split networks”, in *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, 2022, pp. 87–96.

- [114] F. Ahmed, “Federated multi-task learning: An overview and quantitative evaluation”, *Available at SSRN 4627317*, 2023.
- [115] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.
- [116] D. Bertsekas, A. Nedic, and A. Ozdaglar, *Convex analysis and optimization*. Athena Scientific, 2003, vol. 1.
- [117] V. Smith, S. Forte, C. Ma, M. Takáč, M. I. Jordan, and M. Jaggi, “Cocoa: A general framework for communication-efficient distributed optimization”, *Journal of Machine Learning Research*, vol. 18, no. 230, pp. 1–49, 2018.
- [118] A. Şahin and R. Yang, “A survey on over-the-air computation”, *IEEE Communications Surveys & Tutorials*, vol. 25, no. 3, pp. 1877–1908, 2023. DOI: 10.1109/COMST.2023.3264649.
- [119] S. Ioffe, “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, *arXiv preprint arXiv:1502.03167*, 2015.
- [120] A. Vaswani, “Attention is all you need”, *Advances in Neural Information Processing Systems*, 2017.
- [121] K. Cho, “Learning phrase representations using rnn encoder-decoder for statistical machine translation”, *arXiv preprint arXiv:1406.1078*, 2014.
- [122] H. Xie, M. Xia, P. Wu, S. Wang, and K. Huang, “Decentralized federated learning with asynchronous parameter sharing for large-scale iot networks”, *IEEE Internet of Things Journal*, 2024.
- [123] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, “Pruning and quantization for deep neural network acceleration: A survey”, *Neurocomputing*, vol. 461, pp. 370–403, 2021.
- [124] H. Robbins, “Some aspects of the sequential design of experiments”, 1952.
- [125] V. Kuleshov and D. Precup, “Algorithms for multi-armed bandit problems”, *arXiv preprint arXiv:1402.6028*, 2014.
- [126] J. Dai, H. Qi, Y. Xiong, *et al.*, *Deformable convolutional networks*, 2017. arXiv: 1703.06211 [cs.CV]. [Online]. Available: <https://arxiv.org/abs/1703.06211>.
- [127] G. Tang, M. Müller, A. Rios, and R. Sennrich, *Why self-attention? a targeted evaluation of neural machine translation architectures*, 2018. arXiv: 1808.08946 [cs.CL]. [Online]. Available: <https://arxiv.org/abs/1808.08946>.
- [128] J. Park, D.-J. Han, M. Choi, and J. Moon, “Sageflow: Robust federated learning against both stragglers and adversaries”, *Advances in neural information processing systems*, vol. 34, pp. 840–851, 2021.
- [129] Z. Zhang, X. Cao, J. Jia, and N. Z. Gong, “Fldetector: Defending federated learning against model poisoning attacks via detecting malicious clients”, in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 2545–2555.
- [130] X. Cao, J. Jia, and N. Z. Gong, “Provably secure federated learning against malicious clients”, in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, 2021, pp. 6885–6893.

- [131] YAML, *Yaml: Yaml ain't markup language*, Accessed: 2024-10-01, 2024. [Online]. Available: <https://yaml.org/>.
- [132] PyTorch, *Pytorch data documentation*, Accessed: 2024-10-01, 2024. [Online]. Available: <https://pytorch.org>.
- [133] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [134] X. Ying, "An overview of overfitting and its solutions", in *Journal of physics: Conference series*, IOP Publishing, vol. 1168, 2019, p. 022 022.
- [135] Torchvision, *Pytorch*, Accessed: 2024-10-01, 2024. [Online]. Available: <https://pytorch.org/>.
- [136] F. Stulp and O. Sigaud, "Many regression algorithms, one unified model: A review", *Neural Networks*, vol. 69, pp. 60–79, 2015.
- [137] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", *Nature Methods*, vol. 17, pp. 261–272, 2020. DOI: 10.1038/s41592-019-0686-2.
- [138] I. Anaconda, *Conda package management system*, Accessed: 2024-10-10, 2023. [Online]. Available: <https://anaconda.org/anaconda/conda>.
- [139] P. Contributors, *Reproducibility — pytorch 2.4 documentation*, Accessed: 2024-10-05, PyTorch Foundation, 2024. [Online]. Available: <https://pytorch.org/docs/stable/notes/randomness.html>.
- [140] Lightning AI, *Torchmetrics documentation*, Accessed: 2024-10-08. [Online]. Available: <https://lightning.ai/docs/torchmetrics/stable/>.
- [141] A. Krizhevsky, *Learning multiple layers of features from tiny images, cifar-10 and cifar-100 dataset*, Accessed: 2024-10-01, 2009. [Online]. Available: <https://www.cs.toronto.edu/~kriz/cifar.html>.
- [142] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, "Microsoft coco: Common objects in context", in *Computer Vision—ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, Springer, 2014, pp. 740–755.
- [143] P. K. Nathan Silberman Derek Hoiem and R. Fergus, "Indoor segmentation and support inference from rgb-d images", in *ECCV*, 2012.
- [144] A. Woubie, E. Solomon, and J. Attieh, "Maintaining privacy in face recognition using federated learning method", *IEEE Access*, 2024.
- [145] M. Rasouli, T. Sun, and R. Rajagopal, "Fedgan: Federated generative adversarial networks for distributed data", *arXiv preprint arXiv:2006.07228*, 2020.
- [146] PyTorch Lightning Team, *Pytorch lightning - cifar10 baseline*, Accessed: 2024-10-08, 2023. [Online]. Available: https://lightning.ai/docs/pytorch/stable/notebooks/lightning_examples/cifar10-baseline.html.
- [147] E. Budu and M. Simi, *Common causes of nans during training*, Accessed: 2024-10-06, 2024. [Online]. Available: <https://www.baeldung.com/cs/ml-training-nan-errors-fix>.

- [148] A. Gomez, *Common causes of nans during training of neural networks*, Accessed: 2024-10-06, 2015. [Online]. Available: <https://stackoverflow.com/questions/33962226/common-causes-of-nans-during-training-of-neural-networks>.
- [149] P. Contributors, *Torch.nn.utils.clip_grad_norm_ — pytorch documentation*, Accessed: 2024-10-06, 2024. [Online]. Available: https://pytorch.org/docs/stable/generated/torch.nn.utils.clip_grad_norm_.html.
- [150] Y. Wang, Q. Shi, and T.-H. Chang, *Why batch normalization damage federated learning on non-iid data?*, 2023. arXiv: 2301.02982 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2301.02982>.
- [151] R. Guerraoui, R. Pinot, G. Rizk, J. Stephan, and F. Taiani, *Overcoming the challenges of batch normalization in federated learning*, 2024. arXiv: 2405.14670 [cs.LG]. [Online]. Available: <https://arxiv.org/abs/2405.14670>.
- [152] I. H. Sarker, “Machine learning: Algorithms, real-world applications and research directions”, *SN Computer Science*, vol. 2, no. 3, p. 160, 2021, ISSN: 2661-8907. DOI: 10.1007/s42979-021-00592-x. [Online]. Available: <https://doi.org/10.1007/s42979-021-00592-x>.

Abbreviations

FL	Federated Learning
ML	Machine Learning
DFL	Decentralized Federated Learning
MTL	Multi-Task Learning
FMTL	Federated Multi-Task Learning
PFL	Personalized Federated Learning
CFMTL	Centralized Federated Multi-Task Learning
DFMTL	Decentralized Federated Multi-Task Learning
HCFMTL	Heterogeneous Centralized Federated Multi-Task Learning
HDFMTL	Heterogeneous Decentralized Federated Multi-Task Learning
Non-IID	Non-Independently and Non-Identically Distributed

Glossary

Federated Learning (FL) FL allows clients to collaboratively train one (sometimes multiple) models without sharing their data [1].

Machine Learning (ML) ML involves developing predictive algorithms that enable systems to automatically learn and improve from experience without the need for explicit programming [152].

Decentralized Federated Learning (DFL) DFL enables clients to directly exchange model parameters without the need for a central authority [6].

Multi-Task Learning (MTL) The general concept of Multi-Task Learning (MTL) is to train on different yet related tasks concurrently, utilizing advanced mechanisms to enable tasks to mutually influence and benefit from each other [7].

Federated Multi-Task Learning (FMTL) FMTL, popularized by [10], usually treats each client in a federated environment as a task, making it a subfield of Personalized Federated Learning [11].

Personalized Federated Learning (PFL) Is a FL technique that aims to train a personalized model for each client, instead of one global model [74].

Centralized Federated Multi-Task Learning (CFMTL) CFMTL is defined in section 2.3.3 as follows: In Centralized Federated Multi-Task Learning (CFMTL) a centralized node is responsible for the coordination and/or aggregation of network nodes and their respective models. Each node in the network learns a model based on its own data, while taking its relationship with other nodes into account. While heterogeneous data between clients can exist, the task objectives are the same.

Decentralized Federated Multi-Task Learning (DFMTL) DFMTL is defined in section 2.3.3 as follows: In Decentralized Federated Multi-Task Learning (DFMTL) nodes communicate directly with each other, without relying on a central server to conduct coordination and/or aggregation. Each node in the network learns a model based on its own data, while taking its relationship with other nodes into account. While heterogeneous data between clients can exist, the task objectives are the same.

Heterogeneous Centralized Federated Multi-Task Learning (HCFMTL) HCFMTL is defined in section 2.3.3 as follows: In Heterogeneous Centralized Federated Multi-Task Learning (HCFMTL) a centralized node is responsible for the coordination and/or aggregation of network nodes and their respective models. Each node in the network

learns a model based on its own data, while taking its relationship with other nodes into account. The task objectives among clients differ.

Heterogeneous Decentralized Federated Multi-Task Learning (HDFMTL) HDFMTL is defined in section 2.3.3 as follows: In Heterogeneous Decentralized Federated Multi-Task Learning (HDFMTL) nodes communicate directly with each other, without relying on a central server to conduct coordination and/or aggregation. Each node in the network learns a model based on its own data, while taking its relationship with other nodes into account. The task objectives among clients differ.

Non-Independently and Non-Identically Distributed (Non-IID) Data among clients is often non-independently and non-identically distributed (Non-IID). Examples of non-IID data in FL include attribute skew, label skew, and temporal skew [13].

List of Figures

2.1	Dominant and Conflicting Gradients. Illustration by [18]	7
2.2	Depiction of Hard and Soft Parameter Sharing Inspired by [8]	9
2.3	CFL, DFL and Semi-DFL as Depicted by [6]	13
4.1	Proposed Framework	37
5.1	Overview of the Code Repository	44
5.2	Residual Block as Depicted by [133]	46
5.3	Implemented Aggregation Schemes	48
6.1	Cifar-10 Dataset as Visualized by [141]	58
6.2	An Image of the CelebA Dataset with Labels as Depicted by [22]	60
6.3	Average Client Test Metrics for Animal Classification with Different Epochs per Aggregation Round	62
6.4	Average Client Test Metrics for Object Classification with Different Epochs per Aggregation Round	62
6.5	Average Client Validation Metrics for Animal Classification with Different Epochs per Aggregation Round	63
6.6	Average Client Validation Metrics for Object Classification with Different Epochs per Aggregation Round	63
6.7	Average Client Test Metrics for Face Attribute Classification with Different Epochs per Aggregation Round	65
6.8	Average Client Test Metrics for Face Landmark Detection with Different Epochs per Aggregation Round	65
6.9	Average Client Validation Metrics for Face Attribute Classification with Different Epochs per Aggregation Round	65

6.10	Average Client Test Metrics for Animal Classification with Different Degrees of Privatizations	66
6.11	Average Client Test Metrics for Object Classification with Different Degrees of Privatization	66
6.12	Average Client Validation Metrics for Animal Classification with Different Degrees of Privatization	67
6.13	Average Client Validation Metrics for Object Classification with Different Degrees of Privatization	67
6.14	Average Client Test Metrics for Face Attribute Classification with Different Degrees of Privatization	68
6.15	Average Client Test Metrics for Face Landmark Detection with Different Degrees of Privatization	68
6.16	Average Client Validation Metrics for Face Attribute Classification with Different Degrees of Privatization	69
6.17	Average Client Test Metrics for Animal Classification with Different Aggregation Schemes	70
6.18	Average Client Test Metrics for Object Classification with Different Aggregation Schemes	70
6.19	Average Client Validation Metrics for Animal Classification with Different Aggregation Schemes	72
6.20	Average Client Validation Metrics for Object Classification with Different Aggregation Schemes	72
6.21	Average Client Test Metrics for Face Attributed Classification with Different Aggregation Schemes	73
6.22	Average Client Test Metrics for Face Landmark Detection with Different Aggregation Schemes	73
6.23	Average Client Validation Metrics for Face Attributed Classification with Different Aggregation Schemes	74
B.1	Average Client Train and Validation Metrics for Animal Classification with Different Epochs per Aggregation Round	108
B.2	Average Client Train and Validation Metrics for Object Classification with Different Epochs per Aggregation Round	108
B.3	Average Client Train and Validation Metrics for Face Landmark Detection with Different Epochs per Aggregation Round	109

B.4	Average Client Train and Validation Metrics for Face Landmark Detection with Different Epochs per Aggregation Round	109
B.5	Average Client Train and Validation Metrics for Animal Classification with Different Degrees of Privatization	110
B.6	Average Client Train and Validation Metrics for Object Classification with Different Degrees of Privatization	110
B.7	Average Client Train and Validation Metrics for Face Attribute Classification with Different Degrees of Privatization	111
B.8	Average Client Train and Validation Metrics for Face Landmark Detection with Different Degrees of Privatization	111
B.9	Average Client Train and Validation Metrics for Animal Classification with Different Aggregation Schemes	112
B.10	Average Client Train and Validation Metrics for Object Classification with Different Aggregation Schemes	112
B.11	Average Client Train and Validation Metrics for Face Attributed Classification with Different Aggregation Schemes	113
B.12	Average Client Train and Validation Metrics for Face Landmark Detection with Different Aggregation Schemes	113

List of Tables

2.1	Architectural MTL Concepts, Based on [25] and [18]	8
2.2	Heterogeneous Clients as Identified by [64]	13
3.1	Database and Search Queries	20
3.2	Reviewed Research	20
3.3	Centralized Federated Multi-Task Learning	21
3.4	Decentralized Federated Multi-Task Learning	27
3.5	Heterogeneous Federated Multi-Task Learning	30
6.1	Experimental Evaluation of the Proposed Framework	56

Appendix A

Installation Guidelines

The installation is documented in the ReadMe file located on the GitHub repository. Please see Section 1.2.1 for the link.

Appendix B

Additional Plots

Chapter 6 only discussed a subset of the available plots. As such, this appendix lists them. Even more figures can be accessed by following the link provided in Section 1.2.1.

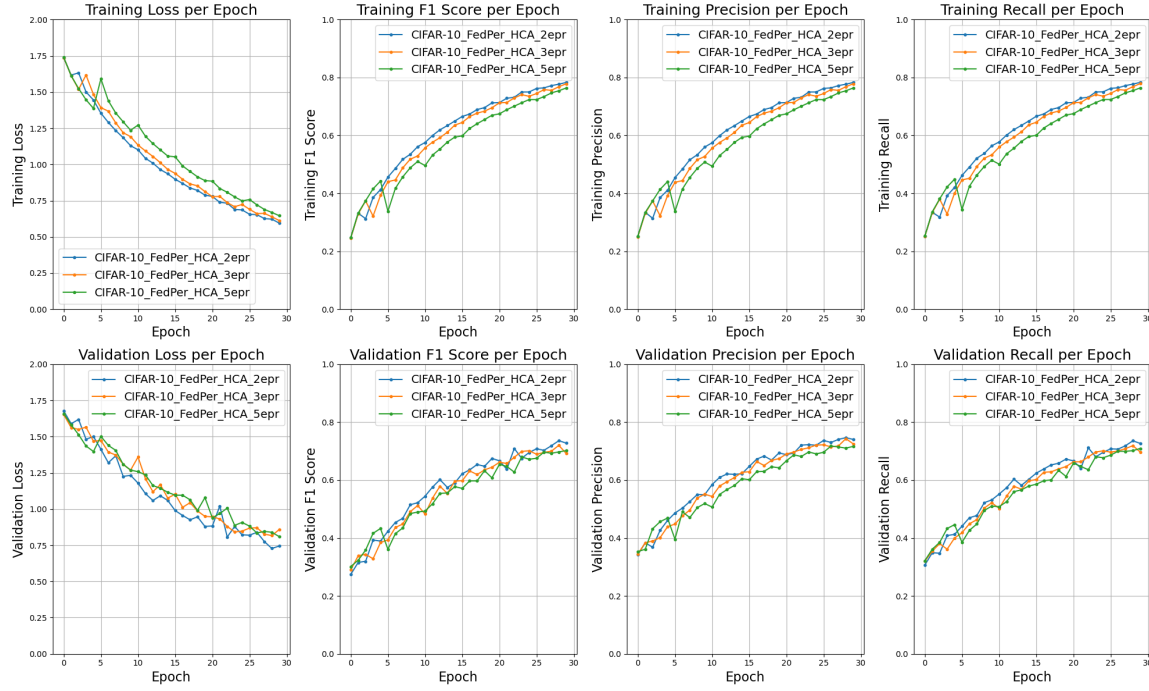


Figure B.1: Average Client Train and Validation Metrics for Animal Classification with Different Epochs per Aggregation Round

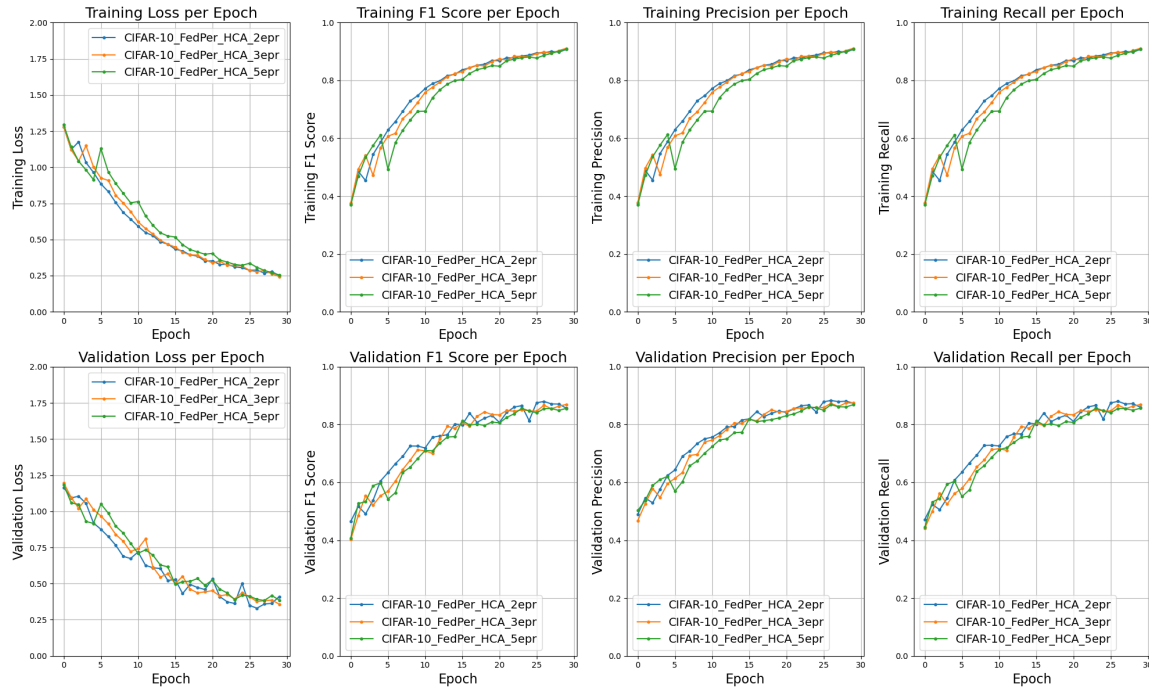


Figure B.2: Average Client Train and Validation Metrics for Object Classification with Different Epochs per Aggregation Round

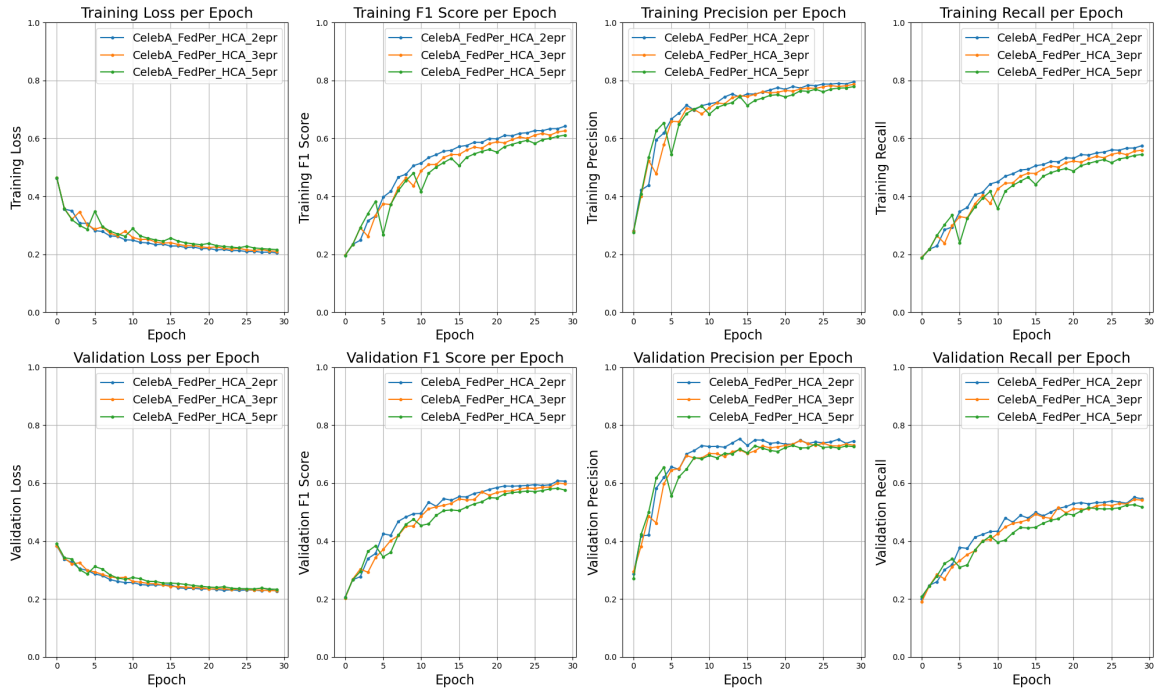


Figure B.3: Average Client Train and Validation Metrics for Face Landmark Detection with Different Epochs per Aggregation Round

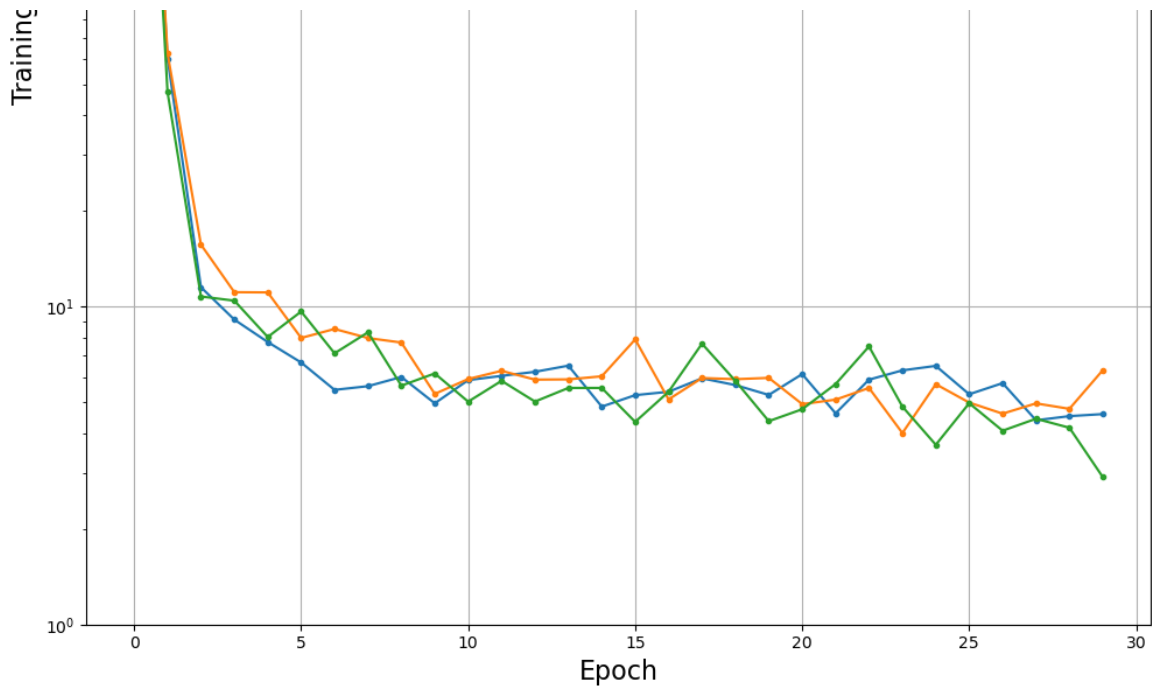


Figure B.4: Average Client Train and Validation Metrics for Face Landmark Detection with Different Epochs per Aggregation Round

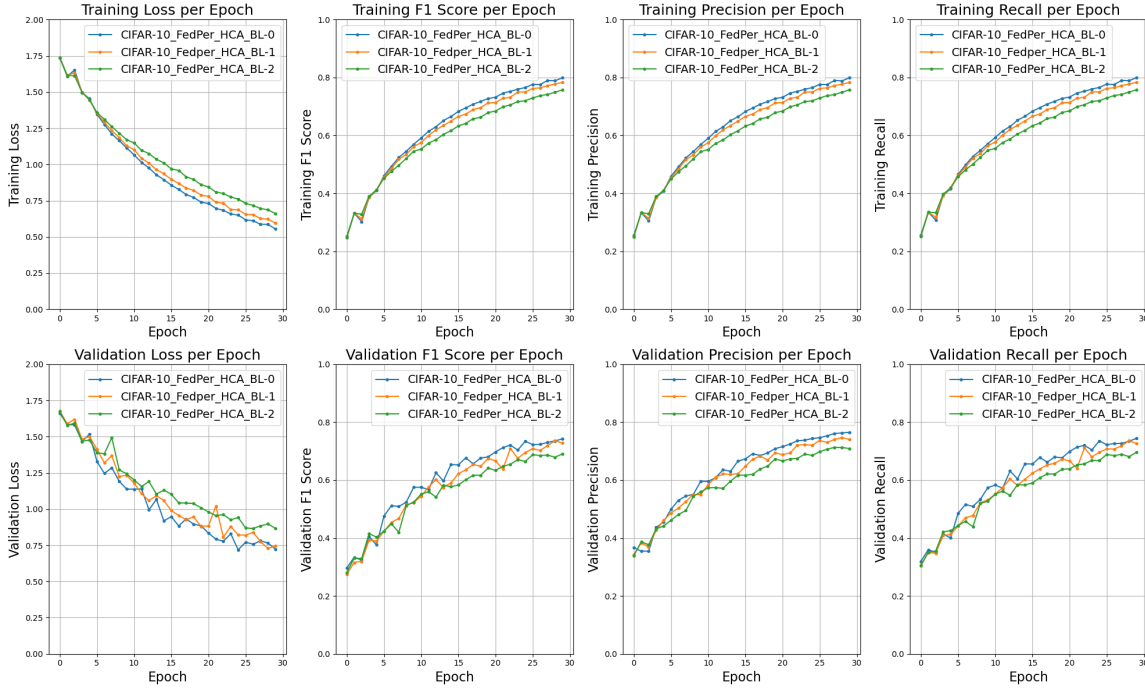


Figure B.5: Average Client Train and Validation Metrics for Animal Classification with Different Degrees of Privatization

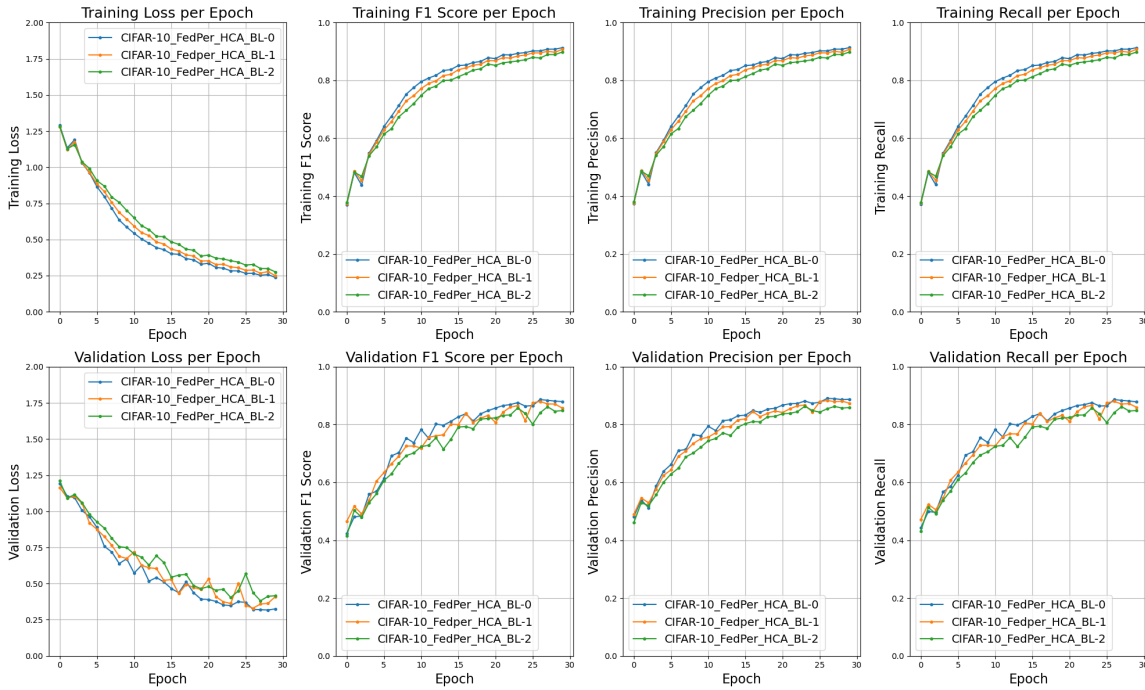


Figure B.6: Average Client Train and Validation Metrics for Object Classification with Different Degrees of Privatization

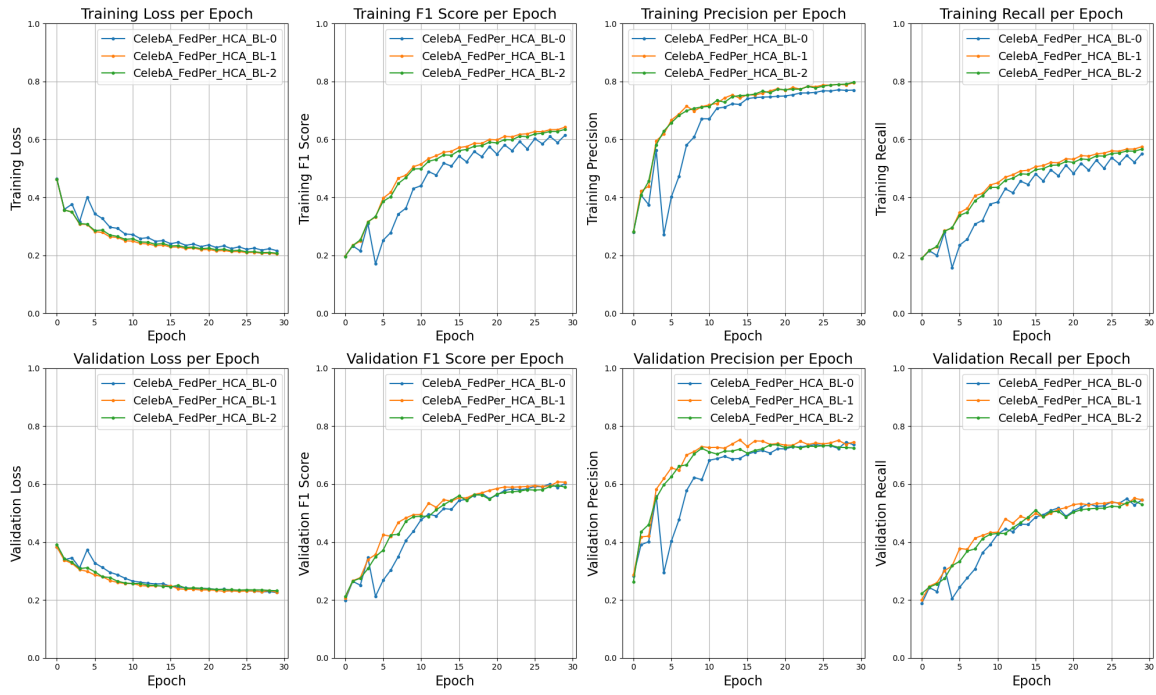


Figure B.7: Average Client Train and Validation Metrics for Face Attribute Classification with Different Degrees of Privatization

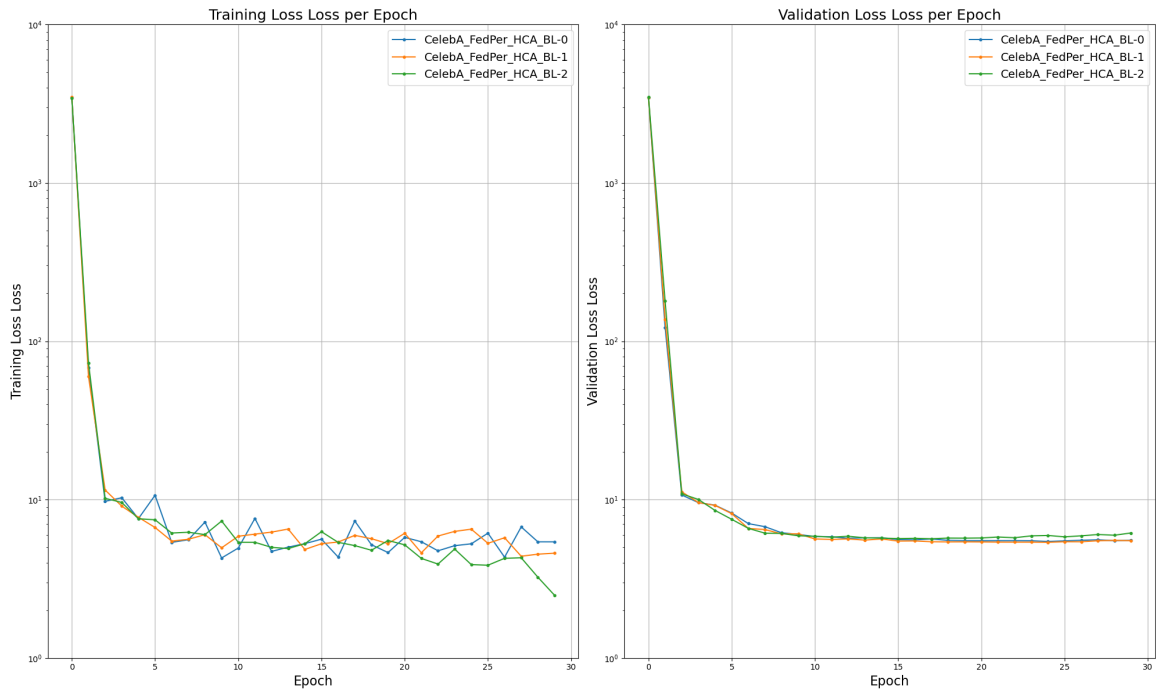


Figure B.8: Average Client Train and Validation Metrics for Face Landmark Detection with Different Degrees of Privatization

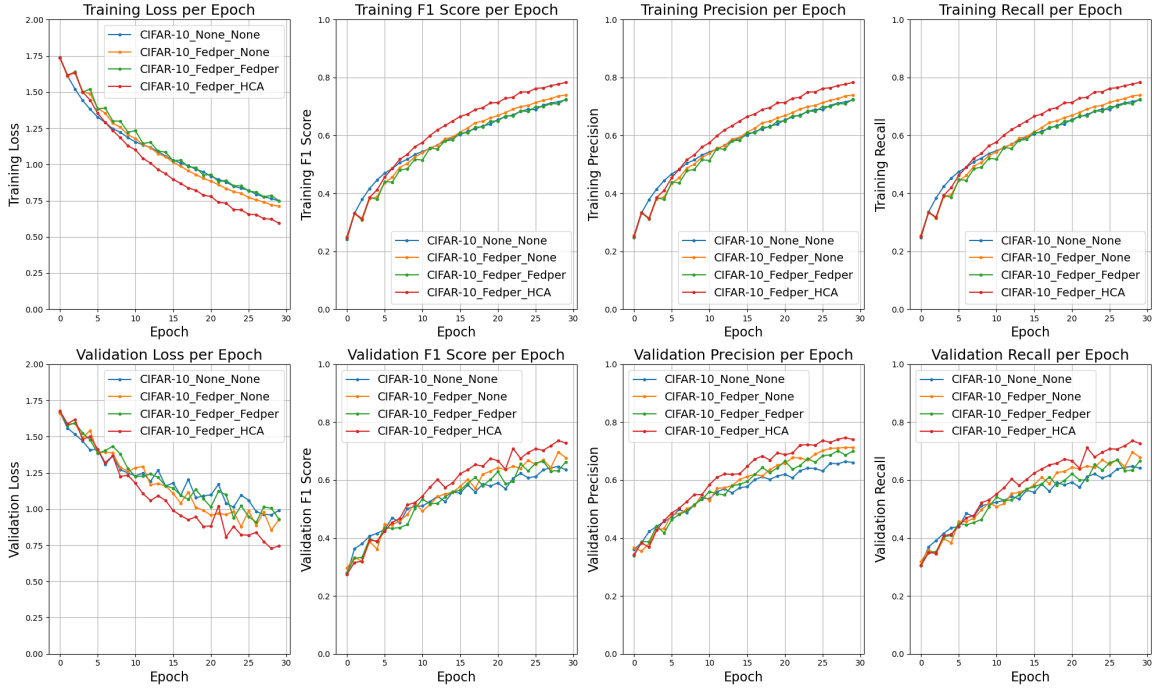


Figure B.9: Average Client Train and Validation Metrics for Animal Classification with Different Aggregation Schemes

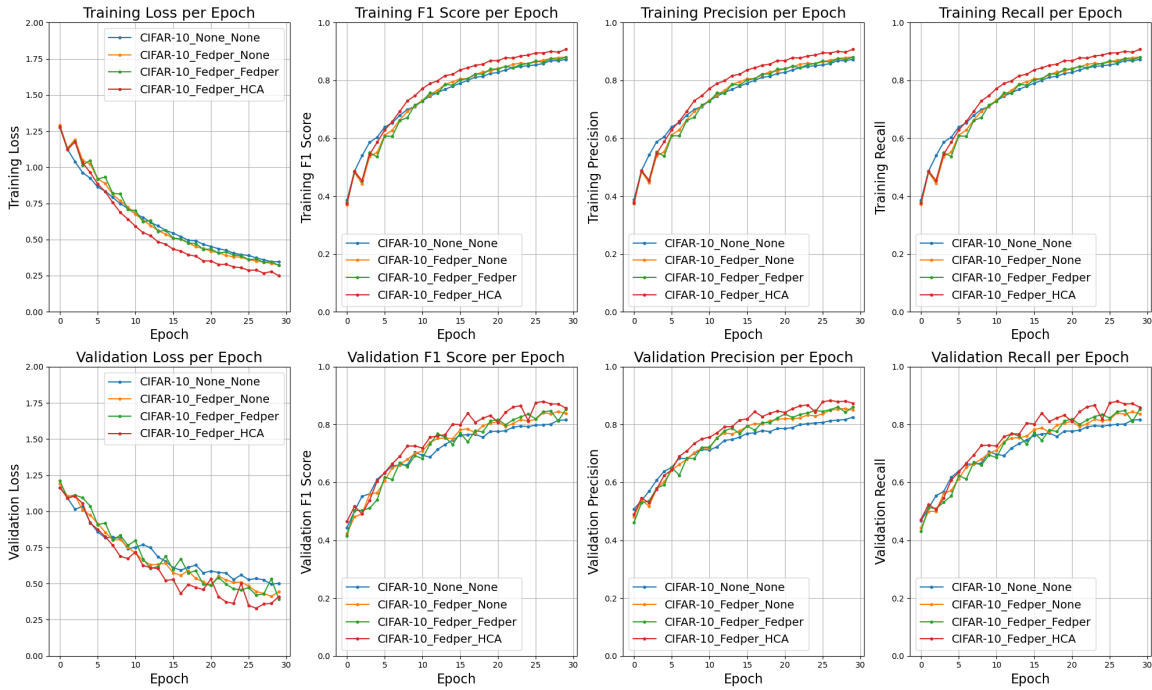


Figure B.10: Average Client Train and Validation Metrics for Object Classification with Different Aggregation Schemes

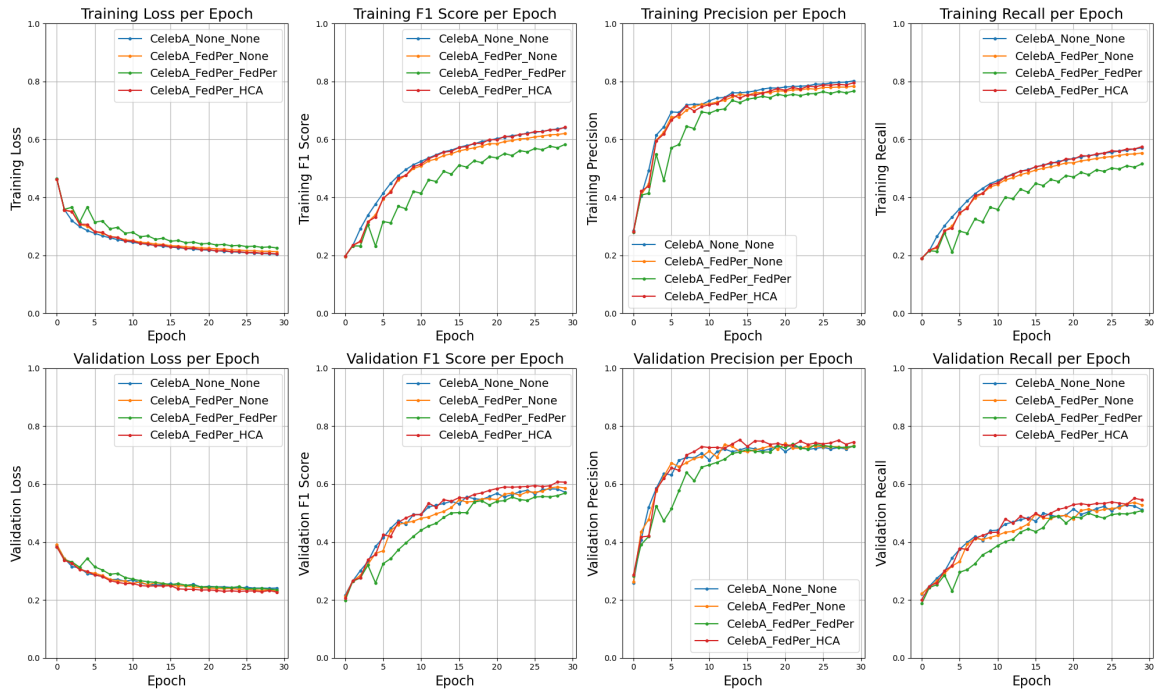


Figure B.11: Average Client Train and Validation Metrics for Face Attributed Classification with Different Aggregation Schemes

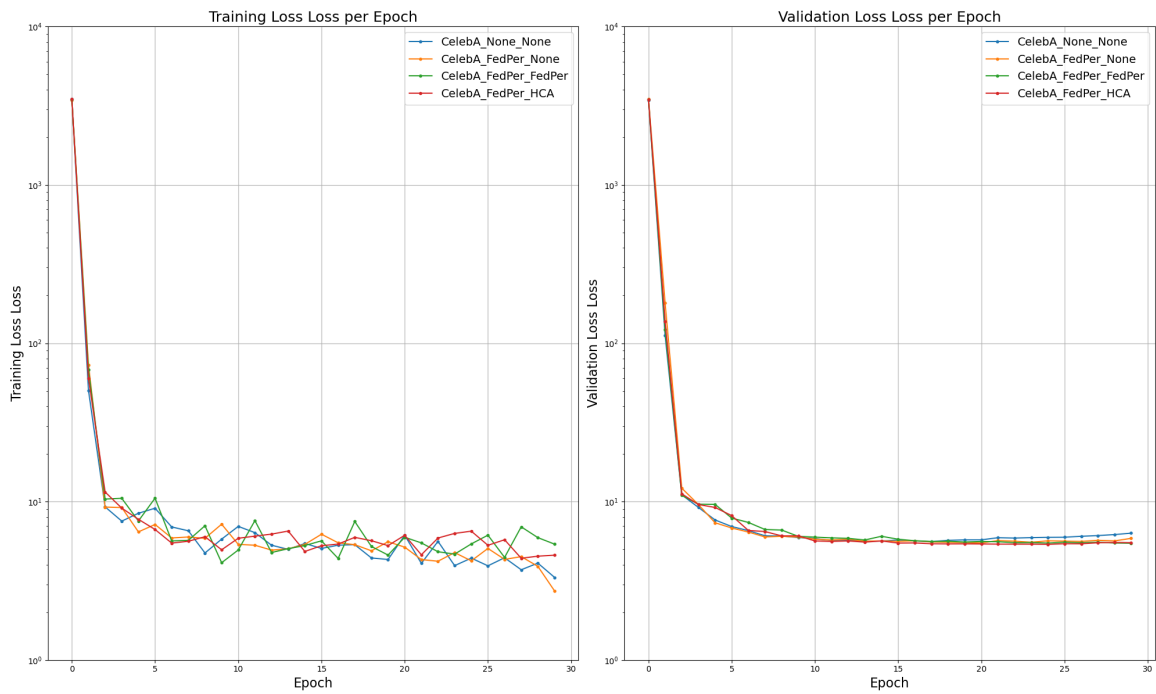


Figure B.12: Average Client Train and Validation Metrics for Face Landmark Detection with Different Aggregation Schemes