Heterogeneous Federated Learning in Robotic Systems

Jan Przybyszewski

Master's Thesis

ROB10

June 2, 2022

AALBORG UNIVERSITY

Aalborg University
Department of Electronic Systems

Department of Electronic Systems
Aalborg University
https://www.aau.dk

AALBORG UNIVERSITY

Title:
Heterogeneous Federated Learning in Robotic Systems

Theme:
Master's Thesis

Project Period:
Spring Semester 2022

Participant(s):
Jan Przybyszewski

Supervisor(s):
Prof. Chen LI, Ph.D.

Copies: 1

Page Numbers: 78

Date of Completion:
June 2, 2022

Abstract:

In modern, data-driven word, privacy becomes a significant concern for users of robotic systems. Federated learning (FL) is a machine learning paradigm in which a federation of clients is trained collaboratively, without sharing local datasets, consequently increasing their privacy. In this work, a novel heterogeneous FL framework is proposed, capable of training federations regardless of the model architectures used. With this framework, grasp prediction models are trained, and a pick-and-place pipeline is deployed, presenting the first application of FL in industrial robotics. In addition, the flexibility of the system is shown in image classification and sentiment analysis tasks. The influence of knowledge distillation on training results is also investigated. As the results show, the presented FL framework can significantly improve client performance compared to training clients in isolation. At the same time, contrary to standard distributed learning approaches, it mitigates privacy risks introduced by data sharing.

Preface

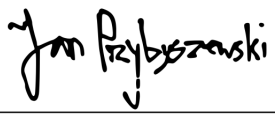
Although the topic of data privacy has recently emerged as one of the key research interests, when I prepared to select a topic for my Master's thesis, it appeared to me that it has not yet been thoroughly investigated in the context of robotics. As more and more systems in this domain are data-driven, sensitive information can be exploited by malicious actors and put user privacy at risk. Fortunately, there are some ways to protect data used by machine learning applications, one of which I find especially interesting is Federated Learning (FL). In this thesis, my goal is to bring FL and robotics together, explaining the benefits of this setting and showing a practical example on how it can be used in robotic systems. As a result, I hope to encourage the reader to pursue the investigation of this interesting topic, which in my opinion could be very beneficial to the fascinating domain of robotics.

Acknowledgements

I express my sincere appreciation for the supervision of Prof. Chen LI, Ph.D. during the work on this thesis. The very valuable feedback provided by Prof. LI in every meeting allowed me to improve the thesis and expand my knowledge. The straightforward attitude of the professor and the calm communication provided the environment to finish this work without unnecessary stress, for which I am very grateful.

I would also like to thank all members of the Sano Center for Computational and Personalized Medicine [1] for collaborating on this thesis and creating a great space in which I could conduct exciting research. Specifically, I want to show my appreciation for Prof. Maciej Malawski, Ph.D. The professor's suggestion to investigate the heterogeneous setting allowed me to come up with an interesting research direction, which resulted in this thesis. Finally, robotic experiments would not be possible without Przemysław Korzeniowski, Ph.D., who is the Head of the VR and Robotics lab at Sano.

This work was supported by the EU H2020 grant 'Sano' No 857533; and by the project 'Sano' carried out within the International Research Agendas Program of the Foundation for Polish Science, co-financed by the European Regional Development Fund. This research was supported in part by the PL-Grid infrastructure.



Jan Przybyszewski
jprzyb20@aau.dk

Contents

Chapter 1

Introduction

In modern robotics, machine learning (ML) has become an essential part of many of the applications. Having great success in computer vision tasks, such as object detection, image segmentation or object classification, as well as in other domains, including natural language processing (NLP) and signal processing, ML can be a solution to nearly every common challenge for a typical robotic system, from perception [2], through sensor fusion [3], to grasp detection in industrial manipulators [4] and path planning for mobile robotics [5]. When the amount of data is sufficient, solving many problems in the aforementioned domains is often trivial. However, data sparsity can greatly restrict ML model performance. In many conventional applications, for example, image classification, data samples are collected from many sources and stored in a central location, to accumulate the largest dataset possible and then train a single model to obtain the best performance [6]. Even though some ML-based robotic systems are deployed in thousands of instances [7], creating a potential to aggregate the information from each robot and collect a sufficiently large dataset, in practice robotic applications are usually unable to create a centralized dataset. This could be the result of two restricting factors: data transmission restrictions and privacy protection. Although many robots have the ability to connect to a centralized service through standard communication protocols, the connection is often not persistent due to application-specific factors. For example, a mobile robot can operate in an environment that cannot be reached by any communication means for an unspecified time. In other instances, the communication may not be stable enough for data transfer. A third example of such constraints may be data transmission costs, where economic reasons prevent the system from sending data over the communication channel, although the connection parameters are sufficient for this purpose. On the other hand, even when communication means are present, many robotic systems process sensitive data, bringing great privacy concerns to the table. Transferring these data to a centralized location may not be advised because of legal regulations protecting the

system's user, inability to collect required consent from the user, or just to mitigate classified information leaks. An example of exposing the user to privacy risks could be any robotic system based on automatic speech recognition (ASR), such as robotic assistants [8]. These systems have access to the user's verbal communication, containing often very sensitive information, which, if accessed by an unauthorized third party, can be exploited and cause harm. Furthermore, in many industrial applications, there may be robotic ML systems that operate on visual data, e.g. for grasp detection or quality control. Passing these data to a third party can also create privacy risks, leading to information leaks about the production process, including product details, procedures or manufacturing equipment, and other industrial secrets, crucial for the company to keep the edge over its competitors. The aforementioned considerations restrict many ML models to use only samples collected by robotic systems locally, resulting in inferior performance, when compared to the centralized models trained on datasets aggregated from a number of clients.

To address similar issues, in a 2017 article by McMahan et al. [9], a novel approach to distributed learning was presented, called Federated Learning (FL). As mentioned above, standard ML services collect training data from client devices in a centralized location to improve the performance of a common model. FL approach breaks this paradigm, creating an improved model without any training data transfer between the client devices. To implement this idea, the authors proposed the Federated Averaging (FedAvg) algorithm, originally used in a so-called cross-device setting, characteristic of services deployed on mobile phones. In this setting, there is a large pool of devices with small computational resources, often unreliable in terms of both availability (owing to communication constraints and/or costs) and reliability (the devices may often break down). The algorithm was successfully used to improve Google Board (GBoard) [10] predictions. GBoard is a service available on Android devices [11], which can prompt the user with suggestions for the next word while typing, using an ML model. Each time the user clicks on one of the words prompted to use it in the sentence, the device stores the text that has been typed and the word chosen as a data sample for model training. Each FL round, a number of devices is chosen, and a local model's training is triggered on each of them. This is done while the device is idle and connected to power, to prevent the user from noticing degradation in performance and availability, as training removes computational resources and battery life. After a number of local training steps, the models are transferred from each client to a central server and aggregated using weighted averaging. The resulting model is used as an updated central model and broadcast to update the clients and improve their performance. Storing the latest model in a central location has another advantage: in case any brand-new devices join the federation, they can receive the most-recent model straightaway from the central server, and provide the best user experience

right from the start.
In robotics, the characteristics of the cross-device setting are typical for the services deployed on small mobile robots. However, FL use cases are not limited to pools of unreliable clients. Another setting that could use the advantages of FL is a so-called cross-silo setting [12], where a smaller number of relatively reliable and well-communicated clients participate in the federation. This could be multiple organizations that collaborate to improve a common process. This sort of setting also applies to robotic applications, e.g. when multiple companies collaborate on training a model for a common robotic task, such as grasp prediction, at the same time avoiding privacy risks related to business secrets outlined earlier. Such applications could be used not only in manufacturing but also in healthcare, when sharing sensitive patient information is not advised.

Due to the clear advantages of FL, the domain has seen a considerable amount of research activity since its inception. The main research topics currently explored include discovering new optimization methods, evaluating the implications for user privacy, defending against malicious attacks, and ensuring fairness and low bias in FL systems. In general, current research in the field focuses on homogeneous environments, which means that all clients employ a model of the same deep learning architecture. In this case, a shared model must be suited for all of the clients. This approach is surely not optimal if the potential heterogeneity is considered - each client may have different computational capabilities, and sharing the architecture means that it has to be suitable for the client with the lowest computational resources, often leaving more powerful clients underutilized. At the same time, as established before, sharing the knowledge between clients is proven to be beneficial; therefore, investigating how the knowledge can be transferred between clients with differing model architectures could be very important, especially in the cross-silo setting, where there are many companies with different resources that would like to participate in a federation. Unfortunately, heterogeneous environments cannot use standard FL optimization methods, e.g. FedAvg, as they are designed with homogeneity in mind.

The objective of this thesis is to contribute to the robotics and FL domains. A novel FL framework is proposed, in which collaborative training of clients with heterogeneous models is possible without any data transfer. As a robotic contribution, the framework is employed in a grasp prediction task, showing the method's suitability in this setting. For this purpose, I train an adaptation of the Grasp Quality Convolutional Neural Network (GQ-CNN) on the grasp prediction dataset and use the trained models in a pick-and-place pipeline deployed with the use of a robotic arm. In this way, to the best of my knowledge, I present the first documented use of FL in a pick-and-place application, expanding the limited amount of research in the robotic federated learning domain. In addition, the applicability of the method for two other ML datasets is investigated, for natural language

processing (NLP) and computer vision (CV) tasks, to showcase its flexibility. I also evaluated the use of knowledge distillation (KD) [13] as a way to improve knowledge transfer during training. Contrary to existing heterogeneous FL approaches, the presented framework allows the federation to be trained without any auxiliary public dataset, which is essential for tasks where data collection is difficult.

The organization of this work is as follows. The current research directions in FL are described in Ch. 2. I reiterate on the motivation of the work and define the research problem in Ch. 3. In Ch. 4, I explain the architecture and workflow of the framework, as well as the experimental setup (including datasets) and implementation details. In Ch. 5 the results of this work obtained during the experiments are presented. In Ch. 6, I discuss the results, analyze the limitations of the framework, and outline possible future improvements. The work is summarized in Ch. 7.

Chapter 2

Literature review

2.1 FedAvg

As described in the Introduction, research in the FL domain was initiated by the paper published by McMahan et al. [9] in 2016. In the paper, the authors defined the decentralized training problem solved by FL, proposed a practical algorithm (FedAvg), and evaluated it in three machine learning tasks. Importantly, they defined the Federated Optimization (FO) problem and outlined key distinctions that differentiate this problem from Distributed Optimization (DO). Key distinctions are:

- Non-IID assumption
- Unbalanced device utilization
- Extreme-scale distribution
- Limited communication

In DO, the usual assumption is that the data for each participating device is drawn from a uniform distribution and that each data sample is independent of previous samples; therefore, the data for each client are independent and identically distributed (IID). This is achievable because in DO a central dataset is distributed in a controlled way to the participating clients, and thus the central server has control over the data distribution. This assumption is, however, not true for FO, as the central server has no control over the distribution, each client is collecting its own data, and there are many factors (geographic factors, time zone, etc.) that in practice make the data non-IID. The notion of unbalanced device utilization comes from the fact that DO usually takes place in data centers, which have highly available client devices, so the goal of the optimization algorithm is to balance the utilization as much as possible and to optimize the available hardware resources.

However, with FO, client devices can be highly unavailable. Some available clients may be used very frequently, and some that are available only from time to time can hardly be used to perform the optimization. The extreme-scale distribution means that FO can potentially use many more clients than the number of training samples on each client, which is not typical for DO. Limited communication is once again related to the optimization infrastructure: in DO, there are no assumptions of limited communication, as it takes place within a data center, with bandwidth restrictions and related bariers. However, FO should be suitable for mobile devices, for which bandwidth restrictions and costs are an important factor. McMahan et al. decided to focus on the first two characteristics of the FO problem. The authors compared two algorithms: the so-called FederatedSGD (FedSGD) and Federated Averaging (FedAvg). FedSGD is a naive implementation of DO in a federated setting: In each training round, the participant k with the learning rate η performs one training epoch, computing the average gradient g_k on all local data, resulting in an updated local model w_t^k (Eq. 2.1). The update is then sent to a central server, which computes a mean of all the received gradients and updates the current model (Eq. 2.2). The model is also sent to each client, so that during the next training round all clients work on the most recent model (Eq. 2.3).

$$\forall k, w_{t+1}^k \leftarrow w_t - \eta g_k \tag{2.1}$$

$$w_{t+1} \leftarrow w_t - \eta \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k \tag{2.2}$$

$$w_{t+1}^k \leftarrow w_{t+1} \tag{2.3}$$

On the other hand, in FedAVG, in each round, the client k performs a number of training epochs, where each local training epoch results in a local model update (Eq. 2.4). Only after a selected number of epochs are performed do the clients send an updated local model to the central server for averaging.

$$w^k \leftarrow w^k - \eta g_k \tag{2.4}$$

An important factor in the FedAvg algorithm is the batch size, which, as the authors find, strongly influences FO performance. With the number of computations E and the batch size B explicitly formulated as hyperparameters of this algorithm, FedAvg can be thought of as a generalization of FedSGD. When $E = 1$ and $B = S$, where S is the number of local training samples in each client, the FedAVG algorithm becomes FedSGD. The authors indicate that they found the averaging of models trained on different subsets of data to work very well if the model weights were initialized using the same weight distribution. The experiments carried out consisted of four machine learning problems: two image classification tasks and

two language modeling tasks. For each parameter set, the authors evaluated algorithm's convergence rate, measured as a number of global updates (communication rounds) necessary to obtain a set performance score, e.g. 0.95 accuracy. The authors explored the influence of three algorithm parameters on convergence: the fraction of federation clients that participate in training, the number of local training rounds, and the batch size. As expected, the increase in the number of participants speeds up the training. More interestingly, the authors found that the higher the number of local updates on clients (which can be the result of decreasing B or increasing E) between communication rounds, the faster the algorithm converges. Additionally, this effect is present for both IID and non-IID data. However, for too large E values, the algorithm can diverge or plateau, which means that this parameter must be carefully tuned. The key findings presented were observed on benchmark datasets, including MNIST [14] and Shakespeare [15], but the authors also run large-scale tests on CIFAR [16] and 10-million post-social network datasets, which supports their research as one that is possible to implement for real-life application.

2.2 Federated optimization methods

After FedAvg was published, several works focused on FO followed. As a motivation for the FedProx [17] method, the authors pointed out that FedAvg does not fully address the issues related to variation of client hardware, referred to as system heterogeneity. It does not allow clients to do a variable amount of work according to their computational potential. Instead, at each round, all clients compute the same number of local epochs, and if any client is a straggler, then this client is just dropped to prevent training stoppage. Moreover, there is an issue of statistical heterogeneity; it was later proven that FedAvg can often diverge in a non-IID setting. FedProx is a generalization of FedAvg with two simple, yet key, changes. First, FedProx introduced partial work aggregation: If during a training round, some clients take more time to locally train the model, then after a set time, the algorithm does not wait for them to finish. Instead, it uses the partial update computed so far by the straggling client to perform the central model update. In addition, the authors introduce a term called the proximal term. Adding this term to the client loss function forces clients not to diverge too much from the global model. Intuitively, even though the FO algorithm should make the best use of clients with more computational capability, these clients can dominate the FO task and reduce global model performance. By keeping such clients close to the global model received at the beginning of the training round, the proximal term restricts the negative impact of the variable number of computations. The authors thoroughly evaluated the algorithm on both synthetic and real datasets. The investigation shows that the proximal term can greatly stabilize the training of the

federated model, but its value has to be carefully tuned. The use of partial updates can also positively influence training, even when operating with many straggling clients.

The authors of the stochastic controlled average algorithm (SCAFFOLD) [18] attempted to solve the FedAvg problem with heterogeneous data. As a result of unfavorable data distribution across the clients, the SCAFFOLD authors have proven that simply averaging the updates misleads the global model far from the optimum, as the clients diverge on their local datasets, especially when a large number of local computations are done at each training round. SCAFFOLD alleviates this problem by introducing control variables for all participating clients (c_i) and the global model (c). Control variables can be initialized with 0 and updated after each training step, both for the global model and the clients. Note that in this setting, the clients are stateful - control variables are adjusted between the rounds, not overwritten. After each calculation of the local model y_i gradient g_i, a term is added to the standard model update, which adjusts the update direction to be similar to the direction of the global model (Eq. 2.5). After computing local updates, the global model and the control variables are updated.

$$y_i \leftarrow y_i - \eta(g_i(y_i) + c - c_i) \tag{2.5}$$

This term resembles the momentum term [19],a popular technique used to stabilize convergence in gradient-based optimization. The authors show that the introduction of control variables acts as a variance reduction mechanism and prevents the global model from divergence in scenarios with many local computation steps. Whereas FedAvg is prone to reduced performance in this case, SCAFFOLD can be resistant to any deterioration. With this characteristic, the FO can be greatly accelerated, as more local computation steps speed up training in general, as also found in the FedAvg paper.

Another method that focuses on ways to alleviate the issue of objective inconsistency, when the number of local updates varies between clients, is the Federated Normalized Averaging Algorithm (FedNova) [20]. As stated when describing the SCAFFOLD algorithm, this can reduce the convergence time as it prevents many local computations between global model updates. The authors of FedNova propose a novel aggregation strategy. In FedAvg, each client runs local computation steps, and, after all steps are computed, the resulting local models are aggregated. In FedNova, instead of aggregating the resulting local models, each client computes the average of the gradients obtained during the local computation. Then those mean gradients are averaged using a weighted average. The weights decide which clients are more important for the global model update and can be tuned accordingly to the application. The algorithm has been evaluated on a synthetic FL dataset, as well as CIFAR [16]. The results show that FedNova allows for faster convergence, compared to FedAvg and FedProx. The authors outline that this method

is also very flexible and can incorporate additional strategies to alleviate the objective inconsistency issue, e.g. by adding the proximal term from the FedProx method.

2.3 Personalized federated learning

In the Iterative Federated Clustering Algorithm (IFCA) [21], the authors focus on one of the formulations of the non-IID FL problem, termed clustered FL. In clustered FL, non-IID data is tackled by clustering the federation participants into groups, e.g. in recommender systems, these may be the users interested in different domains, such as sport, or politics. A global model may not be suitable in this setting, but it may be beneficial for users to personalize the model to better fit each cluster, maximize performance, and improve user experience. To address issues with data heterogeneity and different model objectives, the authors propose that the centralized server will track the k central models, where k is the number of clusters. Each training round, all central models are sent to the clients. Each client finds their respective models by performing inference and finding the model with the lowest loss. Local training is then performed by the client on the best-fit model. In the aggregation step, for each central model, only the updates presented by the clients in the respective cluster are aggregated, and the updated models are broadcast back to the clients. The tests conducted on the CIFAR [16] and MNIST [14] datasets show that tracking a number of models and customizing them according to the group membership of the client allows the algorithm to obtain a higher classification performance than the FedAvg method. However, this comes at the cost of more computations and a larger memory footprint of the algorithm.

The authors of the pFedMe algorithm [22] also investigate how to personalize the global model. Their approach is similar to the one presented in the FedProx algorithm. The global model update works in the same way as the FedAvg algorithm. During local computation, however, clients use a special loss function, with a regularization term based on the l2 norm, to keep the local models as close as possible to the global model, At the same time minimizing standard loss term, respectively, to the objective. This use of the Moreau envelope [23] allows the models to overfit to local data, improving the performance for each client. At the same time, the regularization term prevents the global model from a reduced performance due to the aforementioned objective inconsistency. The results show that this strategy yields better results, compared to the FedAvg algorithm: the average performance of personalized models was improved both on convex and non-convex problems.

2.4 Privacy in federated learning

In addition to investigating FO approaches, many researchers focused on different aspects of privacy related to FL in general. In the DPFL article [24] the authors point out that the fact that data are not transmitted alone does not mean that there are no risks, as the transferred model weights pose privacy risks; it is proven that both image recognition and language models can memorize unique patterns in training data, which can be exploited by privacy attacks [25, 26, 27]. As described by Geiping et al. [28], there are numerical methods that can reconstruct inputs just from gradient information, both for Stochastic Gradient Descent (SGD) [29] optimisation, as well as for batch gradient descent. The methods work not only for fully connected architectures, but also for mixed models, such as convolutional neural networks (CNNs) [30]. Geiping et al. is able to reconstruct low-noise images even when the local training step operated on a large number of images and states that no data transfer does not mean that the privacy of the clients is fully protected. This motivates the authors of DPFL to use a differential privacy (DP) mechanism to hide client contributions while maintaining high optimization performance. DP is a promise of the data holder that the privacy of the data subject will not be hampered, even if their data will be shared to a different data holder. In other words, in differentially private databases, reidentification of the data samples is not possible, even using auxiliary information related to the data samples present outside the database. To date, many works have suggested ways to identify anonymized data samples based on auxiliary information [31]. DP mechanisms can address this problem, preventing such mechanisms from being effective. To achieve this, the authors used a Gaussian mechanism (GM). The local training step is carried out as in the FedAvg algorithm; however, when the updates are aggregated, Gaussian noise is added, effectively hiding client contributions. The results show that with a sufficient number of clients (10000), DPFL has a performance comparable to standard FedAvg, which is consistent with the law of large numbers [32] located at the base of DP. At the same time, introducing a great increase in client privacy, preventing potential attackers from accessing information about local training datasets using client updates. For federations with a small number of clients (100-1000), this method yields inferior results.

Another work related to this topic presented a DP-FedAvg algorithm [33], an improvement over FedAvg to incorporate the DP mechanisms proposed by the original authors. They introduce several steps to improve privacy. Contrary to FedAvg, they randomly sample the number of clients selected for each round, rather than always using the same number of clients. The updates computed by each client during the local computation steps are restricted by adding a L_2 norm term. Bounding the client updates is an essential step to estimate the sensitivity, which in turn is used to scale the Gaussian noise added during the client

updates aggregation, thus ensuring DP. The authors evaluate the algorithm on a next-word prediction task, using a recurrent neural network with long short-term memory layers (LSTM-RNN) [34, 35] in a Reddit post dataset [36]. The authors thoroughly investigate the influence of various update clipping strategies, different levels of noise, and the comparison of predictive performance with vanilla FedAvg. In terms of predictive accuracy, the DP algorithm yields results only minimally decreased scores, compared to its non-DP counterpart. Although the aforementioned approaches to DP in FL used the so-called global DP with noise-adding step conducted in a central location, some representatives of local DP [37] proposed to conduct the privatization on clients, before the updates are sent to the central server.

Apart from using DP to increase the privacy of participating clients, some authors also add secure multiparty computation (SMC) mechanisms to the domain. A hybrid approach was proposed in [38], where SMC is combined with DP to ensure privacy at the same time, preventing a decrease in predictive performance of ML models. In the first step of the training round, a central server sends a query to a subset of clients. This query can, e.g. ask the clients to compute updates to current global model, as in a standard FO scheme. Next, the clients use local DP mechanism, e.g. adding Gaussian noise to the local updates. Updates are then encrypted using a Paillier cryptosystem [39] and sent back to the central server. The Paillier cryptosystem allows SMC to be implemented - using this method, the central server can aggregate the updates in encrypted form, preserving client privacy. To decrypt the aggregated model, the central server has to query a number of clients to participate in the decryption. This results in an updated, decrypted model, which can be broadcast back, as in standard FL. This hybrid approach further increases the security of client information. The authors evaluated the algorithm on three ML predictors: Decision Trees [40], CNNs and multiclass SVM [41]. The results show that this method retains high prediction performance while at the same time reinforcing privacy guarantees over standard DP approaches in the FL domain.

Following the publication of the FedAvg algorithm, another SMC approach was presented in [42], in the form of the Secure Aggregation protocol. This work uses two types of coordinated perturbation applied to model updates after local training steps. The perturbations are agreed between the clients so that they cancel out during aggregation. At the same time, the central server has no information about the perturbation, so even if the server is overly curious about the client's data, it cannot reconstruct client's original updates. The protocol also provides strict communication encryption guidelines, which further increase security, e.g. against man-in-the-middle attacks. This algorithm, together with FedAvg, is implemented in the Gboard application [10].

2.5 Attacks analysis

In addition to investigating ways to improve user privacy, researchers also demonstrated ways to exploit the FL setting in malicious attacks. Bagdasaryan et al. [43] presented a backdoor attack on any standard FO algorithm in which, during the local training step, the attacker can train its local model on a specially prepared dataset. For instance, a classification task is considered, this dataset could make the local model misclassify images containing specific features, e.g. images of purple cars can be misclassified as birds for image classification task. In language modeling, this could mean that, for specific inputs, the attacker wants to output a swear word for the next-word prediction. When attacking the federation, the attacker can influence the global model to obtain a different objective from all the other users. The naive way to do this would be to poison the local dataset with data samples re-labeled according to the malicious objective. Standard FL algorithms are, however, quite robust to such attacks - with many clients, the updates from the attacker's model are canceled and overwhelmed by other updates. To target the FedAvg algorithm, the authors propose using an estimation of the update necessary for the attacker's model to influence the global model sufficiently, so that it can fulfill the malicious objective. In this way, the attacker essentially substitutes the global model for a malicious one, thus executing an attack. As the objective can mean just misclassifying a small fraction of samples in the dataset, it can be impossible to notice tracking only the predictive performance. The authors showed that this attack is very effective and conducting it even for one training round during FO can influence the global model for a number of the following epochs. The authors were able to reproduce this on both image classification and language modeling tasks.

Bhagoji et al. [44] presented a very similar approach to backdoor attacks, with the attacker boosting its model updates to preserve it during the FedAvg aggregation step. In their approach, the authors added a regularization term that keeps malicious updates close to the global model. This allows the attacker to increase the stealthness of the attack, as the global model retains high performance on the original objective and no negative impact can be noticed. At the same time, the global model can implicitly fulfill the attacker's objective. The authors also propose a strategy of alternative minimization, where the attacker's objective is minimized every other round, alternating with the original objective, to mitigate the risk of detection. Another important characteristic of the proposed method is that it is also efficient for Byzantine-resistant aggregation algorithms, such as Krum [45] and coordinate-wise median [46].

Xie et al. [47] show a distributed attack model. Contrary to previous, centralized approaches, the authors show that they are able to divide malicious features (e.g., an image pattern) into parts and distribute crafted data samples containing

those parts between a number of malicious clients. Then, all these clients can participate in FO. The phenomenon observed is that the parts of the attacker's objective fulfilled by the adversarial clients can together make the global model achieve the malicious objective. Furthermore, the authors prove that the distributed nature of the approach not only makes a better use of the available computation, but is also more persistent compared to non-distributed attack methods, even in instances where the attack is carried out during one training round (one-shot attack). In a detailed evaluation of the approach, the authors investigate the influence of the number of adversarial clients, the distribution of features, and other factors on the predictive performance of the attacker's objective.

2.6 Incentive and fairness in federated learning

So far, the methods mentioned evaluated various FO algorithms, analyzed privacy risks, and proposed solutions to these risks. A very practical question that is out of focus for this method is: What motivates the user to join the federation in the first place? This question was asked by Kang et al. [48], who investigated the incentive mechanisms that can influence users to participate in FL. As noted above, without incentives, selfish users will not be motivated to participate in the training, preventing FL from being used in practical applications. The authors propose to classify potential participants with a profit function that assigns value to each of them, depending on the quality and reliability of their dataset. Grades can be based on observations from previous FL tasks in which these users participated. After grading potential users, the task provider should propose each user a custom contract with an incentive proportional to the profit value brought by the user to the federation. This way it can motivate the user by providing a beneficial incentive (material or any other) in exchange for participation. The authors also proposed a formulation of the incentive mechanism metrics that can help assess whether users are properly motivated, as well as whether the task provider benefits from the user contributions.

A topic closely related to incentive mechanisms is fairness in FL. In most of the proposed FO algorithms, the global model accuracy is frequently measured as an average of the accuracies of clients on their respective test sets. The authors of q-Fair Federated Learning (q-FFL) [49] point out that this may not be the best metric. Obtaining a high average accuracy for the federation does not mean that the accuracy for all users is close to this value. In reality, it may happen that some users have a very low accuracy score, but this is offset by users with a very high accuracy. This means that the global model is not fair, as for some users it works very well, while at the same time yielding poor results for the rest. q-FFL focuses on providing a more fair (more uniform) accuracy distribution in the federation. After each local training round, q-FFL can estimate the loss of each client based on

their gradients. When the model updates are aggregated, but using the weighted average, the weights proportional to the loss are used to assign more emphasis on the weaker models. This results in a shift in the distribution of the model performance towards more uniform. The method is tunable; the user can benefit from fairness at the expense of mean model performance, and vice versa. The authors evaluated the algorithm on many FL benchmarks and showed that it can be used efficiently to increase the uniformity of the federation results, providing more fairness.

2.7 Decentralized federated learning

Most traditional FL methods worked in a centralized framework, where the central server coordinates the work of the clients. However, some articles presented different distributed workflows. In [50], the authors show the use of blockchain technology [51] in FL, called BlockFL. As outlined, the centralized approach has several issues: The federation is prone to central server malfunction. Moreover, when the clients participate in FL they do not have any voice, and they are essentially forced to contribute every training round. As mentioned when describing incentive mechanisms, this leaves clients with more data unmotivated because they cannot benefit from the FL as much as smaller clients. BlockFL allows clients to choose peers with whom they want to form a federation, without central server oversight. In addition, the method also encapsulates incentive mechanisms, since clients can seek reward for participating in the training. For this purpose, the so-called mining rewards, which are essentially built into blockchain technology [52], can be used. Therefore, the authors argue that Blockchain is a natural setting for decentralized FL.

Another work on this topic is FedLess [53]. To conduct training, the authors use a Function-as-a-Service (Faas) platform [54, 55, 56, 57]. FaaS functions are small code units that are stateless. They are an emerging paradigm in cloud computing. The reason they are so appealing is that they can be executed using a chosen trigger, e.g. a HTTP request. Only after the function is triggered is the function instance created, and the needed computing resources are allocated by the cloud infrastructure provider. This means that only when the function outcome is really requested, the resources are used, with no idle time-related costs. FL can be implemented using the FaaS paradigm. Contrary to traditional approaches, where clients wait idle until they are selected for training and waste resources, FedLess instantiates the functions responsible for training only when the respective client is selected. This provides better scalability and reduces costs. Note that FaaS is stateless; therefore, client models and datasets still have to be stored in a database instance to be persisted between the rounds; however, database operating costs are far smaller than the cost of idle computing instances. A limitation to this method

is the usual restriction applied by FaaS providers; for example, for AWS Lambda [57], the function execution cannot take longer than 15 minutes and it cannot use more than 15 GBs of memory. Although the authors are aware of these limitations, they state that with fast developments in the FaaS domain, these constraints may soon disappear or be relaxed.

2.8 Heterogeneous federated learning

The FedDF [58] method is one of the very first works that addresses the limitation introduced to FL with standard algorithms, restricting them to use models of the same architecture for all clients. The aim of FedDF is to train a centralized model using a federation of clients that use heterogeneous models, using knowledge distillation (KD). Knowledge distillation was originally proposed by Hinton et al. in [13] as a way to extract knowledge from an ensemble into a single model. In general, using an ensemble of models instead of a single model trained on the same data always yields better results. However, it is often impractical, as there are larger hardware requirements and additional considerations to take into account when deploying ensemble-based machine learning systems. Hinton et al. attempted to distill knowledge from an ensemble into a single model without sacrificing performance. To do that, the authors hypothesize that for classification tasks, much of the cumbersome (ensemble) model's generalization ability lies in the ratios between probability distributions of each class. They assume that a standard training approach, in which the trained model is penalized just for predicting the incorrect class, is not sufficient to transfer this ability. Instead, the authors propose a novel loss function, which they denote as the "distillation loss". Using the multinomial classification task as an example, the standard output produced by a neural network classifier is the class probabilities q_i, calculated from logarithmic results of the last layer z_i using the softmax function denoted in Eq. 2.6.

$$q_i = \frac{\exp z_i / T}{\sum_j \exp z_j / T} \tag{2.6}$$

The parameter T in Eq. 2.6 is the temperature and affects the resulting probability distribution. A higher value of T makes the distribution over classes softer. In standard objective functions, it is set to 1. In distillation loss, knowledge can be distilled (transferred) from the complex model (denoted the teacher model) into a simple model (denoted the student model), using the loss formulation shown in Eq. 2.7.

$$E(x|t) = -\sum_i \hat{y}_i(x|T) \log y_i(x|T) \tag{2.7}$$

Eq. 2.7, presents a cross-entropy value calculation between the probabilities $y_i(x|T)$ outputted by the student model and the probabilities $\hat{y}_i(x|T)$ produced by the

teacher model. Intuitively, this forces the student to match all the probabilities of the teacher, whereas in standard objectives the model would be only forced to assign a high probability to the correct class. To extract knowledge from an ensemble, the authors use a training strategy in which training input is inferred through all ensemble models. The resulting probabilities are then averaged and used as teacher model probabilities. They are calculated using the softmax function; however, the authors found that in distillation loss, higher T values (usually in the [2.0;10.0] interval) often yield better results. This is one of the hyperparameters that should be carefully tuned. After obtaining teacher probabilities, the input is inferred through the student model, and student probabilities are also computed, using softmax with the same T as for the teacher. The teacher and student probabilities are then used to calculate the distillation loss, and on the basis of the loss value, the student weights are updated. Note, that distillation loss essentially does not need labeled training samples. The authors evaluated the distillation loss on computer vision and NLP tasks. The results proved that the use of knowledge distillation during training allows smaller models to obtain performance comparable to that of a model ensemble. Crucially, the authors found that using a loss function consisting of both distillation loss and standard objective loss often leads to better results. An example of such a function for a classification task is shown in Eq. 2.8

$$E(x|t) = -T^2 \sum_i \hat{y}_i(x|T) \log y_i(x|T) - \sum_i \bar{y}_i \log y_i(x|1) \tag{2.8}$$

Here, the function has a term added to the distillation loss, which corresponds to the cross-entropy between a known label $\bar{y}$ and the student's probabilities computed with $T = 1$. Note that in this formulation, the distillation loss term is weighted by T^2. When higher values of T are used in softmax, the gradient of the model weights is scaled by $\frac{1}{T^2}$. This weight mitigates this scaling effect, preventing the distillation loss from being overwhelmed by the second term. Research conducted in conjunction with the work of Hinton et al. further reinforces this notion - the most notable application of knowledge distillation is the DistillBERT model [59], distilled from the BERT model [60]. 40% smaller in terms of training parameters, DistillBERT is able to retain 97% of its teacher performance. Returning to FL, in FedDF, the central model (the student model) is trained to obtain the same output as the ensemble of client models using a special loss function. One by one, the inputs are inferred through k client models, obtaining k logits. The logits are averaged, and this average is treated as the ground truth for the distilled model. Intuitively, the distilled model has the objective of outputting values as close as possible to the average logits obtained from the teacher models. During distillation, an unlabeled central dataset is used, so there is no data transfer between the central server and the clients. The authors evaluated the algorithm using different architectures (ResNet [61], VGG [62], ShuffleNetV2 [63], DistillBERT [59]) in image classification and NLP tasks using the CIFAR [16], ImageNet [64],

AG News [65] and Stanford Sentiment Treebank [66] datasets. An important part of the evaluation was the comparison with the FedAvg and FedProx algorithms. During experiments, for each training dataset, an out-of-domain dataset was used as a distillation dataset; e.g., when the experiments used ImageNet to train local models, samples from CIFAR were used for distillation. The results show that FedDF, contrary to FedAvg, benefits from longer local computational rounds, actually obtaining better results in this setting and being robust to divergence. The opportunity to increase the number of computations per round means that the algorithm can converge faster than its competitors. In non-IID settings, the distilled central model is able to retain high performance, which makes it superior to the compared algorithms. The authors also managed to prove that even in heterogeneous systems, when the ensemble comprises different architectures, FedDF still obtains a well-performing global model. The heterogeneous system tests used an ensemble of ResNet-32, ResNet-20, and ShuffleNetV2. In general, training the central model using KD has been shown to be efficient and robust.

The authors of the FedMD method presented a similar approach [67], however, here the authors focused on improving the client models, regardless of their underlying architecture. Each training round, clients train their models on local data. After local training is finished, the distillation round starts: using a public dataset, each client passes the same data sample through their respective model and advertises the obtained logits to the central server. The server aggregates the logits from all the clients to form a consensus, which is then used by the clients to train their respective models with the distillation loss function. Using an averaged consensus and broadcasting it to clients, the information is exchanged, and clients can benefit from the federated setting. The steps are repeated for each training round. Evaluation on FL benchmark datasets proved that this approach improves federated models beyond their performance obtained during solitary training, while allowing each client to use its own model architecture.

Hu et al. [68] proposed another federated distillation framework. In the MHAT method, each client has two datasets: private and public. First, the private datasets are used to train the models for each client. After this step, clients use their respective public dataset to obtain logs from the trained model. The logits are broadcasted to the server along with their respective inputs. The server then trains a central model on all data transmitted, using KD loss. Following this step, the server uses its own separate data set and sends logits obtained from this dataset along with the respective samples to the clients, which perform distillation locally, improving their models. The experimental results show that this approach is effective and obtains favorable results compared to the standard method that uses logits averaging to aggregate knowledge from the participants.

2.9 Federated learning in robotics

Documented research on FL in robotic applications is very limited. Majcherczyk et al. [69] investigated the use of FL in trajectory forecasting tasks. In the simulation, the robots moved in a swarm, using different behaviors such as flocking, foraging, and diffusion. Each robot continuously collected data on the trajectory of its neighbors. These data samples were used during the local training step to train a local trajectory forecasting model based on LSTM [34]. The aggregated global model was obtained using a standard FedAvg algorithm. Furthermore, the authors evaluated a decentralized FL approach, with models aggregated in a serverless fashion, on a peer-to-peer basis. In experiments, the authors compared FL, non-FL and decentralized FL approaches and showcased the superiority of FL over the non-FL approach in most settings. However, for some swarm behaviors, the decentralized FL showed favorable results over the centralized approach.

Related research was carried out by Yu et al. [70]. The authors investigated the application of FL in vision-based obstacle avoidance. For testing purposes, three simulated environments were designed using the Nvidia Isaac simulator [71]. The authors used a mobile platform to carry out experiments in which the robots trained a convolutional neural network to classify camera input into two classes: blocked and free. Each robot trained his model on data collected in a different environment. Redundant sensors: a lidar and a 2D range finder were used to collect training data. The resulting FL model was superior, compared to a fully centralized approach, in which the data samples collected by all robots were used to train a single model, without resorting to FL. Apart from assessing that the FL approach is beneficial for the performance of the robots, they observed that the FL model trained on simulation is able to obtain good results in reality, with an insignificant sim-2-real change in performance.

Zhang et al. [72] studied the use of FL in dynamic mapping fusion settings for intelligent vehicles. The work used FL to fine-tune feature models on vehicles that aggregate information from sensors and detect objects approached during vehicle movement. In general, the framework achieved good results during experiments carried out on the CARLA simulation platform [73], and provided low communication overhead, which is a vital characteristic for the setting of intelligent vehicles.

As the number of applications of reinforcement learning (RL) in robotics increases every year [74, 75, 76], some researchers attempted to use RL in the FL setting. Liang et al. [77] used RL to train mobile agents to avoid obstacles in different environments. FL allowed autonomous vehicles to share knowledge with each other, resulting in improved obstacle avoidance and a decrease in collision counts. Four agents were deployed in their respective environments (both real and simulated using Airsim [78]). Agents were trained to avoid obstructions using the deep determination policy gradient method (DDPG) [79]. During each training

round, agents trained their models using RL and subsequently exchanged model updates with a central server. The aggregate updates, using FedAvg, were then used to update the global model, which was broadcast to the agents. In this way, the agents exchanged information during the learning process and continuously benefited from receiving the most recent global model. In experiments, the authors investigated in a real-life environment whether the FL improves the training of the agents, compared to the non-FL scenario, with agents not exchanging their models. The tests included driving on an obstacle course. The results showed that knowledge aggregation and the distributed learning characteristic to FL allow agents to achieve better performance. Moreover, the authors found that training the federation of not only real-life agents, but also additionally simulated agents, yields better results. This proves that FL can also be used to minimize the sim-to-real gap in practical applications.

The ability to avoid different types of obstacles by continuously learning from agents operating in distinct environments, as the authors of the Lifelong Federated Reinforcement Learning (LFRL) method [80] point out, can be regarded as a Lifelong Machine Learning (LML) system. During its lifetime, a system can learn many small tasks (navigating in the presence of different types of obstacles). This is a very appealing setting that promises the ability of a system once deployed to improve over time. The authors of LFRL provide a general framework that is very similar at a high level to the work of Liang et al. LFRL assumes the use of agents based on a Q-network [81] method. For each generation of deployed agents, the system provides an initial model which agents improve in their respective environments. After some time, the trained models are transferred to a central server. A central traning dataset is next created by artificially generating sensor input and asking each model for a prediction, together with the confidence score. The predictions are then fused together, producing labeled training samples. The system then trains a new central model on these training data. It can be used by a new generation of actors as an initial local model. The presented approach obtained better results, compared to standard RL without knowledge transfer, allowing agents to achieve better obstacle avoidance performance and accelerating their initial training.

2.10 Summary

In this section, a comprehensive overview of current research directions in the federated learning domain was presented, including the pioneering FedAvg algorithm, the main optimization methods, and personalization. Many topics surrounding this setting were described, such as attack analysis, incentive mechanisms, or fairness. In this work, I will focus on heterogeneous FL in robotic systems. As outlined in Sec. 2.8, current approaches focus on using an auxil-

iary dataset to perform training, which may not be optimal for some applications, when the available training dataset is small. When it comes to robotics, FL is still not widely known as a relevant research topic. This results in a very limited number of articles on FL in this context. Specifically, existing research focuses on mobile robotic systems, ignoring the possibilities that FL can introduce to industrial environments. Therefore, I wish to fill this gap by applying an FL framework to deploy a relevant robotic system and present the suitability of FL in this setting.

Chapter 3

Problem statement and system requirements

3.1 Problem analysis

In Ch. 1 and Ch. 2 a context for the research problem was established. FL is an emerging machine learning setting where a federation of clients benefits from each other's knowledge without any training data exchange, as is the case in standard distributed learning approaches. The collaborative nature of FL allows clients to share knowledge while avoiding privacy risks related to training data leakage. Without data transfer, FL algorithms are also very efficient in terms of communication overhead compared to DL, which is crucial for many applications. In Ch. 1 the reasons why FL is very attractive for robotics were outlined, including potential use cases for mobile robots and industrial applications. As shown in Ch. 2, only a very limited number of publications are available that apply FL in robotic systems [69, 70]. Standard FL methods focus on training a single, centralized model, but some authors relax this assumption and present heterogeneous FL frameworks, where each client has its own unique model of a specific architecture. As described in Ch. 1, the use of heterogeneous FL makes this paradigm more flexible and applicable to settings where clients are not consistent in terms of the architecture used. This can be very attractive in many cases, e.g., when clients have different computational capabilities. In standard FL, all the clients would have to use the architecture suitable for the least computationally able member. On the other hand, in heterogeneous FL, every client can use the architecture best suited for its computational resources, enabling all participants to perform to their full potential.

3.2 Problem statement

Taking into account the use cases outlined in Ch. 1, I see great potential in applying FL in robotics. However, existing limited research on this topic presents only mobile robotics applications. There are no documented examples of FL for tasks that require industrial manipulators, for example, in pick-and-place. I believe that showing practical usefulness of this paradigm is key to increase the amount of research on the intersection of those domains, which in turn could contribute to the improvement of user's privacy in robotic applications. To contribute to this end, heterogeneous FL approaches could be especially interesting, as in industrial, cross-silo settings, cooperating institutions (e.g., manufacturers, e-commerce warehouses) often have different budgets, which could result in disparities in terms of computational capabilities of the federated participants.

Regarding the state of FL, the single-model assumption of most of the current methods is a limitation, which could make it unpractical for federations consisting of clients with heterogeneous hardware. A limited number of heterogeneous methods exist, but they are all using auxiliary public datasets (that can be shared with no privacy concerns) for knowledge transfer, as a workaround to the no-data-exchange assumption. I see this as a drawback, because it forces clients [68] or the central server [67, 58] to collect additional training samples for this purpose. For clients, this is often impossible, as they can collect only a small amount of data in the first place. Moreover, a practical question arises: How can the client collect data samples that can be made public without raising privacy concerns? For central servers, it also means that to run the training, it has to collect a sufficient amount of data, which is an additional and cumbersome responsibility.

In this work, a limited amount of research on FL in robotics is addressed, by using a heterogeneous FL system in a pick-and-place task, to demonstrate practical applicability in industrial settings. Moreover, I expand the current state-of-the-art in FL, by presenting a heterogeneous method that does not require any auxiliary dataset to train the federation, which can be especially useful for environments with limited training data available.

3.3 System requirements

To apply a new heterogeneous framework in a robotic application, I would like to present an FL system that is capable of collaboratively training a federation of clients. The framework should not use any training data transfer between clients but should allow them to share knowledge with their peers. At the same time, the system must allow models of various architectures to participate in the federation. Contrary to existing approaches, the system should not use any auxiliary public datasets. To summarize these requirements, I compared the system with existing

heterogeneous FL approaches in Tab. 3.1. Furthermore, I classified the technical

Table 3.1: The As-Is vs. To-Be table comparing existing heterogeneous FL methods (As-Is) and the proposed approach (To-Be).

Characteristic	As-Is	To-Be
Allows heterogeneous models training	Yes	Yes
Auxiliary training datasets	Required	**Not required**
Applied in an industrial robotic setting	No	**Yes**

requirements using the MoSCoW methodology [82], as follows.
Must:

- Allow for distributed training with no data transfer
- Allow for heterogeneous models training
- Not require additional public datasets

Should:

- Allow for straightforward model customization
- Allow for straightforward dataset customization

Could:

- Allow for straightforward hyperparameters customization

Won't:

- Support different FO algorithms, e.g. FedAvg

Chapter 4

System design

In this chapter, I will describe the proposed collaborative training system, including the architecture, the participating actors, and the overall workflow. In addition, the experimental setup for the robotic application that was deployed with the use of this system and two other machine learning experiments that were investigated in this work will be described.

4.1 Architecture

The system consists of a central server and a federation of clients (Fig. 4.1). In each training round, clients communicate with the central server and participate in federated training. The goal of the system is to train heterogeneous models to obtain the highest possible performance on a common machine learning task. In principle, this framework is not limited to any communication protocol - the server can communicate with clients using any communication means that are available, for example using the Hypertext Transfer Protocol [83] or WebSockets [84].

4.1.1 Central server

The central server is an entity that serves as a proxy between clients. It does not perform any machine learning-related computation or update aggregation. Its function is to orchestrate federated learning: trigger local training, assign models, and start on-peer training (the workflow stages are described in Sec. 4.2). Due to the limited computing responsibilities of the central server, it is a lightweight part of the system, which mitigates maintenance costs.

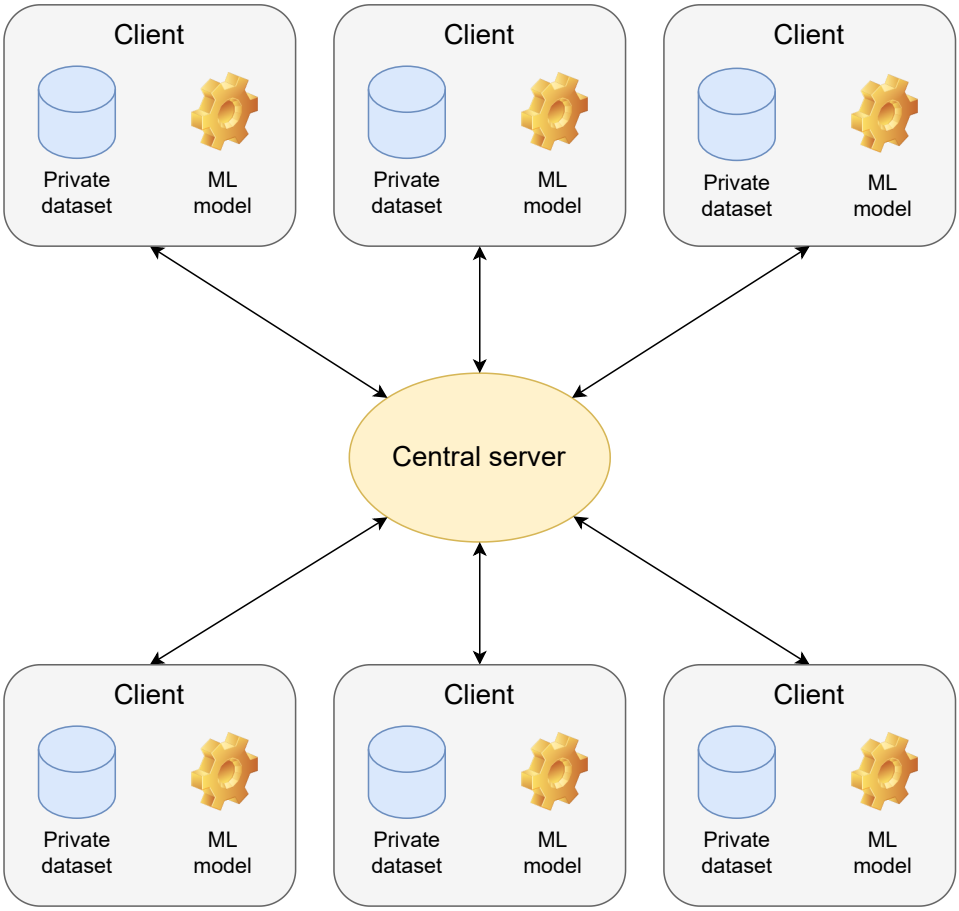


Figure 4.1: A high-level architecture overview. The central server acts as a proxy between the clients, passing models to conduct on-peer training, as well as triggering other federated training stages.

4.1.2 Clients

Clients are entities that can connect to the central server and form a federation, participating in collaborative training. As the presented system is mostly designed to work in a cross-silo scenario, each client represents an institution (silo), which has its own private dataset and a machine learning model. Clients are assumed to have computational capabilities to train their own model, as well as a guest model of a foreign client, assigned by the central server during the on-peer training step of the system workflow (described in Sec. 4.2). In addition, they should have communication channels open to allow models and trigger messages to be exchanged with the central server.

4.2 Workflow

The system is capable of training a number of clients that form a federation to improve their local models. Its workflow consists of five steps that are repeated in sequence for a number of training rounds:

- Local training
- Peer assignment
- On-peer training
- Model reassignment
- Model evaluation

Each step will be described in detail in the following sections. The high-level overview of the workflow is shown in Fig. 4.2 and in Alg. 1.

4.2.1 Local training

The goal of the local training step is to train the local model of each client on the respective private datasets. In this stage, clients work on their own and do not communicate with the central server. During local training, a gradient descent optimization [85] of the local model w_k is performed, using a loss function l, chosen according to the machine learning task carried out by the federation. In this work, the system is evaluated on classification tasks. Therefore, the loss function used is the negative logarithmic likelihood (NLL) loss (Eq. 4.1). The optimizer seeks to find the weights of the model w_k, which minimize the NLL loss on the training dataset. The input x is passed through w_k to obtain the output $\bar{y}_c(x|T)$, which is calculated using the softmax function according to Eq. 2.6 with temperature $T = 1$.

$$l(w_k; b) = -\frac{1}{b_s} \sum_{x} \sum_{c=1}^{C} y_c ln(\bar{y}_c(x|T)), T = 1 \tag{4.1}$$

At each epoch, the client splits the private dataset D_k into b batches, using a suitable batch size b_s. Next, it uses the local model w_k to calculate the loss $l(w_k, b)$ on a batch of training samples b for each input x in the batch. One-hot encoded labels y, contain C values specific to the number of classes in the task. Finally, the client calculates the loss' gradient $\nabla l(w_k; b)$. This gradient is used to update the model, after being scaled by the learning rate η (Alg. 2).

Algorithm 1 HETEROGENEOUS FEDERATED LEARNING with K clients (indexed with k), each with a model w_k; N denotes the number of training rounds; N_p denotes the number of on-peer training rounds; D_t denotes a common test dataset; D_k denotes private dataset of client k; E denotes the number of local training epochs; h denotes the target host id; D_h denotes the private dataset of the target host; E_p denotes the number of on-peer training epochs. The algorithm returns a set of trained models.

inputs: K, D_k, D_t, N_p, E, E_p
Orchestrated by the central server:
for k in range(K) **do**
  initialize w_k
end for
for i in range(N) **do**
  for k in range(K) **do**
    $w_k^l \longleftarrow$ LocalTraining(w_k, D_k, E)
    ModelEvaluation(w_k^l, D_t)
  end for
  for j in range(N_p) **do**
    targets $\longleftarrow$ PeerAssignment(K)
    for k in range(K) **do**
      $h \longleftarrow$ targets[k]
      $w_k^p \longleftarrow$ OnPeerTraining(w_k^l, w_h, D_h, E_p)
      ModelEvaluation(w_k^p, D_t)
      $w_k \longleftarrow w_k^p$ // model reassignment
    end for
  end for
end for
return $\{w_1, w_2, ..., w_k\}$

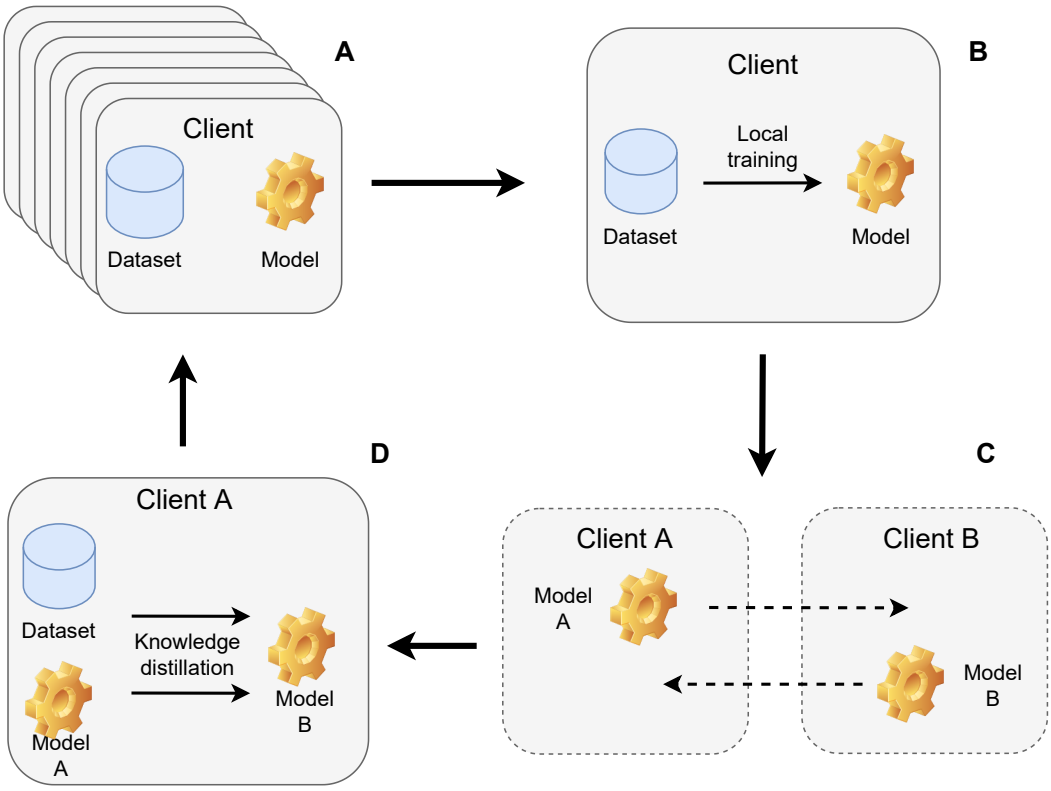


Figure 4.2: System workflow. A: A federation participating in training consists of a population of clients, each with their own private dataset and a ML model. B: In the first step of the training procedure, each client performs the local model training on the private dataset. C: The updated model weights are assigned to foreign clients for the on-peer training. D: The on-peer training trains the model of the guest client not only using the private dataset of the host, but also using the knowledge of the host model by means of knowledge distillation. For evaluation purposes, the models are tested using a common test dataset after steps B and D.

4.2.2 Peer assignment

During the peer assignment step, the central server assigns the model weights of each client to a foreign host to undergo on-peer training. The assignment approach should take into account application-specific requirements. For example, the central server could consider the computational capabilities of the clients so that larger models are not assigned to hosts that cannot fulfill their hardware requirements. Another instance of application-specific conditions could be the availability of the clients - the central server should not assign unavailable clients as hosts for on-peer training, as waiting until such clients become available could result in a large latency.

In this work, it is assumed that all clients are available and have the computational means to train any model in the federation. With this assumption, a

Algorithm 2 LocalTraining. The client model w_k is trained on a private dataset D_k; b_s denotes the batch size; η denotes the learning rate; E is the number of local training epochs; $\nabla l(w_k; b)$ is the gradient of loss function calculated in each step of gradient descent optimization on batch b using model w_k.

inputs: w_k, D_k, E
parameters: b_s
B $\longleftarrow$ split D_k into batches of size b_s
$l(w_k; b) = -\frac{1}{b_s} \sum_x \sum_{c=1}^{C} y_c \log(\bar{y}_c(x|T))$, T = 1
for e in range(E) **do**
  for b in B **do**
    $w_k \longleftarrow w_k - \eta \nabla l(w_k; b)$
  end for
end for
return w_k

stochastic approach in which clients are assigned to hosts uniformly at random is used (Alg. 3).

Algorithm 3 PeerAsignment. Each client c_k is assigned to a foreign host client uniformly at random; K is the number of clients; U is a list of unassigned clients; T is a list of target hosts for each client.

inputs: K
$U \longleftarrow$ client ids from 1 to K
for k in range(K) **do**
  $t \longleftarrow k$
  while $t = k$ **do**
    $t \longleftarrow$ draw sample id from U with uniform distribution
  end while
  remove t from U
  append t to T
end for
return T

4.2.3 On-peer training

After the central server assigns a peer to each client, the client model is sent to the assigned host to perform the on-peer training step. The guest model is trained on the assigned client, using the host's private dataset. Contrary to the local training step, where the loss function used is the NLL loss, during on-peer training, an additional distillation loss term is used to improve training using knowledge

distillation. According to Eq. 2.8, the complete loss function is described in Eq. 4.4, with $\hat{y}_c$ denoting the output of the host model w_h, which is used for distillation as a reference for the guest model.

$$kd(w_k; w_h; x|T) = T^2 \frac{1}{b_s} \sum_x \sum_{c=1}^{C} \hat{y}_c(x|T) \log \bar{y}_c(x|T) \tag{4.2}$$

$$nll(w_k; w_h; x|T) = \frac{1}{b_s} \sum_x \sum_{c=1}^{C} y_c \log(\bar{y}_c(x|T))), T = 1 \tag{4.3}$$

$$l(w_k; w_h; x|T) = -(1-\alpha) * nll(w_k; w_h; x|1) - \alpha * kd(w_k; w_h; x|T) \tag{4.4}$$

There are two terms, scaled by the ratio $\alpha \in [0, 1]$. The first term (Eq. 4.2) is the distillation loss term, described earlier in Sec. 2.8. The second term (Eq. 4.3) is the standard NLL loss term. The composed loss allows the guest model to improve not only thanks to the host's private dataset, but also by distilling the knowledge from the host's model. The tunable parameters of this loss formulation are α and T. Parameter α allows to tune the amount of distillation. With $\alpha = 1.0$, the guest model is trained using only the NLL loss term. With $\alpha = 0.0$, the NLL loss term is ignored and the guest is only trained using knowledge distillation. T is the temperature value used in softmax and can influence the performance of knowledge distillation. The impact of both hyperparameters is evaluated in Sec. 5. Alg. 4 describes the on-peer training procedure.

Algorithm 4 OnPeerTraining. The client model w_k is trained on a private dataset D_h of the host using its model w_h; b_s denotes the batch size; η denotes the learning rate; E_p is the number of on-peer training epochs; $\nabla l(w_k; w_h; b)$ is the gradient of loss function calculated in each gradient descent optimization step on batch b for the guest model w_k using host's model w_h.

inputs: w_k, w_h, D_h, E_p
parameters: b_s, T, α
B $\longleftarrow$ split D_h into batches of size b_s
$l(w_k; w_h; x|T) = -(1-\alpha) * nll(w_k; w_h; x|1) - \alpha * kd(w_k; w_h; x|T)$
for e in range(E_p) **do**
  for b in B **do**
    $w_k \longleftarrow w_k - \eta \nabla l(w_k; w_h; b)$
  end for
end for
return w_k

4.2.4 Model reassignment

After completing the on-peer training step, the improved guest models are sent back and update the models of their respective clients

4.2.5 Model evaluation

After each training step (local and on-peer), the models are evaluated on a common test dataset. As experiments focus on classification tasks on balanced datasets, for each model, the accuracy metric score obtained on the test dataset is calculated [86] (Alg. 5).

Algorithm 5 ModelEvaluation. The mean accuracy metric score is calculated using the ground truth y and the output $\bar{y}$ of the model w_k on the test dataset D_t.

inputs: w_k, D_t
parameters: b_s
$B \longleftarrow$ split D_t into b batches of size b_s
acc $\longleftarrow 0$
for x, y in B **do**
  acc += Accuracy($y(x)$, $\bar{y}(x)$)
end for
return $\frac{acc}{b}$

4.3 Experimental setup

To evaluate the proposed framework, it was applied to three different ML tasks in the robotics, computer vision and NLP domains. Specifically, the framework was used to train federations for grasp prediction (Dex-Net dataset [87]), image classification (MNIST dataset [14], and sentiment analysis (Large Movie Review Dataset [88]. For each task, the participats were divided into 3 groups. Within each group, the clients used neural network models of the same architecture. However, between the groups, the models varied in size (number of parameters); in all experiments, there was a group of small, medium, and large models. With this setup, the heterogeneity of the federation was simulated. In all experiments, the dataset used was divided into training and test subsets. The training subset was divided equally between the clients. On the other hand, all clients shared the same test data set, which was used for evaluation purposes. In addition to showcasing the application of FL to robotics and the flexibility of the proposed system, the following questions were investigated:

- Is the on-peer training step improving the federation's training, compared to the local training only for each client?
- How does the parameter α influence the performance of the models?
- How does the parameter T influence the performance of the models?

To answer the first question, training using the proposed framework, including all the steps described in 4.2 was compared to a situation in which clients were trained only on their private dataset (local-only training), without model exchange in the form of the on-peer training. This mirrored the instance when the clients were not performing federated learning - they were trained in isolation instead. To investigate the influence of α and T, a grid search was performed and the performance of the models was evaluated for each value considered. Furthermore, in the grasp prediction task, the grasp quality convolutional neural network (GQCNN [87]) was trained on the Dex-Net 2.0 dataset using the presented framework, and the models were practically evaluated in a pick-and-place task.

4.3.1 Dex-Net experiments

The presented framework was employed to train an adaptation of grasp quality convolutional neural network (GQ-CNN) for a robotic task, grasp prediction. As a first documented use of FL in industrial application, the models were evaluated with the use of an industrial manipulator. For this purpose, a pick-and-place pipeline was implemented, based on the Dex-Net 2.0 paper [87]. Dexterity Network 2.0 builds on the Dex-Net 1.0 study [89], where 1500 3D objects were analyzed in simulation using grasp wrench space (GWS) analysis [90] for grasp quality metric calculation. Based on their previous work, Mahler et al. created a dataset of 6.7 million depth images with the corresponding grasp success probabilities, which can be used to predict the grasp success. Furthermore, the authors designed a GQ-CNN architecture that can be trained on the aforementioned dataset to perform the grasp prediction task.

Mahler et al. considered parallel-jaw grasps with the grasp axis parallel to the table plane, using the coordinates of the center point of the grasp (x, y, z) and its orientation θ, in a selected coordinate frame. This grasp description is shown in Fig. 4.3.

The dataset consists of depth images of the simulated objects, the grasp height value (the distance between the camera and the grasp center point height), and a binary label denoting the probability of grasp success (1 - grasp successful, 0 - grasp unsuccessful), as shown in Fig. 4.4. To create the Dex-Net 2.0 dataset, each depth image was sampled from the scene by picking the grasp pixel coordinates and grasp orientation. The grasp area was then rotated to align the grasp axis with the horizontal axis of the image, cropped around the center of the grasp, and

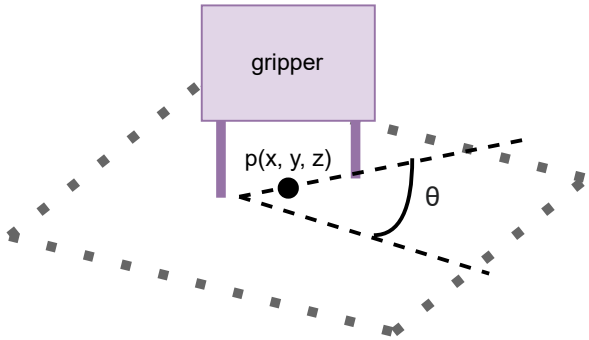


Figure 4.3: In this work, top-down parallel-jaw grasps described by the coordinates of the grasp center point p and a grasp angle θ were considered.

Table 4.1: The group configuration for Dex-Net experiments.

Group name	FC layers size	Number of filters
small	256	16
medium	512	32
large	1024	64

resized to a resolution of 32x32 (Fig. 4.5). GWS analysis allowed for automatic labeling of the sampled grasps.

Instead of a standard GQ-CNN, in the experiments, I used a slightly altered architecture, denoted as alt-GQ-CNN (Fig. 4.6), according to suggestions from [91]. Instead of entering the distance from the center point of the grasp to the camera and the depth image, this distance was subtracted from the pixel values of the corresponding image. In this way, the input consisted of only the image. Furthermore, instead of local response normalization [92], batch normalization [93] was used.

To examine the influence of client cooperation during training, compared to the case where clients only perform solitary training, 6 clients with $\alpha = 1.0$ were first trained in isolation for 70 local training rounds. Then, a federation with the same number of participants was trained using the entire proposed workflow for 35 training rounds (which equals 70 training epochs, 35 in the local training step, and 35 in on-peer training). The client parameters for each group are summarized in Tab. 4.1. In addition to comparing the framework with isolated training, the influence of T and α was investigated using a grid search strategy, with $T \in \{1.0, 1.5, 2.0, 4.0, 7.0, 10.0\}$ and $\alpha \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0\}$. All experiments had the learning rate set to $\eta = 0.001$ and batch size to $b_s = 64$. To evaluate the models, in each experiment we selected a randomly sampled common test

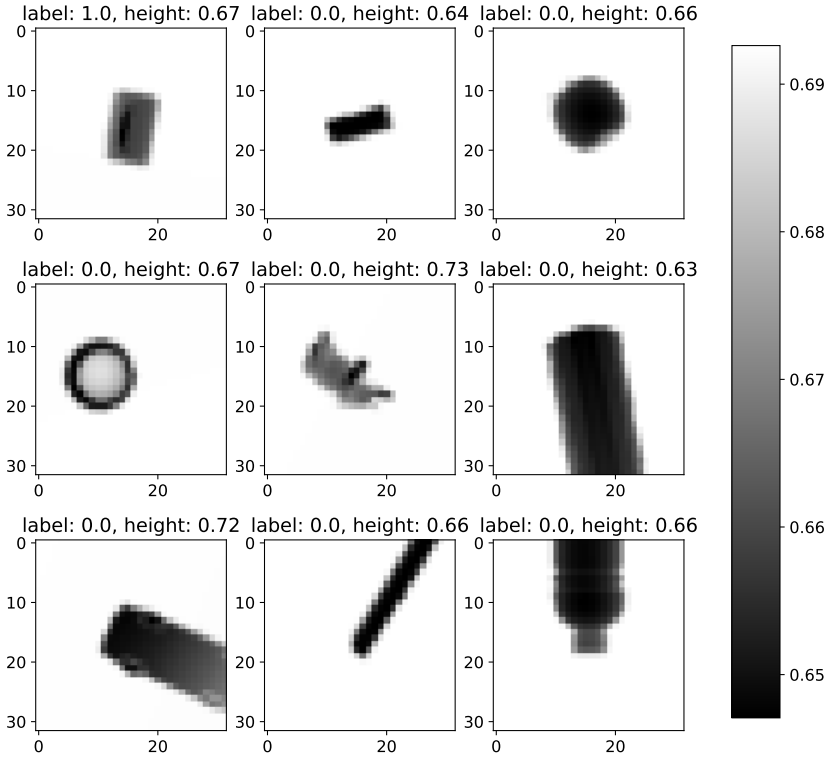


Figure 4.4: Sample images and corresponding heights and labels from the Dex-Net 2.0 dataset. The images were generated from the simulated scenes, so that the grasp center point is at the middle pixel in the image and the grasp axis is aligned with the image horizontal axis.

subset from the original Dex-Net 2.0 dataset, using 0.8:0.2 train-test ratio.

To present how federated learning can be used in an industrial robotics setting, trained models were evaluated in a pick-and-place pipeline, using the UFACTORY xArm 7 robotic arm [94] and Intel RealSense D415 depth camera [95]. The high-level overview of the pipeline is depicted in Fig. 4.7.

The robot was set up in an eye-in-hand configuration, with the camera mounted on the gripper. Before the experiments, the Zhang method [96] was used, to estimate the intrinsic parameters of the camera, and based on the estimates, collected images were undistorted. To perform a hand-eye calibration, the Tsai et al. [97] algorithm was used. The setup is shown in Fig. 4.8.

The pick-and-place pipeline starts by commanding the robot to move to the

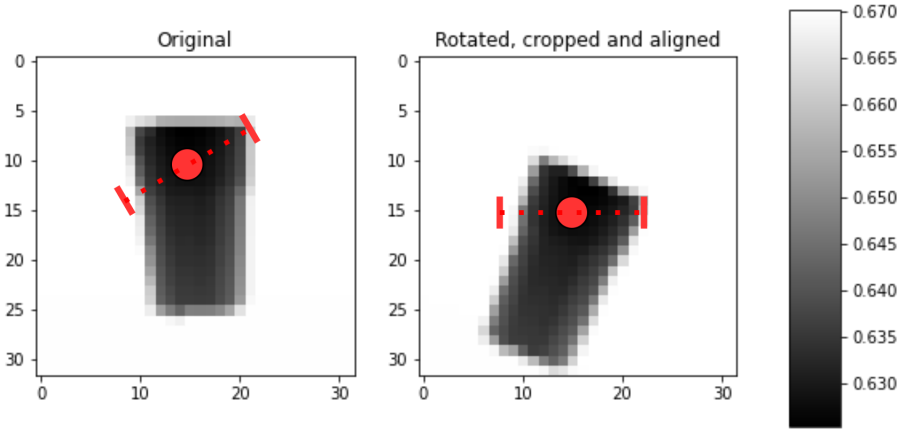


Figure 4.5: To create Dex-Net 2.0 dataset, after sampling a grasp from the scene (including grasp height), the image from the simulated camera was rotated to align the grasp axis with the horizontal axis of the image. Additionally, the image was centered around grasp center point. Before entering the model, the image was normalized. The sampled grasp is visualized in red.

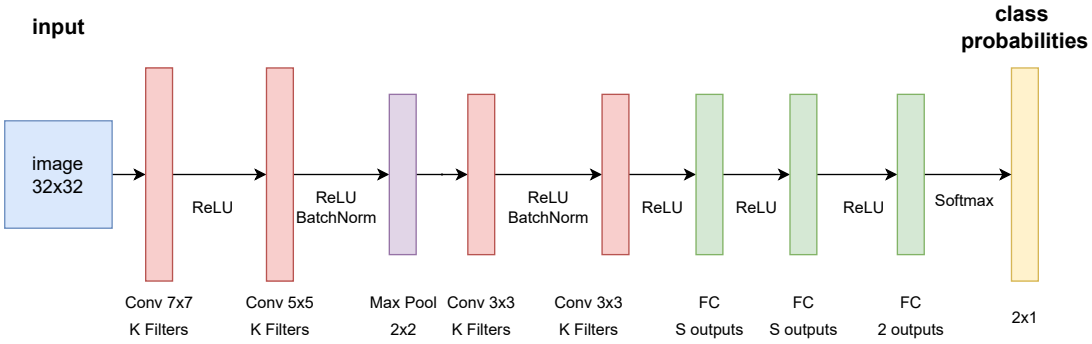


Figure 4.6: Alt-GQ-CNN architecture. The grasp height information is included in the image, by subtracting this value from the depth image pixel values for each sample. The image is then processed by a set of convolutional layers with K filters, which varies between model groups trained in experiments. The image is downsampled with the use of max pooling and a pair of additional convolutional layers. Finally, the output is adjusted using a set of fully connected layers. All convolutional layers use no padding and stride 1. For the second and second-to-last convolutional layer, batch normalization was used.

image acquisition position above a flat workspace. Next, a depth image is acquired from the camera and a grasp sampling strategy similar to that of the Dex-Net 2.0 paper [87], is used to sample a number of grasps for an object placed in a pick region of 0.2x0.2m in the workspace. To implement this sampling strategy, thresholding and Canny edge detection [99] are used to find the edges on the upper surface of the object. Next, two points are sampled uniformly at random from the resulting set. The average position of the point is then considered as the

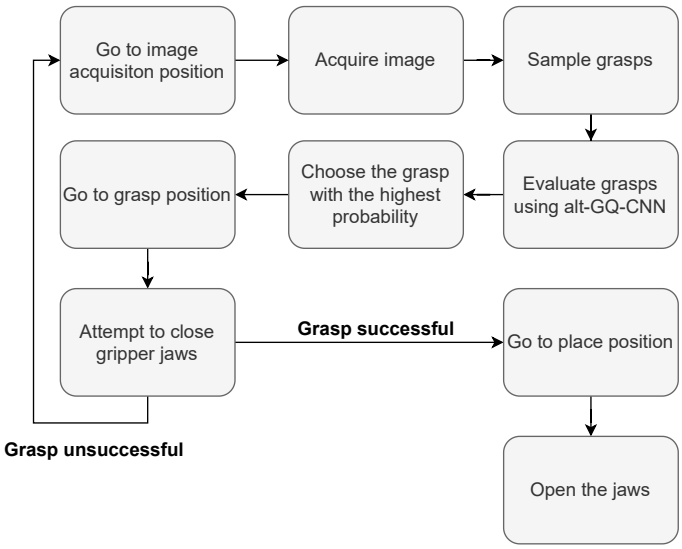


Figure 4.7: The employed pick-and-place pipeline overview. First an image is acquired from the starting position. Next, a number of grasps is sampled from the scene and for each an image is prepared according to Fig. 4.5. The inputs are passed through a trained alt-GQ-CNN model to estimate grasp success for each sample. Then, the grasp with the highest success probability is selected and the robot attempts to place the gripper according to this grasp's position and orientation. If the grasp is successful, the robot is commanded to place the object in the place area. Otherwise, it aborts the operation and goes back to the image acquisition position.

center of grasp. The grasp axis is the line that connects these two points and is used to calculate the angle θ between the grasp axis and the horizontal axis of the image. To obtain the height of the center point of the grasp, a number of values are sampled in the range between the height of the object in the center of the grasp and the height of the base plane of the workspace. Note that in this case, the height of the grasp is considered the difference between the camera height and the height of the center point of the grasp (Fig. 4.9).

After sampling 4096 grasps (512 grasp center points, 8 heights each), for each grasp, an alt-GQ-CNN input is generated, as shown in Fig. 4.5, translating, rotating, and cropping the image, additionally with the grasp height subtracted from the pixel values. The images are then passed through the alt-GQ-CNN model to obtain the success probabilities for each grasp. The grasp with the highest probability of success is selected and its coordinates are used to guide the robot gripper to the grasp location. After a grasp attempt, if it was successful, the object is transported to a location above the target box and dropped.

In robotic experiments, models trained using local training only were compared to those trained using the proposed collaborative training scheme to showcase the improvement over isolated training. For this purpose, two metrics were used: the Success Rate is the percentage of grasps in which the object is successfully placed in

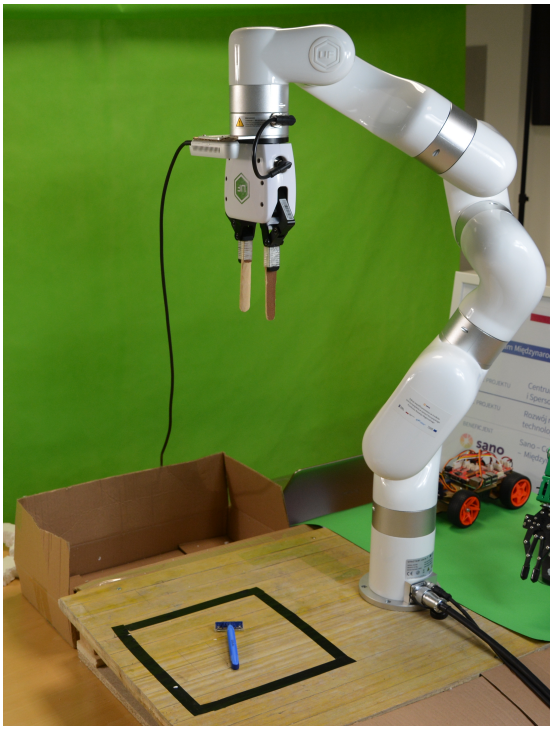


Figure 4.8: The pick-and-place experiments setup. The Intel RealSense D415 RGB-D camera was mounted on the xArm Gripper (eye-in-hand configuration), which had custom fingers to mirror the specification of ABB Yumi gripper [98], used originally in Dex-Net 2.0 experiments. The test objects were placed on the workspace before each grasp attempt. The goal of the pipeline was to pick the object and place it in the box next to the workspace.

the target box. Furthermore, the Robust Grasp Rate metric was computed, which is the percentage of sampled grasps that have a grasp success probability value greater than 0.5. Each model was evaluated on a set of 10 objects (Fig. 4.10) in an uncluttered scene, in 50 trials, five picking attempts for each object. The orientation of the objects was determined at random.

4.3.2 MNIST experiments

To demonstrate the use case of the presented framework outside of robotics, it was evaluated on a computer vision task using the widely known MNIST dataset [14]. The dataset consists of 70000 28x28 grayscale images of handwritten digits (Fig. 4.11), split into training (60000 images) and test datasets (10000 images). The objective of the models trained on this dataset is to classify the digit on the image into one of 10 classes. Before using the images, the pixel values were standardized. The smaller size of this dataset allowed to experiment with a larger number of clients - in each run, a federation of 24 clients was trained, divided into 3 groups,

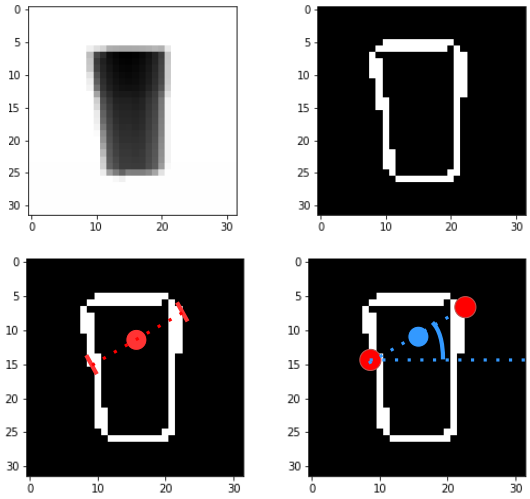


Figure 4.9: To sample grasps, the original depth image (top left) is first thresholded and edges are detected (top right) on the object's top surface. Next, from the edges found, for each grasp two points are sampled (bottom right) and the grasp center point and the grasp axis angle are computed. Finally, the resulting grasp description is passed to the input preparation procedure (Fig. 4.5).

Figure 4.10: Pick-and-place test objects consisted of 6 custom shapes and 4 everyday use objects.

with the configuration of each group summarized in Tab. 4.2. All clients used a fully connected neural network with two hidden layers of size $S \in \{8, 16, 32\}$ (Fig. 4.12). The Rectified Linear Unit (ReLU) activations are used with dropout regularization. In all experiments, the batch size was set to $b_s = 128$ and the learning rate to $\eta = 0.001$. To evaluate the models, we used 10000 test images as a common test dataset. To evaluate the influence of the on-peer training step introduced by the proposed system, a federation was trained for 200 full rounds

Table 4.2: Group configuration for MNIST experiments.

Group name	Hidden layers sizes
small	8
medium	16
large	32

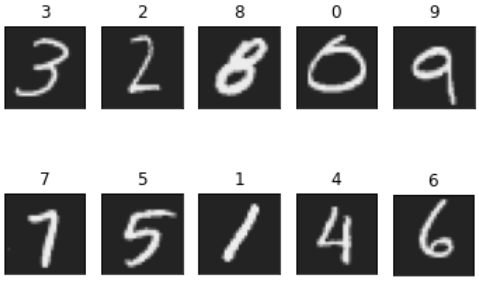


Figure 4.11: Example MNIST data samples. Each sample is a 28x28 grayscale image of a handwritten digit, representing one of 10 classes (indicated above the images). This makes the MNIST dataset suitable as a benchmark for multiclass classification tasks.

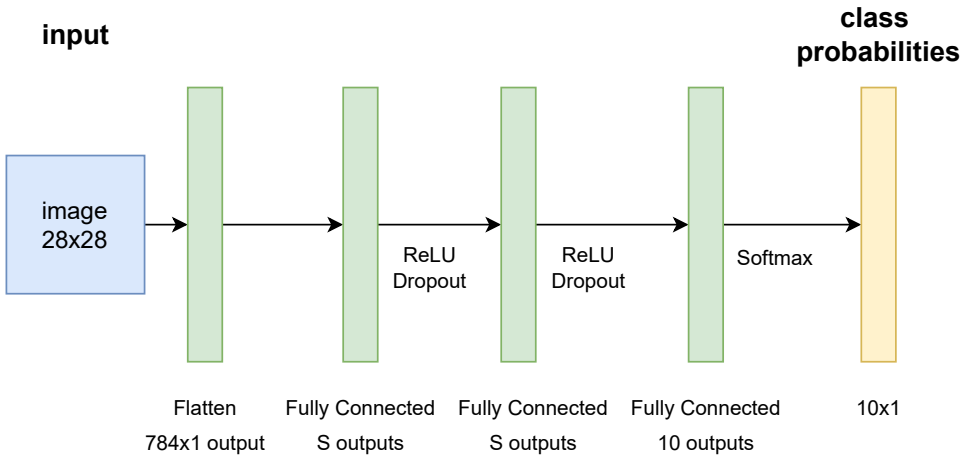


Figure 4.12: Neural network architecture used in the MNIST experiments. For each hidden layer size $S \in \{8, 16, 32\}$ there were 8 representatives in the federation, totalling in 24 clients. The input image is first flattened and then passed through a set of two fully connected layers with ReLu activations and Dropout regularization. The output is a vector of 10 values, scaled using softmax to represent probabilities for each class.

until convergence and I compared the results with the setup, where the on-peer

training step was omitted, and the clients were only conducting local-training. The clients were trained locally for 400 epochs, to equal the amount of training resulting from 200 full rounds in the former scenario (200 full rounds equal to 200 local training rounds and 200 on-peer rounds, giving 400 training epochs overall). In this scenario α was set to 1.0, disregarding the distillation term in the on-peer training.

Furthermore, to evaluate the impact of the distillation procedure and the distillation parameters, a grid search was performed in a separate experiment, training the federation for 200 full rounds in each trial. The hyperparameter values investigated included $\alpha \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0\}$ and $T \in \{1.0, 1.5, 2.0, 4.0, 7.0, 10.0\}$.

4.3.3 IMDB experiments

To show the applicability in an NLP task, sentiment analysis experiments were carried out, using the Large-Scale Movie Review dataset (also known as IMDB dataset), created by Maas et al. [88]. It consists of 50000 highly polar movie reviews that are suitable for binary classification. Contrary to the MNIST dataset, which only requires standarization of the inputs, a couple of data preprocessing steps had to be implemented to pass the reviews into the classifier in a suitable form. First, HTML tags were removed because they were present in the review text. The data samples were then lowercased and tokenized [100], resulting in a list of lowercase tokens for each review. To create a numerical representation for the reviews, word2vec embeddings [101] were chosen. For this purpose, the continuous bag-of-words (CBOW) model was trained on the available corpus to predict the target word from the context. After training, the hidden layer of the CBOW model consisted of embeddings (dense vectors) for each token in the training corpus. Intuitively, training stimulates the model to place tokens that appear in a similar context close to each other in the latent space. This method was used successfully in the NLP domain and has been proven to improve classification performance over standard encoding approaches. The input of the classifier was constructed by stacking embeddings of size 16 for each word in the review. Each embedding vector was also normalized to unit length. As a classifier, a recurrent neural network (RNN) architecture [102] was used, based on long short-term memory (LSTM) layers [34]. LSTM layers address the problem of vanishing gradients, commonly encountered in vanilla RNN layers. The introduction of forget and input gates allows LSTM to improve modeling long-range dependencies in input sequences. The architecture of the models used for sentiment prediction is shown in Fig. 4.13. Similarly to the MNIST experiments described in Sec. 4.3.2, a federation of 24 clients with $\alpha = 1.0$ was trained for 800 full training rounds, and model performance is compared with a group of isolated models trained using local-only training, for 1600 epochs. Again, a different model architecture was used for each group of clients,

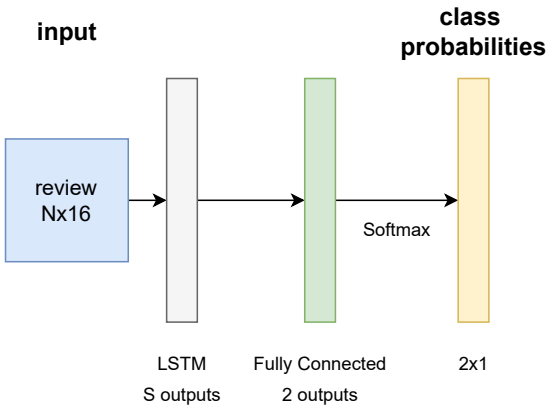


Figure 4.13: The architecture used in sentiment analysis experiments. It consists of an LSTM layer with hidden size $S \in \{4, 8, 16\}$ followed by a fully connected layer is used. The input to the network is a stack of N word2vec embeddings with dimensionality 16, one for each word in the review.

Table 4.3: Group configuration for IMDB experiments.

Group name	Hidden layer size
small	4
medium	8
large	16

using the hidden layer size of the LSTM $S \in \{4, 8, 16\}$. The batch size was set to $b_s = 1024$ and the learning rate was $\eta = 0.01$ (Tab. 4.3). As a common test dataset, we used the 25000 reviews originally separated for this purpose by the dataset creators. In grid search experiments, the influence of $\alpha \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 1.0\}$ and $T \in \{1.0, 1.5, 2.0, 4.0, 7.0, 10.0\}$ was investigated.

4.4 Implementation details

All experiments code was written in Python (version 3.8) [103]. The distributed training framework was implemented using the tools provided by the Ray library (version 1.11.0) [104]. Deep learning models were implemented in Pytorch (version 1.11.0) [105] using the Pytorch Lightning wrapper (version 1.5.10) [106], to reduce the amount of boilerplate code and handle the logging of the training procedures. For image pre-processing, the OpenCV library (version 4.5.5.64) [107] was used. In the sentiment analysis task, the text was pre-processed and word embeddings were obtained using the Gensim module (version 4.1.2) [108].

Training during the prototyping phase was executed on a local workstation with a NVIDIA 3080TI GPU, an Intel Core i9-11900K processor, and 64 GB of RAM. For the evaluation phase, training for all experiments was performed on the Prometheus High Performance Computing Cluster [109] within the PL-Grid initiative [110], with access granted by the Academic Computer Centre Cyfronet AGH [111]. For each experiment, a single node with Intel Xeon Gold 5220 processor (24 cores) and 8 NVIDIA V100 SXM2 GPUs with 32 GBs of VRAM was used.

The Github repository containing all the code used to implement the framework and the experiments is available at [112] (please request access from the author by email).

Chapter 5

Results

In Sec. 4.3, the setup for robotic, NLP and computer vision experiments was described. In the following section, the results obtained with that experimental configuration are presented, including the results from the pick-and-place pipeline and federated learning on each dataset. The average score values for all model groups are summarized in tables. In addition, insightful plots for each experiment are provided. Note that the scores for each training step showcased in the graphs were processed with a rolling window of size 5 and mean aggregation, to improve the visibility of the trends observed in the collected data.

5.1 Dex-Net experiments

5.1.1 Pick-and-place evaluation

The experiments carried out on the robot compared the models trained in isolation (local-only strategy) with the FL training scheme presented in a real-world pick-and-place task. Using the federated training framework to train alt-GQ-CNN models of various sizes on the Dex-Net 2.0 dataset, the proposed pipeline obtained a grasp success rate of 0.72 for 10 test objects in 50 trials. The results of these experiments are provided in Tab. 5.1. Most importantly, they indicate that alt-GQ-CNN models trained in isolation obtained grasp success rates lower by 0.12 for all model groups, compared to models trained collaboratively. Robust grasp rates are also generally higher for isolated models, the difference being 0.0976 for small models and 0.007 for large models. This trend does not hold for medium models - in this group, the models trained collaboratively had a robust grasp rate higher by 0.0072.

Table 5.1: The results obtained during evaluation of the alt-GQ-CNN models trained using different strategies (local-only vs. full proposed federated training) on a pick-and-place task for six models, one representative for each group and strategy. The success rate values are listed with 95% confidence intervals. The best results for each group are highlighted in bold. All models were tested by making 50 trials, 5 for each test object.

Training strategy	FC size	Number of filters	Success Rate	Robust Grasp Rate
Local-only	256	16	0.6 ± 0.069	0.0106
Full	256	16	**0.72**$\pm$0.063	0.0084
Local-only	512	32	0.58 ± 0.070	0.0031
Full	512	32	**0.70**$\pm$0.065	0.0103
Local-only	1024	64	0.6 ± 0.069	0.0011
Full	1024	64	**0.72**$\pm$0.064	0.0004

Table 5.2: Mean accuracy values for all alt-GQ-CNN model groups, showcasing the comparison between local-only model training vs. the proposed training framework, with an on-peer training step included, at training step 68, obtained on the test subset of Dex-Net 2.0 data. The best results for each group of models are highlighted in bold.

Training strategy	FC layers size	Number of filters	Accuracy	Std
Local-only	256	16	0.950	0.004
Full	256	16	**0.959**	0.003
Local-only	512	32	0.952	0.004
Full	512	32	**0.966**	0.009
Local-only	1024	64	0.934	0.001
Full	1024	64	**0.966**	0.016

5.1.2 Full vs local-only training

At the end of the training procedure, the federation of clients trained on the Dex-Net 2.0 dataset using the proposed framework obtained higher accuracy scores on test Dex-Net 2.0 dataset, compared to when models were trained in isolation, only on their respective local datasets. This was the case for all model groups. For small alt-GQ-CNN models, the average result of collaborative training at step 68 was 0.959 (local-only - 0.950). For medium models, the result was 0.966 (local-only 0.952). Finally, for large models, it was 0.966 (local-only 0.934). These results are visualized in Fig. 5.1 and summarized in Tab. 5.2.

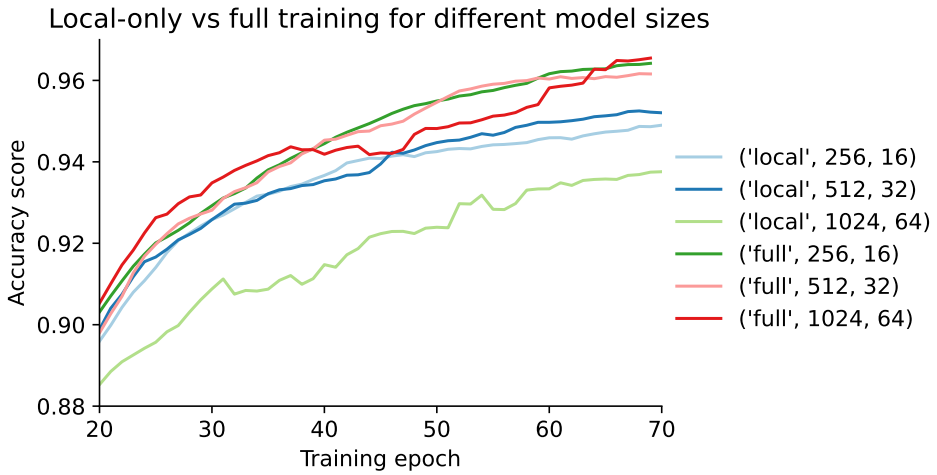


Figure 5.1: Comparison between accuracy scores for local-only ('local') and the full proposed training framework ('full') obtained on the test subset of Dex-Net 2.0 data, for different alt-GQ-CNN model groups (small - 256 neurons in the FC layers and 16 convolutional filters, medium - 512 neurons and 32 filters and large - 1024 neurons and 64 filters.

5.1.3 Influence of the α parameter

For small models, the tuning of α parameter influenced the obtained predictive performance in the following way: for small models group, the $\alpha \in \{0.4, 0.5\}$ provided the best performing clients (Fig. 5.2). For medium models $\alpha = 0.5$ produced the best results on average, although experiments with $\alpha = 0.4$ and $\alpha = 0.3$ acquired similar performance (Fig. 5.3). For large models $\alpha = 0.5$ was superior over other values of this parameter, as shown in Fig. 5.4. The results of this experiment are summarized in Tab. 5.3.

5.1.4 Influence of the T parameter

The influence of the temperature parameter for each model group at the finish of the training is summarized in Tab. 5.4. For small models (Fig. 5.5), the best value of T was 1.0, indicating the beneficial influence of the hard-softmax probabilities ratio. For the medium and large model groups, two peak values for T were observed: for the former, they were 1.0 and 2.0, and for the latter, the values were 1.0 and 4.0, although due to very similar performance, it is hard to visualize the results clearly (Fig. 5.6 and Fig. 5.7).

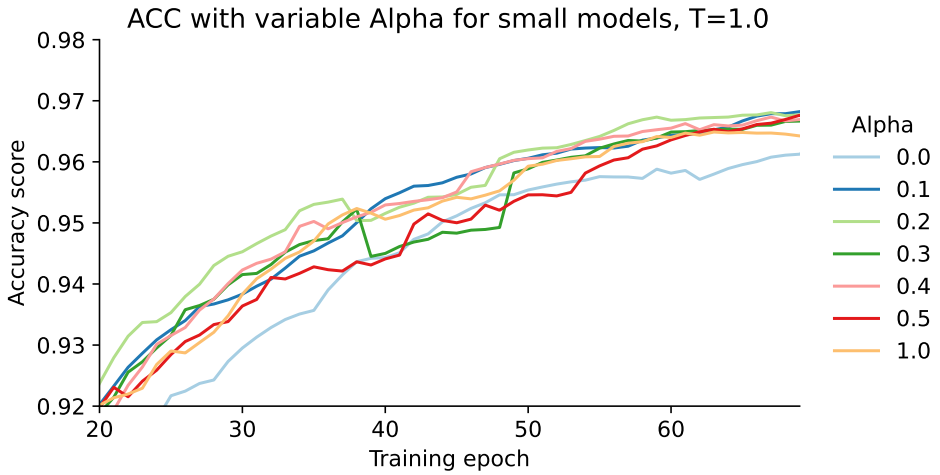


Figure 5.2: Mean accuracy scores comparison for small alt-GQ-CNN models group with $T = 1.0$, using variable α values.

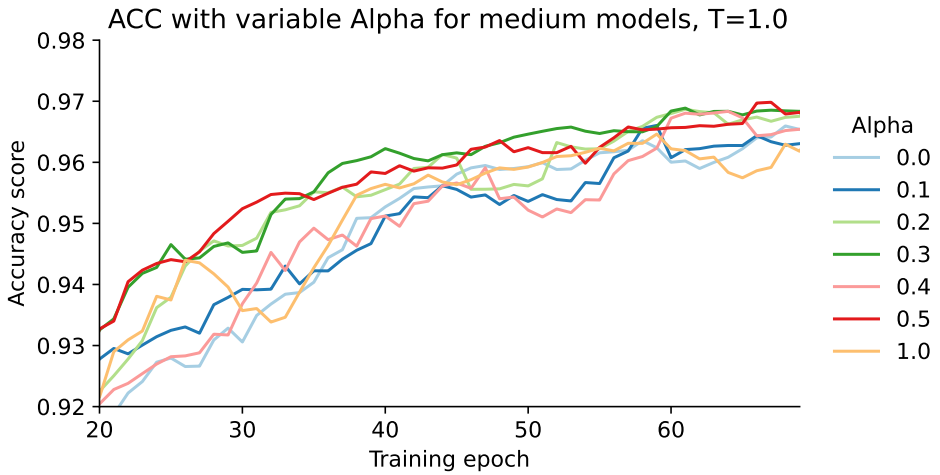


Figure 5.3: Mean accuracy scores comparison for medium alt-GQ-CNN models group with $T = 1.0$, using variable α values.

5.2 MNIST experiments

5.2.1 Full vs local-only training

The comparison between models trained using only local training and models trained using the full framework is shown in Fig. 5.8. In these experiments, the results are proportional to model size; best results are obtained for the largest models, regardless of the training strategy used. Improvement of training in a federation, compared to when models are isolated, can be as high as 0.049 in the

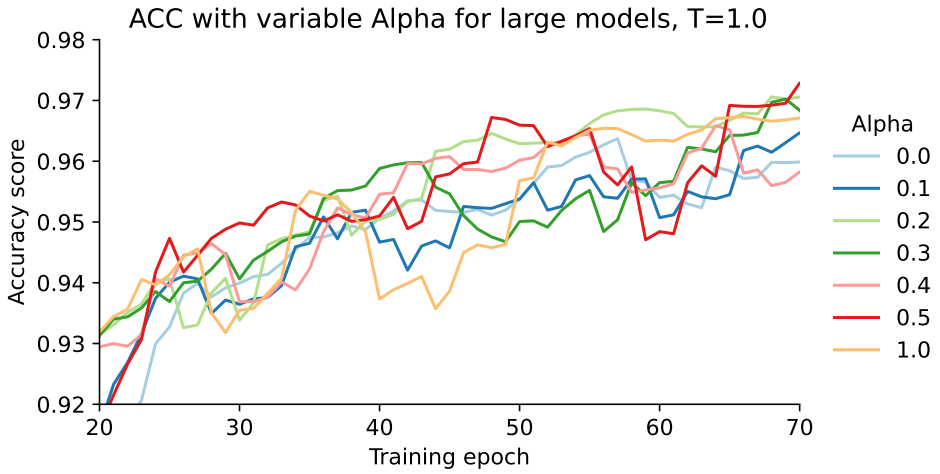


Figure 5.4: Mean accuracy scores comparison for large alt-GQ-CNN models group with $T = 1.0$, using variable α values.

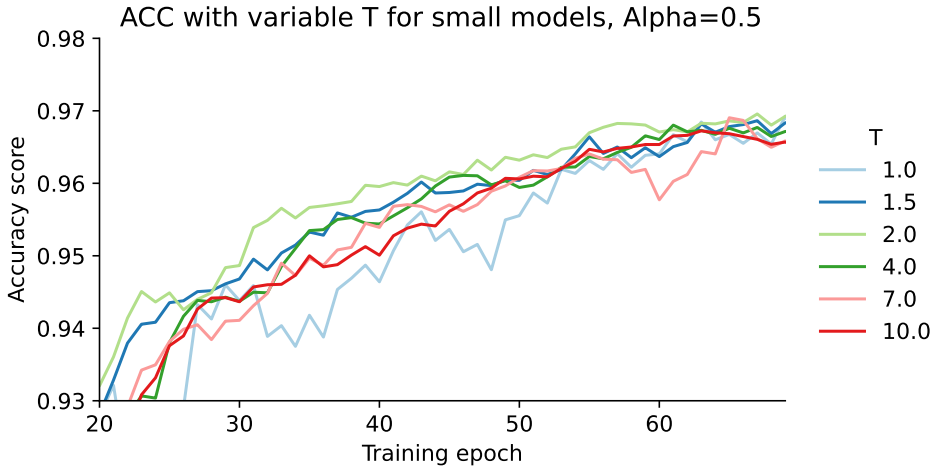


Figure 5.5: Mean accuracy scores comparison for small alt-GQ-CNN models group with $\alpha = 0.5$, using variable T values.

accuracy score for the small model group at step 200. The mean accuracy scores for step 200 for all models are summarized in Tab. 5.5.

5.2.2 Influence of the α parameter

To evaluate the influence of the α parameter on the accuracy for the models trained using the proposed framework, the scores for each group of models with T set to 1.0 and variable α was investigated (Fig. 5.9, Fig. 5.10, and Fig. 5.11). The results show that for the smallest models, the best mean accuracy scores were obtained

Table 5.3: Mean accuracy scores for all alt-GQ-CNN model groups with $T = 1.0$ and variable value of α in step 1600. The best configuration in each group has the accuracy score highlighted in bold.

α	FC layers size	Number of filters	Accuracy	Std
0.0	256	16	0.966	0.005
0.1	256	16	0.973	0.001
0.2	256	16	0.972	0.004
0.3	256	16	0.970	0.002
0.4	256	16	**0.974**	0.001
0.5	256	16	**0.974**	0.001
1.0	256	16	0.964	0.001
0.0	512	32	0.967	0.004
0.1	512	32	0.972	0.001
0.2	512	32	0.971	0.001
0.3	512	32	0.972	0.003
0.4	512	32	0.974	0.003
0.5	512	32	**0.975**	0.002
1.0	512	32	0.965	0.012
0.0	1024	64	0.960	0.012
0.1	1024	64	0.966	0.004
0.2	1024	64	0.968	0.005
0.3	1024	64	0.972	0.001
0.4	1024	64	0.972	0.004
0.5	1024	64	**0.976**	0.001
1.0	1024	64	0.972	0.003

when using α equal to 0.3. The second and third best α values were 0.5 and 0.2. With α equal to 1.0, when the on-peer training does not use knowledge distillation, only the fourth best result was obtained. A significant decrease in performance is observed for $\alpha \in \{0.0, 0.1\}$. Similarly to the small model group, for the large and medium model groups, the lowest performance scores were obtained using $\alpha \in \{0.0, 0.1\}$. However, in these two groups, the best performance was observed by a large margin when $\alpha = 1.0$, with no KD used. When KD was used, the best values of α were 0.5, 0.4, 0.3 and 0.2, in this order from the best score to the worst score. The complete results obtained for each model group and the values of α for step 380 are presented in Tab. 5.6.

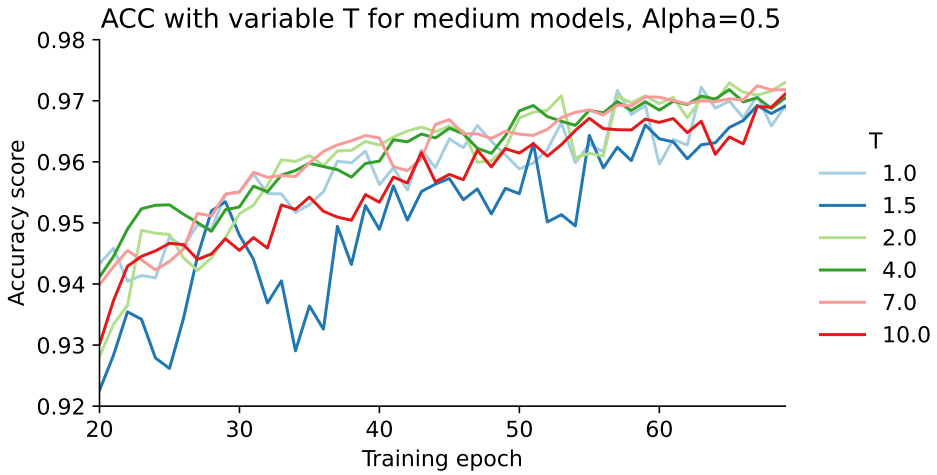


Figure 5.6: Mean accuracy scores comparison for medium alt-GQ-CNN models group with $\alpha = 0.5$, using variable T values.

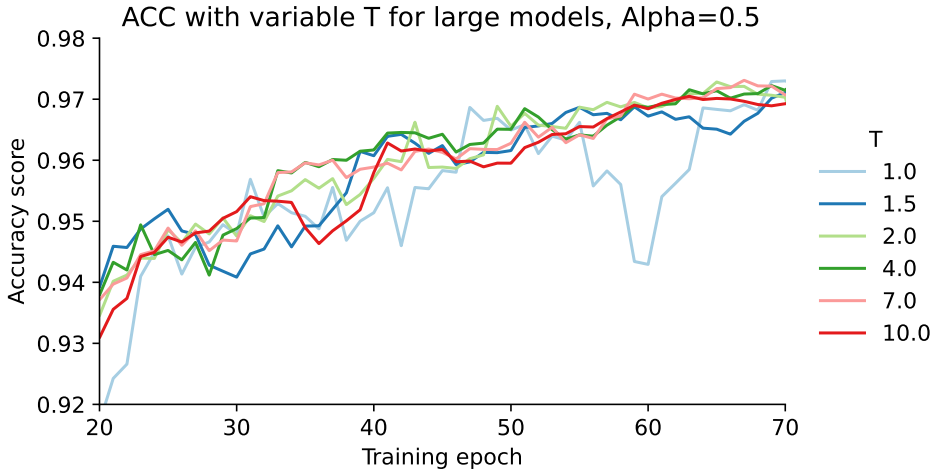


Figure 5.7: Mean accuracy scores comparison for large alt-GQ-CNN models group with $\alpha = 0.5$, using variable T values.

5.2.3 Influence of the T parameter

The comparison of accuracy scores obtained for different T values are shown on Fig. 5.12, Fig. 5.13, and Fig. 5.14, for $\alpha = 0.5$. For small models, the best value T was equal to 2.0. The worst results were obtained using a high-temperature value of 7.0. All other T values resulted in similar performance. Also, for medium and large models, $T = 7.0$ was the worst parameter configuration. $T = 10.0$ gave a similarly poor performance. For large models, the best temperature value turned out to be equal to 1.0, with 2.0 and 1.5 performing similarly. For medium-sized

Table 5.4: Mean accuracy scores for all alt-GQ-CNN model groups with $\alpha = 0.5$ and variable value of T at step 70. The best configuration in each group has the accuracy score highlighted in bold.

T	FC layers size	Number of filters	Accuracy	Std
1.0	256	16	**0.974**	0.001
1.5	256	16	0.972	0.003
2.0	256	16	0.973	0.001
4.0	256	16	0.970	0.001
7.0	256	16	0.971	0.001
10.0	256	16	0.967	0.002
1.0	512	32	**0.975**	0.002
1.5	512	32	0.974	0.001
2.0	512	32	**0.975**	0.003
4.0	512	32	0.971	0.005
7.0	512	32	0.972	0.003
10.0	512	32	0.973	0.001
1.0	1024	64	**0.976**	0.001
1.5	1024	64	0.975	0.001
2.0	1024	64	0.972	0.003
4.0	1024	64	**0.976**	0.002
7.0	1024	64	0.973	0.003
10.0	1024	64	0.968	0.005

Table 5.5: Mean accuracy values for all MNIST model groups, showcasing the comparison between local-only model training vs. full proposed training framework, with an on-peer training step included, at step 200, obtained on the test dataset. The best results for each model group are highlighted in bold.

Training strategy	Hidden layer size	Accuracy	Std
Local-only	8	0.825	0.021
Full	8	**0.874**	0.011
Local-only	16	0.884	0.008
Full	16	**0.921**	0.003
Local-only	32	0.909	0.003
Full	32	**0.945**	0.003

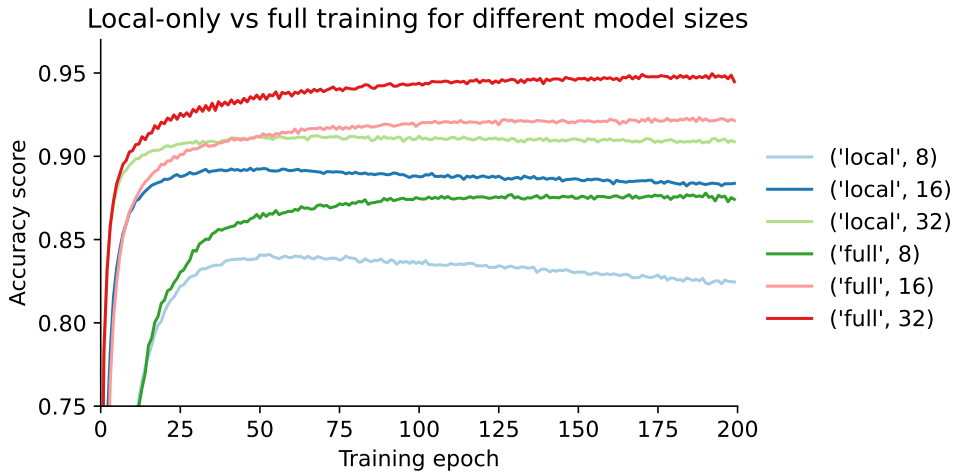


Figure 5.8: Comparison between accuracy scores for local-only ('local') and the full training framework ('full') for different MNIST model groups (with 8, 16 or 32 neurons in the hidden layer), obtained on the test dataset.

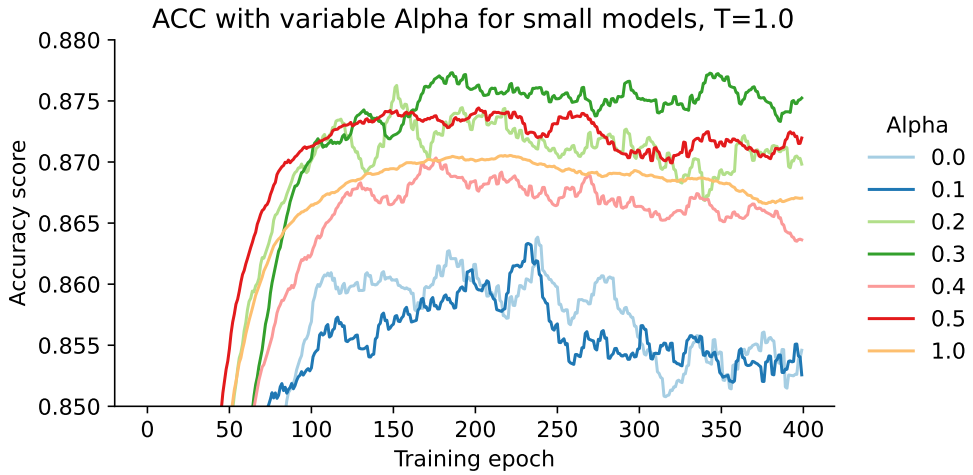


Figure 5.9: Mean accuracy scores comparison for small MNIST models group with $T = 1.0$, using variable α values.

models, $T = 1.5$ allowed the models to obtain the best mean accuracy at the end of training, with $T \in \{1.0, 2.0, 4.0\}$ providing medicore scores. The results show that the optimal value of T can improve the performance by up to 0.019 in terms of the accuracy score, for some model groups (small models in Tab. 5.7, at step 380.)

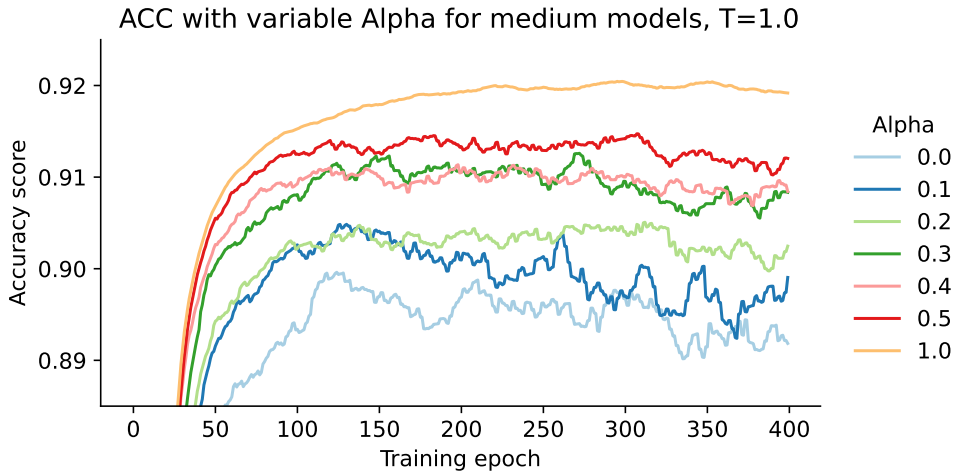


Figure 5.10: Mean accuracy scores comparison for medium-size MNIST models group with $T = 1.0$, using variable α values.

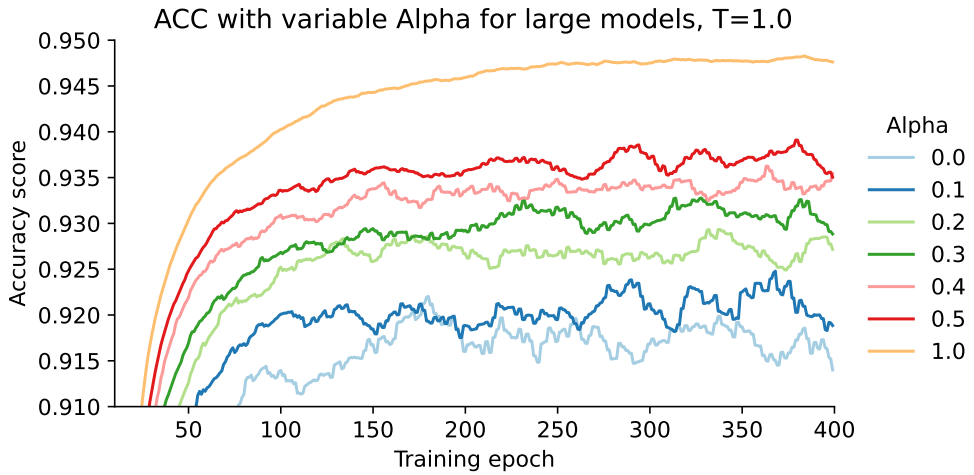


Figure 5.11: Mean accuracy scores comparison for large MNIST models group with $T = 1.0$, using variable α values.

5.3 IMDB experiments

5.3.1 Full vs local-only training

In this investigation, an improvement of 0.123 was observed in the accuracy score for the small model group, 0.185 for medium models, and 0.188 for large models when using the proposed training framework, compared to local-only training (Tab. 5.8, for step 1600). This result is also visualized in Fig. 5.15.

Table 5.6: Mean accuracy scores for all MNIST model groups with $T = 1.0$ and variable value of α at step 380. The best configuration in each group has the accuracy score highlighted in bold.

α	Hidden layer size	Accuracy	Std
0.0	8	0.843	0.035
0.1	8	0.832	0.047
0.2	8	0.854	0.032
0.3	8	**0.874**	0.016
0.4	8	0.849	0.033
0.5	8	0.871	0.014
1.0	8	0.865	0.011
0.0	16	0.861	0.062
0.1	16	0.909	0.007
0.2	16	0.897	0.029
0.3	16	0.893	0.032
0.4	16	0.896	0.023
0.5	16	0.894	0.026
1.0	16	**0.917**	0.007
0.0	32	0.889	0.062
0.1	32	0.883	0.047
0.2	32	0.907	0.039
0.3	32	0.918	0.024
0.4	32	0.932	0.026
0.5	32	0.828	0.018
1.0	32	**0.946**	0.002

5.3.2 Influence of the α parameter

When considering variable α parameter influence on the predictive performance, the results show, that for small models, the highest mean accuracy scores were obtained with α set to 1.0 (Fig. 5.16). For this group of models, the lower the α value, the worse the accuracy obtained at the end of the training. This is also the case for the medium model group; however, the disparities in scores obtained with $\alpha = 1.0$ and $\alpha \in \{0.3, 0.4\}$ are smaller than for the small model group (Fig. 5.17). For large models, this trend does not hold; the best mean accuracy within this group was obtained with $\alpha = 0.5$ (Fig. 5.18). Also, for large models, the difference between the best α values and the worst values was the largest: the improvement in the score for $\alpha = 0.5$ (best value) compared to $\alpha = 0.0$ (worst value) was 0.086. The results at the end of the training (step 1600) are shown in Tab. 5.9.

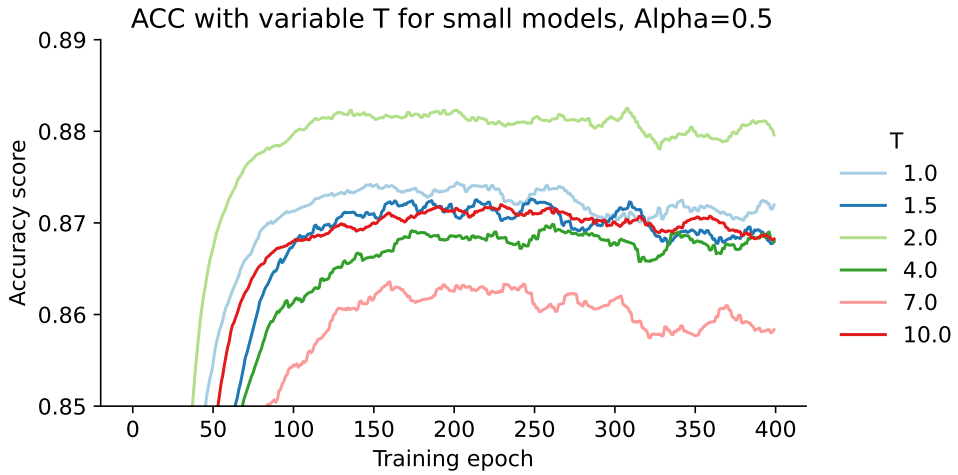


Figure 5.12: Mean accuracy scores comparison for small MNIST models group with $\alpha = 0.5$, using variable T values.

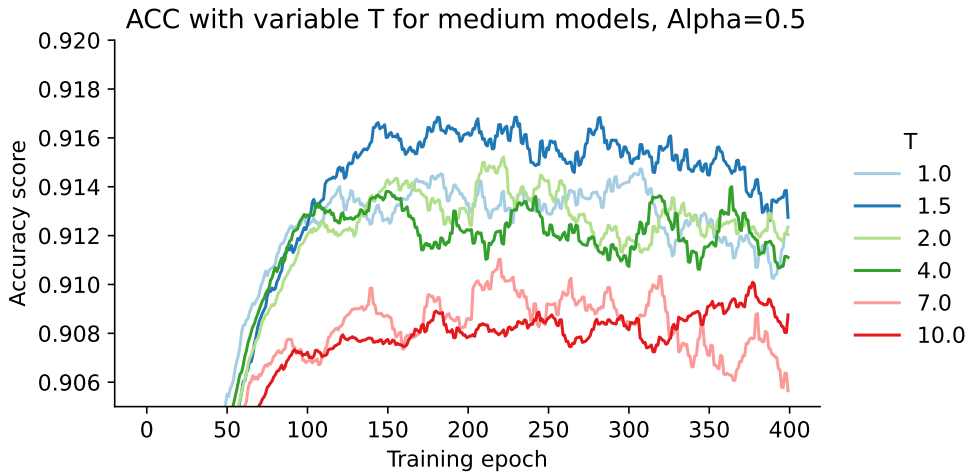


Figure 5.13: Mean accuracy scores comparison for medium MNIST models group with $\alpha = 0.5$, using variable T values.

5.3.3 Influence of the T parameter

The study of the influence of the T parameter value for the IMDB dataset shows, that the small models improved their predictive performance with smaller T values - the best result at the training finish was obtained with $T = 2.0$ (Fig. 5.19). The model group with the best average performance, containing medium-size models, provided the best performance with the 0.759 accuracy score at step 1600 using $T = 7.0$, as shown in Fig. 5.20. For the large model group, the best average scores were obtained for $T = 1.0$ (Fig. 5.21). The complete results for this investigation at

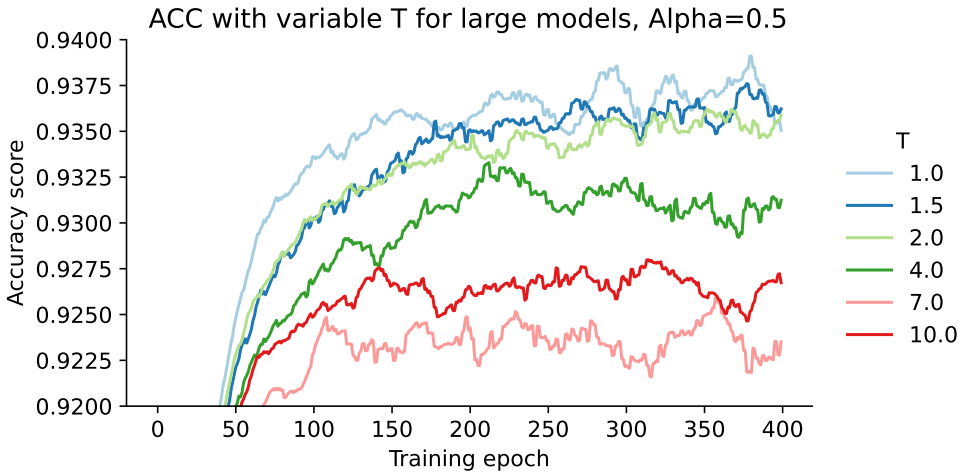


Figure 5.14: Mean accuracy scores comparison for large MNIST models group with $\alpha = 0.5$, using variable T values.

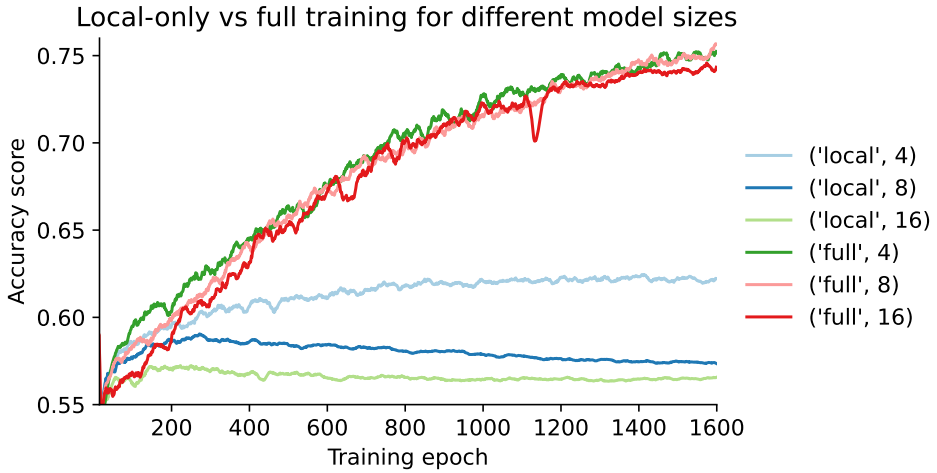


Figure 5.15: Comparison between accuracy scores for local-only ('local') and the full proposed training framework ('full') for different IMDB model groups (with 4, 8 or 16 neurons in the hidden layer of the LSTM architecture used), obtained on the test dataset.

the last training step (step 1600) are shown in Tab. 5.10

Table 5.7: Mean accuracy scores for all MNIST model groups with $\alpha = 0.5$ and variable value of T at step 380. The best configuration in each group has the accuracy score highlighted in bold.

T	Hidden layer size	Accuracy	Std
1.0	8	0.876	0.012
1.5	8	0.873	0.008
2.0	8	**0.885**	0.006
4.0	8	0.871	0.023
7.0	8	0.866	0.019
10.0	8	0.866	0.01
1.0	16	0.919	0.006
1.5	16	**0.923**	0.005
2.0	16	0.919	0.002
4.0	16	0.918	0.002
7.0	16	0.915	0.003
10.0	16	0.912	0.005
1.0	32	**0.945**	0.002
1.5	32	0.944	0.002
2.0	32	0.942	0.003
4.0	32	0.938	0.003
7.0	32	0.932	0.004
10.0	32	0.932	0.003

Table 5.8: Mean accuracy values for all IMDB model groups, showcasing the comparison between local-only model training vs. the full proposed training framework, with an on-peer training step included, at step 1600, obtained on the test datase. The best results for each group of models are highlighted in bold.

Training strategy	Hidden layer size	Accuracy	Std
Local-only	4	0.620	0.032
Full	4	**0.743**	0.035
Local-only	8	0.571	0.015
Full	8	**0.756**	0.012
Local-only	16	0.564	0.019
Full	16	**0.742**	0.020

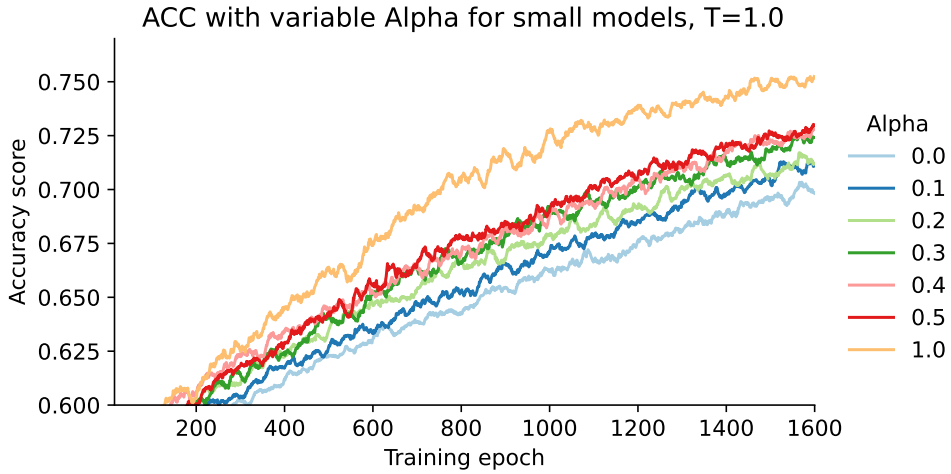


Figure 5.16: Mean accuracy scores comparison for small IMDB models group with $T = 1.0$, using variable α values.

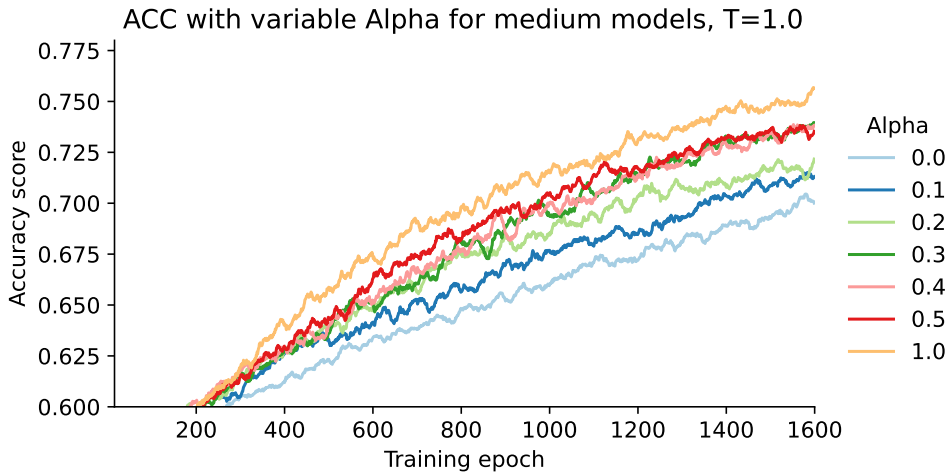


Figure 5.17: Mean accuracy scores comparison for medium IMDB models group with $T = 1.0$, using variable α values.

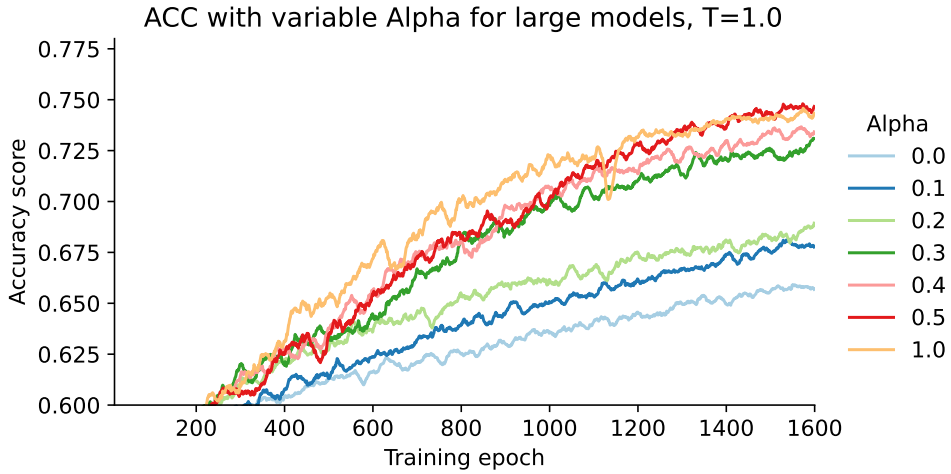


Figure 5.18: Mean accuracy scores comparison for large IMDB models group with $T = 1.0$, using variable α values.

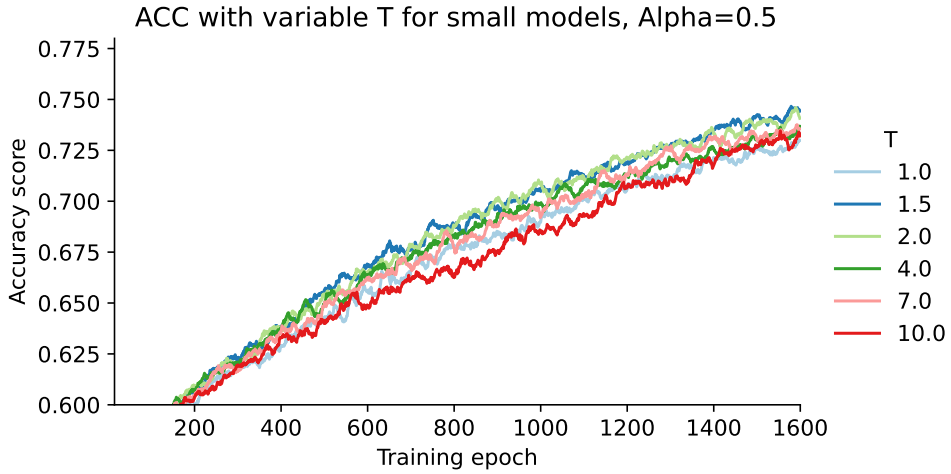


Figure 5.19: Mean accuracy scores comparison for small IMDB models group with $\alpha = 0.5$, using variable T values.

Table 5.9: Mean accuracy scores for all IMDB model groups with $T = 1.0$ and variable value of α at step 1600. The best configuration in each group has the accuracy score highlighted in bold.

α	Hidden layer size	Accuracy	Std
0.0	4	0.712	0.021
0.1	4	0.710	0.022
0.2	4	0.710	0.033
0.3	4	0.716	0.028
0.4	4	0.701	0.045
0.5	4	0.737	0.030
1.0	4	**0.742**	0.035
0.0	8	0.705	0.017
0.1	8	0.713	0.029
0.2	8	0.730	0.023
0.3	8	0.748	0.021
0.4	8	0.752	0.030
0.5	8	0.740	0.034
1.0	8	**0.756**	0.012
0.0	16	0.664	0.018
0.1	16	0.681	0.023
0.2	16	0.690	0.026
0.3	16	0.737	0.019
0.4	16	0.739	0.0162
0.5	16	**0.750**	0.014
1.0	16	0.742	0.021

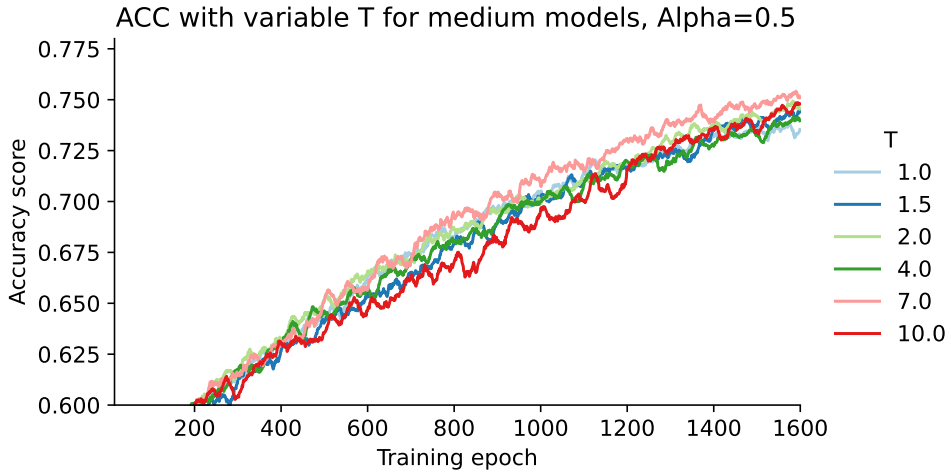


Figure 5.20: Mean accuracy scores comparison for medium IMDB models group with $\alpha = 0.5$, using variable T values.

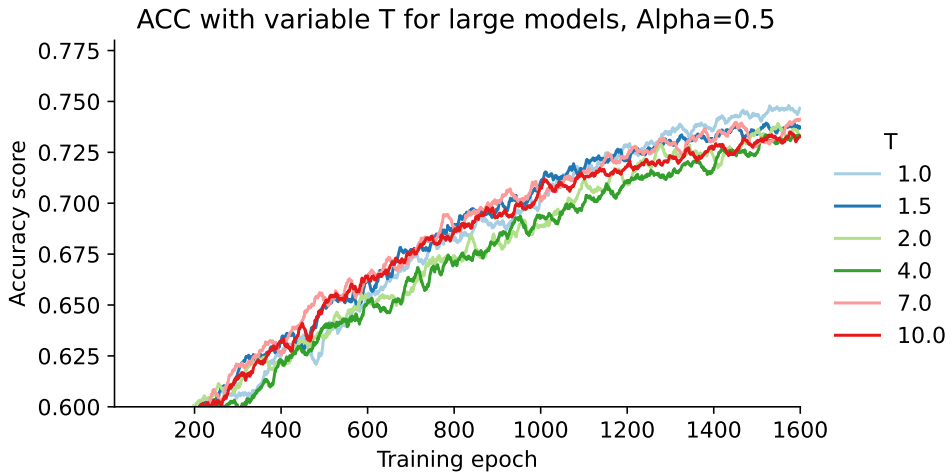


Figure 5.21: Mean accuracy scores comparison for large IMDB models group with $\alpha = 0.5$, using variable T values.

Table 5.10: Mean accuracy scores for all IMDB model groups with $\alpha = 0.5$ and variable value of T at step 1600. The best configuration in each group has the accuracy score highlighted in bold.

T	Hidden layer size	Accuracy	Std
1.0	4	0.737	0.003
1.5	4	0.750	0.003
2.0	4	**0.753**	0.025
4.0	4	0.744	0.023
7.0	4	0.746	0.025
10.0	4	0.721	0.029
1.0	8	0.740	0.034
1.5	8	0.750	0.021
2.0	8	0.745	0.025
4.0	8	0.723	0.046
7.0	8	**0.759**	0.014
10.0	8	0.721	0.029
1.0	16	**0.750**	0.014
1.5	16	0.733	0.024
2.0	16	0.735	0.037
4.0	16	0.736	0.022
7.0	16	0.731	0.038
10.0	16	0.723	0.037

Chapter 6

Discussion

6.1 System requirements fulfillment

The proposed framework complied with the technical requirements outlined in Ch. 3, as summarized in Tab. 6.1. In terms of its main purpose, this collaborative training system allows a federation of clients with heterogeneous architectures to be trained on a common machine learning task without any data transfer, which was the main requirement. The proposed objective function used during on-peer training step allowed the incorporation of knowledge distillation, which proved to boost performance for some model groups. Moreover, by conducting guest models training directly on the clients, it alleviated the need to use an auxiliary dataset for this purpose, which distinguished this approach from the existing heterogeneous solutions. In terms of implementation, the framework also meets the requirements. A general interface was used to set up the training: it allows for easy integration with any dataset and architecture suitable for knowledge distillation, which was showcased in three different ML tasks and architectures used in this study. In addition, the distillation parameters can be fully customized for each client, which expands the flexibility of the implementation.

6.2 Robotics applicability

I expand the state-of-the-art in FL for robotics, employing the proposed framework to train an alt-GQ-CNN model designed for grasp prediction. Models trained collaboratively in a federation on the Dex-Net 2.0 dataset were used to create a real-world pick-and-place pipeline, successfully deployed on a robotic arm. A success rate of 0.72 was obtained for the best models, which is a lower result than what was achieved in the original Dex-Net 2.0 article. I believe that this inferior result can be attributed to the available hardware. The original study was conducted using an ABB Yumi robot and a Primesense Carmine camera. I was unable to obtain

Table 6.1: The proposed system met all the technical requirements considered in Ch. 3, listed in this table.

Technical requirement	Fulfilled
Allows distributed training with no data transfer	Yes
Allows heterogeneous models training	Yes
Does not require additional public datasets	Yes
Allows straightforward model customization	Yes
Allows straightforward dataset customization	Yes
Allows straightforward hyperparameters customization	Yes

the original setup; instead, a different manipulator, UFACTORY xArm7 (crucially, with different gripper fingers) was used, as well as a different camera - RealSense D415. This may have played a major role in the inability to reproduce the original results, as the Dex-Net 2.0 dataset was prepared using the geometrical properties of this gripper and camera combination during dataset generation. Moreover, I have used a different grasp sampling strategy. It sampled grasps uniformly at random, whereas in the original article the best results were obtained by resampling grasps around the most probable grasp points (which was referred to as the cross-entropy sampling policy). This could reduce the number of high-quality grasps that the algorithm presented was able to sample and, in effect, reduce the overall performance.

With that said, I believe that the absolute values obtained by the pipeline compared to the original article are not the main point of interest. My goal was to show that taking into account all potential benefits of heterogeneous FL in robotics, which were outlined in Sec. 1, I can propose a framework that is suitable for training client federations on a robotic ML task and then present a practical application of such framework. To this end, I have contributed successfully. The influence of the FL training system presented on performance in this task was investigated, compared to the situation in which clients are trained in isolation on their private datasets. The results show the major advantage of the proposed training strategy: Using this framework, the success rate in the pick-and-place pipeline increased by 0.12 over isolated clients (Tab. 5.1). This indicates that the use of the framework presented in industrial tasks, such as pick-and-place, can positively influence the quality of the service provided by the robotic system. At the same time, the proposed framework reduces privacy risks - no private dataset is shared, which makes it interesting for entities that value increased data security, e.g., to protect sensitive information from rival companies. I believe that this improvement in performance

could be even greater when the number of participating clients is larger and their private datasets are smaller. Due to hardware constraints, I was unable to test a federation with more than six clients, which meant that each client had about 890000 training samples in the local training dataset. If the original dataset was divided between a larger number of clients resulting in smaller local datasets, I hypothesize that the discrepancies would be even larger.

These results are especially relevant for industries such as the automotive industry, where classified information about the manufacturing process and part design is very valuable. Furthermore, I believe that there is great potential to use this framework in healthcare. Sano Center for Computational Personalized Medicine, which was the institution that collaborated on this study, sees potential in FL for medical applications. In discussions with senior researchers, I learned that using FL to train ML models on data from hospitals around the world could increase their overall performance while avoiding problems with cross-border distributed learning. These issues are related to legal barriers that prevent the transfer of sensitive medical data abroad from the original institutions. In this context, a natural application of robotic FL could be surgical robots. Senior Sano researchers indicated their interest in FL also in this context, as they become more and more aware that traditional distributed training may not be practical for healthcare. Therefore, the consultants positively evaluated the presented system, indicating that the possibility of training heterogeneous models is a strong advantage, as differences in the computational infrastructure between hospitals can be very large, especially when collaborating with institutions across different countries.

6.3 Collaborative training flexibility

As a result of a generalized implementation, the framework is suitable not only for robotic ML tasks. I have presented its application to three different ML problems: grasp prediction, image classification, and sentiment analysis. Each problem used different datasets and architectures. As shown in Ch. 5, in all cases, the improvements introduced by the FL system were clear. In Sec. 5.1.2, Sec. 5.2.1, and Sec. 5.3.1, I have presented the difference in model performance on validation datasets, for isolated and collaboratively trained models. For the MNIST experiments, the increase in mean accuracy scores at the end of federated training was equal to 0.049, 0.037 and 0.045 for the small, medium, and large model groups, respectively. Even higher differences were observed for the IMDB dataset: 0.123, 0.185 and 0.178. This trend was also reinforced by Dex-Net experiments, although with smaller differences in performance (0.009, 0.014, 0.032). Intuitively, these results are expected; participation in the collaborative training exposes the client models to a larger number of training samples, thus allowing them to obtain better performance compared to a situation where the model is trained only on a private

dataset. However, contrary to a standard distributed learning approach, where the dataset is shared between clients, the proposed system allows models to benefit from this larger amount of data without raising concerns about privacy. It is also worth noting that the improvement observed when using this system could be more significant than that indicated by the MNIST and Dex-Net experiments. In the case of MNIST, it is one of the most straightforward datasets, and even clients trained on a small subset of data can obtain very high performance, therefore, concealing the benefits of the on-peer training step. On a more difficult dataset, when each sample counts, the difference in the number of samples to which the models are exposed when using collaborative training, compared to isolated training, can be even more significant. The difference could also potentially increase for DexNet, as explained in Sec 6.2.

6.4 The influence of α and T

By using an objective function that not only incorporated standard loss functions for the ML tasks considered, but also introduced a KD term, I allowed the framework to benefit from this interesting approach to model training. The heterogeneity of the participants was simulated using three different model groups in each experiment. The introduction of the parameter α allowed for easy adjustment of the amount of KD that was used in the experiments. This way, it was possible to investigate the influence of KD on the predictive performance of the models. It was observed that the models that benefit the most from KD are the ones that perform the worst within the federation. This was visible in all experiments. For the MNIST dataset, on average, the worst performing models were representatives of the small model group. In Fig. 5.9 I could observe that those models obtained an improved accuracy score when using $\alpha \in \{0.3, 0.5, 0.2\}$. This indicates that the models were able to extract additional knowledge from the host model during the on-peer training procedure. However, for the medium and large model groups, the best results were provided using $\alpha = 1.0$, without knowledge distillation. These observations are also visible in the experiments of the IMDB (Fig. 5.18) and Dex-Net 2.0 (Tab. 5.3) experiments, although I was unable to obtain as large difference in average performance between the groups, as with the MNIST dataset, which could hide the dynamics between models. In Dex-Net, all model groups benefited from the KD. In general, observations show that KD can indeed improve training results, but if the differences between model groups are large, only the weakest performers will benefit. If the disparity between the performance of the models is too large, it is recommended to use a personalized value α, which can be tuned according to a historical performance comparison between the guest and host models, which could disable distillation if the guest model performed better than the host. It is also important to note that, when using KD, the temperature parameter

has to be carefully tuned. The results shown in Tab. 5.4, Tab. 5.7, and Tab. 5.10 show that this parameter can greatly influence the results and is an additional hyperparameter to consider when using the proposed framework, although no clear trends were observed, indicating that the optimal values are application specific.

6.5 Study limitations

In the proposed training workflow, the main limitation that could create difficulties when using the framework in practice are the costs associated with the on-peer training step. This step in the workflow uses resources of the host, such as energy and storage, which means that this cost must be taken into account when participating in federation. Theoretically, the more proficient models are participating in the federation, the better for the participants, which should motivate clients to help each other; however, there is the question of the incentive for the best clients. They benefit the least from the training and therefore may not be motivated to share their resources with other participants, which could effectively be a disadvantage of the proposed scheme.

Another limitation of this study is the potential privacy risks that should be thoroughly analyzed. Although the proposed collaborative training scheme reduces privacy risks compared to standard distributed learning approaches, it does not guarantee the full security of sensitive data. As presented in Ch. 2, there are documented methods that can extract information directly from models. In the proposed training scheme, models that were trained on private datasets of other clients are indeed exchanged. Potentially, this could lead to information leaks that could be used by malicious users.

6.6 Future research directions

The presented study sets the basis for several topics that could be investigated in the future. The use of FL is not widely investigated in robotics; therefore, finding more applications for the presented framework could prove essential to reinforce FL in this domain, which, as indicated, can be very beneficial for robotic systems in many industries. Investigations could focus on documenting more ML methods for robotic tasks that are suitable for training with the use of the proposed system. One direction could be to test a different grasp prediction method. Another possibility would be to apply the framework to mobile robotics. An important topic is also finding applications of the framework on a higher level, by conducting case studies for different industries, which would raise awareness of the applicability of federated learning in robotics and potentially increase the amount of research in this domain. As always, evaluation of the framework on a larger number of

datasets is advised to better understand the dynamics between model groups and system advantages.

A question relevant to the FL domain is how the presented framework performs on the non-IID data. In this study, I assumed that the data are IID between clients, which will not be the case for most federations. Investigating how the non-IID state influences collaborative training is, therefore, very important. Furthermore, I did not investigate the influence of some of the training parameters. Specifically, for each training round, I conducted exactly one local training epoch and one on-peer training epoch. Future experiments could investigate how different values of these parameters influence the results of federation training. An interesting study direction could also consider incentive analysis. As indicated in Sec. 6.5, if the weakest clients benefit the most from participating in the federations, what could be the incentive mechanism that will attract clients with larger computational capabilities? What could be their motivation to join the training? These questions are very relevant, and without answering them through a number of case studies, frameworks like the one proposed cannot be used in practice. Another question related to the limitations of the study is: How secure is this framework? The priority of future work should be to identify privacy risks and methods to prevent malicious behavior. An interesting direction relevant to this matter, which could be the focus point in the future, is the incorporation of secure multiparty computation methods and differential privacy mechanisms to improve the safety of the federation.

Chapter 7

Conclusion

In this work, my aim was to turn the reader's attention to the privacy issues that have so far been widely overlooked in the robotics domain, but, in my opinion, pose great risks for the users of many data-driven robotic applications. I believe that data protection will be a matter of great relevance in the near future; therefore, in Ch. 1 I showed how FL can alleviate potential privacy risks. In a comprehensive review of the literature (Ch. 2), I provided the reader with current FL research directions and proved that this setting is not yet widely investigated in the robotic context. To contribute to this end, a novel heterogeneous FL framework was proposed, which allows a federation of clients to be trained collaboratively, so that they can increase performance compared to isolated training, without sharing private datasets with their peers, as in standard distributed learning approaches. I showed the suitability of this method in an industrial robotics task, by training alt-GQ-CNN models on the Dex-Net 2.0 grasp prediction dataset. With the trained models, a pick-and-place pipeline was deployed and evaluated, using a robotic manipulator. The results show an increase in pipeline performance when the proposed training system is used, even though it does not transfer local datasets between participants, thus improving privacy. To the best of my knowledge, this is the first documented use of an FL framework in industrial robotics. I believe that such frameworks can increase robotic system performance when privacy is an important factor, which is crucial, e.g., in manufacturing and healthcare.

In addition, to contribute to the FL domain, this framework addressed two problems with existing FL approaches. Standard FL methods are limited to federations containing clients that are heterogeneous in terms of the deep learning architectures used, which is not optimal, as they often have inconsistent computational capabilities. To solve this issue, I presented a heterogeneous federated learning framework, which can be used to train a federation of clients, regardless of their architecture, on a common machine learning task. In experiments, the framework was evaluated and compared with the results of clients trained in iso-

lation, demonstrating the superiority of the proposed FL approach in sentiment analysis and image classification, in addition to the aforementioned positive implications for the grasp prediction task. I also evaluated the influence of the two term objective function on the clients' performance. I found that knowledge distillation can increase accuracy scores for less-proficient clients, compared to training with a standard objective function. What distinguishes this method from existing heterogeneous FL approaches is that it can perform heterogeneous FO without any auxiliary dataset. This trait is especially useful for tasks in which data collection is difficult.

The use of FL in robotic systems is a broad topic and this study focused only on its part. As described in Ch. 6, there are many possible directions for future research, including investigation of privacy risks, case study of robotic applications, influence of non-IID data, and incentive analysis.

I believe that FL is providing solutions to privacy issues related to information sharing that are becoming more and more apparent in the increasingly data-driven world. User privacy can soon find itself as a main point of interest for many domains, including robotics. I encourage the reader to investigate FL and generally privacy in this context, as this emerging field could soon become one of the cornerstones of modern robotic systems.

Bibliography

[1] *Sano Center for Computational and Personalized Medicine*. https://sano.science/. Accessed: 2022-05-14.

[2] Thomas George Thuruthel et al. "Soft robot perception using embedded soft sensors and recurrent neural networks". In: *Science Robotics* 4.26 (2019), eaav1488.

[3] Jamil Fayyad et al. "Deep learning sensor fusion for autonomous vehicle perception and localization: A review". In: *Sensors* 20.15 (2020), p. 4220.

[4] Shehan Caldera, Alexander Rassau, and Douglas Chai. "Review of deep learning methods in robotic grasp detection". In: *Multimodal Technologies and Interaction* 2.3 (2018), p. 57.

[5] Yang Yang, Li Juntao, and Peng Lingling. "Multi-robot path planning based on a deep reinforcement learning DQN algorithm". In: *CAAI Transactions on Intelligence Technology* 5.3 (2020), pp. 177–183.

[6] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[7] *iRobot Vacuums*. https://www.irobot.com/roomba. Accessed: 2022-04-01.

[8] Chen Li et al. "How can i help you? an intelligent virtual assistant for industrial robots". In: *Companion of the 2021 ACM/IEEE International Conference on Human-Robot Interaction*. 2021, pp. 220–224.

[9] H. Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: (2016). DOI: 10.48550/ARXIV.1602.05629. URL: https://arxiv.org/abs/1602.05629.

[10] *Gboard*. https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin&hl=en&gl=US. Accessed: 2022-04-01.

[11] *Android OS*. https://developer.android.com/about.

[12] Peter Kairouz et al. "Advances and open problems in federated learning". In: *Foundations and Trends® in Machine Learning* 14.1–2 (2021), pp. 1–210.

[13] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* 2.7 (2015).

[14] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[15] William Shakespeare. *The complete works of William Shakespeare*. URL: https://www.gutenberg.org/ebooks/100.

[16] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. "CIFAR-10 (Canadian Institute for Advanced Research)". In: (). URL: http://www.cs.toronto.edu/~kriz/cifar.html.

[17] Tian Li et al. "Federated optimization in heterogeneous networks". In: *Proceedings of Machine Learning and Systems* 2 (2020), pp. 429–450.

[18] Sai Praneeth Karimireddy et al. "Scaffold: Stochastic controlled averaging for federated learning". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5132–5143.

[19] Ning Qian. "On the momentum term in gradient descent learning algorithms". In: *Neural networks* 12.1 (1999), pp. 145–151.

[20] Jianyu Wang et al. "Tackling the objective inconsistency problem in heterogeneous federated optimization". In: *Advances in neural information processing systems* 33 (2020), pp. 7611–7623.

[21] Avishek Ghosh et al. "An efficient framework for clustered federated learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 19586–19597.

[22] Canh T Dinh, Nguyen Tran, and Josh Nguyen. "Personalized federated learning with moreau envelopes". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 21394–21405.

[23] Abderrahim Jourani, Lionel Thibault, and Dariusz Zagrodny. "Differential properties of the Moreau envelope". In: *Journal of Functional Analysis* 266.3 (2014), pp. 1185–1237.

[24] Robin C Geyer, Tassilo Klein, and Moin Nabi. "Differentially private federated learning: A client level perspective". In: *arXiv preprint arXiv:1712.07557* (2017).

[25] Reza Shokri et al. "Membership inference attacks against machine learning models". In: *2017 IEEE symposium on security and privacy (SP)*. IEEE. 2017, pp. 3–18.

[26] Chiyuan Zhang et al. "Understanding deep learning requires rethinking generalization (2016)". In: *arXiv preprint arXiv:1611.03530* (2017).

[27] Nicholas Carlini et al. “The secret sharer: Evaluating and testing unintended memorization in neural networks”. In: *28th USENIX Security Symposium (USENIX Security 19)*. 2019, pp. 267–284.

[28] Jonas Geiping et al. “Inverting gradients-how easy is it to break privacy in federated learning?” In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 16937–16947.

[29] Léon Bottou. “Large-scale machine learning with stochastic gradient descent”. In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[30] Keiron O'Shea and Ryan Nash. “An introduction to convolutional neural networks”. In: *arXiv preprint arXiv:1511.08458* (2015).

[31] Arvind Narayanan and Vitaly Shmatikov. “How to break anonymity of the netflix prize dataset”. In: *arXiv preprint cs/0610105* (2006).

[32] Pao-Lu Hsu and Herbert Robbins. “Complete convergence and the law of large numbers”. In: *Proceedings of the National Academy of Sciences of the United States of America* 33.2 (1947), p. 25.

[33] Galen Andrew et al. “Differentially private learning with adaptive clipping”. In: *Advances in Neural Information Processing Systems* 34 (2021).

[34] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[35] Ralf C Staudemeyer and Eric Rothstein Morris. “Understanding LSTM–a tutorial into long short-term memory recurrent neural networks”. In: *arXiv preprint arXiv:1909.09586* (2019).

[36] Rami Al-Rfou et al. “Conversational contextual cues: The case of personalization and history for response ranking”. In: *arXiv preprint arXiv:1606.00372* (2016).

[37] Abhishek Bhowmick et al. “Protection against reconstruction and its applications in private federated learning”. In: *arXiv preprint arXiv:1812.00984* (2018).

[38] Stacey Truex et al. “A hybrid approach to privacy-preserving federated learning”. In: *Proceedings of the 12th ACM workshop on artificial intelligence and security*. 2019, pp. 1–11.

[39] Michael O'Keeffe. “The paillier cryptosystem”. In: *Mathematics Department April* 18 (2008), pp. 1–16.

[40] John F Magee. *Decision trees for decision making*. Harvard Business Review Brighton, MA, USA, 1964.

[41] Corinna Cortes and Vladimir Vapnik. “Support vector machine”. In: *Machine learning* 20.3 (1995), pp. 273–297.

[42] Keith Bonawitz et al. “Practical secure aggregation for federated learning on user-held data”. In: *arXiv preprint arXiv:1611.04482* (2016).

[43] Eugene Bagdasaryan et al. “How to backdoor federated learning”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 2938–2948.

[44] Arjun Nitin Bhagoji et al. “Analyzing federated learning through an adversarial lens”. In: *International Conference on Machine Learning*. PMLR. 2019, pp. 634–643.

[45] Peva Blanchard et al. “Machine learning with adversaries: Byzantine tolerant gradient descent”. In: *Advances in Neural Information Processing Systems* 30 (2017).

[46] Haibo Yang et al. “Byzantine-resilient stochastic gradient descent for distributed learning: A lipschitz-inspired coordinate-wise median approach”. In: *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE. 2019, pp. 5832–5837.

[47] Chulin Xie et al. “Dba: Distributed backdoor attacks against federated learning”. In: *International Conference on Learning Representations*. 2019.

[48] Jiawen Kang et al. “Incentive design for efficient federated learning in mobile networks: A contract theory approach”. In: *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)*. IEEE. 2019, pp. 1–5.

[49] Tian Li et al. “Fair resource allocation in federated learning”. In: *arXiv preprint arXiv:1905.10497* (2019).

[50] Hyesung Kim et al. “Blockchained on-device federated learning”. In: *IEEE Communications Letters* 24.6 (2019), pp. 1279–1283.

[51] Michael Nofer et al. “Blockchain”. In: *Business & Information Systems Engineering* 59.3 (2017), pp. 183–187.

[52] Satoshi Nakamoto. “Bitcoin whitepaper”. In: *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)* (2008).

[53] Andreas Grafberger et al. “FedLess: Secure and Scalable Federated Learning Using Serverless Computing”. In: *2021 IEEE International Conference on Big Data (Big Data)*. IEEE. 2021, pp. 164–173.

[54] Sarah Allen et al. “Cncf serverless whitepaper v1. 0”. In: *Dosegljivo: https://github.com/cncf/wg-serverless/tree/master/whitepapers/serverless-overview* (2018).

[55] Apache OpenWhisk. “Apache openwhisk is a serverless, open source cloud platform”. In: *Apache Foundation,[Online]. Available: http://openwhisk. apache. org/.[Accessed 21 June 2018]* (2018).

[56] Google Cloud Platform. *Cloud Functions — Google Cloud*.

[57] Amazon AWS. *AWS Lambda-Serverless Compute-Amazon Web Services*.

[58] Tao Lin et al. "Ensemble distillation for robust model fusion in federated learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 2351–2363.

[59] Victor Sanh et al. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv preprint arXiv:1910.01108* (2019).

[60] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[61] Kaiming He et al. "Identity mappings in deep residual networks". In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[62] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[63] Ningning Ma et al. "Shufflenet v2: Practical guidelines for efficient cnn architecture design". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 116–131.

[64] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[65] Xiang Zhang, Junbo Zhao, and Yann LeCun. "Character-level convolutional networks for text classification". In: *Advances in neural information processing systems* 28 (2015).

[66] Richard Socher et al. "Recursive deep models for semantic compositionality over a sentiment treebank". In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1631–1642.

[67] Daliang Li and Junpu Wang. "Fedmd: Heterogenous federated learning via model distillation". In: *arXiv preprint arXiv:1910.03581* (2019).

[68] Li Hu et al. "MHAT: An efficient model-heterogenous aggregation training scheme for federated learning". In: *Information Sciences* 560 (2021), pp. 493–503.

[69] Nathalie Majcherczyk, Nishan Srishankar, and Carlo Pinciroli. "Flow-fl: Data-driven federated learning for spatio-temporal predictions in multi-robot systems". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 8836–8842.

[70] Xianjia Yu, Jorge Pena Queralta, and Tomi Westerlund. "Federated Learning for Vision-based Obstacle Avoidance in the Internet of Robotic Things". In: *arXiv preprint arXiv:2204.06949* (2022).

[71] NVIDIA. *NVIDIA Isaac*.

[72] Zijian Zhang et al. "Distributed dynamic map fusion via federated learning for intelligent networked vehicles". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 953–959.

[73] Alexey Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.

[74] Hai Nguyen and Hung La. "Review of deep reinforcement learning for robot manipulation". In: *2019 Third IEEE International Conference on Robotic Computing (IRC)*. IEEE. 2019, pp. 590–595.

[75] Wenshuai Zhao, Jorge Pena Queralta, and Tomi Westerlund. "Sim-to-real transfer in deep reinforcement learning for robotics: a survey". In: *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2020, pp. 737–744.

[76] Neziha Akalin and Amy Loutfi. "Reinforcement learning approaches in social robotics". In: *Sensors* 21.4 (2021), p. 1292.

[77] Xinle Liang et al. "Federated transfer reinforcement learning for autonomous driving". In: *arXiv preprint arXiv:1910.06001* (2019).

[78] Shital Shah et al. "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles". In: *Field and Service Robotics*. 2017. eprint: arXiv:1705.05065. URL: https://arxiv.org/abs/1705.05065.

[79] Xinle Liang et al. "Federated transfer reinforcement learning for autonomous driving". In: *arXiv preprint arXiv:1910.06001* (2019). URL: https://arxiv.org/abs/1509.02971.

[80] Boyi Liu, Lujia Wang, and Ming Liu. "Lifelong federated reinforcement learning: a learning architecture for navigation in cloud robotic systems". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 4555–4562.

[81] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

[82] Janet Kuhn. "Decrypting the MoSCoW analysis". In: *The workable, practical guide to Do IT Yourself* 5 (2009).

[83] R. Fielding et al. *RFC 2616, Hypertext Transfer Protocol – HTTP/1.1*. 1999. URL: http://www.rfc.net/rfc2616.html.

[84] I. Fette and A. Melnikov. *The WebSocket Protocol*. RFC 6455. http://www.rfc-editor.org/rfc/rfc6455.txt. RFC Editor, 2011. URL: http://www.rfc-editor.org/rfc/rfc6455.txt.

[85] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv preprint arXiv:1609.04747* (2016).

[86] Gavin Hackeling. *Mastering Machine Learning with scikit-learn*. Packt Publishing Ltd, 2017.

[87] Jeffrey Mahler et al. "Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics". In: *arXiv preprint arXiv:1703.09312* (2017).

[88] Andrew Maas et al. "Learning word vectors for sentiment analysis". In: *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*. 2011, pp. 142–150.

[89] Jeffrey Mahler et al. "Dex-net 1.0: A cloud-based network of 3d objects for robust grasp planning using a multi-armed bandit model with correlated rewards". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 1957–1964.

[90] Domenico Prattichizzo and Jeffrey C Trinkle. "Grasping". In: *Springer handbook of robotics*. Springer, 2016, pp. 955–988.

[91] Vishal Satish, Jeffrey Mahler, and Ken Goldberg. "On-Policy Dataset Synthesis for Learning Robot Grasping Policies Using Fully Convolutional Deep Networks". In: *IEEE Robotics and Automation Letters* (2019).

[92] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[93] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[94] *UFACTORY xArm collaborative robots*. `https://www.ufactory.cc/xarm-collaborative-robot`. Accessed: 2022-05-14.

[95] *Intel RealSense D415 Camera*. `https://www.intelrealsense.com/depth-camera-d415/`. Accessed: 2022-05-14.

[96] Zhengyou Zhang. "A flexible new technique for camera calibration". In: *IEEE Transactions on pattern analysis and machine intelligence* 22.11 (2000), pp. 1330–1334.

[97] Roger Y Tsai, Reimar K Lenz, et al. "A new technique for fully autonomous and efficient 3 d robotics hand/eye calibration". In: *IEEE Transactions on robotics and automation* 5.3 (1989), pp. 345–358.

[98] *ABB Yumi Grippers*. `https://new.abb.com/products/robotics/collaborative-robots/irb-14050-single-arm-yumi`. Accessed: 2022-05-14.

[99] John Canny. "A computational approach to edge detection". In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), pp. 679–698.

[100] Dan Jurafsky and James H. Martin. *Speech and language processing : an introduction to natural language processing, computational linguistics, and speech recognition*. 2009. ISBN: 9780131873216 0131873210. URL: http://www.amazon.com/Speech-Language-Processing-2nd-Edition/dp/0131873210/ref=pd_bxgy_b_img_y.

[101] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[102] Tomas Mikolov et al. "Recurrent neural network based language model." In: *Interspeech*. Vol. 2. 3. Makuhari. 2010, pp. 1045–1048.

[103] Guido vanRossum. "Python reference manual". In: *Department of Computer Science [CS]* R 9525 (1995).

[104] Philipp Moritz et al. "Ray: A distributed framework for emerging {AI} applications". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018, pp. 561–577.

[105] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).

[106] William Falcon and The PyTorch Lightning team. *PyTorch Lightning*. Version 1.4. Mar. 2019. DOI: 10.5281/zenodo.3828935. URL: https://github.com/PyTorchLightning/pytorch-lightning.

[107] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[108] Radim Rehurek and Petr Sojka. "Gensim–python framework for vector space modelling". In: *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic* 3.2 (2011).

[109] *Prometheus HPC information*. https://kdm.cyfronet.pl/portal/Prometheus:en. Accessed: 2022-05-14.

[110] Marian Bubak, Tomasz Szepieniec, and Kazimierz Wiatr. *Building a National Distributed E-Infrastructure–PL-Grid: Scientific and Technical Achievements*. Vol. 7136. Springer Science & Business Media, 2012.

[111] *Academic Computer Centre CYFRONET AGH*. https://kdm.cyfronet.pl/portal/Main_page. Accessed: 2022-05-14.

[112] *Heterogeneous Federated Learning framework*. https://github.com/SanoScience/heterogeneous-federated. Accessed: 2022-06-02.