MULTI-TASK FEDERATED LEARNING FOR PERSONALISED DEEP NEURAL NETWORKS IN EDGE COMPUTING

A PROJECT REPORT

Submitted in the partial fulfillment of requirements to

R.V.R & J.C COLLEGE OF ENGINEERING

For the award of the degree
B.Tech. in CSE

By

Thatti Jatin (Y20CS173)
Varikuti Pavan Kumar (Y20CS184)
Udayagiri Leela Krishna (Y20CS181)

May, 2024

R.V.R. & J.C. COLLEGE OF ENGINEERING (Autonomous)
(Affiliated to Acharya Nagarjuna University)
Chandramoulipuram::Chowdavaram
GUNTUR – 522 019

R.V.R.& J.C. COLLEGE OF ENGINEERING
(Autonomous)

DEPARTMENT OF COMPUTER SCIENCE& ENGINEERING

CERTIFICATE

This is to certify that this project work titled **"Multi-Task Federated Learning for Personalised Deep Neural Networks in Edge Computing"** is the work done by Thatti Jatin (Y20CS173), Varikuti Pavan Kumar (Y20CS184) and Udayagiri Leela Krishna (Y20CS181) under my supervision, and submitted in partial fulfillment of the requirements for the award of the degree, B.Tech. in Computer Science & Engineering, during the Academic Year **2023-2024**.

S.Karthik	**Dr. K. Siva Kumar**	**Dr. M. Sreelatha**
Project Guide	**In-charge, Project Work**	**Prof.&Head**

ACKNOWLEDGEMENT

The successful completion of any task would be incomplete without proper suggestions, guidance and environment. Combination of these three factors acts like backbone to our Project **"Multi-Task Federated Learning for Personalised Deep Neural Networks in Edge Computing"** .

We are very glad to express our special thanks to **S.Karthik**, Project Guide, who has inspired us to select this topic, and also for his/her valuable suggestions in completion of the project.

We are very thankful to **Dr. K.Siva Kumar**, Project Incharge who extended his encouragement and assistance in ensuring the smooth progress of the project work.

We would like to express our profound gratitude to **Dr.M.Sreelatha**, Head of the Department of Computer Science and Engineering for her encouragement and support to carry out this Project successfully.

We are very much thankful to **Dr. Kolla Srinivas** , Principal, for having allowed us to conduct the study needed for the project.

Finally we submit our heartfelt thanks to all the staff in the Department and to all our friends for their cooperation during the project work.

T. Jatin(Y20CS173)

V. Pavan Kumar (Y20CS184)

U. Leela Krishna (Y20CS181)

ABSTRACT

Federated Learning (FL) presents a compelling solution for training Deep Neural Networks (DNNs) on edge devices without compromising user privacy. However, challenges such as non-IID data distribution and the need for personalized model accuracy persist. In this paper, a novel Multi-Task FL (MTFL) algorithm is proposed to address these challenges. MTFL enhances user model accuracy (UA) and convergence speed by incorporating non-federated Batch-Normalization (BN) layers into federated DNNs. MTFL allows users to train personalized models tailored to their own data while preserving privacy. The approach leverages the benefits of distributed optimization techniques like Adam within the FL framework. UA is introduced as a new metric for measuring FL performance, emphasizing individual model accuracy over global model accuracy. Through extensive experiments on MNIST, CIFAR10, MTFL with FedAvg-Adam outperforms existing FL algorithms, achieving fewer rounds to reach target UA and significantly improving average UA compared to state-of-the-art personalized FL algorithms. These findings highlight the feasibility and effectiveness of MTFL in enhancing FL performance while addressing practical deployment challenges in edge computing environments.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

S.No.	Abbreviation	Full Form
1.	FL	Federated Learning
2.	DNNs	Deep Neural Networks
3.	non-IID	non-Independent and Identically Distributed
4.	UA	User model Accuracy
5.	MTFL	Multi-Task Federated Learning
6.	BN	Batch-Normalization
7.	FedAvg	Federated Averaging
8.	MEC	MULTI-ACCESS Edge Computing
9.	MTL	Multi-Task Learning
10.	SGD	Stochastic Gradient Descent
11.	SINR	Signal-to-Interference plus-Noise Ratio
12.	MAML	Model Agnostic Meta-Learning
13.	FC	Fully Connected
14.	CNN	Convolutional Neural Networks
15.	P2P	Peer-To-Peer

CHAPTER 1
INTRODUCTION

1.1 Background

MULTI-ACCESS Edge Computing (MEC) moves Cloud services to the network edge, enabling low-latency and real-time processing of applications via content caching and computation offloading [2], [3]. Coupled with the rapidly increasing quantity of data collected by smart phones, Internet-of-Things (IoT) devices, and social networks (SNs),MEC presents an opportunity to store and process huge quantities of data at the edge, close to their source.

Deep Neural Networks (DNNs) for Machine Learning (ML) are becoming increasingly popular for their huge range of potential applications, ease of deployment, and state-of-the-art performance. Training DNNs in supervised learning, however, can be computationally expensive and require an enormous amount of training data, especially with the trend of increasing DNN size. The use of DNNs in MEC has typically involved collecting data from mobile phones/IoT devices/SNs, performing training in the cloud, and then deploying the model at the edge. Concerns about data privacy, however, mean that users are increasingly

unwilling to upload their potentially sensitive data, raising the question about how these models will be trained. Federated Learning (FL) [4] opens new horizons for ML at the edge. In FL, participating clients collaboratively train an ML model (typically DNNs), without revealing their private data. McMahan et al. [5] published an initial investigation into FL with the Federated Averaging (FedAvg) algorithm. FedAvg works by initializing a model at a coordinating server before distributing this model to clients. These clients perform a round of training on their local datasets and push their new models to the server. The server averages these models together before sending the new aggregated model to the clients for the next round of training. referred to the people/institutions/etc. that own data for FL as ‘users’, and to the devices that actually participate in FL as ‘clients’. FL is a very promising approach for distributed ML in situations where data cannot be uploaded for protecting clients’ privacy. Therefore, FL is well suited for real-world scenarios such as analyzing sensitive healthcare data [6], [7], next-word prediction on mobile keyboards [8], and content recommendation [9]. However, FL presents multiple unique challenges:

- Clients usually do not have Independent and Identically Distributed (IID) training data. Each client has data generated by itself, and can have noisy data or only a subset of all features/labels. These factors can all substantially hinder training of the FL model.
- FL research typically uses the performance metric of global-model accuracy on a centralized test-set. However, in many cases, individual model accuracy on clients is the real objective – motivating 'personalized FL' that creates unique models for FL clients to improve local performance. However, the best way of incorporating personalization into FL remains an under-researched topic.
- Due to the non-IID nature of client datasets, the performance of the global FL model may be higher on some clients than others. This could even lead some clients to receive a worse model than the one they could have trained independently.

This paper addresses the above challenges by proposing a Multi-Task FL algorithm (MTFL), that allows clients to train personalized DNNs that both improve local model accuracy, and help to further enhance client privacy. MTFL has lower storage cost of personalization, and lower computing cost compared with other personalized FL algorithms (not requiring extra steps of SGD on clients during the training loop or at personalization time) [10], [11], [12], [13].As client datasets in FL are usually non-IID, clients can be viewed as attempting to optimize their models during local training for disparate tasks. Our MTFL approach takes the Batch-Normalization (BN) layers that are commonly incorporated into DNN architectures, and keeps them private to each client. Mudrarkarta et al. [14] previously showed that private BN layers improved Multi-Task Learning (MTL) performance for joint training on ImageNet/Places-365 in the centralized setting. Using private BN layers has the dual benefit of personalizing each model to the clients' local data as well as helping to preserve data privacy: as some parameters of client models are not uploaded to the server, less information about a client's data distribution can be gleaned from the uploaded model. Our MTFL approach using BN layers also has a storage-cost benefit compared to other personalized FL algorithms: BN layers typically contain a tiny fraction of the total parameters of a DNN, and only these BN parameters need to be stored between FL rounds, compared to entire personalized DNN models of competing algorithms [10],[12],[13].MTFL adds

personalization on top of the typical iterative FL framework. FedAvg and other popular algorithms are instances of this iterative optimization framework [5], [16]. Most of these FL algorithms use vanilla Stochastic Gradient Descent (SGD) on clients, however, momentum-based optimization strategies such as Adam [17] have the potential to improve convergence speed of FL training. showed that a distributed optimization technique using Adam (FedAvg Adam) shows substantial speedup in terms of communication rounds compared to FedAvg, and works very well within the MTFL algorithm.

Our work makes the following contributions:

- Proposed an MTFL algorithm that adds Multitask learning on top of general iterative-FL algorithms, allowing users to learn DNN models that are personalized for their own data. MTFL uses private Batch Normalization (BN) layers to achieve this personalization, which provides an added privacy benefit.
- Proposed a new metric for measuring the performance of FL algorithms: User model Accuracy (UA). UA better reflects a common objective of FL (increasing test accuracy on clients), as opposed to the standard global-model accuracy.
- Analyzed the impact that private BN layers have on the activations of MTFL models during inference, providing insights into the source of their impact. analyzed the training and testing performance of MTFL when keeping either the trained parameters or statistics of BN layers private, demonstrating that MTFL provides a better balance between convergence and regularization compared to FL or independent training.
- Conducted extensive simulations on the MNIST and CIFAR10 datasets. The results show that MTFL with FedAvg is able to reach a target UA in up to 5× less rounds than when using only FL, with FedAvg- Adam providing a further 3× improvement. Other experiments show that MTFL is able to significantly improve average UA compared to other state-of-the- art personalized FL algorithms.
- Performed experiments using an MEC-like testbed consisting of Raspberry Pi clients and a FL server. The results show that MTFL with FedAvg-Adam's overheads are outweighed by its substantial UA and convergence speed benefits.

The rest of this paper is organized as follows: Section 2 describes related work; Section 3 details the proposed MTFL algorithm, the effect that keeping private BN layers within MTFL has on training and inference, and the FedAvg-Adam optimization strategy; Section 4 presents and discusses experiments using both simulations and an MEC-like testbed; and Section 5 concludes the paper.

1.1.1 Overcoming Non-IID Data

FL optimization faces unique challenges, primarily stemming from the non-Independent and Identically Distributed (non-IID) nature of user data. This challenge affects the convergence speed of FL algorithms, making it crucial to develop strategies that can handle diverse and noisy datasets efficiently. Additionally, traditional FL approaches often measure performance solely based on global-model accuracy, overlooking the importance of improving individual model accuracy for personalized user experiences.

1.1.2 Personalized Federated Learning

Personalized Federated Learning aims to address the limitations of traditional FL by tailoring models to individual users' data, thereby improving local model accuracy and user satisfaction. However, incorporating personalization into FL algorithms poses significant research challenges, particularly in achieving a balance between model customization and preserving user privacy. Finding effective techniques for personalization while ensuring data privacy remains an ongoing area of investigation.

1.1.3 Multi-Task Federated Learning

Multi-Task Federated Learning (MTFL) presents a promising solution to enhance FL by allowing users to train personalized DNN models suited to their specific data. MTFL introduces non-federated Batch-Normalization (BN) layers into federated DNN architectures, enabling users to achieve better model accuracy while preserving data privacy. By leveraging techniques such as private BN layers and distributed optimization strategies like FedAvg-Adam, MTFL aims to improve convergence speed and user model accuracy, addressing critical challenges in FL optimization.

1.2 Problem Definition

The main aim is to improve the convergence speed and individual User model Accuracy (UA) in Federated Learning (FL) for training Deep Neural Networks (DNNs) on edge devices and to overcome the challenges posed by non-Independent and Identically Distributed (non-IID) user data in FL algorithms, where data from different users may not be uniformly distributed or independently sampled and to enable users to train personalized models based on their own data while keeping the private user data on their devices.

1.3 Significance of Work

The need for the present study, as outlined in the abstract, is to address the limitations and challenges faced in Federated Learning (FL) when training Deep Neural Networks (DNNs) on mobile devices. The specific needs are:

- Efficient Convergence: There is a requirement to improve the convergence speed of FL algorithms, as the presence of non-Independent and Identically Distributed (non-IID) user data can slow down the training process.

- Personalized User Models: Many applications, such as user content-recommendation, demand personalized User model Accuracy (UA) rather than just focusing on global-model accuracy.

- User Data Privacy: It is essential to ensure that user data remains private and does not leave the mobile devices during the collaborative training process.

- Practicality in Real-World Mobile Scenarios: The study aims to provide a practical and effective solution that can be applied in real-world mobile environments, especially for edge-computing scenarios.

- Compatibility with Existing FL Optimization Algorithms: The proposed approach should be compatible with popular iterative FL optimization algorithms, like Federated Averaging (FedAvg).

- Empirical Validation: The study seeks to empirically evaluate the proposed solution using MNIST and CIFAR10 datasets to showcase its effectiveness and benefits.

- Comparison with Competing Algorithms: The study aims to compare the performance of the proposed approach against other existing personalized FL algorithms to demonstrate its superiority.

- Overcoming Limitations: There is a need to overcome the limitations posed by non-IID data and improve the UA of individual User models.

Overall, the present study seeks to develop a Multi-Task Federated Learning (MTFL) algorithm, incorporating non-federated Batch-Normalization (BN) layers, to achieve the objectives of improving convergence speed, enhancing User model Accuracy (UA), and ensuring user data privacy in FL on mobile devices. The evaluation on an edge-computing testbed further validates its practicality for real-world mobile settings.

CHAPTER 2
LITERATURE SURVEY

The literature survey of the context for the "Multi-Task Federated Learning for Personalized Deep Neural Networks in Edge Computing" can be summarized as follows:

1. Federated Learning (FL): The paper highlights that Federated Learning is an emerging approach for training Deep Neural Networks collaboratively on mobile devices. It allows devices to learn from each other without sharing their private user data, preserving user privacy while improving model accuracy.

2. Non-Independent and Identically Distributed (non-IID) Data: Previous works have shown that FL algorithms struggle with non-IID user data, where the data distribution among devices is not uniform. This can lead to slower convergence of the FL algorithms.

3. Individual User Model Accuracy (UA): While most existing work on FL focuses on global-model accuracy, the paper emphasizes the importance of individual user model accuracy (UA). In scenarios like user content recommendation, improving UA becomes the primary objective to enhance user experience.

4. Multi-Task Federated Learning (MTFL) Algorithm: To address the issues of non-IID data and improve UA, the paper proposes the MTFL algorithm. It introduces non-federated Batch-Normalization (BN) layers into the federated DNN, allowing users to personalize their models based on their own data.

5. Compatibility with Existing FL Optimization Algorithms: MTFL is designed to work with popular iterative FL optimization algorithms like Federated Averaging (FedAvg). Additionally, a distributed form of the Adam optimization algorithm (FedAvg-Adam) is utilized to further enhance convergence speed within MTFL.

6. Experimental Validation: The paper conducts experiments using MNIST and CIFAR10 datasets to validate the effectiveness of MTFL. The results demonstrate significant reductions in the number of rounds required to reach the target UA, outperforming existing FL optimization strategies and personalized FL algorithms in terms of UA.

7. Edge Computing Testbed Evaluation: The paper evaluates MTFL with FedAvg-Adam on an edge computing testbed, confirming that the convergence and UA benefits of MTFL outweigh any additional overhead incurred during the process.

2.1 Review of the project

Y. Mao, et al. , "A survey on mobile edge computing: The communication perspective," [1]. Driven by the visions of Internet of Things and 5G communications, recent years have seen a paradigm shift in mobile computing, from the centralized Mobile Cloud Computing towards Mobile Edge Computing (MEC). The main feature of MEC is to push mobile computing, network control and storage to the network edges (e.g., base stations and access points) so as to enable computation-intensive and latency-critical applications at the resource-limited mobile devices. MEC promises dramatic reduction in latency and mobile energy consumption, tackling the key challenges for materializing 5G vision. The promised gains of MEC have motivated extensive efforts in both academia and industry on developing the technology. A main thrust of MEC research is to seamlessly merge the two disciplines of wireless communications and mobile computing, resulting in a wide-range of new designs ranging from techniques for computation offloading to network architectures. This paper provides a comprehensive survey of the state-of-the-art MEC research with a focus on joint radio-and-computational resource management. Discussed a set of issues, challenges and future research directions for MEC research, including MEC system deployment, cache-enabled MEC, mobility management for MEC, green MEC, as well as privacy-aware MEC. Advancements in these directions will facilitate the transformation of MEC from theory to practice. Finally, introduced recent standardization efforts on MEC as well as some typical MEC application scenarios.

B. Yang, et al. , "Computation offloading in multi-access edge computing: A multi-task learning approach," [2] . Multi-access edge computing (MEC) has already shown the potential in enabling mobile devices to bear the computation-intensive applications by offloading some tasks toa nearby access point (AP) integrated with a MEC server(MES). However, due to the varying network conditions and limited computation resources of the MES, the offloading decisions taken by a mobile device and the computational resources allocated by the MES may not be efficiently achieved with the lowest cost. In this paper, proposed a dynamic offloading framework for the MEC network, in which the uplink non-orthogonal multiple access (NOMA) is used to enable multiple devices to upload their tasks via the same frequency band. Formulated the offloading decision problem as a multiclass classification problem and formulate the MES computational resource allocation problem as a regression problem. Then a multi-task learning based feedforward neural network (MTFNN) model is designed to jointly optimize the offloading decision and computational resource allocation. Numerical results illustrate that the proposed MTFNN outperforms the conventional optimization method in terms of inference accuracy and computation complexity

Y. Li, et al. ,"Learning aided computation offloading for trusted collaborative mobile edge computing," [3] . Cooperative offloading in mobile edge computing enables resource-constrained edge clouds to help each other with computation-intensive tasks. However, the power of such offloading could not be fully unleashed, unless trust risks in collaboration are properly managed. As tasks are outsourced and processed at the network edge, completion latency usually presents high variability that can harm the offered service levels. By jointly considering these two challenges, propose OLCD, an Online Learning-aided Cooperative off loading mechanism under the scenario where computation offloading is organized based on accumulated social trust. Under co-provisioning of computation, transmission, and trust services, trust propagation is performed along the multi-hop offloading path such that tasks are allowed to be fulfilled by powerful edge clouds. harness Lyapunov optimization to exploit the spatial-temporal optimality of long-term system cost minimization problem. By gap-preserving transformation, decouple the series of bidirectional offloading problems so that it suffices to solve a separate decision problem for each edge cloud. The optimal offloading control can not materialize without complete latency

knowledge. To adapt to latency variability, resort to the delayed online learning technique to facilitate completion latency prediction under long-duration processing, which is fed as input to queued-based offloading control policy. Such predictive control is specially designed to minimize the loss due to prediction errors over time. theoretically prove that OLCD guarantees close-to-optimal system performance even with inaccurate prediction, but its robustness is achieved at the expense of decreased stability. Trace-driven simulations demonstrate the efficiency of OLCD as well as its superiorities over prior related work.

T. Li,et al. "Federated learning: Challenges, methods, and future directions," [4] . Federated learning involves training statistical models over remote devices or siloed data centres, such as mobile phones or hospitals, while keeping data localized. Training in heterogeneous and potentially massive networks introduces novel challenges that require a fundamental departure from standard approaches for large-scale machine learning, distributed optimization, and privacy-preserving data analysis. In this article, discuss the unique characteristics and challenges of federated learning, provide a broad overview of current approaches, and outline several directions of future work that are relevant to a wide range of research communities.

B. McMahan,et al. "Communication-efficient learning of deep networks from decentralized data," [5] . Modern mobile devices have access to a wealth of data suitable for learning models, which in turn can greatly improve the user experience on the device. For example, language models can improve speech recognition and text entry, and image models can automatically select good photos. However, this rich data is often privacy sensitive, large in quantity, or both, which may preclude logging to the data center and training there using conventional approaches. Advocate an alternative that leaves the training data distributed on the mobile devices, and learns a shared model by aggregating locally-computed updates. termed this decentralized approach Federated Learning. present a practical method for the federated learning of deep networks based on iterative model averaging, and conduct an extensive empirical evaluation, considering five different model architectures and four datasets. These experiments demonstrate the approach is robust to the unbalanced and non-IID data distributions that are a defining characteristic of this setting. Communication costs are

the principal constraint, and showed a reduction in required communication rounds by 10–100× as compared to synchronized stochastic gradient descent.

A. G. Roy, et al. "Braintorrent: A peer-to-peer environment for decentralized federated learning,"[6]. Access to sufficient annotated data is a common challenge in training deep neural networks on medical images. As annotating data is expensive and time-consuming, it is difficult for an individual medical center to reach large enough sample sizes to build their own, personalized models. As an alternative, data from all centers could be pooled to train a centralized model that everyone can use. However, such a strategy is often infeasible due to the privacy-sensitive nature of medical data. Recently, federated learning (FL) has been introduced to collaboratively learn a shared prediction model across centers without the need for sharing data. In FL, clients are locally training models on site-specific datasets for a few epochs and then sharing their model weights with a central server, which orchestrates the overall training process. Importantly, the sharing of models does not compromise patient privacy. A disadvantage of FL is the dependence on a central server, which requires all clients to agree on one trusted central body, and whose failure would disrupt the training process of all clients. In this paper, introduced Brain Torrent, a new FL framework without a central server, particularly targeted towards medical applications. Brain Torrent presents a highly dynamic peer-to-peer environment, where all centers directly interact with each other without depending on a central body. Demonstrate the overall effectiveness of FL for the challenging task of whole brain segmentation and observe that the proposed server-less Brain Torrent approach does not only outperform the traditional server-based one but reaches a similar performance to a model trained on pooled data.

L. Huang, et al. "Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records," [7] . Electronic medical records (EMRs) supports the development of machine learning algorithms for predicting disease incidence, patient response to treatment, and other healthcare events. But insofar most algorithms have been centralized, taking little account of the decentralized, non-identically independently

distributed (non-IID), and privacy-sensitive characteristics of EMRs that can complicate data collection, sharing and learning. To address this challenge, Introduced a community-based federated machine learning (CBFL) algorithm and evaluated it on non-IID ICU EMRs. Our algorithm clustered the distributed data into clinically meaningful communities that captured similar diagnoses and geological locations, and learnt one model for each community. Throughout the learning process, the data was kept local on hospitals, while locally-computed results were aggregated on a server. Evaluation results show that CBFL outperformed the baseline FL algorithm in terms of Area Under the Receiver Operating Characteristic Curve (ROC AUC), Area Under the Precision-Recall Curve (PR AUC), and communication cost between hospitals and the server. Furthermore, communities' performance difference could be explained by how dissimilar one community was to others.

A. Hard et al., "Federated learning for mobile keyboard prediction," [8]. Trained a recurrent neural network language model using a distributed, on-device learning framework called federated learning for the purpose of next-word prediction in a virtual keyboard for smartphones. Server-based training using stochastic gradient descent is compared with training on client devices using the Federated Averaging algorithm. The federated algorithm, which enables training on a higher-quality dataset for this use case, is shown to achieve better prediction recall. This work demonstrates the feasibility and benefit of training language models on client devices without exporting sensitive user data to servers. The federated learning environment gives users greater control over the use of their data and simplifies the task of incorporating privacy by default with distributed training and aggregation across a population of client devices.

M. Ammad-ud-din, et al. "Federated collaborative filtering for privacy-preserving personalized recommendation system," [9] . The increasing interest in user privacy is leading to new privacy preserving machine learning paradigms. In the Federated Learning paradigm, a master machine learning model is distributed to user clients, the clients use their locally stored data and model for both inference and calculating model updates. The model updates are sent back and aggregated on the server to update the master model then redistributed to the clients. In this paradigm, the user data never

leaves the client, greatly enhancing the user' privacy, in contrast to the traditional paradigm of collecting, storing and processing user data on a backend server beyond the user's control. In this paper introduce, as far as are aware, the first federated implementation of a Collaborative Filter. The federated updates to the model are based on a stochastic gradient approach. As a classical case study in machine learning, explore a personalized recommendation system based on users' implicit feedback and demonstrate the method's applicability to both the Movie Lens and an in-house dataset. Empirical validation confirms a collaborative filter can be federated without a loss of accuracy compared to a standard implementation, hence enhancing the user's privacy in a widely used recommender application while maintaining recommender performance.

C. T. Dinh,et al. "Personalized federated learning with moreau envelopes," [10]. Federated learning (FL) is a decentralized and privacy-preserving machine learning technique in which a group of clients collaborate with a server to learn a global model without sharing clients' data. One challenge associated with FL is statistical diversity among clients, which restricts the global model from delivering good performance on each client's task. To address this, propose an algorithm for personalized FL (pFedMe) using Moreau envelopes as clients' regularized loss functions, which help decouple personalized model optimization from the global model learning in a bi-level problem stylized for personalized FL. Theoretically, show that pFedMe's convergence rate is state-of-the-art: achieving quadratic speedup for strongly convex and sublinear speedup of order 2/3 for smooth nonconvex objectives. Experimentally, verify that pFedMe excels at empirical performance compared with the vanilla FedAvg and Per-FedAvg, a meta-learning based personalized FL algorithm.

Y. Jiang, et al. , "Improving feder-ated learing personalization via model agnostic meta learning," [11] . Federated Learning (FL) refers to learning a high quality global model based on decentralized data storage, without ever copying the raw data. A natural scenario arises with data created on mobile phones by the activity of their users. Given the typical data heterogeneity in such situations, it is natural to ask how can the global model be personalized for every such device, individually. In this work, point out that the setting of Model Agnostic Meta Learning (MAML), where one optimizes for a fast, gradient-based, few-shot adaptation to a heterogeneous distribution of tasks, has a

number of similarities with the objective of personalization for FL. present FL as a natural source of practical applications for MAML algorithms, and make the following observations. 1) The popular FL algorithm, Federated Averaging, can be interpreted as a meta learning algorithm. 2) Careful fine-tuning can yield a global model with higher accuracy, which is at the same time easier to personalize. However, solely optimizing for the global model accuracy yields a weaker personalization result. 3) A model trained using a standard data center optimization method is much harder to personalize, compared to one trained using Federated Averaging, supporting the first claim. These results raise new questions for FL, MAML, and broader ML research.

A. Fallah, et al., "Personalized federated learning with theoretical guarantees: A model-agnostic meta learning approach,"[12]. In Federated Learning, aim to train models across multiple computing units (users), while users can only communicate with a common central server, without exchanging their data samples. This mechanism exploits the computational power of all users and allows users to obtain a richer model as their models are trained over a larger set of data points. However, this scheme only develops a common output for all the users, and, therefore, it does not adapt the model to each user. This is an important missing feature, especially given the heterogeneity of the underlying data distribution for various users. In this paper, study a personalized variant of the federated learning in which our goal is to find an initial shared model that current or new users can easily adapt to their local dataset by performing one or a few steps of gradient descent with respect to their own data. This approach keeps all the benefits of the federated learning architecture, and, by structure, leads to a more personalized model for each user. show this problem can be studied within the Model-Agnostic Meta-Learning (MAML) framework. Inspired by this connection, study a personalized variant of the well-known Federated Averaging algorithm and evaluate its performance in terms of gradient norm for non-convex loss functions. Further, characterize how this performance is affected by the closeness of underlying distributions of user data, measured in terms of distribution distances such as Total Variation and 1-Wasserstein metric.

V. Smith, et al., "Federated multi-task learning," [13] . Federated learning poses new statistical and systems challenges in training machine learning models over distributed networks of devices. In this work, show that multi-task learning is naturally suited to

handle the statistical challenges of this setting, and propose a novel systems-aware optimization method, MOCHA, that is robust to practical systems issues. Our method and theory for the first time consider issues of high communication cost, stragglers, and fault tolerance for distributed multi-task learning. The resulting method achieves significant speedups compared to alternatives in the federated setting, as demonstrate through simulations on real-world federated datasets.

P. K. Mudrakarta, et al. , "K for the price of 1: Parameter efficient multi-task and transfer learning," [14] . introduce a novel method that enables parameter-efficient transfer and multi-task learning with deep neural networks. The basic approach is to learn a model patch - a small set of parameters - that will specialize to each task, instead of fine-tuning the last layer or the entire network. For instance, show that learning a set of scales and biases is sufficient to convert a pretrained network to perform well on qualitatively different problems (e.g. converting a Single Shot MultiBox Detection (SSD) model into a 1000-class image classification model while reusing 98% of parameters of the SSD feature extractor). Similarly, show that re-learning existing low-parameter layers (such as depth-wise convolutions) while keeping the rest of the network frozen also improves transfer-learning accuracy significantly. Our approach allows both simultaneous (multi-task) as well as sequential transfer learning. In several multi-task learning problems, despite using much fewer parameters than traditional logits-only fine-tuning, match single-task performance.

D. Leroy, et al., "Federated learning for keyword spotting," [15] . propose a practical approach based on federated learning to solve out-of-domain issues with continuously running embedded speech-based models such as wake word detectors. conduct an extensive empirical study of the federated averaging algorithm for the "Hey Snips" wake word based on a crowdsourced dataset that mimics a federation of wake word users. empirically demonstrate that using an adaptive averaging strategy inspired from Adam in place of standard weighted model averaging highly reduces the number of communication rounds required to reach our target performance. The associated upstream communication costs per user are estimated at 8 MB, which is a reasonable in the context of smart home voice assistants. Additionally, the dataset used for these experiments is being open sourced with the aim of fostering further transparent research in the application of federated learning to speech data.

S. Reddi et al., "Adaptive federated optimization,"[16]. Federated learning is a distributed machine learning paradigm in which a large number of clients coordinate with a central server to learn a model without sharing their own training data. Standard federated optimization methods such as Federated Averaging (FedAvg) are often difficult to tune and exhibit unfavourable convergence behaviour. In non-federated settings, adaptive optimization methods have had notable success in combating such issues. In this work, propose federated versions of
adaptive optimizers, including Adagrad, Adam, and Yogi, and analyze their convergence in the presence of heterogeneous data for general non-convex settings. Our results highlight the interplay between client heterogeneity and communication efficiency. also perform extensive experiments on these methods and show that the use of adaptive optimizers can significantly improve the performance of federated learning
.

**Y. Huang et al., "Personalized cross-silo federated learning on non-IID data,"[17]
.** Non-IID data present a tough challenge for federated learning. In this paper, explore a novel idea of facilitating pairwise collaborations between clients with similar data. propose FedAMP, a new method employing federated attentive message passing to facilitate similar clients to collaborate more. establish the convergence of FedAMP for both convex and non-convex models, and propose a heuristic method to further improve the performance of FedAMP when clients adopt deep neural networks as personalized models. Our extensive experiments on benchmark data sets demonstrate the superior performance of the proposed methods. Non-IID data present a tough challenge for federated learning. In this paper, explore a novel idea of facilitating pairwise collaborations between clients with similar data. propose FedAMP, a new method employing federated attentive message passing to facilitate similar clients to collaborate more. establish the convergence of FedAMP for both convex and non-convex models, and propose a heuristic method to further improve the performance of FedAMP when clients adopt deep neural networks as personalized models. Our extensive experiments on benchmark data sets demonstrate the superior performance of the proposed methods. Federated learning (Yang et al. 2019) facilitates collaborations among a set of clients and preserves their privacy so that the clients can achieve better machine learning performance than individually working alone. The underlying idea is to collectively learn from data from all clients. The initial idea of federated learning starts

from aggregating models from clients to achieve a global model so that the global model can be more general and capable. The effectiveness of this global collaboration theme that is not differentiating among all clients highly depends on the data distribution among clients. It works well on IID data, that is, clients are similar to each other in their private data distribution. In many application scenarios where collaborations among clients are needed to train machine learning models, data are unfortunately not IID. For example, consider the cases of personalized cross-silo federated learning (Kairouz et al. 2019), where there are tens or hundreds of clients and the private data of clients may be different in size, class distributions and even the distribution of each class. Global collaboration without considering individual private data often cannot achieve good performance for individual clients. Some federated learning methods try to fix the problem by conducting an additional fine-tuning step after a global model is trained (Ben-David et al. 2010; Cortes and Mohri 2014; Mansour et al. 2020; Mansour, Mohri, and Rostamizadeh 2009; Schneider and Vlachos 2020; Wang et al. 2019).While those methods work in some cases, they cannot solve the problem systematically as demonstrated in our experimental results (e.g., data set CIFAR100 in Table 3). argue that the fundamental bottleneck in personalized cross-silo federated learning with non-IID data is the misassumption of one global model can fit all clients. Consider the scenario where each client tries to train a model on customers' sentiments on food in a country. Different clients collect data in different countries. Obviously, customers' reviews on food are likely to be related to their cultures, life-styles, and environments. Unlikely there exists a global model universally fitting all countries. Instead, pairwise collaborations among countries that share similarity in culture, life-styles, environments and other factors may be the key to accomplish good performance in personalized cross-silo federated learning with non-IID data. Carrying the above insight, in this paper, tackle the challenging personalized cross-silo federated learning problem by a novel attentive message passing mechanism that adaptively facilitates the underlying pairwise collaborations between clients by iteratively encouraging similar clients to collaborate more. make several technical contributions. propose a novel method federated attentive message passing (FedAMP) whose central idea is the attentive message passing mechanism. FedAMP allows each client to own a local personalized model, but does not use a single global model on the cloud server to conduct collaborations. Instead, it maintains a personalized cloud model on the cloud server for each client, and realizes the attentive message passing mechanism by

attentively passing the personalized model of each client as a message to the personalized cloud models with similar model parameters. Moreover, FedAMP updates the personalized cloud model of each client by a weighted convex combination of all the messages it receives. This adaptively facilitates the underlying pairwise collaborations between clients and significantly improves the effectiveness of collaboration. prove the convergence of FedAMP for both convex and non-convex personalized models. Furthermore, propose a heuristic method to further improve the performance of FedAMP on clients using deep neural networks as personalized models. conduct extensive experiments to demonstrate the superior performance of the proposed methods.

C. Dinh et.al, "Fedu: A unified framework for federated multi-task learning with laplacian regularization,"[18]. Federated multi-task learning (FMTL) has emerged as a natural choice to capture the statistical diversity among the clients in federated learning. To unleash the potential of FMTL beyond statistical diversity, formulate a new FMTL problem FedU using Laplacian regularization, which can explicitly leverage relationships among the clients for multi-task learning. first show that FedU provides a unified framework covering a wide range of problems such as conventional federated learning, personalized federated learning, few-shot learning, and stratified model learning. then propose algorithms including both communication-centralized and decentralized schemes to learn optimal models of FedU. Theoretically, show that the convergence rates of both FedU's algorithms achieve linear speedup for strongly convex and sublinear speedup of order 1/2 for nonconvex objectives. While the analysis of FedU is applicable to both strongly convex and nonconvex loss functions, the conventional FMTL algorithm MOCHA, which is based on CoCoA framework, is only applicable to convex case. Experimentally, verify that FedU outperforms the vanilla FedAvg, MOCHA, as well as pFedMe and Per-FedAvg in personalized federated learning Recently, federated learning (FL) has been considered as a promising distributed and privacy-preserving method for building a global model from a massive number of handheld devices (Kairouz et al., 2021; McMahan et al., 2017). FL has a wide range of futuristic applications, such as detecting the symptoms of possible diseases (e.g., stroke, heart attack, diabetes) from wearable devices in health-care systems (Brisimi et al., 2018; Rieke et al., 2020; Xu & Wang, 2020), or predicting disaster risks from internet-of-things devices in smart cities (Ahmed et al., 2020; Jiang

et al., 2020). In FL, one of the key challenges is statistical diversity, i.e., the naturally distinct (non-i.i.d) data distributions among clients (Haddadpour & Mahdavi, 2019; Karimireddy et al., 2020). The generalization error of the FL global model on each client's local data significantly increases when the statistical diversity increases (Deng et al., 2020; Li & Wang, 2019). As solutions, personalized federated learning (Dinh et al., 2020; Fallah et al., 2020) and federated multi-task learning (FMTL) (Smith et al., 2017) have been proposed to handle the statistical diversity in FL. Personalized FL aims to build a global model which is then leveraged to find a "personalized model" that is stylized for each client's data. Here, the global model is considered as an "agreed point" for each client to start personalizing its local model based on its heterogeneous data distribution. Motivated by multi-task learning frameworks (Kumar & Daume´, 2012; Zhang & Yeung, 2010), FMTL directly addresses the statistical diversity challenge in FL by learning simultaneously separate models for each client. In FMTL, the clients' models are separated but normally correlated, since the clients with similar features (e.g, location, time, age, gender) are likely to share similar behaviors. The relationships among the clients' models are captured by a regularization term which is minimized to encourage the correlated clients' models to be mutually impacted. Unfortunately, the conventional FMTL framework MOCHA (Smith et al., 2017) is limited to only applications having convex objective functions and a communication-centralized mode. Therefore, proposing a new framework to fully explore the potential of FMTL is the ultimate goal of our work.

H. Jiang et.al, "Distributed deep learning optimized system over the cloud and smart phone devices,"[19] . Edge devices are currently used for various applications in many areas including transportation and healthcare [1– 4]. These applications often deploy machine learning (ML) frameworks using data collected by the edge devices' sensors. ML models have transformed into more complex and larger Deep Neural Networks (DNNs). These DNNs are memory and computationally expensive in training due to their complexity and size. On the contrary, an edge device usually does not have sufficient memory or computation resources to conduct the entire DNN model training job. Thus, a DNN is usually trained (or updated) in the cloud, then compressed and deployed on the edge nodes for inference. Such cloud-based training can generate significant delays when the network is intermittent (e.g., disaster, network congestion), and cannot provide data privacy protection [5] for sensitive applications (e.g., medical

records) as the data needs to be transferred to the cloud. Inthis case, distributing the job of training or updating a DNN model among only edge nodes becomes a promising solution. Recent works [6, 5, 7] for distributed training on edges handle either data parallelism or model parallelism while involving the cloud at a certain stage. Data parallelism methods [5, 7] deploy replicas of an entire neural network on the edges, and these edges have their subsets of training data. Each edge processes its training data subset and synchronizes model parameters in a parameter server running on an initiator edge or the cloud. Model parallelism methods [6] divide a neural network and distribute the shallow (earlier) layers to edges and the deep layers to the cloud and let edges communicate with cloud for model parameters transfer from the previous layer transfer to the next layer. The data parallelism methods sometimes may not be feasible as deploying a large DNN model on a single edge may not be feasible. In contrast, the model parallelism methods suffer from the long delay of communication and intermittent network between edges and cloud. A concurrent data and model parallelism based deep learning (DL) system can handle these problems. In such a system [8], clusters of edges are created according to geographical locations, and each cluster trains a replica of the DL model using model parallelism based on its locally collected data. Each cluster has a cluster head that has relatively high capacity, and it assigns the partitions of a DNN model (i.e., tasks) to the cluster edge nodes with a goal of minimizing training time. Each edge within a cluster collects its own data and sends the data to the first-layer node, which collects the sensed data from all cluster edges as the training data of the model replica in the cluster. The cluster head assigns tasks to the edges based on the resource demands of the tasks and the available resources of the edges. Thus, the cluster head needs to continuously observe the workload conditions of all the edge nodes in its cluster. With this cluster-wide knowledge, the cluster head can avoid overloading the edges in assigning the tasks. However, such a centralized scheduling method imposes a significant workload on the cluster head, which ultimately impacts the performance of the training. Recently, RL [9, 10] has also been used for such scheduling in the recent times with similar high load. To lessen this load, first propose a multi-agent RL method (MARL) that enables each edge node to schedule its own jobs among its neighboring edge nodes (i.e., edge nodes in its transmission range) using RL. In MARL, each edge device works as an independent agent and makes the scheduling decision among its nearby edges, thus relieving the cluster head from the extra burden. However, without the coordination between the edge nodes, action

collision may occur, in which multiple nodes may schedule tasks to the same node and make it overloaded. To avoid this problem, propose Shielded ReinfOrcement learning based DL Training on Edges (SROLE) on top of the MARL-based assignment approach. The shielding approach [11] works as a separate monitor that suggests alternative actions to avoid action collision by observing the states and actions that will be taken by the agents. In SROLE, each edge node schedules its own jobs using MARL, and the shield, which is deployed on the cluster head, checks the action collisions among the schedules of the edges in its cluster. Edge nodes report to the shield their action decisions, and it checks action collisions and provides alternative actions to avoid the collisions. However, the computational cost of a centralized shield grows dramatically with the number of edges in a cluster, and it may become a bottleneck for the entire cluster. Thus, further propose a decentralized shielding method, in which different shields are responsible for different regions in the cluster and they coordinate to avoid action collisions on the region boundaries. Specifically, a large cluster is divided into multiple sub-clusters according to the geographical proximity, and a shield monitors the edges within each sub-cluster and communicates with its neighboring shields for the edges at the boundary of the sub-clusters to avoid action collisions, i.e., unsafe actions. The computational cost of each shield in the decentralized method is lower than that in the centralized shielding as the centralized shield's workload is distributed to a number of shields. In summary, the contributions of this work are as follows: To avoid overloading a cluster head due to scheduling DL training jobs in its cluster, initially propose a multiagent RL-based method (MARL) that enables each edge node to use RL to schedule its own jobs. For a given DL training job, an edge node makes scheduling decisions for DNN partitions among its nearby edges depending on their resource availability to minimize training time.To avoid action collisions in MARL, use the shielding approach in each cluster. The shield in a cluster collects the scheduling decisions of all edge nodes in the cluster, checks the action collisions and provides alternative actions to avoid the action collisions. To avoid overloading a shield in a cluster, they distribute the shielding workload to multiple shields in a cluster and each shield is responsible for a sub-cluster. The neighboring shields communicate with each other to avoid action collisions from the edges on the boundaries of sub-clusters.measured the performance of the SROLE system on container based emulation on Amazon EC2 instances. Our evaluation shows that the SROLE system shows up to 59% reduction in training time and up to 48% reduction in the number of action

collisions compared to the centralized RL and MARL approach. Our real device experiments also show up to 53% reduction in training time and up to 46% reduction in the number of action collisions in comparison with the centralized RL and MARL approach. The rest of the paper is organized as follows. Section II presents the related work. Section IV describes our SROLE system. Section V presents the performance evaluation. Finally, Section VI concludes the paper with remarks on our future work.Researchers have been studying federated ML training and DNN partition distribution across cloud/fog and edge devices [6, 7, 12–17]. In federated learning [7], edge devices update a trained model on the cloud. Individual edges download a replica of the model and update the models using their own available datasets. Finally, the updated models from individual edges are aggregated through averaging or using a control theorem on the cloud to produce the final model. Many proposed ML inference approaches partition the ML model distributed edge nodes [18–20]. The works in [18, 20] simply consider each or multiple convolution layers as a partition, and the work [19] vertically divides the convolutional layers in a convolutional neural network (CNN). The work in [6] distributes DNN partition across cloud/fog and edge devices to accelerate training or inference. The resource-constraint edge devices run lighter (earlier) layers of the model and the cloud or fog run heavier (later) layers of the model. Resilinet [21] achieves failure-resilient inference in model-parallel ML at the edge. Data parallelism training methods distribute training data among different edges for training and accumulate training updates. Wang et al. [5] analyzed the convergence rate of replicas for ML models such as Support Vector Machine (SVM) and K-means, and accordingly proposed a control method that dynamically adjusts the frequency of global update from each replica to the ML initiator in real time to minimize the learning loss under a fixed computation resource budget for the edges. To efficiently run ML models on edge devices in terms of inference time, energy and memory, researchers have introduced techniques for compressing the neural networks (NNs) [22– 35].

K. Bonawitz et al., “Towards federated learning at scale: System design,” [20] . Federated Learning (FL) (McMahan et al., 2017) is a distributed machine learning approach which enables training on a large corpus of decentralized data residing on devices like mobile phones. FL is one instance of the more general approach of “bringing the code to the data, instead of the data to the code” and addresses the fundamental problems of privacy, ownership, and locality of data. The general

description of FL has been given by McMahan & Ramage (2017), and its theory has been explored in Konecny et al. (2016) McMahan et al. (2017; 2018). A basic design decision for a Federated Learning infrastructure is whether to focus on asynchronous or synchronous training algorithms. While much successful work on deep learning has used asynchronous training, e.g., Dean et al. (2012), recently there has been a consistent trend towards synchronous large batch training, even in the data center (Goyal et al., 2017; Smith et al., 2018). The Federated Averaging algorithm of McMahan et al. (2017) takes a similar approach. Further, several approaches to enhancing privacy guarantees for FL, including differential privacy (McMahan et al., 2018) and Secure Aggregation (Bonawitz et al., 2017), essentially require some notion of synchronization on a fixed set of devices, so that the server side of the learning algorithm only consumes a simple aggregate of the updates from many users. For all these reasons, chose to focus on support for synchronous rounds, while mitigating potential synchronization overhead via several techniques describe subsequently. Our system is thus amenable to running large-batch SGD-style algorithms as well as Federated Averaging, the primary algorithm run in production; pseudo-code is given in Appendix B for completeness. In this paper, it was reported on a system design for such algorithms in the domain of mobile phones (Android). This work is still in an early stage, and do not have all problems solved, nor are able to give a comprehensive discussion of all required components. Rather, attempt to sketch the major components of the system, describe the challenges, and identify the open issues, in the hope that this will be useful to spark further systems research. Our system enables one to train a deep neural network, using TensorFlow (Abadi et al., 2016), on data stored on the phone which will never leave the device. The weights are combined in the cloud with Federated Averaging, constructing a global model which is pushed back to phones for inference. An implementation of Secure Aggregation (Bonawitz et al., 2017) ensures that on a global level individual updates from phones are uninspectable. The system has been applied in large scale applications, for instance in the realm of a phone keyboard. Our work addresses numerous practical issues: device availability that correlates with the local data distribution in complex ways (e.g., time zone dependency); unreliable device connectivity and interrupted execution; orchestration of lock-step execution across devices with varying availability; and limited device storage and compute resources. These issues are addressed at the communication protocol, device, and server levels. have reached a state of maturity sufficient to deploy the system in production and solve

applied learning problems over tens of millions of real-world devices; anticipate uses where the number of devices reaches billions.

M. Duan et.al, "Self-balancing federated learning with global imbalanced data in mobile systems,"[21] . Federated Learning (FL) is a promising distributed neural network training approach for deep learning applications such as image classification [1] and nature language process [2]. FL enables the mobile devices to collaboratively train a shared neural network model with the training data distributed on the local devices. In a FL application, any mobile device can participate in the neural network model training task as a client. Each client independently trains the neural network model based on its local data. A FL server then averages the models' updates from a random subset of FL clients and aggregates them into a new global model. In this way, FL not only ensures privacy, costs lower latency, but also makes the mobile applications adaptive to the changes of its data. Nevertheless, a main challenge of the mobile federated learning is that the training data is unevenly distributed on the mobile devices, which results in low prediction accuracy. Several efforts have been made to tackle the challenge. McMahan et al. propose a communication-efficient FL algorithmFederated Averaging (FedAvg) [3], and show that the CNN model trained by FedAvg can achieve 99% test accuracy on non-IID MNIST dataset, i.e., any particular user of the local dataset is not representative of the population distribution. Zhao et al. [4] point out that the CNN model trained by FedAvg on non-IID CIFAR-10 dataset has 37% accuracy loss. Existing studies assume that the expectation of the global data distribution is balanced even though the volume of data on the devices may be disproportionate. In most real scenarios of distributed mobile devices, however, the global data distribution is imbalanced. In this paper, consider one more type of imbalanced distribution, named global imbalanced, of distributed training data. In global imbalanced distribution, the collection of distributed data is class imbalanced. draw a global imbalanced subset from EMNIST dataset and explore its impact on the accuracy of FL in Section II-B. The experimental results show that the global imbalanced training data leads to 7.92% accuracy loss for FedAvg. The accuracy degradation caused by imbalances drives us to design a novel self-balancing federated learning framework, called Astraea. The Astraea framework counterweighs the training of FL with imbalanced datasets by two strategies. First, before training the model, Astraea performs data augmentation [5] to alleviate global imbalance. Second, Astraea proposes to use some mediators to

reschedule the training of clients according to the KLD between the mediators and the uniform distribution. By combining the training of skewed clients, the mediators may be able to achieve a new partial equilibrium. With the above methods, Astraea improves 5.59% top-1 accuracy on the imbalanced EMNIST and 5.89% on imbalanced CINIC-10 [6] over FedAvg. Our rescheduling strategy can significantly reduce the impact of local imbalance and decrease the mean of the KLD between the mediators and the uniform distribution to below 0.2. The proposed framework is also communication-efficient. For example, the experimental results show that Astraea can reduce 92% communication traffic than that of FedAvg in achieving 75% accuracy on imbalanced EMNIST. The main contributions of this paper are summarized as follows.

2.2 Limitations of Existing Techniques

1. Limited Personalization:

Existing FL algorithms often prioritize optimizing the global model accuracy, neglecting the need for personalized model accuracy tailored to individual users. While some personalized FL algorithms have been proposed, they may suffer from scalability issues and computational overhead due to the complexity of maintaining separate models for each user or performing fine-tuning steps.

2. Non-IID Data Challenges:

FL operates in a decentralized environment where each client holds its own dataset, characterized by non-identical and non-distributed (non-IID) data distributions. This heterogeneity among client datasets poses significant challenges for model training, as traditional FL algorithms may struggle to effectively aggregate information from diverse data sources. Consequently, convergence speed and model performance may vary across clients, leading to suboptimal results.

3. Privacy Concerns:

Preserving user privacy is a fundamental aspect of FL, as it involves training models on decentralized data without exposing sensitive information to external parties. However, ensuring privacy while effectively training models on distributed data

remains a challenging task. Existing FL algorithms may not provide robust privacy guarantees, potentially exposing user data during model aggregation or updates.

4. Communication Overhead:

FL relies on communication between clients and a central server for model aggregation and updates. However, the communication overhead incurred by transmitting model parameters and updates over wireless networks can be substantial, particularly in edge computing environments with limited bandwidth and high latency. This communication overhead can hinder the scalability and efficiency of FL systems, especially when dealing with a large number of clients or data-intensive tasks.

5. Convergence Speed:

Traditional FL optimization algorithms, such as Federated Averaging (FedAvg), may suffer from slow convergence due to the decentralized nature of training and the challenges posed by non-IID data distributions. Slow convergence can prolong the training process and increase resource consumption, limiting the applicability of FL in time-sensitive applications or environments with strict performance requirements.

6. Edge Computing Challenges:

Deploying FL in edge computing environments introduces additional challenges related to resource constraints, network connectivity, and scalability. Existing FL systems may not be optimized for edge computing architectures, leading to inefficiencies and performance bottlenecks. Moreover, edge devices often have limited computational capabilities and storage capacity, which may impact the feasibility and effectiveness of FL training tasks.

7. Lack of Standardization:

The FL research community lacks standardized frameworks and evaluation metrics, making it challenging to compare and replicate results across studies. Additionally, the diversity of FL applications and deployment scenarios requires tailored solutions, further complicating the development of universal FL algorithms. The absence of standardized practices may hinder collaboration and hinder the widespread adoption of FL in various domains.

Addressing these limitations requires innovative approaches that prioritize personalized model accuracy, ensure robust privacy protection, optimize communication efficiency, enhance convergence speed, and adapt FL algorithms for edge computing environments. The proposed Multi-Task FL (MTFL) algorithm aims to overcome these challenges by introducing personalized training techniques, optimizing communication protocols, and leveraging edge computing capabilities to enhance the efficiency and effectiveness of FL systems.

CHAPTER 3

SYSTEM ANALYSIS

3.1 Requirements Specification

Requirements analysis, also called requirements engineering is the process of determining user expectations for a new or modified product. These features called requirements must be quantifiable, relevant and detailed. In software engineering, such requirements are often called functional specifications. Requirements analysis is critical to the success or failure of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

The requirements specification for the project "Multi-Task Federated Learning for Personalized Deep Neural Networks in Edge Computing" would involve detailing the specific needs, objectives, and constraints of the project. Given the complexity and technical nature of the project, the requirements specification would likely encompass several key elements: project objectives, functional requirements, nonfunctional requirements, data requirements, model evaluation, user requirements, implementation constraints, ethical and legal considerations, timeline and milestones, budget and resource allocation.

3.1.1 Functional Requirements

The system must effectively handle and partition datasets like MNIST and CIFAR10 for experimentation purposes. This involves implementing a data partitioning scheme that ensures a non-IID distribution across clients, adhering to the specified methodology. Additionally, the system should support the configuration of experiments with various parameters, including the number of clients (W), client participation rates (C), and optimization strategies. It should allow for the execution of experiments across multiple trials with different random seeds to ensure the reliability and robustness of the results.

For model training and optimization, the system needs to facilitate the training of deep neural network architectures, such as the 2NN for MNIST and CNN for CIFAR10, with specified layers and configurations. It should also provide support for different optimization algorithms, including FedAvg, FedAvg-Adam, and FedAdam, and allow for the tuning of hyperparameters for each experimental scenario. Additionally, the system must implement mechanisms for Model Transfer with Federated Learning (MTFL) to enable the privacy-preserving transfer of certain model parameters, such as BN-layer statistics and trainable parameters. It should ensure that these privacy-preserving techniques do not significantly hinder the learning process or degrade performance.

The system should include tools for analyzing experiment results, including the number of rounds required to reach the target average UA, the impact of private BN values, and comparisons between FL and MTFL with different optimization strategies. It should support the visualization of training and testing accuracies over communication rounds to provide insights into the trade-offs and effects of different configurations. Furthermore, the system should evaluate the performance of different FL and MTFL setups in terms of convergence speed, robustness to noisy clients, and the ability to reach target accuracies.

In addition to virtual experiments, the system should be capable of simulating real-world scenarios, such as a Mobile Edge Computing (MEC) environment with heterogeneous client devices like Raspberry Pi. This involves providing insights into the performance of FL and MTFL in such environments, considering factors like device capabilities, computational cost, and communication overhead. The system should analyze the computational cost per round, including the time spent on communication, local training, and model uploading/downloading, to provide a comprehensive understanding of system performance.

3.1.2 Non-Functional Requirements

A Non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions.

1. The model should be easy to develop and reliable.
2. The system should be able to scale to the increasing dataset.

3. The system should be able to perform various algorithms effectively.

4. The computer system must have the ability to adapt to increased demands.

Software Requirements

- Operating System : Windows
- Language : Python3

Hardware Requirements

- Processor - Pentium IV or higher
- Speed – 2.4GHz
- RAM - 256 MB(min)
- Hard Disk - 512 MB(min)

3.2 UML Diagrams for the Project Work

3.2.1 System view diagram for MTFL

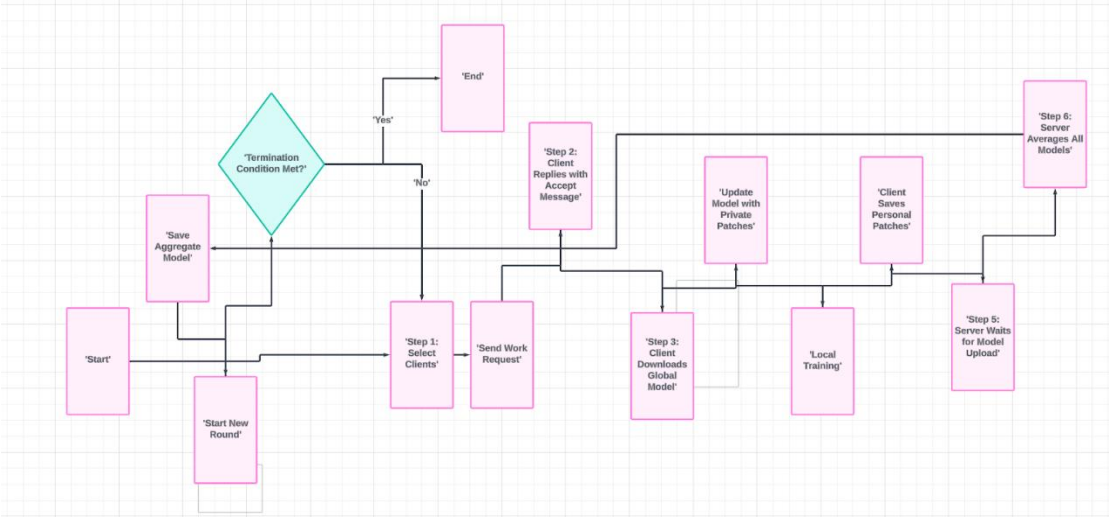

Fig: 3.2.1 System view diagram for MTFL

The MTFL algorithm operates in rounds within an Edge Computing environment. Initially, the server selects a subset of clients and sends them a work request, to which clients respond based on their current physical state and local preferences. Subsequently, clients download the global model from the server and incorporate

private patches, such as Batch Normalization layers. Following this, clients conduct local training using their updated models. Once local training is completed, clients upload their non-private model and optimizer values to the server, or until a predetermined time limit is reached. Finally, the server aggregates all received models, saves the aggregate, and initiates a new round, thus perpetuating the iterative process of model refinement and aggregation.

3.2.2 Detailed view diagram for MTFL

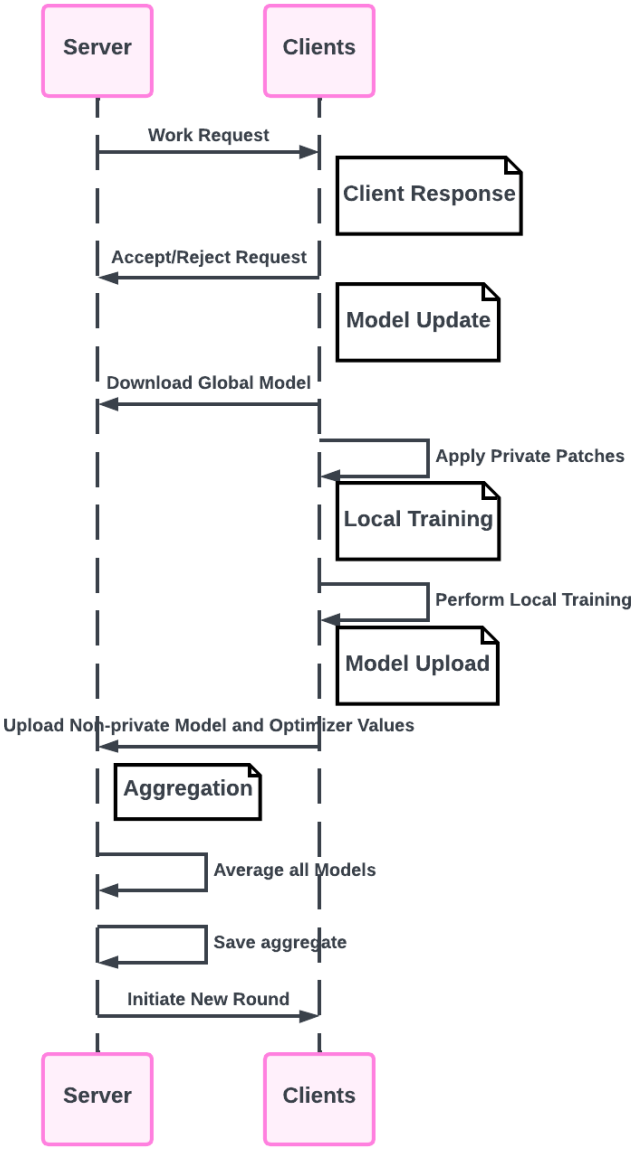

Fig: 3.2.2 Detailed view diagram for MTFL

In the context of Edge Computing, the operation of the MTFL (Multi-Task Federated Learning) algorithm involves a series of steps conducted in rounds until a termination condition is met. Initially, the server selects a subset of clients from its database to participate in the round and sends them a work request. Subsequently, clients respond with an accept message based on their physical state and local preferences. Upon acceptance, clients download the global model along with any optimization parameters from the server and update their copy of the global model with private patches, typically implemented using techniques like Batch Normalization (BN) layers. Following this, clients perform local training using their updated models before saving their personal patches for the next round. Meanwhile, the server awaits a fraction of clients, denoted as C, to upload their non-private model and optimizer values or until a predefined time limit is reached. Upon receiving the required fraction of uploads or reaching the time limit, the server averages all models, saves the aggregate, and initiates a new round of training. This iterative process of communication and model aggregation facilitates collaborative learning while respecting privacy constraints and leveraging edge computing resources effectively.

CHAPTER 4

SYSTEM DESIGN

4.1 Architecture of the Proposed System

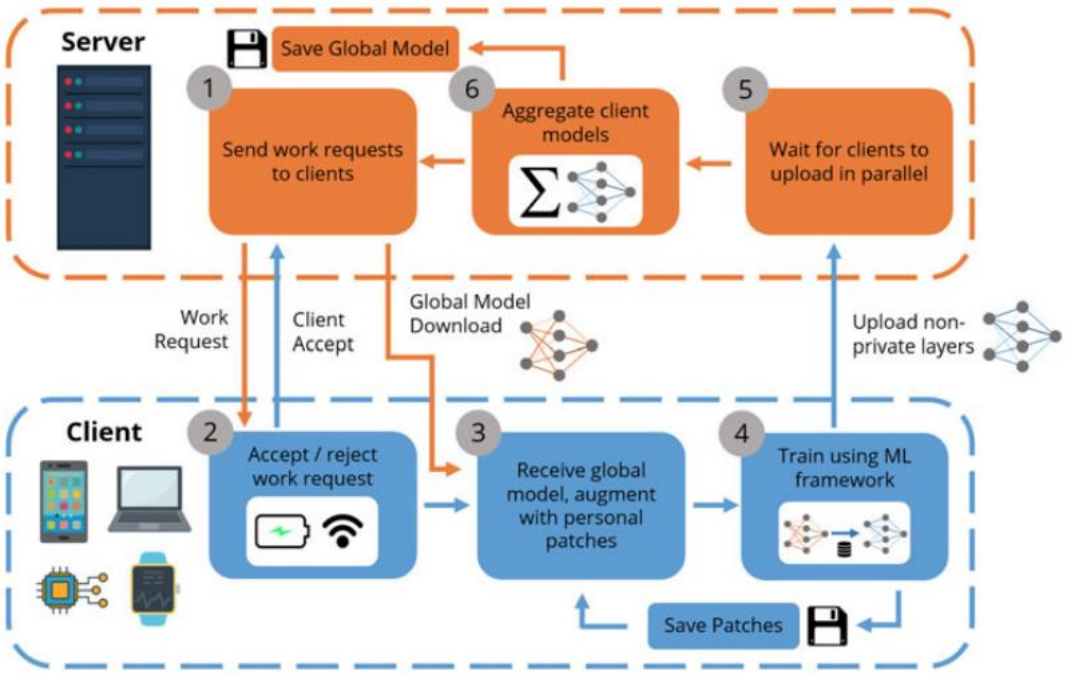

Fig 4.1 Operation of the MTFL algorithm in Edge Computing. Training is performed in rounds until a termination condition is met.

The operation of the MTFL (Model Transfer with Federated Learning) algorithm follows a structured process. Initially, the server selects a subset of clients from its database to engage in the training round, issuing work requests to them. Subsequently, clients respond with an acceptance message based on their physical state and local preferences. Upon acceptance, clients retrieve the global model, along with any optimization parameters, from the server. They then integrate their private patches, typically represented by Batch Normalization (BN) layers, into the global model. Following this, clients conduct local training using their updated model, saving their personal patches for subsequent rounds. Meanwhile, the server awaits the upload of models and optimizer values from a specified fraction of clients or until a time limit is reached. Upon receipt, the server aggregates the models, calculates the average, and initiates a new training round for further iteration. This structured workflow ensures

collaborative model training while preserving privacy in distributed Edge Computing environments.

4.2 Module Description

- User model accuracy and MTFL
- Batch normalization
- Federated optimization within MTFL

A . User model accuracy and MTFL :

In many FL works, such as the original FedAvg paper [5], the authors use a central IID test-set to measure FL performance. Depending on the FL scenario, this metric may or may not be desirable. If the intention is to create a single model that has good performance on IID data, then this method would be suitable. However, in many FL scenarios, the desire is to create a model that has good performance on individual user devices. For example, Google have used FedAvg for their GBoard next-word-prediction software [8]. The objective was to improve the prediction score for individual users. As users do not typically have non-IID data, a single global model may display good performance for some users, and worse performance for others.

Propose using the average User model Accuracy (UA) as an alternative metric of FL performance. UA is the accuracy on a client using a local test-set. This test-set for each client should be drawn from a similar distribution as its training data. In this paper, perform experiments on classification problems, but UA could be altered for different metrics (e.g., error, recall).

4.3 Workflow of the Proposed System

The MTFL algorithm is based on the client-server framework, however, rounds are initiated by the server, as shown in Fig. 1. First, the server selects all, or a subset of all, known clients from its database and asks them to participate in the FL round (Step 1), and sends a Work Request message to them. Clients will accept a Work Request depending on user preferences (for example, users can set their device to only participate in FL if charging and connected to WiFi). All accepting clients then send an Accept message to the server (Step 2). The server sends the global model (and any

associated optimization parameters) to all accepting clients, who augment their copy of the global model with private patches (Step 3). Clients then perform local training using their own data, creating a different model. Clients save the patch layers from their new model locally, and upload their nonprivate model parameters to the server (Step 4). The server waits for clients to finish training and upload their models (Step 5). It can either wait for a maximum time limit, or for a given fraction of clients to upload before continuing, depending on the server preferences. After this, the server will aggregate all received models to produce a single global model (Step 6) which is saved on the server, before starting a new round. MTFL therefore offloads the vast majority of computation to client devices, who perform the actual model training. It preserves users' data-privacy more strongly than FedAvg and other personalised-FL algorithms: not only is user data not uploaded, but key parts of their local models are not uploaded. The framework also accounts for client stragglers with its round time/uploading client fraction limit. Moreover, MTFL utilises patch layers to improve local model performance on individual users' non-IID datasets, making MTFL more personalised.

CHAPTER 5

IMPLEMENTATION

5.1 Algorithms

Algorithm 1. MTFL

Initialise global model Ω and global optimiser values V
while termination criteria not met **do**
  Select round clients, $S_r \subset S, |S_r| = C \cdot |S|$
  for each client $s_k \in S_r$ in parallel **do**
    Download global parameters $\mathcal{M}_k \leftarrow \Omega$
    Download optimiser values $V_k \leftarrow V$
    for $i \in$ patchIdxs **do** ▷ Apply local patches
      $\mathcal{M}_{k,i} \leftarrow P_{k,i}, V_{k,i} \leftarrow W_{k,i}$
    end for
    for batch b drawn from local data D_k **do**
      $\mathcal{M}_k, V_k \leftarrow \text{LocalUpdate}(\mathcal{M}_k, V_k, b)$
    end for
    for $i \in$ patchIdxs **do** ▷ Save local patches
      $P_{k,i} \leftarrow \mathcal{M}_{k,i}, W_{k,i} \leftarrow V_{k,i}$
    end for
    for each $i \notin$ patchIdxs **do**
      Upload $\mathcal{M}_{k,i}, V_{k,i}$ to server
    end for
  end for
  for $i \notin$ nonPatchIndexes **do**
    $\Omega_i \leftarrow \text{GlobalModelUpdate}(\Omega_i, \{\mathcal{M}_{k,i}\}_{k \in S_r})$
    $V_i \leftarrow \text{GlobalOptimUpdate}(V_i, \{V_{k,i}\}_{k \in S_r})$
  end for
end while

n : total number of samples across all clients

k : loss function on client k

Ω : set of global model parameters

Mk : patched model on client k

Pki : patch parameters

In FL, user data is often non-IID, so users could be considered as having different but related learning tasks. It is possible for an FL scheme to achieve good global-model accuracy, but poor UA, as the aggregate model may perform poorly on some clients' datasets (especially if they have a small number of local samples, so are weighted less in the FedAvg averaging step).The proposed MTFL algorithm that allows clients to build different models, while still benefiting from FL, in order to improve the average UA. Mudrakarta et al. [14] have previously shown that adding small per-task 'patch' layers to DNNs improved their performance in MTL scenarios. Patches are therefore a good candidate for training personalised models for clients. In FL, the aim is to minimise the following object.

$$F_{\mathrm{FL}} = \sum_{k=1}^{K} \frac{n_k}{n} \ell_k(\mathbf{\Omega}),$$

As shown in Algorithm 1, MTFL runs rounds of communication until a given termination criteria (such as target UA) is met (Line 2). At each round, a subset Sr of clients are selected to participate from the set of all clients S (Line 3). These clients download the global model V, which is a tuple of model parameters, and the global optimiser V , if used (Lines 5-6). The clients then update their copy of the global model and optimiser with their private patch layers (Lines 7-9), where the 'patchIdxs' variable contains the indexes of patch layer placement in the DNN. Clients perform training using their now-personalised copy of the global model and optimiser on their local data (Line 10). Depending on the choice of FL optimisation strategy to be used within MTFL, the Local Update function represents local training of the model. For FedAvg, Local Update is simply minibatch-SGD. The discussed and the proposed FedAvg-Adam optimisation strategy, further in Section 3.3. After local training, the updated local patches are saved (Lines 11-13), and the non patch layers and optimiser values are uploaded to the server (Lines 14-16). At the end of the round, the server makes a new global model and optimiser according to the GlobalModelUpdate and

GlobalOptimUpdate functions (Lines 18-20). These functions are again dependent on the FL optimisation strategy used, and are discussed further in Section 3.3. FedAvg, for example, uses a weighted average of client models for GlobalModelUpdate. The updated global model marks the end of the round and a new round is begun. The total per-round computation complexity of MTFL scales with |Sr|, where |Sr| is the number of clients participating per round. The computation performed by each client is independent of the total number of clients. As clients perform local computation in parallel, MTFL (like FedAvg) is eminently scalable. Scalability is important in FL as real-world deployments are expected to have huge numbers of low-powered clients [4], [8]. The global model and optimiser updates (Lines 20-23 in Algorithm 1) depend on the optimisation strategy used. For FedAvg and FedAvgAdam, GlobalModelUpdate is essentially map-reduce (averaging after local training) . For FedAdam, the Adam step following the map-reduce in GlobalOptimUpdate is not dependent on the number of clients (only on the DNN architecture). There are numerous works investigating FL in the PeerTo-Peer (p2p) setting, which is not considered in this paper. Simple p2p FL algorithms involve sending all client models to all participating peers for decentralised aggregation. Extension of MTFL to these schemes is trivial: peers would simply just send/aggregate the non-private layers. More sophisticated p2p FL algorithms may require more complex ways of incorporating private layers – an interesting direction is left for future works.

5.2 Datasets used

1. MNIST Dataset:

The MNIST dataset is a cornerstone in the realm of machine learning and computer vision. Comprising a collection of 28x28 pixel grayscale images, MNIST provides a diverse array of handwritten digits, ranging from 0 to 9. With a total of 10 classes, each corresponding to a specific digit, MNIST serves as an ideal benchmark for digit classification tasks. Its grayscale format encodes pixel intensities, representing shades of gray from black to white. The dataset includes a substantial training set of 60,000 images and a separate test set of 10,000 images, enabling robust model evaluation. Researchers often utilize the MNIST dataset to develop and assess various classification algorithms, particularly in the context of handwritten character recognition.

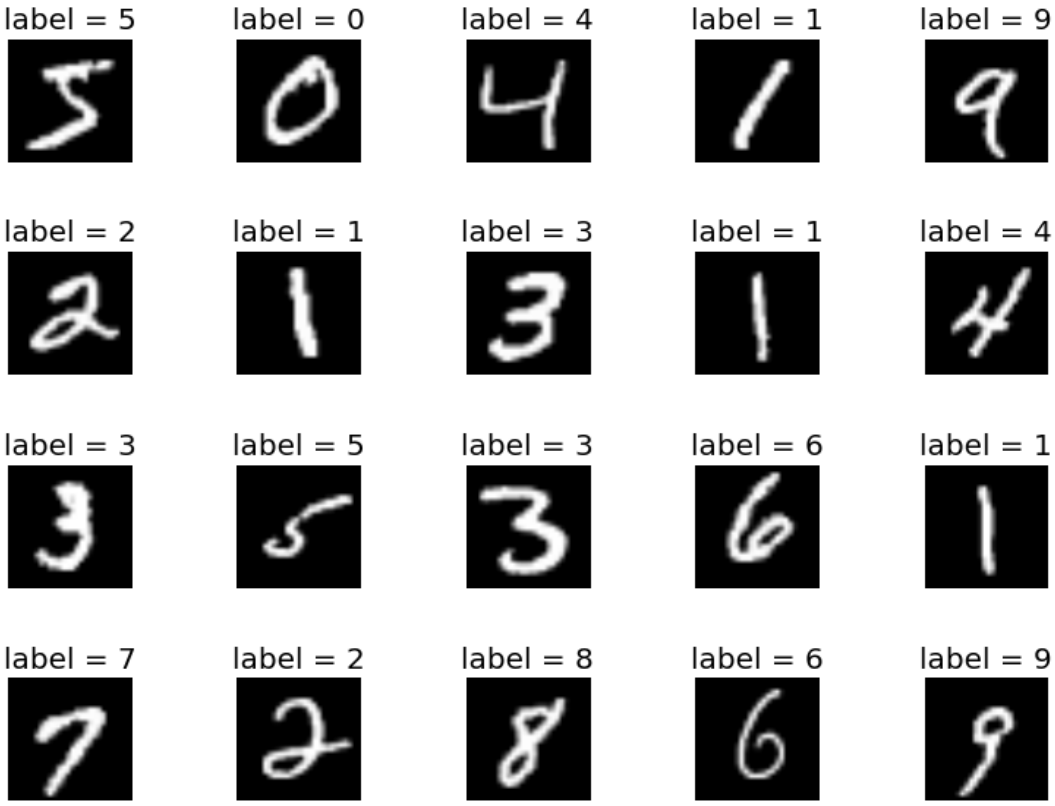

Fig : 5.2.1 MNIST data set images

For experimentation with the MNIST dataset, a network architecture dubbed '2NN' is commonly employed. This architecture typically consists of fully connected (FC) layers, along with batch normalization (BN) layers to enhance training stability. Additionally, a softmax output layer facilitates probability-based classification, enabling models to predict the likelihood of each digit class.

2. CIFAR10 Dataset:

In contrast to MNIST, the CIFAR10 dataset offers a broader scope for image classification tasks. CIFAR10 comprises 32x32 pixel RGB (color) images depicting a variety of everyday objects across 10 distinct classes. These classes encompass common objects such as airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. With its RGB format,

CIFAR10 images contain three color channels—red, green, and blue—providing richer visual information compared to grayscale images. The dataset features a sizable training set of 50,000 images, along with a separate test set of 10,000 images, facilitating comprehensive model evaluation. Researchers commonly leverage CIFAR10 for

testing and comparing the performance of image classification algorithms, particularly those employing convolutional neural network (CNN) architectures. For experimentation with CIFAR10, sophisticated CNN architectures are typically employed. These architectures incorporate convolutional layers, batch normalization, rectified linear unit (ReLU) activation functions, and max-pooling layers to extract and process visual features effectively. Fully connected layers, followed by a softmax output layer, enable the final classification of images into their respective object classes.

Fig: 5.2.2 CIFAR 10 Dataset images

Both MNIST and CIFAR10 datasets play pivotal roles in advancing research and development efforts within the machine learning community. By providing standardized benchmarks for digit and object classification tasks, these datasets facilitate the evaluation and comparison of diverse algorithms and models, driving innovation in the field of computer vision and pattern recognition.

Conducted experiments using two image-classification datasets: MNIST [33] and CIFAR10 [34], and two DNN architectures. MNIST: greyscale images of handwritten digits from 10 classes. The '2NN' network used on this dataset had one Fully Connected (FC) layer of 200 neurons, a BN layer, a second 200-neuron FC layer, and a softmax output layer. CIFAR10 RGB images of objects from 10 classes. The 'CNN' network used on this dataset had one convolutional (conv) layer with 32 filters followed by BN, ReLU and max pooling; a second conv ReLU layer with 64 filters, BN, ReLU and max pooling; a 512 neuron ReLU FC layer; and a softmax output layer. Experiments were run with different numbers of clients W, client participation rates C and optimisation strategies, on non-IID clients. To produce non-IID client data ,the popular approach was

taken from [5]: order the training and testing data by label, split each into 2W shards, and assign each client two shards at random. Using the same assignment indexes for the testing data means that the classes in each client's training set are the same as those in their test set. This splitting produces a strongly non-IID distribution across clients. All results are an average over 5 trials with different random seeds.

5.3 Metrics Calculated

In many FL works, such as the original FedAvg paper [5], the authors use a central IID test-set to measure FL performance. Depending on the FL scenario, this metric may or may not be desirable. If the intention is to create a single model that has good performance on IID data, then this method would be suitable. However, in many FL scenarios, the desire is to create a model that has good performance on individual user devices. For example, Google have used FedAvg for their GBoard next-word-prediction software [8]. The objective was to improve the prediction score for individual users. As users do not typically have non-IID data, a single global model may display good performance for some users, and worse performance for others. It was proposed using the average User model Accuracy (UA) as an alternative metric of FL performance. UA is the accuracy on a client using a local test-set. This test-set for each client should be drawn from a similar distribution as its training data. In this paper, experiments are performed on classification problems, but UA could be altered for different metrics (e.g., error, recall). In FL, user data is often non-IID, so users could be considered as having different but related learning tasks. It is possible for an FL scheme to achieve good global-model accuracy, but poor UA, as the aggregate model may perform poorly on some clients' datasets (especially if they have a small number of local samples, so are weighted less in the FedAvg averaging step). MTFL algorithm was proposed , that allows clients to build different models, while still benefiting from FL, in order to improve the average UA. Mudrakarta et al. [14] have previously shown that adding small per-task 'patch' layers to DNNs improved their performance in MTL scenarios. Patches are therefore a good candidate for training personalised models for clients. In FL, the aim is to minimise the following objective function:

$$F_{\mathrm{FL}} = \sum_{k=1}^{K} \frac{n_k}{n} \ell_k(\Omega),$$

where K is the total number of clients, nk is the number of samples on client k, n is the total number of samples across all clients, ‘k is the loss function on client k, and V is the set of global model parameters. Adding unique client patches to the FL model changes the objective function of MTFL to

$$F_{\mathrm{MTFL}} = \sum_{k=1}^{K} \frac{n_k}{n} \ \ell_k(\mathcal{M}_k)$$

$$\mathcal{M}_k = (\Omega_1 \cdots \Omega_{i_1}, P_{k_1}, \Omega_{i_1+1} \cdots \Omega_{i_m}, P_{k_m}, \Omega_{i_m+1} \cdots \Omega_j),$$

where Mk is the patched model on client k, composed of Federated model parameters V1 Vj (j being the total number of Federated layers) and patch parameters Pk1 Pkm (m being the total number of local patches, fig being the set of indexes of the patch parameters) unique to client k. Fig. 2 shows an example composition of a DNN model used in MTFL.

Mudrakarta et al. [14] showed that Batch Normalisation (BN) layers can act as model patches for MTL in the centralised setting. Later it was shown that BN layers work well as patches in MTFL, considering that they are very lightweight in terms of number of parameters. BN layers are given by

$$\hat{x}_i = \frac{z_i - \mathbb{E}(z_i)}{\sqrt{\mathrm{Var}(z_i) + \epsilon}}$$
$$\mathrm{BN}(\hat{x}_i) = \gamma_i \hat{x}_i + \beta_i,$$

where E(Zi) and Var(Zi) are the mean and variance of a neuron's activations (zi, post-nonlinearity) across a minibatch, and gi and bi are parameters learned during training. BN layers track a weighted moving average of E(Zi) and Var(Zi) during training: μi and σ2(i) , for use at inference time. In Section 4 investigate the benefit of keeping statistics μ; σ and/or trainable parameters γ,β as part of private patch layers .

5.4 Methods Compared:

BN layers are chosen to use for personalisation within MTFL. The reason for this choice is twofold: 1) they show excellent personalisation performance and 2) the storage cost of BN parameters is very small (< 1% of total model size for the tested model architectures). Mudrakarta et al. [14] also investigated the use of depth wise-convolution patches for centralised Multi-Task learning. Any model layers could in principle be kept private during MTFL, however, there is an inherent trade-off between the number of parameters kept private and the ability of the global model to converge

Effect of BN Patches on Inference

To understand the impact that BN-patch layers have on UA, considering the change in internal DNN activations over a client's local test-set immediately before and immediately after the FL aggregation step. As illustrated in Fig. 3a, UA typically drops after the aggregation step in iterative FL. This is because the model has been tuned on the local training set for several epochs, and suddenly has its model weights replaced by the Federated weights, which are unlikely to have better test performance than the pre-aggregation model. This idea is further examined in [32] and showed later in our experimental section. Consider a simple DNN consisting of dense layers followed by BN and then nonlinearities. The vector of first layer neuron activations over the client's test-set (X) from applying weights and biases (W0, b0), can be modelled as a normal distribution, which BN relies on to work.

$$z_i \triangleq [W_0 X + b_0]_i$$
$$z_i \sim N(\mathbb{E}[z_i], Var[z_i]).$$

During local training, the client's model has been adapted to the local dataset, and the BN-layer statistics used for inference (μ , σ^2) have been updated from the layer activations. Assuming, after local training (and before aggregation), $\mu_i \sim E[z_i]$, and $\sigma^2_i \sim Var[z_i]$, then the BN-layer (ignoring)

$$\hat{x}_i \triangleq \frac{z_i - \mu_i}{\sigma_i}$$
$$\hat{x}_i \sim N(0, 1)$$
$$\mathrm{BN}(\hat{x}_i) \sim N(\beta_i, \gamma_i^2),$$

where β_i ,$\gamma_i^2$are the learned BN parameters. If the client is participating in FL or MTFL, then the model parameters W_0, b_0 are updated after downloading the global model with federated values: W_0, b_0. The activations of the first layer are then

$$\overline{z}_i \triangleq [\overline{W}_0 X + \overline{b}_0]_i$$
$$\overline{z}_i \sim N(\mathbb{E}[\overline{z}_i], \mathrm{Var}[\overline{z}_i]).$$

Defining the difference in mean and variance between pre and post-aggregation activations

$$\hat{\bar{x}}_i \sim N\left(\frac{\Delta\mu_i}{\sigma_i}, 1 + \frac{\Delta\sigma_i^2}{\sigma_i^2}\right)$$
$$\mathrm{BN}(\hat{\bar{x}}_i) \sim N\left(\gamma\frac{\Delta\mu_i}{\sigma_i} + \beta_i, \gamma_i^2\left(1 + \frac{\Delta\sigma_i^2}{\sigma_i^2}\right)\right).$$

If the BN layer is not a patch layer (i.e., the client is participating in FL, with federated BN values μ , σ ,β, γ), the output of the BN layer is

$$\hat{\tilde{x}}_i \sim N\left(\frac{\mu_i + \Delta\mu_i - \bar{\mu}_i}{\bar{\sigma}_i}, \frac{\sigma_i^2 + \Delta\sigma_i^2}{\bar{\sigma}_i^2}\right)$$
$$\overline{\mathrm{BN}}(\hat{\tilde{x}}_i) \sim N\left(\bar{\gamma}\frac{\mu_i + \Delta\mu_i - \bar{\mu}_i}{\bar{\sigma}_i} + \bar{\beta}_i, \bar{\gamma}_i^2\frac{\sigma_i^2 + \Delta\sigma_i^2}{\bar{\sigma}_i^2}\right).$$

BN-patch layers in MTFL constrains neuron activations to be closer to what they were before the aggregation step, compared to non-patch BN layers as part of FL as illustrated in Fig. 3b. I.e., the difference in means and variances pre- and post-aggregation using MTFL is smaller than when using FL.

$$\left|\gamma\frac{\Delta\mu_i}{\sigma_i}\right| < \left|\beta_i - \bar{\gamma}\frac{\mu_i + \Delta\mu_i - \bar{\mu}_i}{\bar{\sigma}_i} - \bar{\beta}_i\right|$$
$$\left|\gamma_i^2\frac{\Delta\sigma_i^2}{\sigma_i^2}\right| < \left|\gamma_i^2 - \bar{\gamma}_i^2\frac{\sigma_i^2 + \Delta\sigma_i^2}{\bar{\sigma}_i^2}\right|.$$

Assuming the above inequality holds, it is easy to see how the values propagated through the network after the first layer are closer to the pre-aggregation values when using BN-patches as opposed to federated BN layers. If BN patches are added throughout the network, the intermediate DNN values will be regularly ‘constrained’ to be closer to the pre-aggregation values, resulting ultimately in network outputs closer to the pre-aggregation outputs.

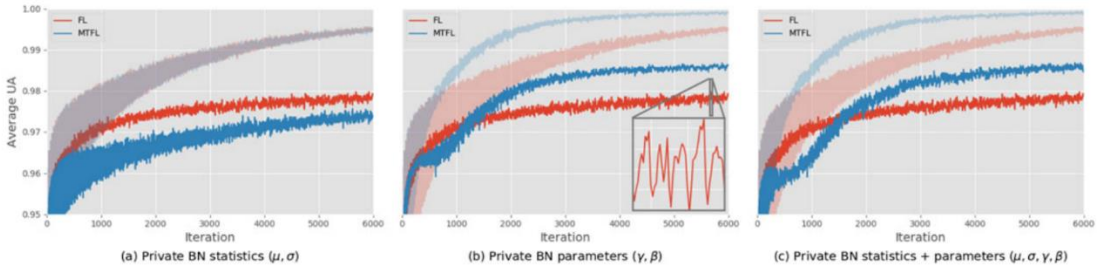

Fig :5.3.1 Average training (faint) and testing (solid).User Accuracy (UA) curves for every step of local SGD on the MNIST, W = 200, C= 1:0 scenario, using FL(FedAvg) (red), and MTFL(FedAvg) (blue). Each plot compares keeping different values within the BN layers of MTFL private: either statistics (μ , σ) and/or trainable parameters (β, γ), to FL. All curves have been smoothed with an averaging kernel for presentation, except the inset of plot (b), which shows the cyclic drops in accuracy due to model averaging characteristic of FL.

These experiments also examine which BN values, when kept private, give the best performance, and compare FL and MTFL with different optimisation strategies. After that, why different private BN values have different impacts on training was investigated, and compare MTFL to other state-of-the-art personalised FL algorithms. Finally, evaluated the cost in terms of computation time of MTFL(FedAvg-Adam) on an MEC-like testbed.

MNIST - 2NN																
	FL				**MTFL**											
	Private values = None				$\mu, \sigma, \gamma, \beta$				μ, σ				γ, β			
Optimisation	W = 200		400		200		400		200		400		200		400	
Strategy	C = 0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0
FedAvg	99	102	107	110	85	58	101	68	X	X	X	X	29	21	34	26
FedAdam	85	69	88	65	56	37	75	77	109	90	194	262	31	25	31	27
FedAvg-Adam	44	49	40	50	17	41	19	32	131	151	170	198	**9**	**9**	**10**	**9**
CIFAR10 - CNN																
FedAvg	139	138	171	164	49	33	55	36	231	280	258	266	37	24	45	30
FedAdam	105	90	83	80	21	14	22	16	67	45	48	38	24	14	25	16
FedAvg-Adam	57	43	36	31	11	9	14	**8**	82	79	62	63	**10**	**7**	**11**	**8**

Table: 5.3.1 Communication Rounds Required to Reach Target Average User Accuracies for Different Tasks Using FL and MTFL (With Private Statistics μ , σ and Trained Parameters β, γ), for Different Numbers of Total Clients W, Client Participation Rates C, and Optimisation Strategies

Optimisation Strategy	FL				MTFL $\mu, \sigma, \gamma, \beta$				MTFL μ, σ				MTFL γ, β			
	Private values = None															
	W = 200		400		200		400		200		400		200		400	
	C = 0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0	0.5	1.0
MNIST - 2NN																
FedAvg	276	X	290	X	115	76	144	144	85	58	102	68	50	36	65	48
FedAdam	X	X	X	X	76	47	110	89	56	37	75	77	43	33	46	53
FedAvg-Adam	133	260	191	X	20	16	24	27	17	41	19	32	**12**	**8**	**15**	**40**
CIFAR10 - CNN																
FedAvg	148	208	202	250	47	32	52	35	239	186	260	88	36	24	43	28
FedAdam	159	91	92	93	21	14	21	15	74	49	51	42	34	16	21	14
FedAvg-Adam	193	X	X	X	14	10	16	**9**	103	111	67	74	**12**	**8**	**13**	**9**

Table 5.3.2 Communication Rounds Required to Reach Target Average User Accuracies (of Non-Noisy Clients) for Different Tasks Using FL and MTFL (With Private Statistics μ , σ and Trained Parameters β, γ), When 20 percent of Clients Have Noisy Training Data, for Different Numbers of Total Clients W, Client Participation Rates C, and Optimisation Strategies.

CHAPTER 6

TESTING

6.1 Testing -1

Testing involves a comprehensive evaluation process to ensure the reliability, efficiency, security, and effectiveness of the federated learning system. Unit testing focuses on testing individual units or components of the system, such as the federated learning algorithm, personalized model training modules, communication interfaces, and data preprocessing functions. It helps identify and address bugs in specific components.

Performance testing to compare the Multi-Task Federated Learning (MTFL) algorithm with traditional Federated Learning (FL) algorithms involves a systematic evaluation of various key metrics to understand their relative effectiveness in real-world scenarios. The objective of this testing is to provide insights into how MTFL performs compared to FL algorithms. Both MTFL and FL algorithms are implemented within this environment to ensure consistency in testing conditions. Standardized datasets and training scenarios are used to conduct the tests, allowing for a fair and unbiased comparison between the two approaches. Convergence speed is another essential metric evaluated during performance testing. It measures how quickly models trained using MTFL and FL algorithms reach a desired level of accuracy. Faster convergence is generally desirable, especially in time-sensitive applications where rapid model updates are required.

6.2 Testing - 2

The number of rounds required to attain user accuracy is a critical aspect of performance evaluation in federated learning systems, including both Multi-Task Federated Learning (MTFL) and traditional Federated Learning (FL) algorithms. This metric directly influences the efficiency and speed at which models are trained and updated across distributed edge devices.

In performance testing, the number of rounds refers to the iterations or cycles of training and model aggregation that occur between the edge devices and the central server. Each round typically involves edge devices performing local training on their respective data samples, followed by the aggregation of model updates on the central server to produce a global model.

The goal of federated learning is to achieve a target level of accuracy while minimizing the number of rounds required to reach that accuracy. This is because reducing the number of rounds can lead to faster convergence and more efficient model updates, resulting in lower communication overhead and reduced computational burden on edge devices.

The number of rounds needed to attain user accuracy depends on various factors, including the complexity of the model, the size and diversity of the dataset, the heterogeneity of edge devices, and the optimization algorithms employed. More complex models or larger datasets may require more rounds to achieve the desired accuracy, while optimization techniques such as federated averaging can help expedite the convergence process.

Performance testing involves experimenting with different configurations and parameters to identify the optimal number of rounds needed to attain user accuracy for a given federated learning scenario. This may involve varying the learning rate, batch size, or aggregation strategy to determine their impact on convergence speed and model performance.

By analysing the results of performance testing, stakeholders can gain insights into the trade-offs between the number of rounds and model accuracy. Balancing these factors is essential for designing efficient federated learning systems that meet performance requirements while minimizing resource consumption and latency. Additionally, ongoing monitoring and optimization of the federated learning process can help adapt to changing conditions and improve overall system performance over time.

6.3 Test Cases

Test case ID	01
Test Scenario	Classification of the image
Test Case	Testing the classification using Federated Learning algorithm
Test Steps	Perform the training of neural network using FL algorithm and
Test data	MNIST handwritten digits
Expected Result	Model should be trained to attain 97% user accuracy
Actual Result	Model is trained to attain the desired accuracy with more number of rounds
Status(Pass/Fail)	Pass

Test case ID	02
Test Scenario	Classification of the image
Test Case	Testing the classification using MTFL algorithm
Test Steps	Perform the training of neural network using MTFL algorithm and
Test data	MNIST handwritten digits
Expected Result	Model should be trained to attain 97% user accuracy
Actual Result	Model is trained to attain the desired accuracy
Status(Pass/Fail)	Pass

Test case ID	03
Test Scenario	Comparing the performance of multi-task federated learning with traditional federated learning approaches
Test Case	Benchmark the performance of multi-task federated learning against standard federated learning methods
Test Steps	Perform the training of neural network using MTFL algorithm and FL algorithm
Test data	MNIST handwritten digits
Expected Result	MTFL algorithm should take less number of rounds to attain the desired user accuracy than FL algorithm
Actual Result	MTFL takes less number of rounds to attain the target user accuracy that FL algorithm
Status(Pass/Fail)	Pass

Test case ID	04
Test Scenario	Classification of the image
Test Case	Testing the classification using Federated Learning algorithm
Test Steps	Perform the training of neural network using FL algorithm and
Test data	CIFAR-10 dataset handwritten digits
Expected Result	Model should be trained to attain 65% user accuracy
Actual Result	Model is trained to attain the desired accuracy with more number of rounds
Status(Pass/Fail)	Pass

Test case ID	05
Test Scenario	Classification of the image
Test Case	Testing the classification using MTFL algorithm
Test Steps	Perform the training of neural network using MTFL algorithm and
Test data	CIFAR-10 dataset handwritten digits
Expected Result	Model should be trained to attain 65% user accuracy
Actual Result	Model is trained to attain the desired accuracy
Status(Pass/Fail)	Pass

Test case ID	06
Test Scenario	Comparing the performance of multi-task federated learning with traditional federated learning approaches
Test Case	Benchmark the performance of multi-task federated learning against standard federated learning methods
Test Steps	Perform the training of neural network using MTFL algorithm and FL algorithm
Test data	CIFAR-10 dataset handwritten digits
Expected Result	MTFL algorithm should take less number of rounds to attain the desired user accuracy than FL algorithm
Actual Result	MTFL takes less number of rounds to attain the target user accuracy that FL algorithm
Status(Pass/Fail)	Pass

CHAPTER 7

RESULT ANALYSIS

7.1 Actual Results obtained from the work

Compared the personalisation performance of MTFL(FedAvg) with two other state-of-the-art Personalised FL algorithms: Per-FedAvg [12] and pFedMe [10], and FL (FedAvg) (where no model layers are private) [5]. Tuned the hyperparameters of each algorithm to achieve the maximum average UA within 200 communication rounds. Presented MTFL(FedAvg) with private (β, γ), not MTFL (FedAvg-Adam), only to compare the personalisation algorithms, not the benefit of the adaptive

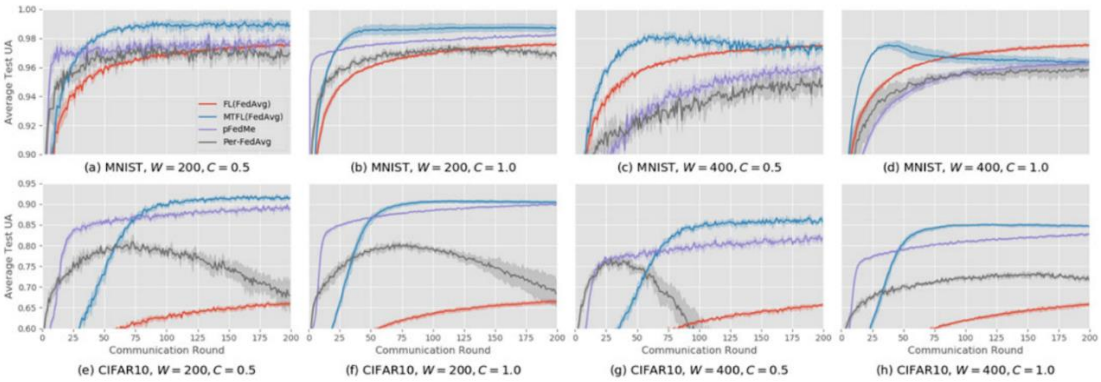

Fig 7.1:Per-round testing User Accuracy (UA) of four FL algorithms: FL(FedAvg) [5], MTFL(FedAvg) (using private β, γ), pFedMe [10] and Per-FedAvg [12]. Experiments are conducted on MNIST and CIFAR10, with data divided in a non-IID fashion.

optimisation strategy. The amount of local computation was also fixed to be roughly constant for the algorithms: performed E = 1 epoch of local training for MTFL(FedAvg) and FL(FedAvg). For MNIST, using a batch size of 20, this is equivalent to 15 and 8 steps of local SGD for W = 200 and W = 400, respectively. For CIFAR10, this is equivalent to 13 and 7 steps of local SGD for W = 200 and W = 400, respectively. Per-FedAvg uses the value K for local iterations, so it was fixed that the same number of steps for FL and MTFL. pFedMe uses has two inner loops, and set the number of outer-loops R to the same value as K from Per-FedAvg, and fix the inner-loop number for pFedMe to 1 for all scenarios. This setup results in the same number of local steps performed for each algorithm, however, the cost per local step of Per-FedAvg and

pFedMe is considerably higher than FL (FedAvg) and MTFL(FedAvg). Note also that MTFL (FedAvg) and FL(FedAvg) have only one hyperparameter, h to tune, whereas Per-FedAvg and pFedMe both have two. This makes the hyperparameter search for Per-FedAvg and pFedMe considerably more costly

Results - The plots in Fig. 5 show that MTFL(FedAvg) was able to achieve a higher UA compared to the other schemes in all tested scenarios. Per-FedAvg and pFedMe were able to reach a higher UA than FL(FedAvg) in the W = 200 cases for MNIST, but were actually slower in the W = 400 cases. All the personalised-FL schemes were able to achieve good UA faster than FL(FedAvg) for the CIFAR10 experiments, however. This is likely due to the CIFAR10 task being a much harder one than MNIST. It is interesting to note that PerFedAvg appeared to overfit quickly on this task. Also worthy of note is the fact that MTFL(FedAvg) was able to beat Per-FedAvg and pFedMe whilst also having one less hyperparameter to tune, and being computationally cheaper. MTFL also provides the extra benefit to privacy of keeping some model parameters private (pFedMe and PerFedAvg both upload entire models) (RPi) 2B's and 5 RPi 3B's, connected over WiFi to a server, in order to emulate a low-powered, heterogeneous set of clients. The RPi's used Tensorflow to perform local training. The server did not perform any model testing, only receiving, averaging and distributing models. The average time over 10 rounds was taken, along with the percentage of time spent per round in downloading models from the server, local training, uploading models and work performed on the server

Results - Table 4 shows the average time taken per round for FL(FedAvg), MTFL(FedAvg-Adam), and Independent learning, when one local epoch of training is performed. Each round is also split by time spent for each task within the round. As would be expected, Independent learning took the least time per round as clients did not have to download / upload any models. FL(FedAvg) took longer per round due to uploading / downloading, and MTFL(FedAvg-Adam) took the longest per round due to the increased number of weights that FedAvg-Adam communicates over FedAvg, indicated by the higher percentage of round time spent downloading and uploading models. However, the increase in communication time is likely to be outweighed in most cases by the far fewer

Learning Scheme	Round Time (s)	Percentage of Round Time (%) *Down*	*Client*	*Up*	*Server*
MNIST - 2NN					
FL(FedAvg)	30	5	88	6	1
MTFL(FedAvg-Adam)	38	11	76	12	1
Independent	29	0	100	0	0
CIFAR10 - CNN					
FL(FedAvg)	108	5	86	5	4
MTFL(FedAvg-Adam)	136	11	74	12	3
Independent	100	0	100	0	0

Table 7.1 : Average Time Per Round of Different Learning Schemes on the MNIST and CIFAR10 Datasets, and Percentage of Time Spent Downloading the Model (Down), Training the Model (Client), Uploading the Model (Up), and Model Aggregation/Distribution on the Server (Server) Took

7.2 Analysis of the Results obtained

The excerpt delves into a comparison of various personalized Federated Learning (FL) algorithms, aiming to gauge the effectiveness of MTFL(FedAvg) against other contemporary approaches. In this setup, MTFL(FedAvg) is pitted against Per-FedAvg and pFedMe, alongside the standard FL(FedAvg) method. The primary objective lies in assessing each algorithm's personalization performance, with hyperparameters carefully tuned to maximize average User Accuracy (UA) within a set number of communication rounds, standing at 200 in this case. Notably, MTFL(FedAvg) is presented with private parameters (β, γ), rather than MTFL(FedAvg-Adam), underlining a focus on comparing personalization algorithms devoid of additional optimization strategies.

To ensure a fair comparison, the amount of local computation is standardized across algorithms, with the number of epochs of local training kept consistent for MTFL(FedAvg) and FL(FedAvg). The fixed number of local steps varies based on the dataset utilized—MNIST or CIFAR10—and the number of participating clients (W). However, it's acknowledged that while the same number of local steps are performed for each algorithm, the cost per local step differs. Per-FedAvg and pFedMe entail higher costs per local step compared to FL(FedAvg) and MTFL(FedAvg), impacting the computational expense of these approaches.

Upon analysis of the results, MTFL(FedAvg) emerges as a standout performer, consistently achieving higher UA across all tested scenarios. While Per-FedAvg and pFedMe demonstrate superior UA compared to FL(FedAvg) in certain cases, they lag behind in others. Interestingly, all personalized FL schemes outshine FL(FedAvg) in the CIFAR10 experiments, indicating their efficacy in tackling more complex tasks. However, concerns arise regarding Per-FedAvg's susceptibility to overfitting on the challenging CIFAR10 dataset, suggesting potential limitations.

Further scrutiny reveals MTFL(FedAvg)'s competitive edge, boasting superior performance while being computationally cheaper and requiring fewer hyperparameters to tune compared to Per-FedAvg and pFedMe. This finding underscores MTFL's favorable balance between efficacy and resource efficiency, positioning it as a promising candidate for practical FL deployments. Additionally, extending the analysis to a real-world Mobile Edge Computing (MEC) environment underscores the scalability and feasibility of MTFL, particularly in scenarios featuring heterogeneous client devices with limited computational resources.

CHAPTER 8
CONCLUSION AND FUTURE WORK

a) Conclusion

Proposed a Multi-Task Federated Learning (MTFL) algorithm that builds on iterative FL algorithms by introducing private patch layers into the global model. Private layers allow users to have personalised models and significantly improves average User model Accuracy (UA). Analysed the use of BN layers as patches in MTFL, providing insight into the source of their benefit. MTFL is a general algorithm that requires a specific FL optimisation strategy, and also proposed the FedAvg-Adam optimisation scheme that uses Adam on clients. Experiments using MNIST and CIFAR10 show that MTFL with FedAvg significantly reduces the number of rounds to reach a target average UA compared to FL, by up to 5 . Further experiments show that MTFL with FedAvg-Adam reduces this number even further, by up to 3 . These experiments also indicate that using private BN trainable parameters (β, γ) instead of statistics (μ , σ) in model patches gives better convergence speed. Comparison to other state-of-the-art personalised FL algorithms show that MTFL is able to achieve the highest average UA given limited communication rounds. Lastly, showed in experiments using a MEC-like testbed that the communication overhead of MTFL with FedAvg-Adam is outweighed by its significant benefits over FL with FedAvg in terms of UA and convergence speed.

b) Future Work

The future work of the above research work could involve several potential directions for further exploration and improvement. Some possible avenues for future research are:

- Extension to Other Datasets: While the research work demonstrated the effectiveness of MTFL on MNIST and CIFAR10 datasets, further experiments on a broader range of datasets could be conducted. This would help validate the generalizability and robustness of the MTFL algorithm across different types of data.

- Comparison with More Algorithms: Though the research compared MTFL with competing personalized FL algorithms, it could be extended to include more algorithms and methodologies for FL. This would provide a more comprehensive analysis of MTFL's performance and competitiveness in various scenarios.

- Different Neural Network Architectures: The research focused on MTFL with specific neural network architectures. Exploring MTFL's performance with other DNN architectures, such as ResNet or Transformer-based models, would allow researchers to understand its versatility and adaptability to different network designs.

- Privacy and Security Enhancements: As data privacy is a critical aspect in FL, future work could focus on enhancing the privacy measures in MTFL. Exploring techniques like differential privacy or secure aggregation could strengthen user data protection in the federated learning process.

- Application to Real-World Use Cases: The research primarily focused on tasks like MNIST and CIFAR10 classification. Future work could extend the application of MTFL to real-world use cases, such as personalized recommendation systems, natural language processing tasks, or healthcare applications, to assess its performance in practical scenarios.

- Edge Computing Optimization: Optimizing the MTFL algorithm specifically for edge-computing environments, considering resource constraints and communication overhead, could further enhance its efficiency and effectiveness.

Overall, these future research directions could contribute to a more comprehensive understanding of MTFL's capabilities and potential for practical applications in the domain of Federated Learning and edge computing.

CHAPTER 9

REFERENCES

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. Letaief, “A survey on mobile edge computing: The communication perspective,” IEEE Commun. Surv. Tut., vol. 19, no. 4, pp. 2322–2358, Fourthquarter 2017.

[2] B. Yang, X. Cao, J. Bassey, X. Li, and L. Qian, “Computation offloading in multi-access edge computing: A multi-task learning approach,” in Proc. IEEE Int. Conf. Commun., 2019, pp. 1–6.

[3] Y. Li, X. Wang, X. Gan, H. Jin, L. Fu, and X. Wang, “Learningaided computation offloading for trusted collaborative mobile edge computing,” IEEE Trans. Mobile Comput., vol. 19, no. 12, pp. 2833–2849, Dec. 2020.

[4] T. Li, A. Sahu, A. Talwalkar, and V. Smith, “Federated learning: Challenges, methods, and future directions,” IEEE Signal Process. Mag., vol. 37, no. 3, pp. 50–60, May 2020.

[5] B. McMahan, E. Moore, D. Ramage, and B. A. Y. Arcas, “Communication-efficient learning of deep networks from decentralized data,” in Proc. 20th Int. Conf. Artif. Intell. Stat., 2017, pp. 1273–1282.

[6] A. G. Roy, S. Siddiqui, S. Polsterl, N. Navab, and C. Wachinger, € “Braintorrent: A peer-to-peer environment for decentralized federated learning,” 2019, arXiv:1905.06731.

[7] L. Huang, A. L. Shea, H. Qian, A. Masurkar, H. Deng, and D. Liu, “Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records,” J. Biomed. Inform., vol. 99, 2019, Art. no. 103291.

[8] A. Hard et al., “Federated learning for mobile keyboard prediction,” 2018, arXiv:1811.03604.

[9] M. Ammad-ud-din, E. Ivannikova, S. A. Khan, W. Oyomno, Q. Fu, K. E. Tan, and A. Flanagan, “Federated collaborative filtering for privacy-preserving personalized recommendation system,” 2019, arXiv:1901.09888.

[10] C. T. Dinh, N. Tran, and J. Nguyen, “Personalized federated learning with moreau envelopes,” in Proc. Conf. Neural Inf. Process. Syst., 2020, vol. 33, pp. 21394–21405.

[11] Y. Jiang, J. Konecny, K. Rush, and S. Kannan, “Improving feder- ated learing personalization via model agnostic meta learning,” 2019, arXiv:1909.12488.

[12] A. Fallah, A. Mokhtari, and A. Ozdaglar, “Personalized federated learning with theoretical guarantees: A model-agnostic metalearning approach,” in Proc. Conf. Neural Inf. Process. Syst., 2020, vol. 33, pp. 3557–3568.

[13] V. Smith, C.-K. Chiang, M. Sanjabi, and A. Talwalkar, “Federated multi-task learning,” in Proc. Conf. Neural Inf. Process. Syst., 2017, pp. 4427–4437.

[14] P. K. Mudrakarta, M. Sandler, A. Zhmoginov, and A. Howard, “K for the price of 1: Parameter efficient multi-task and transfer learning,” in Proc. Int. Conf. Learn. Representations, 2019.

[15] D. Leroy, A. Coucke, T. Lavril, T. Gisselbrecht, and J. Dureau, “Federated learning for keyword spotting,” in Proc. IEEE Int. Conf. Acoust., Speech Signal Process., 2019, pp. 6341–6345.

[16] S. Reddi et al., “Adaptive federated optimization,” in Proc. Int. Conf. Learn. Representations, 2021.

[17] Y. Huang et al., “Personalized cross-silo federated learning on non-IID data,” in Proc. Assoc. Adv. Artif. Intell., 2021, pp. 7865– 7873.

[18] C. Dinh, T. Vu, N. Tran, M. Dao, and H. Zhang, “Fedu: A unified framework for federated multi-task learning with laplacian regu- larization,” 2021, arXiv:2102.07148.

[19] H. Jiang, J. Starkman, Y.-J. Lee, H. Chen, X. Qian, and M.-C. Huang, “Distributed deep learning optimized system over the cloud and smart phone devices,” IEEE Trans. Mobile Comput., vol. 20, no. 1, pp. 147–161, Jan. 2021.

[20] K. Bonawitz et al., “Towards federated learning at scale: System design,” in Proc. Syst. Mach. Learn. Conf., 2019.

[21]M. Duan, D. Liu, X. Chen, R. Liu, Y. Tan, and L. Liang, “Self-bal- ancing federated learning with global imbalanced data in mobile systems,” IEEE Trans. Parallel Distrib. Syst., vol. 32, no. 1, pp. 59– 71, Jan. 2021.

[22] D. Kingma and J. Ba, “Adam: A method for stochastic opti- mization,” in Proc. Int. Conf. Learn. Representations, 2014.

[23] F. Hanzely and P. Richt´arik, “Federated learning of a mixture of global and local models,” 2020, arXiv:2002.05516.