Data Heterogeneity and Personalization in Federated Learning

by

Jonathan Scott

January, 2026

A thesis submitted to the
Graduate School
of the
Institute of Science and Technology Austria
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Committee in charge:

Veronika Sunko, Chair
Christoph Lampert
Dan Alistarh
Virginia Smith

Institute of
Science and
Technology
Austria

The thesis of Jonathan Scott, titled *Data Heterogeneity and Personalization in Federated Learning*, is approved by:

Supervisor: Christoph Lampert, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Dan Alistarh, ISTA, Klosterneuburg, Austria

Signature: _____

Committee Member: Virginia Smith, Carnegie Mellon University, Pittsburgh, USA

Signature: _____

Defense Chair: Veronika Sunko, ISTA, Klosterneuburg, Austria

Signature: _____

Signed page is on file

ISTA Thesis, ISSN: 2663-337X

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I accept full responsibility for the content and factual accuracy of this work, including the data and their analysis and presentation, and the text and citation of other work.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: _____

Jonathan Scott
January, 2026

Signed page is on file

Abstract

In recent years there has been a massive increase in the amount of data generated in a decentralized manner. Ever more powerful edge devices, such as smartphones, have become ubiquitous in most societies on earth. Through text typed, photos taken and apps used, these devices, which we refer to as clients, generate enormous amounts of high quality and complex data. Moreover, the nature of these devices means the data they generate is often sensitive and privacy concerns prevent it being gathered and stored in a central location. This presents a challenge to the modern machine learning paradigm that requires central access to large amounts of data. Federated learning (FL) has emerged as one of the answers to this problem. Rather than bringing the data to the model, FL sends the model to the data. Model training takes place on device, with periodically synchronized updates, allowing data to remain locally stored. While this approach offers significant privacy advantages it comes with its own set of unique challenges. These include: data heterogeneity, the notion that different devices generate data in distinct ways which can negatively impact training dynamics; systems heterogeneity, meaning that different devices may have differing hardware specifications; high communication costs, which are induced by the repeated transferring of models over the network and low device computational power, which limits the use of larger models on device.

In this thesis we present a range of methods for federated learning. We focus primarily on the challenge of data heterogeneity, though the methods presented are designed to be well adapted to the other challenges of a federated setting, such as the constraints of limited compute and communication overhead. We first present a method for explicitly modeling client data heterogeneity. The approach formulates clients as samples from a certain probability distribution and infers the parameters of this distribution from the available training clients. This learned distribution then represents the heterogeneity present among the clients and can be sampled from in order to create new simulated clients that are similar to the real clients we have observed so far. Following this we present two methods for directly dealing with data heterogeneity through personalization. Highly heterogeneous client data distributions can mean that learning a single global model becomes suboptimal, and some form of personalization of models to each individual client is required. Our approaches are based around hypernetworks, which we use to generate personalized model parameters without the need for additional training or finetuning. In the first approach we focus on generating full parameterizations of client models using learned embeddings of client data and labels, with a hypernetwork located on the central server. In the second approach we address the more challenging scenario where we want to generate a personalized model for a client without any label information. The hypernetwork is trained to generate a low dimensional representation of a client's personalized model parameters, allowing it to be transferred to and run on the client devices. In our final presented method, we change our focus and rather than aim to directly address the challenge of data heterogeneity, we instead ensure we are unaffected by it. This is done in the context of k-means clustering and we present a method for federated clustering with a focus on added privacy guarantees.

Acknowledgements

"In the end, it's not the things we do that matter, but the people we meet—and how they change us."

I would like to thank my supervisor, Christoph, for his tireless enthusiasm, valuable suggestions, and active engagement with ideas and challenges. I am very grateful to him for all that I have learned about research over the last years. I would also like to thank my committee members, Dan and Virginia; I have been fortunate to benefit from their feedback and perspectives over the years. I also thank Veronika for being a great chair.

I have been fortunate to spend my time at ISTA surrounded by kind and intelligent people. Particular thanks go to my coauthors Hossein, David, and Michelle for their ideas and enthusiasm, which made the projects we worked on so much better and more enjoyable. To my office mates over the years: Hossein, Edwige, Nikita, Alexandra, Niko, and Mary, thanks for the camaraderie and laughs. And to all the other group members whom I was lucky to work with: Amelie, Paul, Bernd, Peter, Egor, Max, and Fabian, I have always appreciated your generosity with time for feedback and discussions. Finally, on the academic side, my thanks go out to my coauthor and mentor while I was at Apple, Áine, as well as the rest of the privacy team for making my time there so enjoyable and valuable.

My PhD was both an academic and a personal journey. Through the highs and lows, something that never changed was the presence of wonderful people around me. I would like to thank my parents, Beata and Mike, and my brother Jeremy, for their never-ending support and encouragement, and for always believing in me. Thank you, Elena, for helping me start this journey and for the enjoyable early years in Vienna, despite all the lockdowns and uncertainty. To my friends from WBH Wien, in particular Arvid, Marcel, and Max, thanks for many great rallies and "eine kleine Bier noch"s at Bauernbräu. Thank you: Andreas for the endless psych and cycling to the sea and back with me; Katya for great food and fascinating conversations; Dan for flashing my projects and always motivating me to be better; and Nick for listening hard and talking slow. Finally, to Robin, for making my life better in every way.

Funding Sources This research was funded in part by the Austrian Science Fund (FWF) [10.55776/COE12]. Furthermore, the candidate acknowledges the support from the Scientific Service Units (SSU) of ISTA through resources provided by Scientific Computing (SciComp).

About the Author

Jonathan Scott obtained a BSc in mathematics from the University of Edinburgh and a MASt in Mathematics from Cambridge University. He joined ISTA as a PhD student in September 2020 and was supervised by Professor Christoph Lampert. During this time Jonathan worked on developing algorithms for federated learning, with a focus on data heterogeneity and privacy. His work was published in international conferences and journals.

List of Collaborators and Publications

This thesis is based on the following first-author publications of Jonathan Scott. We provide a summary of the contributions of each author, referred to by their initials.

1. Jonathan Scott and Áine Cahill. Improved modelling of federated datasets using mixtures-of-Dirichlet-multinomials. In *International Conference on Machine Learning (ICML)*, 2024

    - Chapter 3 is based on this paper.
    - Initial motivation to study the problem of improved server-side simulations suggested by AC.
    - Weekly or biweekly in depth discussions took place between JS and AC, leading to many improvements to the algorithm and code over the course of the project.
    - Algorithm and central methodology proposed and developed by JS.
    - Theoretical results stated and proven by JS.
    - Initial code written and initial results produced by JS.
    - Code rewritten, and additional experiments run, producing most plots in the final paper, done by AC.
    - Additional folktables experiments done by JS.
    - Paper mostly written by JS, with substantial editing and feedback from AC.

2. Jonathan Scott, Hossein Zakerinia, and Christoph H. Lampert. PeFLL: Personalized federated learning by learning to learn. In *The Twelfth International Conference on Learning Representations*, 2024

    - Chapter 4 is based on this paper.
    - Initial project idea by JS, out of many discussions with CL.
    - Link between theory and method arose in discussions between JS and HZ.
    - Regular meetings between JS, HZ and CL led to development and refinement of the algorithm.
    - Theoretical results derived by HZ and CL.
    - Code written and experiments run by JS.
    - JS, HZ and CL all contributed significantly to the writing of the paper.

3. Hossein Zakerenia, Jonathan Scott, and Christoph H. Lampert. Federated learning with unlabeled clients: Personalization can happen in low dimensions, 2025. Preprint

- Chapter 5 is based on this paper.
- HZ and JS contributed equally to this project (joint first authorship)
- Initial idea to use a low dimensional parameterization proposed by HZ.
- Code written, ideas tested and experiments run by JS.
- Theory stated and proved by HZ and CL.
- Paper mostly written by JS, theory and proof sections by HZ. JS, HZ and CL all edited and refined the writing.

4. Jonathan Scott, Christoph H. Lampert, and David Saulpic. Differentially private federated k-means clustering with server-side data. In *International Conference on Machine Learning (ICML)*, 2025

    - Chapter 6 is based on this paper.
    - Initial idea for the project suggested by JS.
    - Algorithm developed by JS and DS.
    - JS, CL and DS all contributed during regular meetings to improve the algorithm and methodology.
    - Code written and experiments run by JS.
    - Theoretical results stated and proven by DS.
    - JS, CL and DS all contributed substantially to the writing of the paper.

The following first author publication by Jonathan Scott also arose out of the PhD but does not appear in the thesis.

5. Jonathan Scott, Michelle Yeo, and Christoph H. Lampert. Cross-client label propagation for transductive and semi-supervised federated learning. *Transactions on Machine Learning Research*, 2023

Table of Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

In recent years, due to the proliferation of ever more powerful edge devices, such as smartphones and wearable devices, there has been an explosion in the amount of data generated in a decentralized fashion. Both the scale, and often sensitive nature of these data, has necessitated a major shift in how and where machine learning on such data takes place. Federated learning (FL) [MMR+17] has emerged as the de-facto standard for privacy-preserving machine learning in a distributed setting. Data holding devices, known as *clients* or *users*, collaboratively learn predictive models without sharing their data directly with any other party. This is done under the supervision of a central server. The key principle underlying federated learning is that rather than taking the data, centralizing it on the server and training on it there, we keep data decentralized and run training on the edge where the data is, periodically synchronizing the training through sharing model updates with the server. While this approach offers potentially significant privacy advantages, it comes with a number of challenges which we describe now in detail.

Data heterogeneity is the phenomenon by which the underlying data distributions of different client datasets can be different. To give a concrete example, two different phone users will likely take pictures of very different things, leading to very different processes generating the image datasets stored on their respective devices. Data heterogeneity has a major effect on learning dynamics particularly affecting final performance of globally trained models [ZLL+18, HQB19], as well as the speed of convergence [KKM+20, LHY+20]. Data heterogeneity can occur in variety of ways including, but not limited to, differences between label distributions, feature distributions or the number of samples clients have [SGT22].

Systems heterogeneity occurs due to the presence of differing types of user devices, e.g. multiple generations of smartphones. This manifests itself primarily in that some clients have access to significantly more compute than others, likely in terms of memory and/or computational speed. It can lead to biases in which devices are able to participate in training, either due to hardware constraints or by failing to compute updates in time or dropping out due to network connectivity issues. This can impact convergence of training and/or impact the final model performance as well as fairness of the learned models [LSZ+20, KMA+21].

Communication costs in federated learning can be a major factor impacting the efficiency and feasibility of algorithms [KMRR16, KMY+16, SWMS19]. These costs are incurred by the transmission of statistics, typically model weights or gradient updates, between server and client. This can lead to a trade-off between efficiency and accuracy, for instance by

communicating less often and running fewer rounds or using smaller models. It can also necessitate the usage of communication efficient techniques such as model compression or sparsification.

Privacy issues can still occur despite the data being kept locally on each device. A number of works have shown that successful attacks are possible in federated learning when no additional privacy preserving measures are used [MSDCS19, ZLH19, GBDH20]. If privacy is the primary focus, then other techniques, such as differential privacy [Dwo06] and secure aggregation [BIK+16, TWM+24] are needed to enforce stricter privacy guarantees. However, as we discuss in Section 2.4, these methods come with the cost of potentially lower accuracy and increased communication overhead. Moreover, there is an added tension regarding algorithm design, since getting utility out of privacy primitives might restrict the sorts of operations we allow our algorithms to utilize, e.g. enforcing the condition that quantities seen by the server are only expressed as summations.

In this thesis we focus primarily on the challenge of data heterogeneity, although the presented methods are designed with the restrictions of federated learning in mind, such as communication overhead and privacy desires. The remainder of this thesis is organized as follows.

- In Chapter 2 we introduce the required background and notation for the rest of the thesis. We present the fundamentals of supervised and unsupervised learning and formally introduce federated learning, discussing in more detail the aspects of personalization and privacy.
- In Chapter 3, which is based on [SC24], we present a method for modeling data heterogeneity in federated learning using a mixture of Dirichlet-multinomial distributions. The algorithm is able to infer the maximum likelihood parameters of the distribution using the training clients. By sampling from this distribution with the learned parameters we are then able to simulate new clients. We illustrate how this can be used to partition server-side data in a way that better represents the heterogeneity present among the clients, with the goal of running training simulations on the server.
- In Chapter 4 we present an approach to dealing with data heterogeneity through personalization. This Chapter is based on [SZL24]. The key focus of our approach is obtaining personalized models for clients without the need for additional training or fine-tuning. This is done by training a hypernetwork, which is kept on the server, and an embedding network, which is sent to the clients. The embedding network takes client data as input, and outputs an embedding vector, which summarizes the client's data distribution. This embedding vector can then be fed into the hypernetwork, which then outputs the full model parameters of a personalized model for that client. The method comes with theoretical guarantees on generalization to new clients, as well as convergence of the optimization procedure.
- In Chapter 5, which is based on [ZSL25], we again look at tackling personalization without finetuning, this time in the more challenging setting of obtaining personalized models for clients that do not hold any labeled data. We again follow a hypernetwork approach. The hypernetwork now takes in a batch of (unlabeled) data, and outputs a low dimensional parameterization of a client's personalized model. This construction significantly reduces the size of the hypernetwork, allowing it to be sent to, and run on, the clients. We prove a transductive generalization bound and derive from this bound a training objective for our method. This objective includes a regularization term that

can be computed using only unlabeled data, meaning that unlabeled clients can also contribute to training.

- In Chapter 6, which is based on [SLS25], we switch perspectives and instead of modelling and/or dealing with data heterogeneity directly we instead aim to be unaffected by it. We do this in the context of federated clustering, with a particular focus on ensuring differential privacy. The primary contribution is an initialization method that leverages a small amount of (potentially) out-of-distribution server-side data, that is able to output good initial centers to the k-means problem. These centers lie close enough to the optimal ones that typically few, if any, steps of refinement using a naive version of federated and differentially private Lloyd's algorithm, are needed. We prove, in the setting of data sampled from a mixture of Gaussians, that our method converges exponentially fast to the true centers.
- Finally, in Chapter 7 we conclude and provide an outlook on future work.

CHAPTER 2

Background

In this Chapter we present an overview of machine learning in a federated setting. We provide a brief introduction to the fundamentals of supervised and unsupervised machine learning as well as introducing some of the standard notation that will appear in the remainder of this thesis. We then introduce the federated learning paradigm and highlight the challenges associated with it. We introduce personalized federated learning, as a solution to the challenge of data heterogeneity, and we conclude by providing background on essential privacy techniques that appear frequently in the context of federated learning.

2.1 Fundamentals of Machine Learning

Machine learning aims to automate the process of using data to solve some desired task. Typically *training data* will be utilized by a *learning algorithm* to gain an understanding of patterns present within the data. This is referred to as training, and the output of this process is a *model*, which will contain the learned patterns of the training data and can (hopefully) be used to make predictions on future, previously unseen, data. In this thesis we deal with both supervised and unsupervised learning, and we now provide a brief introduction to each of them.

2.1.1 Supervised Learning

In supervised learning we possess both data and labels. For instance, images together with labels detailing their contents. We denote the data space by $\mathcal{X} \subseteq \mathbb{R}^d$ and the label space by $\mathcal{Y}$. In this thesis we deal primarily with classification tasks and denote the number of classes by C, meaning that $\mathcal{Y} = \{1, \dots, C\}$ is a discrete space. We assume there is a probability distribution $\mathcal{D} \in \mathcal{P}(\mathcal{X} \times \mathcal{Y})$ that defines the relationship between data and labels. The goal is, generally speaking, to learn a predictive function from data to labels $f : \mathcal{X} \to \mathcal{Y}$ using training data (X, Y) which is assumed to be sampled i.i.d. from the underlying distribution $\mathcal{D}$. We denote the number of samples by $|X| = |Y| = m$. The model f is often a parameterized function, typically a deep neural network, with parameters $\theta \in \mathbb{R}^D$. A deep neural network consists of (possibly a large number of) functional compositions of *layers*. A layer l is generally some form of linear transformation followed by a non-linear one. For example, a fully connected layer can be written as:

$$f_l(z) = \sigma(W_l z + b_l),$$

where $\sigma(t) = \max(0, t)$ is applied elementwise. In this case the model parameters are $\theta = (W_l, b_l)_{l=1}^{L}$. Learning consists of finding a setting of the parameters θ that performs well on the training data, where performance is measured by a loss function $\ell : \mathcal{X} \times \mathcal{Y} \times \mathbb{R}^D \to \mathbb{R}_+$. This is therefore an optimization problem of the form:

$$\min_{\theta} \frac{1}{m} \sum_{(x,y)\in(X,Y)} \ell(x, y, \theta), \tag{2.1}$$

which is typically solved by a gradient based optimization method, such as *stochastic gradient descent (SGD)*. An SGD update has the following form:

$$\theta_{t+1} \approx \theta_t - \eta \nabla_\theta \ell(\theta_t), \tag{2.2}$$

where $\eta > 0$ is the learning rate, or step size. In minibatch SGD a batch, i.e. subset, of the training data is used to compute the gradient. A single full pass through the training data (in batches) is called an *epoch*. Training typically consists of a number of epochs.

2.1.2 Unsupervised Learning

In unsupervised learning, we only have a data space $\mathcal{X}$, and the goal is to uncover some underlying properties of the data. The most popular form of unsupervised learning, and the one that will appear in this thesis, is clustering. The goal here is, given a dataset X, to uncover a partitioning of X into k subsets, aka clusters, so that points within clusters exhibit similarity. We consider k-means clustering which computes a clustering by optimizing the k-means objective. Formally, given a set of data points, $X = (x_1, \ldots, x_m)$ and any $2 \leq k \leq m$, the goal of k-means clustering is to find *cluster centers* $\nu_1, \ldots, \nu_k$ that minimize the k-means cost:

$$\sum_{i=1}^{m} \min_{j=1,\ldots,k} \|x_i - \nu_j\|^2. \tag{2.3}$$

The cluster centers induce a partition of the data points: a point x belongs to cluster j, if $\|x - \nu_j\| \leq \|x - \nu_{j'}\|$ for all j, j', with ties broken arbitrarily (but deterministically). It is well established that solving the k-means problem optimally is NP-hard in general [Das08]. However, efficient approximate algorithms are available, the most popular being Lloyd's algorithm [Llo82], which is shown in Algorithm 1. Given an initial set of centers, it iteratively refines their positions until a local minimum of (2.3) is found. A characteristic property of Lloyd's algorithm is that the number of steps required to converge and the quality of the resulting solution depend strongly on the initialization. The most commonly used initialization is the k-means++ algorithm [AV07], shown in Algorithm 2. We will revisit clustering at the end of this thesis in Chapter 6, where we deal with private clustering in a federated setting.

2.2 Federated Learning

As introduced in the previous Chapter, federated learning has emerged as the de facto approach to doing machine learning on distributed and privacy-sensitive data. In this section we introduce standard notation that will be used throughout this thesis as well as the standard framework for doing federated learning. In the sections that follow we go on to introduce personalization and privacy techniques in the context of federated learning.

Algorithm 1 Lloyd's Algorithm

input Data points $X = \{x_1, \ldots, x_m\}$, number of clusters k, initial centers $\nu_1^{(0)}, \ldots, \nu_k^{(0)}$

Set iteration counter $t \leftarrow 0$
repeat
  Assignment step:
  for $i = 1$ to m **do**
    Assign x_i to the nearest center: $c_i^{(t)} \leftarrow \arg\min_{j=1,\ldots,k} \|x_i - \nu_j^{(t)}\|^2$
  end for
  Update step:
  for each cluster $j = 1$ to k **do**
    Update center: $\nu_j^{(t+1)} \leftarrow \frac{1}{|\{i:c_i^{(t)}=j\}|} \sum_{i:c_i^{(t)}=j} x_i$
  end for
  $t \leftarrow t + 1$
until assignments $c_1^{(t)}, \ldots, c_m^{(t)}$ do not change

output Final cluster centers $\nu_1, \ldots, \nu_k$ and assignments $c_1, \ldots, c_m$

Algorithm 2 k-means++ Initialization

input Data points $X = \{x_1, \ldots, x_m\} \subset \mathbb{R}^d$, number of clusters k

Choose the first center $\nu_1^{(0)}$ uniformly at random from X
for $j = 2$ to k **do**
  for each point $x_i \in X$ **do**
    Compute $D(x_i) = \min_{1 \leq \ell < j} \|x_i - \nu_\ell^{(0)}\|^2$
  end for
  Choose $\nu_j^{(0)}$ from X with probability proportional to $D(x_i)$
end for

output Initial centers $\nu_1^{(0)}, \ldots, \nu_k^{(0)}$

There are two commonly considered federated learning scenarios: *cross-device* federated learning and *cross-silo* federated learning. Cross-device FL is usually characterized by having a large number of clients, typically each with little data. Clients tend to have limited compute and memory capacity and may be unreliable, and prone to dropping out during training. The canonical example of cross-device FL is training on a large network of smartphone devices. In contrast, in cross-silo FL we have a small number of clients, typically in the tens or hundreds. Clients are much larger institutions, such as hospitals or banks, with ample computational resources and larger amounts of data. While cross-device and cross-silo FL share common attributes, such as the decentralized nature of the data and the potential statistical heterogeneity of the clients, in other crucial aspects, such as scale of local data and computational power, they differ significantly. As such, while methods in FL are likely transferable between both settings, the motivations and focus might be better suited to one setting than the other. In this thesis we will implicitly focus more on cross-device FL where, for instance, personalization without finetuning as presented in Chapters 4 and 5 is more relevant. Our probabilistic approach to modeling heterogeneous clients in Chapter 3 also makes most sense when considered in a cross-device setting, since clients correspond to samples from a (meta) distribution, and there should be a large number of these. In contrast, however, Chapter 6 can be equally applied to cross-device and cross-silo, and we explicitly consider both settings in our evaluation.

Algorithm 3 FederatedAveraging - Server

input Number of clients n, fraction of clients p, number of rounds T

Initialize global model weights $\theta^{(0)}$
for each round $t = 0, 1, \ldots, T-1$ **do**
  $C^{(t)} \leftarrow$ (random set of pn clients)
  for each client $i \in C^{(t)}$ **in parallel do**
    Server sends $\theta^{(t)}$ to client i
    $\theta_i^{(t+1)} \leftarrow \text{ClientUpdate}(i, \theta^{(t)})$
  end for
  $m^{(t)} \leftarrow \sum_{i \in C^{(t)}} m_i$
  $\theta^{(t+1)} \leftarrow \sum_{i \in C^{(t)}} \frac{m_i}{m^{(t)}} \theta_i^{(t+1)}$
end for

output Final global model weights $\theta^{(T)}$.

Algorithm 4 FederatedAveraging - ClientUpdate

input Client ID i, global model weights θ, local batch size B, local learning rate η, number of local epochs E

$\mathcal{B} \leftarrow$ (split S_i into batches of size B)
for each local epoch e from 1 to E **do**
  for batch $b \in \mathcal{B}$ **do**
    $\theta \leftarrow \theta - \eta \nabla \ell(b, \theta)$
  end for
end for
return θ to server

We now formalize the federated learning setting and introduce some of the notation that will appear throughout the remainder of this thesis. We will use n to denote the number of clients, aka data holding entities, in a federated learning system. We assume each client i has access to a dataset S_i of m_i datapoints sampled from a data distribution D_i. We will always allow that clients may be statistically heterogeneous, meaning that we can have $D_i \neq D_j$ for $i \neq j$. We also always assume the existence of a server, whose role is to coordinate training. Federated algorithms typically follow a standard training framework that takes place over T rounds. This approach is well illustrated by the original federated algorithm proposed by [MMR+17] called federated averaging, shown in Algorithms 3 and 4, which aims to train a single global model using decentralized gradient updates at the clients. The model is typically a neural network, denoted by $f : \mathcal{X} \to \mathcal{Y}$, with parameters θ. Each round the server samples a subset of the total n clients. We refer to this subset as a *cohort*. The server then broadcasts the current model weights θ to each client in the sampled cohort. The clients then run E epochs of training locally on their data before sending the updated model weights back to the server. The server receives the aggregate of the client updates, and from this updates the global model weights.

As illustrated by federated averaging, the key tenant of FL is keeping data decentralized and stored locally on the client, rather than centralizing it on the server. While enabling this is the primary focus of federated learning, in practical applications the server often has access to some of its own data. This could take the form of a proxy public dataset, synthetically generated data or anonymized data from a subset of consenting clients. Typically, this data is of a much smaller scale and of a different distribution to the true data that is held by the

clients. As such, it is rarely good enough to rely on, on its own for training. However, with additional techniques it can be useful in certain scenarios as we examine later in Chapters 3 and 6.

2.3 Personalized Federated Learning

In practice, the client data distributions may differ significantly from each other, which makes learning a single global model suboptimal [LSTS20, KMA+21]. Personalized federated learning (pFL) [SCST17a] is a means of dealing with such statistical heterogeneity, by allowing clients to learn individual models while still benefiting from each other. So while federated averaging trained a single global model θ, in personalized FL the goal is for each client i to obtain its own personalized model parameters θ_i. Existing approaches can be broadly categorized into the following groups.

Multitask methods, as seen in [SCST17a, MNB+21, DVT+22, LHBS21, DTN20, HHHR20, HR20, YNW+23, ZLD+23, QYW+23], learn individual per-client models, while sharing information between clients. Often this is done through the training of global and local models with some form of regularization towards the global model.

Meta-learning methods learn a shared model, which the clients can quickly adapt or finetune to their individual data distributions with a small number of gradient updates [FMO20a, JKRK19, WBAH23]. For instance, by incorporating the MAML [FAL17] objective into the training of a global model, one explicitly seeks a model that clients can quickly personalize to themselves.

Decomposition-based methods such as [AASC19, CHMS21, BMG+19, LLL+20, CYG+23, WZY+23] split the learnable parameters into two groups: those that are meant to be shared between clients and those that are learned on a per-client basis. For instance, the feature extractor could be a shared across all clients, and globally trained, while the classifier head is unique to each client.

Clustering-based methods [GCYR20a, MMRS20a] divide the clients into a fixed number, K of subgroups and learn individual models for each cluster. Typically this is done by iterating between assigning clients to clusters, using which of the K models best fits each client, and updating the models using these cluster assignments.

A key challenge that arises uniquely in the context of personalized federated learning is how to generate a personalized model for a client that was not seen during the training process. This is typical in cross-device FL, where the number of clients is very large and not all clients can be accessed during training due to practical constraints for running training: device online, plugged in and charging, not in use etc. Additionally, new clients are constantly joining the network. In standard federated learning obtaining a model for such a client is not an issue since the goal is to train a single global model, and, post training, this model can be deployed, as is, on any client: present or future. However, in personalized FL, we seek some form of personalized model, also on future clients, which necessitates a procedure to obtain it. In nearly all the personalized FL literature this is done by some form of additional training or finetuning of, for instance, globally learned parameters obtained during the training process. However, this approach comes with added computational overhead, and potentially also communication costs when personalization requires multiple interactions with the server. It can also be prone to overfitting when client data is limited, as is often the case in personalized federated learning. Finally it also requires the presence of labeled information on the clients. Chapters 4 and 5

aim to directly tackle this challenge. The approaches presented there focus on training-free personalization, which are applicable in the setting of clients without labels.

2.4 Privacy Techniques in Federated Learning

Privacy, in the form of data minimization, is the primary motivation for doing federated learning. However, multiple works have shown that simply training on decentralized data without additional privacy tools can lead to serious information leakage [MSDCS19, ZLH19, GBDH20]. This has led to the development and deployment of additional privacy preserving technologies in federated learning. We introduce here the two most important and most popular, namely differential privacy and secure aggregation.

2.4.1 Differential Privacy

The goal of differential privacy (DP) is to provide a privacy guarantee of the following form: "Whether any single individual's data was used or not will not change the outcome of training by much". Differential privacy is the mathematical quantification of this statement. Formally, for any $\varepsilon, \delta > 0$, a (necessarily randomized) algorithm $\mathcal{A} : \mathcal{P} \to \mathcal{S}$ that takes as input a data collection $P \in \mathcal{P}$ and outputs some values in a space $\mathcal{S}$, is called (ϵ, δ) *differentially private*, if it fulfills that for every $S \subset \mathcal{S}$,

$$\Pr[\mathcal{A}(P) \in S] \leq e^{\varepsilon} \Pr[\mathcal{A}(P') \in S] + \delta, \tag{2.4}$$

where P and P' are two arbitrary *neighboring* datasets. We consider two notions of *neighboring*: for standard *data-point-level privacy*, two datasets are neighbors if they are identical except that one of them contains an additional element compared to the other. In the more restrictive *client-level privacy*, we think of two datasets as a collection of per-client contributions, and two datasets are neighbors if they are identical, except that all data points of one of the individual client are missing in one of them. In the context of federated learning, the correct notion of neighborhood depends on the application in question. Typically the privacy unit to be used (data point vs client) should be that which corresponds to a human. In a cross-silo setting that is usually a single datapoint, making data-point-level privacy appropriate, while in cross-device a client is usually a device belonging to some individual and client-level DP is required. Condition (2.4) then ensures that no individual data item (a data point or a client's data set) can influence the algorithm output very much. As a consequence, from the output of the algorithm it is not possible to reliably infer if any specific data item occurred in the client data or not. An important property of DP is its *compositionality*: if algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_t$ are DP with corresponding privacy parameters $(\varepsilon_1, \delta_1), \ldots, (\varepsilon_t, \delta_t)$, then any combination or concatenation of their outputs is DP at least with privacy parameters $(\sum_{s=1}^{t} \varepsilon_s, \sum_{s=1}^{t} \delta_s)$. In fact, stronger guarantees hold, which in addition allows trading off between ε and δ, see [KOV15]. These cannot, however, be easily stated in closed form. Due to compositionality, DP algorithms can be designed easily by designing individually private steps and composing them.

The two most popular mechanisms to make computational steps DP, and those that appear in this thesis are: The *Laplace mechanism* [DMNS06], which achieves $(\varepsilon, 0)$ privacy by adding Laplace-distributed noise with scale parameter $\frac{S}{\varepsilon}$ to the output of the computation. Here, S is the *sensitivity* of the step, i.e. the maximal amount by which its output can change when operating on two neighboring datasets, measured by the L^1-distance. The *Gaussian*

mechanism [DR14a], which instead adds Gaussian noise of variance $\sigma_G^2(\varepsilon, \delta; S) = \frac{2\log(1.25/\delta)S^2}{\varepsilon^2}$, to ensure (ε, δ)-privacy. Here, the sensitivity S is measured with respect to L^2-distance. The above formulas show that stronger privacy guarantees, i.e. a smaller *privacy budget* (ε, δ), require more noise to be added. This, however, might reduce the accuracy of the output. Additionally, the more processing steps there are that access private data, the smaller the privacy budget of each step has to be in order to not exceed an overall target budget. In combination, this causes a counter-intuitive trade-off for DP algorithms that does not exist in this form for ordinary algorithms: accessing the data more often, might lead to lower accuracy results, because the larger number of steps has to be compensated by more noise per step. Consequently, a careful analysis of the privacy-utility trade-off is crucial for DP algorithms. In general, however, algorithms are preferable that access the private data as rarely as possible. In the context of federated learning, this means potentially limiting the number of rounds we run, and/or employing techniques such as early stopping, for better privacy guarantees.

2.4.2 Secure Aggregation

Many of the quantities of interest in a machine learning algorithm are summations of other terms, e.g. a minibatch of gradients in stochastic gradient descent or the mean of points in a newly assigned cluster of Lloyd's algorithm. Secure aggregation is a technique that allows the computation of a summation, without exposing any of the individual terms of the sum. On a high level, it works by masking all terms with correlated random noise that sums to 0. Thus all individual terms are indistinguishable from random noise, but still sum to the correct quantity. In the context of federated learning, this means the server sees a summation of client statistics, without seeing any of the individual statistics. On its own, this already adds a layer of enhanced privacy protection, however, it is most crucial when working with differential privacy. When differential privacy is to be ensured relative to the server, secure aggregation allows us to avoid noising every individual summand, and rather just the final summation, thereby saving a factor of cohort size in the noise scale.

A number of protocols have been proposed for running secure aggregation in federated settings. These can be based on secure multi-party computation protocols using either a single server [BIK$^+$16, BIK$^+$17], but requiring multiple communication rounds, or assuming two non-colluding servers [TWM$^+$24] with only a single round. Alternatives exist using trusted execution environments [BEM$^+$17, HNM$^+$22], although these can be difficult to scale and typically require specialized hardware. We shall not focus on the details of secure aggregation protocols in this thesis. Instead, we simply note that a desirable design principle for federated algorithms with a focus on privacy is ensuring that all terms used by the server can be expressed as a summation of client statistics. That way, we allow for easy integration of secure aggregation techniques into our methods.

CHAPTER 3

Improved Modelling of Federated Datasets using Mixtures-of-Dirichlet-Multinomials

3.1 Motivation and Outline

As discussed in section 2.2 running federated learning in practice faces a number of challenges including: statistical and systems heterogeneity, high communication costs, high latency, low client compute and scheduling difficulties that arise from practical restrictions, such as a device needing to be charging and not in use to participate in training [KMA+21]. This makes on device training not only technically challenging but also significantly more time-consuming than standard centralized training. This effect is compounded by the fact that in most modern machine learning pipelines we do not train a model just once, but rather many times, to select the right architecture, optimization algorithm, hyperparameters (HP) etc. Running large scale model selection and HP tuning on real clients in a federated network is often infeasible, making it difficult to obtain good performance. One way to address this issue is using simulations on the server. For instance, to select good model hyperparameters, which are then used directly in live training, thereby avoiding expensive HP tuning in the true federated network. This is possible because, in practice, it is common for the server to have some related proxy data that can be used to simulate the real federated training. For instance this could be public data from some related task, data from a particular subsample of consenting clients, or client data with certain sensitive features removed. Usually, however, the server-side data come without a client identifier, that is to say, it is a single central dataset with no natural partitioning into distinct FL clients. The question then arises how to use these data to create simulations that actually match the dynamics of live. The naive approach would be to create clients from the proxy data by simply subsampling the data points in an IID fashion. However, this approach can fail to capture client data distribution heterogeneity, and it is a well documented fact in FL that clients having non-IID data has a significant effect on FL training dynamics [ZLL+18, LSZ+20, LHY+20].

In this chapter we address the challenge of partitioning centralized data in a way that reflects the statistical heterogeneity of the true FL clients. The goal being to use this partition to run server-side federated training simulations, see Figure 3.1. On a high level our approach does this by choosing some categorical feature of the client data (for example the target

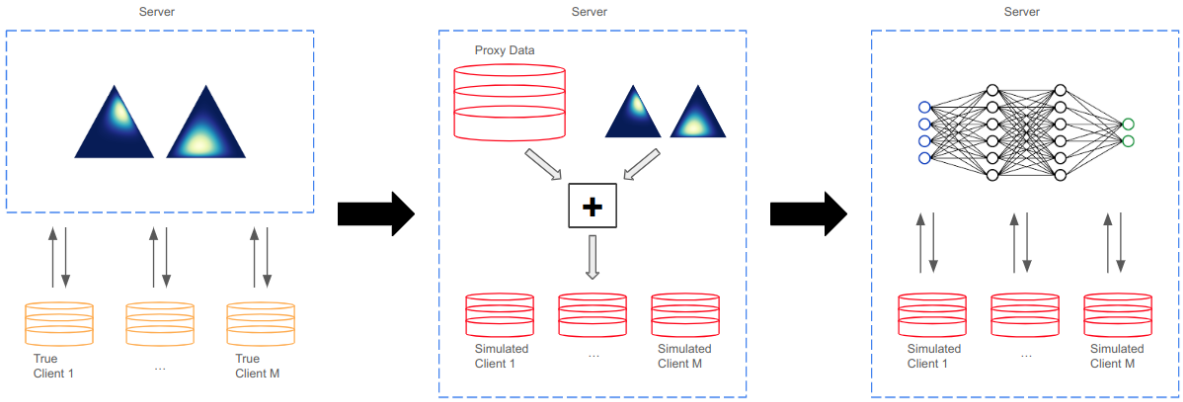

Figure 3.1: Proposed approach to server-side simulations. From left to right: learn Mixture-of-Dirichlet-Multinomials distribution from true federated clients (in this case 2 mixture components); use learned distribution to partition server proxy data into simulated clients; run server-side simulated federated model training using the simulated clients.

class) and representing each client as a histogram over this feature. We propose using a Mixture-of-Dirichlet-Multinomials (MDM) as a probability distribution over these histograms, meaning that each client corresponds to a single draw from this distribution. We then use the observed client histograms to infer the parameters of our distribution by maximum likelihood estimation (MLE) using the true FL clients. Thus we obtain a learned distribution, which we can sample from, and these samples are histograms that look like the histograms we observed in the true clients. It is now simple to create simulated clients from our centralized data. First sample a histogram from our learned distribution, then subsample the centralized data so that its histogram matches this sampled histogram. In this way we create simulated clients whose data distributions match the true client distributions along the chosen feature. There are several key features of our approach that make it effective in overcoming the challenges of a federated setting. Firstly, the inference algorithm is specifically designed to preserve client privacy. It adheres to the hard restriction that the server only works with aggregates of client statistics. This means a secure aggregator can be used, which is crucial to not exposing any single client's data. It also ensures the algorithm is compatible with differential privacy (DP) both on a user or an example level. By post-processing this means that simulations using DP inferred parameters do not degrade our privacy budget. Secondly, inference is efficient in terms of computation and communication. Clients do not run expensive model training, rather they only compute statistics of their data. These statistics, which are transferred between the server and clients, are low dimensional. This ensures low communication overhead. Finally, the proposed inference algorithm is designed not to have hyperparameters that need to be tuned, as such it can be run one-shot on the true clients. This is crucial given our goal is to limit how often we train on the true federated clients. The only hyperparameter choice that needs to be made is the number of mixture components of the distribution and, given the aforementioned efficiency, in practice we simply run a few reasonable options for this HP independently but in parallel. The experiments are implemented using the pfl-research framework [GSC+24].

3.2 Related Work

Dirichlet Distributions in FL Dirichlet distributions are a popular tool used by FL researchers and practitioners to create heterogeneous federated datasets out of existing centralized

data to evaluate new learning methods [YAG+19, HQB19, LDCH22]. Given some chosen value of the $\boldsymbol{\alpha}$ parameter, a federated dataset is created by sampling a class proportion vector $\mathbf{p} \sim \text{Dir}(\boldsymbol{\alpha})$ for each client and assigning data points to the client so that the assigned classes match the sampled class proportions. The level of heterogeneity is controlled by the magnitude of $\boldsymbol{\alpha}$. In this approach it is assumed that we know a priori how we would like to choose $\boldsymbol{\alpha}$, and the resulting federated dataset will reflect the desired level of heterogeneity specified by $\boldsymbol{\alpha}$. This paper focuses on the opposite scenario in which there exists some federated dataset, and we would like to choose the distribution parameters, such as $\boldsymbol{\alpha}$, that well reflect the heterogeneity of this dataset. This is a classic Bayesian inference problem which has been well studied in the context of Dirichlet distributions [Min00, BNJ01, Hua05]. The distribution we would like to infer is our own proposed Mixture-of-Dirichlet-Multinomials. A similar approach is taken by [HHQ12]. Here the authors propose using a mixture of Dirichlet multinomials to model Microbial Metagenomic data, though their approach differs to ours in several key ways. Firstly, the proposed mixture distribution is different as [HHQ12] do not explicitly model the number of sample distributions for each component. Secondly, the inference procedures differ since [HHQ12] take a Maximum a posteriori (MAP) approach and solve the resulting MAP optimization using EM and leveraging optimization algorithms such as BFGS. This approach assumes all data is centrally available and is incompatible with federated learning. In contrast, we take an approach of maximum likelihood estimation, and solve the MLE optimization using parameter update rules that are derived using generalized EM and compatible with the restrictions imposed by an FL setting.

Server-side data in FL Although a key aspect of FL is that data are decentralized and stored locally on client devices, it is common in practice that the central server possesses related proxy data. A number of works propose using server-side data to improve federated training in a range of settings. For instance, [HRM+19] use public data to pre-train language models for improved next word prediction, before running federated training. Other works use server-side data during federated training. [SOK22] use server data to improve meta-gradient calculation in a meta learning approach to personalized FL. [GPZ+22, DKG+20] alternate between training on the clients and training on the server to mitigate the effects of client statistical heterogeneity and improve convergence and performance of acoustic models. [MLZ22] analyze the convergence behavior of such an alternating update scheme theoretically. Finally, in federated semi supervised learning, it is often assumed that the server possesses a small amount of labeled data which is used in combination with unlabeled client data [DDT22, JYYH21]. Our viewpoint and approach differ from the above in that we are interested in how to use server-side data to run realistic simulations that guide our downstream federated training.

Clustered and meta FL The notion that clients in a federated network can belong to different groups or clusters has received ample attention, particularly with respect to personalization in federated learning [GCYR20b, SMS20, MMRS20b]. Closely related to this is the multitask or meta learning viewpoint of FL [SCST17b, FMO20b, SZL24] in which clients are thought of as samples (or tasks) drawn from some meta-distribution over clients. We draw inspiration from both the meta and clustered viewpoints in FL. We propose a meta-distribution over clients and infer the parameters of this distribution that explain the observed clients. Moreover, the form of this distribution is motivated by the view that clients naturally form clusters, with shared statistical properties within clusters.

3.3 Method

3.3.1 Background

Setup Again, we let n denote the number of clients in a federated learning system. In this chapter we assume a general setting where each client $i \in [n]$ has m_i data points $\mathbf{z}_i^1, \dots \mathbf{z}_i^{m_i} \in \mathbb{R}^d$. For instance, for a classification task we could have $\mathbf{z}_i^j = (\mathbf{x}_i^j, y_i^j)$ or in an unsupervised setting we could have $\mathbf{z}_i^j = \mathbf{x}_i^j$ The clients may be statistically heterogeneous. We assume we have an upper bound M on the number of samples any client holds, so that $m_i \leq M$ for all $i \in [n]$. Let $f \in [d]$ be some categorical feature of the data, which can take up to C different values, i.e. $z_{jf}^i \in [C]$ for all i, j. For instance f could be the target class in the case of a C-class classification task. We call f the *modelled feature*. For the modelled feature f each client computes a histogram, or count vector, $\mathbf{c}_i \in \mathbb{Z}^C$ of the feature. Namely, entry l of this vector counts how often f took value l across the client's dataset: $c_{il} = \sum_{j=1}^{m_i} \mathbf{1}_{z_{jf}^i = l}$. Note that $\sum_{l=1}^{C} c_{il} = m_i$. Thus each client is now represented as a tuple $(\mathbf{c}_i, m_i)$.

Mixture-of-Dirichlet-Multinomials Our goal is now to construct a statistical model (probability distribution) of the $(\mathbf{c}_i, m_i)$. Then drawing a new sample from this distribution is like drawing the modelled feature histogram and the number of samples information of a new simulated client. The building block of our distribution is the Dirichlet-Multinomial (DM) distribution. DM is a compound distribution over vectors of discrete counts parameterized by $\boldsymbol{\alpha} \in \mathbb{R}_+^C$ and $m \in \mathbb{N}$. A vector of counts $\mathbf{c} \in \mathbb{Z}^C$ is sampled by first drawing a probability vector from a Dirichlet distribution $\mathbf{p} \sim \text{Dir}(\boldsymbol{\alpha})$ then sampling the counts from a multinomial distribution $\mathbf{c} \sim \text{Mult}(m, \mathbf{p})$. The probability mass function for DM is given by

$$p(\mathbf{c} \mid m, \boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)\Gamma(m+1)}{\Gamma(m+\alpha_0)} \prod_{j=1}^{C} \frac{\Gamma(c_j + \alpha_j)}{\Gamma(\alpha_j)\Gamma(c_j+1)}, \tag{3.1}$$

where Γ is the gamma function and $\alpha_0 \coloneqq \sum_{j=1}^{C} \alpha_j$. We extend DM in two ways to better model clients in real federated learning scenarios. Firstly, since users may have different numbers of samples, we are interested in a distribution where m is not fixed. Hence, we jointly model the distribution of the number of samples and the count vector. We assume independence of m and $\boldsymbol{\alpha}$ so that we have

$$p(\mathbf{c}, m \mid \boldsymbol{\alpha}, \boldsymbol{\pi}) = p(\mathbf{c} \mid m, \boldsymbol{\alpha})\pi_m, \tag{3.2}$$

where $\pi_m \coloneqq p(m)$ parameterizes the probability that a client has m samples in total. Since we assume that the number of samples a client can possess is upper bounded by some constant M, we have that $\boldsymbol{\pi} \in \mathbb{R}^M$ and $\sum_{j=1}^{M} \pi_j = 1$.

Our second extension comes from the observation that in practice clients in a federated learning scenario might be naturally partitioned into a number of groups/clusters, where each cluster comes from a different meta distribution over clients. The natural way to model such an observation is using a mixture model. Let K be the number of components in the mixture. Then we define our Mixture-of-Dirichlet-Multinomials (MDM) model as the joint distribution over histograms and sample counts with the following probability mass function

$$q(\mathbf{c}, m \mid \boldsymbol{\tau}, \mathbf{A}, \boldsymbol{\Pi}) = \sum_{k=1}^{K} \tau_k p(\mathbf{c}, m \mid \boldsymbol{\alpha}_k, \boldsymbol{\pi}_k), \tag{3.3}$$

where $\boldsymbol{\tau} \in \mathbb{R}^K$ is the weight of each component, $\mathbf{A} = [\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_K] \in \mathbb{R}^{K \times C}$ are the per component Dirichlet parameters and $\boldsymbol{\Pi} = [\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_K] \in \mathbb{R}^{K \times M}$ are the per component number of samples distributions. Additionally, p is defined through equations (3.1) and (3.2).

3.3.2 Parameter Inference

Given n clients in a federated learning system, each of which has a modelled feature histogram, number of samples tuple: $(\mathbf{c}_i, m_i)$, our goal is to infer $\boldsymbol{\tau}$, $\boldsymbol{\Pi}$ and $\mathbf{A}$ from Equation (3.3) that well explain the observed clients. Note that under the statistical model (3.3) each client corresponds to a single draw, or data point, from this distribution. Therefore, we can think of (3.3) as a 'meta' distribution over the clients. We take the approach of maximum likelihood estimation (MLE) to infer the statistical model parameters. Therefore, we aim to maximize the log likelihood of the observed data under the MDM statistical model (3.3). The log likelihood is:

$$L(\boldsymbol{\tau}, \mathbf{A}, \boldsymbol{\Pi}) = \sum_{i=1}^{n} \log q(\mathbf{c}_i, m_i \mid \boldsymbol{\tau}, \mathbf{A}, \boldsymbol{\Pi}). \tag{3.4}$$

The objective function (3.4) is in general non-concave and we propose an EM-based algorithm to maximize it in a federated setting. In the remainder of this section we state this algorithm, which consists of two main parts: a single round of initialization of the statistical model parameters; followed by a multi-round iterative procedure that aims to maximize (3.4). Prior to starting inference it is necessary to specify the number of components K to use. In this section we assume the value of K has already been chosen. In Appendix A.1 we outline a procedure for the server to choose the best value of K to use. In Section 3.3.3 we prove that the parameter updates used in the algorithm converge by deriving them as part of a generalized EM approach to maximizing the objective.

Initialization We start by initializing the parameters $\boldsymbol{\tau}$, $\boldsymbol{\Pi}$ and $\mathbf{A}$. The full procedure is given in Algorithm 5. The simplest parameter to initialize is $\boldsymbol{\tau}$, we set

$$\boldsymbol{\tau}^{(0)} = [\frac{1}{K}, \ldots, \frac{1}{K}], \tag{3.5}$$

namely all mixture components start with equal weight. Next we must initialize $\boldsymbol{\Pi}$ and $\mathbf{A}$, both of which consist of a parameter for each component k, namely $\boldsymbol{\pi}_k$ and $\boldsymbol{\alpha}_k$. To do this the server samples a single cohort of clients S_0 and each client $i \in S_0$ uniformly at random chooses a component k_i to contribute to initializing. To initialize the estimated number of samples distribution the server obtains a histogram of the number of samples of each client in each component and then normalizes this per component. Specifically, client i initializes a matrix of zeros $\mathbf{E}^i \in \mathbb{R}^{K \times M}$ and sets $E^i_{k_i m_i} = 1$. The server obtains the aggregate $\mathbf{E} = \sum_{i \in S_0} \mathbf{E}^i$ and then initializes $\boldsymbol{\Pi}$ by normalizing each row of $\mathbf{E}$ to obtain a per component probability distribution:

$$\boldsymbol{\pi}_k^{(0)} = \frac{1}{n_k} \mathbf{E}_k, \tag{3.6}$$

where $n_k = \sum_{j=1}^{S} E_{kj}$ is the number of clients that contributed to component k. Finally, we use a per component moment matching estimate to initialize $\mathbf{A}$. Namely, $\boldsymbol{\alpha}_k^{(0)}$ is chosen so that the first two moments of a Dirichlet distribution with parameter $\boldsymbol{\alpha}_k^{(0)}$ match the empirical moments of the normalized histograms of the clients that are assigned to component k. That is to say, we assume that the client normalized histograms are drawn from a Dirichlet and we

Algorithm 5 Dirichlet-Multinomial Mixture Initialization

input client count tuples $(\mathbf{c}_i, m_i)_{i=1}^n$

$S_0 \leftarrow$ server samples a cohort of clients
for client i **in** S_0 **do**
  $k_i \sim \mathcal{U}\{1, \ldots, K\}$
  $\mathbf{E}^i = \mathbf{0}^{K\times M}$, $E^i_{k_i m_i} = 1$
  $\mathbf{P}^i = \mathbf{0}^{K\times C}$, $\mathbf{P}^i_{k_i} = \frac{1}{m_i}\mathbf{c}_i$
  $\mathbf{Q}^i = \mathbf{0}^{K\times C}$, $\mathbf{Q}^i_{k_i} = (\frac{1}{m_i}\mathbf{c}_i) \odot (\frac{1}{m_i}\mathbf{c}_i)$
end for
$\mathbf{E} = \sum_{i\in S_0} \mathbf{E}^i$, $\mathbf{P} = \sum_{i\in S_0} \mathbf{P}^i$, $\mathbf{Q} = \sum_{i\in S_0} \mathbf{Q}^i$
$n_k = \sum_{j=1}^S E_{kj}$
$\bar{\mathbf{P}}_k = \frac{1}{n_k}\mathbf{P}_k$, $\bar{\mathbf{Q}}_k = \frac{1}{n_k}\mathbf{Q}_k$
$\boldsymbol{\tau}^{(0)} = [\frac{1}{K}, \ldots, \frac{1}{K}]$
$\boldsymbol{\pi}^{(0)}_k = \frac{1}{n_k}\mathbf{E}_k$
$\boldsymbol{\alpha}^{(0)}_k = \frac{\bar{P}_{k1} - \bar{Q}_{k1}}{\bar{Q}_{k1} - \bar{P}^2_{k1}}\bar{\mathbf{P}}_k$

output $\boldsymbol{\tau}^{(0)}$, $\boldsymbol{\Pi}^{(0)}$, $\mathbf{A}^{(0)}$

initialize under the constraint that moments of this Dirichlet match the empirical moments of the normalized histograms. Client i initializes zero matrices $\mathbf{P}^i, \mathbf{Q}^i \in \mathbb{R}^{K\times C}$ and sets $\mathbf{P}^i_{k_i} = \frac{1}{m_i}\mathbf{c}_i$ and $\mathbf{Q}^i_{k_i} = (\frac{1}{m_i}\mathbf{c}_i) \odot (\frac{1}{m_i}\mathbf{c}_i)$ where $\odot$ denotes entry-wise product. The server then obtains the aggregates

$$\mathbf{P} = \sum_{i\in S_0} \mathbf{P}^i \qquad\qquad \mathbf{Q} = \sum_{i\in S_0} \mathbf{Q}^i \tag{3.7}$$

and normalizes both row-wise by the number of clients contributing to each row (component), which is n_k.

$$\bar{\mathbf{P}}_k = \frac{1}{n_k}\mathbf{P}_k \qquad\qquad \bar{\mathbf{Q}}_k = \frac{1}{n_k}\mathbf{Q}_k. \tag{3.8}$$

Thus, the server has computed the empirical estimates of the first two moments of the client normalized histograms for each component. Under the constraint that the first two moments of the kth Dirichlet must match the empirical moments (3.8) we have that

$$\boldsymbol{\alpha}^{(0)}_k = \frac{\bar{P}_{k1} - \bar{Q}_{k1}}{\bar{Q}_{k1} - \bar{P}^2_{k1}}\bar{\mathbf{P}}_k, \tag{3.9}$$

which gives $\mathbf{A}$'s initialization. Appendix A.2.1 derives (3.9).

Solving the MLE Our goal is to infer the maximum likelihood estimates for $\boldsymbol{\tau}$, $\boldsymbol{\Pi}$ and $\mathbf{A}$ which we do via an EM based approach, stated in full in Algorithm 6. In Section 3.3.3 we derive the update formulas and prove their convergence. The server starts by running the initialization procedure from Algorithm 5. Inference of the parameters then takes place over T rounds. During each round the server samples a cohort of clients and sends to each the previous rounds computed estimates of the parameters. Each sampled client i then computes how well each component k describes their data using this latest estimate of the parameters.

Let Z_i denote the latent (unobserved) random variable indicating to which mixture component client i belongs. The client then computes:

$$\omega_k^i = \Pr\{Z_i = k \mid \mathbf{c}_i, m_i, \boldsymbol{\tau}^{(t)}, \boldsymbol{\Pi}^{(t)}, \mathbf{A}^{(t)}\}, \tag{3.10}$$

$$\propto p(Z_i = k \mid \boldsymbol{\tau}^{(t)}) p(\mathbf{c}_i, m_i \mid \boldsymbol{\pi}_k^{(t)}, \boldsymbol{\alpha}_k^{(t)}), \tag{3.11}$$

$$\propto \tau_k^{(t)} p(\mathbf{c}_i \mid m_i, \boldsymbol{\alpha}_k^{(t)}) \pi_{k,m_i}^{(t)}, \tag{3.12}$$

using equation (3.1). The aggregates of the ω_k^i are needed by the server to compute the update to $\boldsymbol{\tau}$. For $\boldsymbol{\Pi}$ and $\mathbf{A}$ the client uses ω_k^i to weight their contribution to the kth component parameter updates on the server. Intuitively, ω_k^i tells us how likely it is that client i was drawn from component k and the higher this value is the more client i contributes to the kth component parameter update. For $\boldsymbol{\Pi}$ each client i initializes $\mathbf{E}^i = \mathbf{0}^{K \times M}$ and sets column m_i equal to $\boldsymbol{\omega}^i$, so that $E_{km_i}^i = \omega_k^i$ for $k = 1, \ldots, K$. This corresponds to a soft assignment of the clients number of samples to the overall histogram computed across all clients. For the update to $\mathbf{A}$ the client computes two quantities to be aggregated by the server to be used in the parameter update:

$$\mathbf{u}_k^i = \omega_k^i \left(\psi(\mathbf{c}_i + \boldsymbol{\alpha}_k^{(t)}) - \psi(\boldsymbol{\alpha}_k^{(t)}) \right), \tag{3.13}$$

$$v_k^i = \omega_k^i \left(\psi(m_i + (\alpha_k^{(t)})_0) - \psi((\alpha_k^{(t)})_0) \right), \tag{3.14}$$

where ψ is the digamma function, which in (3.13) is applied entry-wise. The server gets aggregates of the client statistics

$$\boldsymbol{\omega} = \sum_{i \in S_{t+1}} \boldsymbol{\omega}^i \qquad \mathbf{E} = \sum_{i \in S_{t+1}} \mathbf{E}^i \tag{3.15}$$

$$\mathbf{u}_k = \sum_{i \in S_{t+1}} \mathbf{u}_k^i \qquad v_k = \sum_{i \in S_{t+1}} v_k^i \tag{3.16}$$

and uses these to compute the parameter updates

$$\boldsymbol{\tau}^{(t+1)} = \frac{1}{|S_{t+1}|} \boldsymbol{\omega}, \tag{3.17}$$

$$\boldsymbol{\pi}_k^{(t+1)} = \frac{1}{\omega_k} \mathbf{E}_k, \tag{3.18}$$

$$\boldsymbol{\alpha}_k^{(t+1)} = \frac{1}{v_k} \boldsymbol{\alpha}_k^{(t)} \odot \mathbf{u}_k. \tag{3.19}$$

3.3.3 Theoretical Results

In this section we state our theoretical result, in which we derive the previously stated parameter update formulas.

Theorem 1. *Let $(\mathbf{c}_i, m_i)_{i=1}^n$ be observed histogram, sample count data and $(\boldsymbol{\tau}^{(0)}, \boldsymbol{\Pi}^{(0)}, \mathbf{A}^{(0)})$ be an initialization of the parameters of the Mixture-of-Dirichlet-Multinomials model (3.3). For $t \geq 1$, $i = 1, \ldots, n$ and $k = 1, \ldots, K$ let*

$$\omega_k^i = p(Z_i = k \mid \mathbf{c}_i, m_i, \boldsymbol{\tau}^{(t)}, \boldsymbol{\Pi}^{(t)}, \mathbf{A}^{(t)}),$$

Algorithm 6 Dirichlet-Multinomial Mixture MLE

input client count tuples $(\mathbf{c}_i, m_i)_{i=1}^n$, steps T

Initialize $\boldsymbol{\tau}^{(0)}$, $\boldsymbol{\Pi}^{(0)}$, $\mathbf{A}^{(0)}$ using Algorithm 5.
for $t = 0$ **to** $T-1$ **do**
  $S_{t+1} \leftarrow$ server samples cohort of clients
  Server sends $(\boldsymbol{\tau}^{(t)}, \boldsymbol{\Pi}^{(t)}, \mathbf{A}^{(t)})$ to each client in S_{t+1}
  for client i **in** S_{t+1} **do**
    $\omega_k^i = \frac{\tau_k^{(t)} p(\mathbf{c}_i | m_i, \boldsymbol{\alpha}_k^{(t)}) \pi_{k,m_i}^{(t)}}{\sum_{k=1}^K \tau_k^{(t)} p(\mathbf{c}_i | m_i, \boldsymbol{\alpha}_k^{(t)}) \pi_{k,m_i}^{(t)}}$
    $\mathbf{E}^i = \mathbf{0}^{K \times M}$, $E_{km_i}^i = \omega_k^i$
    $\mathbf{u}_k^i = \omega_k^i \left(\psi(\mathbf{c}_i + \boldsymbol{\alpha}_k^{(t)}) - \psi(\boldsymbol{\alpha}_k^{(t)}) \right)$
    $v_k^i = \omega_k^i \left(\psi(m_i + (\boldsymbol{\alpha}_k^{(t)})_0) - \psi((\boldsymbol{\alpha}_k^{(t)})_0) \right)$
  end for
  $\boldsymbol{\omega} = \sum_{i \in S_{t+1}} \boldsymbol{\omega}^i$, $\mathbf{E} = \sum_{i \in S_{t+1}} \mathbf{E}^i$
  $\mathbf{u}_k = \sum_{i \in S_{t+1}} \mathbf{u}_k^i$, $v_k = \sum_{i \in S_{t+1}} v_k^i$
  $\boldsymbol{\tau}^{(t+1)} = \frac{1}{|S_{t+1}|} \boldsymbol{\omega}$
  $\boldsymbol{\pi}_k^{(t+1)} = \frac{1}{\omega_k} \mathbf{E}_k$
  $\boldsymbol{\alpha}_k^{(t+1)} = \frac{1}{v_k} \boldsymbol{\alpha}_k^{(t)} \odot \mathbf{u}_k$
end for

output $\boldsymbol{\tau}^{(T)}$, $\boldsymbol{\Pi}^{(T)}$, $\mathbf{A}^{(T)}$

where Z_i is the latent variable indicating the mixture component that the ith sample was drawn from. Then the iteration

$$\tau_k^{(t+1)} = \frac{1}{n} \sum_{i=1}^n \omega_k^i$$

$$\pi_{kj}^{(t+1)} = \frac{1}{\sum_{i=1}^n \omega_k^i} \sum_{i=1}^n \omega_k^i \mathbf{1}_{m_i = j}$$

$$\alpha_{kj}^{(t+1)} = \alpha_{kj}^{(t)} \frac{\sum_{i=1}^n \omega_k^i \left(\psi(c_{ij} + \alpha_{kj}^{(t)}) - \psi(\alpha_{kj}^{(t)}) \right)}{\sum_{i=1}^n \omega_k^i \left(\psi(m_i + (\alpha_k^{(t)})_0) - \psi((\alpha_k^{(t)})_0) \right)}$$

corresponds to a generalized EM update of the log likelihood (3.4) and hence converges to a stationary point.

The proof is in Appendix A.2.2. These update rules are identical to those in Algorithm 6 except that the latter are computed on a subset of the data (equivalently a cohort of clients). Thus the updates in Algorithm 6 can be thought of as stochastic versions of the iteration given in Theorem 1.

3.4 Experiments

We separate our empirical evaluation into two orthogonal aspects. In Section 3.4.1 we investigate the inference of the MDM parameters, $\boldsymbol{\tau}, \mathbf{A}$ and $\boldsymbol{\Pi}$. We evaluate Algorithms 5 and 6 in terms of how accurately they recover ground truth parameters. We also examine whether they infer meaningful parameters on real federated datasets for which no ground truth

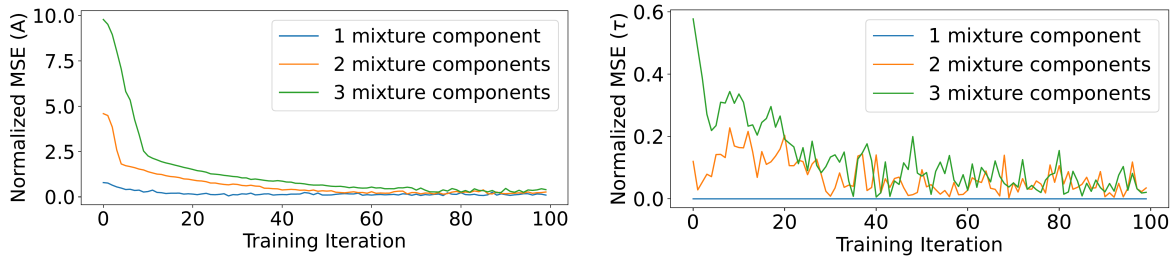

Figure 3.2: Normalized mean squared error (MSE) between the ground truth distribution parameter value and the inferred parameter value over time. Ground truth corresponds to medium levels of client statistical heterogeneity. On the left for $\mathbf{A}$, on the right for $\boldsymbol{\tau}$.

values exist. In Section 3.4.2 we investigate the utility of the parameters after they have been inferred. We evaluate how well simulations run on the server using these inferred parameters reflect training on the real federated clients.

Datasets We evaluate using synthetic data that follows the MDM distribution. To do this we use the following datasets: CIFAR10 [Kri09], FEMNIST [CWL+18] and Folktables [DHMS21]. CIFAR10 is a non-federated multi-class classification dataset with 10 classes which we partition into clients with target class histograms that follow an MDM distribution. For inference we use the target class as the modelled feature over which we compute our histograms. FEMNIST is a federated dataset with a predefined partitioning into clients. It is a character recognition dataset with 62 classes (including digits and lower and upper case letters). Each character is uniquely identified with one of the 3,550 clients in the dataset. For FEMNIST we use the target class as the modelled feature. Folktables is a US census dataset where each datapoint corresponds to an individual person present in the census. The dataset has a natural partitioning into 2,373 clients based on geographical location. Specifically, a client holds the data of all individuals that live in the same PUMA code region. We model two separate features of the data: the race feature and the income feature. For full details of the dataset and its federated partitioning see Appendix A.3.2

Baselines The primary baseline for simulating users on the server is the IID approach. This first computes the true per client number of samples distribution. Then, to simulate a client, it draws a number of samples count m from this distribution and IID samples m points from the centralized dataset. In this baseline each client's distribution of the modelled feature (and all other features) is the same as the marginal distribution of that feature in the centralized dataset. We therefore call this baseline *fully IID* simulation. As a sanity check we also consider an oracle that cannot be used in practice but gives a valuable comparison to our MDM simulation. In this oracle, which we call *conditionally IID* simulation, we ensure that the simulated clients exactly follow the marginal distributions of the modelled feature of the true clients, but when conditioned on the modelled feature, the remaining features are IID. This is done as follows. First, each true client computes a histogram over the modelled feature. For each of these true histograms we randomly sample points from the centralized dataset while ensuring that the histogram of the modelled feature for these samples matches the true histogram. The simulated clients in this oracle capture the heterogeneity in the modelled feature but nothing more. The best we can hope for is for our MDM simulations to match this oracle as we also only capture the heterogeneity in the modelled feature. Note that in a real FL setting this oracle cannot be used as it requires each client to share with the server their histogram of the modelled feature, which could be a serious privacy leak.

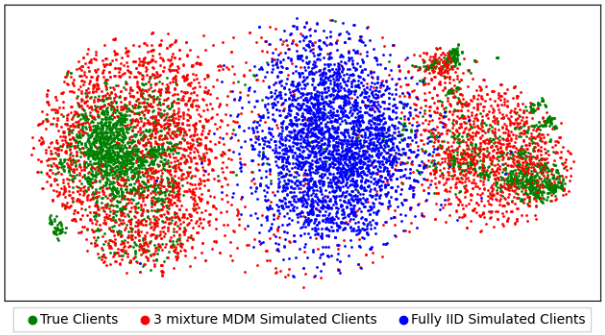

Figure 3.3: t-SNE visualization of FEMNIST clients, each point corresponds to a single client's class histogram. True clients (green), fully IID simulated clients (blue) and MDM clients (red).

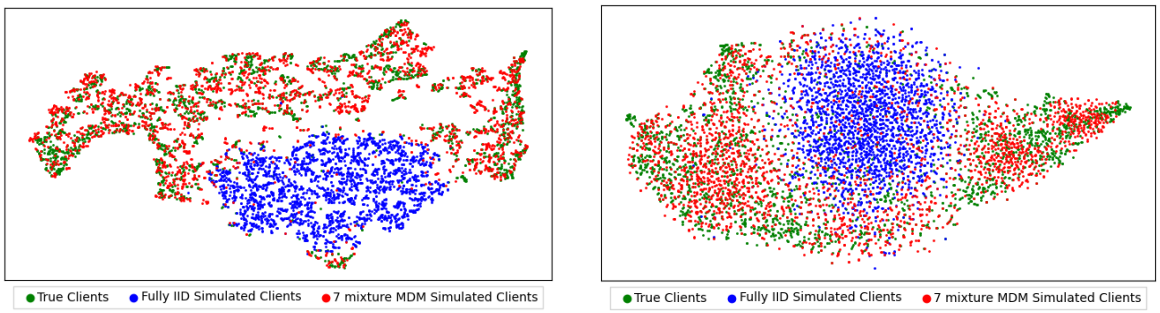

Figure 3.4: t-SNE visualizations of Folktables clients, each point corresponds to a single client's histogram over the race feature (left) and over the binned income feature (right). True clients (green), fully IID simulated clients (blue) and MDM clients (red). Inferred in both cases using $K = 7$.

3.4.1 Distribution Parameter Inference

Inference with known parameters We first assess the correctness of Algorithms 5 and 6 in terms of how accurately they recover ground truth distribution parameters for synthetic clients that follow our assumed MDM distribution. We test this over a range of different settings for $\boldsymbol{\tau}, \mathbf{A}$ and $\boldsymbol{\Pi}$. For 1, 2 and 3 mixture components we choose parameter values corresponding to low, medium and high levels of client heterogeneity. The exact values as well as additional experiment details can be found in Appendix A.3.3. The results for medium levels of heterogeneity can be seen in Figure 3.2, with the others in Appendix A.4.1. We plot the mean squared error (MSE) of the inferred parameters from the ground truth values, normalized by the size of the ground truth parameter (see Appendix A.3.1), against the number of training iterations, T. As we can see in all cases we obtain approximate convergence and quickly in terms of the number rounds required. As expected, convergence is slower when we have more mixture components to infer.

Inference without known parameters We now consider the more challenging and interesting scenario where the true clients do not necessarily follow our assumed MDM distribution. We use the FEMNIST and Folktables datasets which are naturally partitioned into heterogeneous federated clients. For FEMNIST we take the target class to be the modelled feature and represent each client as a histogram over the classes they hold. For Folktables we model two features of the data independently, namely the race feature and the income feature. The race feature is categorical, with $C = 9$ possible values. The income feature is continuous, so in

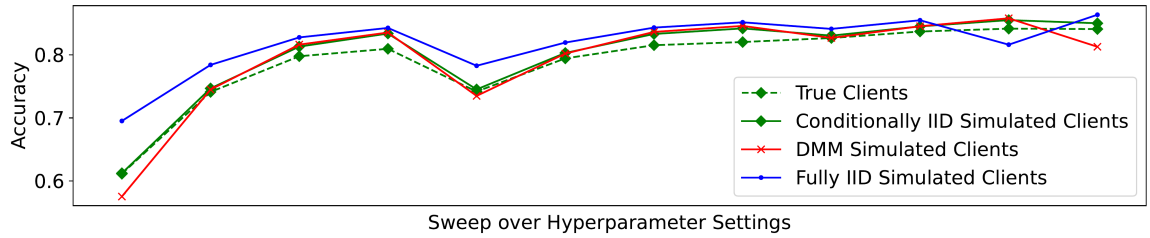

Figure 3.5: FEMNIST test accuracy when training with FedAvg for different settings of local learning rate and local epochs. True clients (dotted green), conditionally IID simulated clients (green), learned MDM simulated clients (red) and fully IID simulated clients (blue).

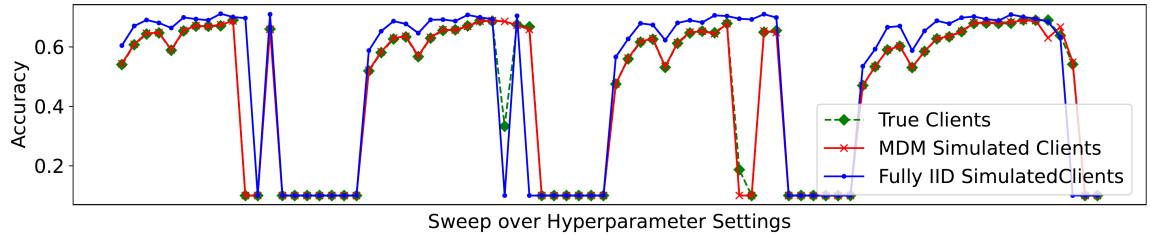

Figure 3.6: CIFAR10 test accuracy when training with FedAvg for different settings of local batch size, local learning rate and local epochs. True clients (dotted green), learned MDM simulated clients (red) and fully IID simulated clients (blue).

order to model it using MDM we convert it into a categorical feature by binning. Specifically, each client bins their continuous income value into one of $C = 41$ bins, with each bin having a range of \$5,000. We now represent clients as histograms over this binned income feature. For full details of the feature modelling on Folktables see Appendix A.3.2. We then run MDM inference on these histograms to obtain values for $\boldsymbol{\tau}, \mathbf{A}$ and $\boldsymbol{\Pi}$. Using these inferred parameter values we then sample from our distribution to obtain the histograms of new simulated clients. We then sample data for each client from the centralized version of the corresponding dataset so that each simulated client matches their sampled class histogram. As a baseline we also consider histograms corresponding to fully IID sampled clients. We plot 2-dimensional t-SNE visualizations of the results, for FEMNIST in Figure 3.3, and Folktables in Figure 3.4. Each point corresponds to a client (histogram), in green we have the true clients, in blue we have the simulated fully IID clients and in red the simulated MDM clients. We can draw several conclusions from these visualizations. Firstly, we observe in all three cases that the true users (green) exist in several distinct clusters, indicating that there are indeed different types of client heterogeneity present in the modelled feature. This validates our motivation for using a mixture model over the clients. Secondly, we see that in all cases the fully IID approach clearly fails to capture the heterogeneity in the client modelled feature distributions. Each of these simulated clients look like a subset of the centralized dataset, which poorly represents the true clients. Finally, in all cases the MDM distribution, with the parameters learned by Algorithms 5 and 6, does an admirable job of simulating the modelled feature heterogeneity of the true clients. This can be seen in the closeness between the true client distribution (green points) and the simulated client distribution (red points) in the t-SNE visualizations. In Appendix A.4.1 we include ablation studies where we infer the MDM parameters using a range of values for K and compare the differences in the corresponding simulated clients.

3.4.2 Federated Training Simulations

We now turn our attention to using our inferred parameters to run simulations on the server. We evaluate how closely model performance when training on simulated clients reflects performance when training on the true federated clients. As federated datasets we use FEMNIST as well as CIFAR10, partitioned into clients which follow an MDM distribution using the same settings of the parameters as in Section 3.4.1. The server-side proxy data is the centralized version of our federated dataset, i.e. the same data but without client identifiers. This setup ensures there is no distribution shift between the server and client data, which could be a confounding factor when evaluating the differences between simulated and true model training, beyond just the partitioning of the server data. In Appendix A.4.2 we provide results for additional experiments that simulate the practically relevant scenario when the server simulation data and client data are disjoint. We choose a range of HP settings and for each setting we train a model with federated averaging on the true clients, the MDM simulated clients, the fully IID simulated clients and the conditionally IID simulated clients. For FEMNIST we vary the local learning rate and local number of epochs used in FedAvg while for CIFAR10 we vary local batch size, local learning rate and local number of epochs. See Appendix A.3.4 for model and HP details. We compare the final accuracies of these trained models across the range of HP settings. The closer the final accuracy of the model trained on simulation clients is to the one trained on the true clients, the better, and more predictive the simulation is. Figures 3.5 and 3.6 show the results for FEMNIST and CIFAR10. The results for CIFAR10 are for true clients generated using 2 mixtures and high heterogeneity, see Section 3.4.1. The results for the other settings are in Appendix A.4.2. We can draw several conclusions from these plots. Firstly, for both datasets the accuracies are highest on the fully IID simulated clients across (nearly) all settings of the hyperparameters. In other words this baseline suffers from giving an overly optimistic prediction as to the performance on the true clients. This is to be expected given that federated averaging tends to perform better in the presence of low client heterogeneity. Secondly, training on the MDM simulated clients (red) gives a better indicator of performance on the true clients (dotted green) as seen by the closeness of the curves. Across all HP settings the mean of the absolute accuracy difference between the true clients and our MDM simulated clients was 1.6% compared to 3.3% for the fully IID simulated clients for FEMNIST and 0.8% compared to 6.6% for CIFAR10. Finally, while on CIFAR10 the MDM simulation exhibits near identical performance to the true clients, with non-trivial differences occurring on two particular settings on HPs which we attribute to randomness in the training process, for FEMNIST there is a non-negligible gap between MDM and the true clients. In general the MDM simulations overestimate performance on the true clients. We do observe, however, that the MDM simulations very closely match the conditionally IID oracle (green) on FEMNIST. Recall that this oracle captures exactly the label heterogeneity of the true clients (but nothing more). This again confirms that the parameters we inferred for FEMNIST in Section 3.4.1 successfully model the true client label distributions, and that this transfers over to federated model training. It also confirms the existence of client heterogeneity within the true dataset that goes beyond just the client label distributions. In other words the MDM simulated clients make it part of the way in capturing the true client heterogeneity but there is still remaining heterogeneity in the other features that is affecting training.

3.5 Conclusion

In this chapter we presented an approach to modelling heterogeneous FL clients using a Mixture-of-Dirichlet-Multinomials. We proposed an efficient and fully federated optimization procedure to infer the maximum likelihood estimates of the distribution parameters and showed theoretically the convergence of the update formulas. We empirically evaluated both the correctness of the proposed algorithm, in terms how well it infers the distributional parameters, as well as the utility of the inferred parameters themselves. We found that simulations run using clients generated with the inferred parameters were more representative of true federated training than those using IID clients. The proposed algorithm is private and compatible with differential privacy, on either a user or example level. In future work it would be interesting to further investigate the combination of DP with our proposed inference method, both in the central DP setting, where a secure aggregator adds noise to the aggregated statistics, and the local DP setting where the clients themselves add noise prior to aggregation.

CHAPTER 4

PeFLL: Personalized Federated Learning by Learning to Learn

4.1 Motivation and Outline

As discussed in 2.3, a key challenge of personalized federated learning lies in balancing the benefit of increased data that is available for joint training with the need to remain faithful to each client's own distribution. Most methods achieve this through a combination of global model training and some form of finetuning or training of a client-specific model. However, such an approach has a number of shortcomings. Consider, for instance, a federated learning system over millions of mobile devices, such as *Gboard* [HRM+19]. Only a fraction of all possible devices are likely to be seen during training, devices can drop out or in at any time, and it should be possible to produce models also for devices that enter the system at a later time. Having to train or finetune a new model for each such new device incurs computational costs. This is particularly unfortunate when the computation happens on the client devices, which are typically of low computational power. Furthermore, high communication costs emerge when model parameters or updates have to be communicated repeatedly between the server and the client. The summary effect is a high latency before the personalized model is ready for use. Additionally, the quality of the personalized model is unreliable, as it depends on the amount of data available on the client, which in personalized federated learning tends to be small.

In this chapter, we address the above challenges by introducing a new personalized federated learning algorithm. In PeFLL (for *Personalized Federated Learning by Learning to Learn*), it is the server that produces ready-to-use personalized models for any client by means of a single forward pass through a *hypernetwork*. The hypernetwork's input is a descriptor vector, which any client can produce by forward passing some of its data through an *embedding network*. The output of the hypernetwork is the parameters of a personalized model, which the client can use directly without the need for further training or finetuning. Thereby PeFLL overcomes the problems of high latency and client-side computation cost in model personalization from which previous approaches suffered when they required client-side optimization and/or multi-round client-server communication.

The combined effect of evaluating first the embedding network and then the hypernetwork constitutes a *learning method*: it produces model parameters from input data. The process

is parametrized by the networks' weights, which PeFLL trains end-to-end across a training set of clients. This *learning-to-learn* viewpoint allows for a number of advantageous design choices. For example, PeFLL benefits when many clients are available for training, where other approaches might suffer in prediction quality. The reason is that the client models are created by end-to-end trained networks, and the more clients participate in the training and the more diverse these are, the better the networks' generalization abilities to future clients. Also, PeFLL is able to handle clients with small datasets better than previous approaches. Because the descriptor evaluation is a feed-forward process, it is not susceptible to overfitting, as any form of on-client training or finetuning would be. At the same time, already during training the client descriptors are computed from mini-batches, thereby anticipating the setting of clients with little data.

PeFLL's strong empirical results, on which we report in Section 4.4 are not a coincidence, but they can be explained by its roots in theory. Its training objective is derived from a new generalization bound, which we prove in Section 4.3.2 and which yields tighter bounds in the federated setting than previous ones. PeFLL's training procedure distributes the necessary computation between the server and the clients in a way that ensures that the computationally most expensive task happen on the server yet stays truthful to the federated paradigm. The optimization is stateless, meaning that clients can drop out of the training set or enter it at any time. Nevertheless, as we show in Section 4.3.1, it converges at a rate comparable to standard federated averaging.

4.2 Related Work

Hypernetworks [HDL17] have only recently been employed in the context of pFL [MZGX22, YSW+23]. Typically, the hypernetworks reside on the clients, which limits the methods' efficiency. Closest to our work are [SNFC21] and [LCW+23], which also generate each clients' personalized model using server-side hypernetworks. These works adopt a non-parametric approach in which the server learns and stores individual descriptor vectors per client. However, such an approach has a number of downsides. First, it entails a stateful optimization problem, which is undesirable because it means the server has to know at any time in advance which client is participating in training to retrieve their descriptors. Second, the number of parameters stored at the server grows with the number of clients, which can cause scalability issues in large-scale applications. Finally, in such a setting the hypernetwork can only be evaluated for clients with which the server has interactive before. For new clients, descriptors must first to be inferred, and this requires an optimization procedure with multiple client-server communication rounds. In contrast, in PeFLL the server learns just the embedding network, and the client descriptors are computed on-the-fly using just forward-evaluations on the clients. As a consequence, clients can drop in or out at any time, any number of clients can be handled with a fixed memory budget, and even for previously unseen clients, no iterative optimization is required.

Learning-to-Learn. Outside of federated learning, the idea that learning from several tasks should make it easier to learn future tasks has appeared under different names, such as *continual learning* [Rin94, MCH+15], *lifelong learning* [TM95, CL18, PKP+19], *learning to learn* [TP98, ADG+16], *inductive bias learning* [Bax00], *meta-learning* [VD02, FAL17, HAMS21]. In this work, we use the term *learning-to-learn*, because it best describes the aspect that the central object that PeFLL learns is a mechanism for predicting model parameters from training data, i.e. essentially it learns a learning algorithm.

The central question of interest is if the learned learning algorithm *generalizes*, i.e., if it produces good models not only for the datasets used during its training phase, but also for future datasets. Corresponding results are typically generalization bounds in a PAC-Bayesian framework [PL14, AM18, RFJK21, GL22, Rez22], as we also provide in this work. Closest to our work in this regard is [Rez22], which addresses the task of hyperparameter learning. However, their bound is not well adapted to the federated learning setting in which the number of clients is large and the amount of data per client might be small. For a more detailed comparison, see Section 4.3.2 and Appendix B.2.

4.3 Method

We now formally introduce the problem we address and introduce the proposed PeFLL algorithm. In this chapter we assume a standard supervised federated learning setting. There are n clients and each of these has a data set, $S_i = \{(x_i^1, y_i^1), \dots, (x_i^{m_i}, y_i^{m_i})\} \sim D_i$, for $i \in \{1, \dots, n\}$. Again we adopt a *non-i.i.d.* setting, so the data distributions can differ between clients, $D_i \neq D_j$ for $i \neq j$. For any model, $\theta \in \mathbb{R}^d$ (we identify models with their parameter vectors), the client can compute its training loss, $\mathcal{L}(\theta; S_i) = \frac{1}{m_i} \sum_{j=1}^{m_i} \ell(x_i^j, y_i^j, \theta)$, where $\ell : \mathcal{X} \times \mathcal{Y} \times \mathbb{R}^d \to \mathbb{R}_+$ is a loss function. The goal is to learn client-specific models, θ_i, in a way that exploits the benefit of sharing information between clients, while adhering to the principles of federated learning.

PeFLL adopts a *hypernetwork* approach: for any client, a personalized model, $\theta \in \mathbb{R}^d$, is produced as the output of a shared deep network, $h : \mathbb{R}^l \to \mathbb{R}^d$, that takes as input a *client descriptor* $v \in \mathbb{R}^l$. To compute client descriptors, we use an embedding network, $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^l$, which takes individual data points as input and averages the resulting embeddings:[1] $v(S) = \frac{1}{|S|} \sum_{(x,y) \in S} \phi(x, y)$. We denote the hypernetwork's parameters as η_h and the embedding network's parameters as η_v. As shorthand, we write $\eta = (\eta_h, \eta_v)$.

Evaluating the embedding network followed by the hypernetwork can be seen as a *learning algorithm*: it transforms a set of training data points into model parameters. Consequently, learning η can be seen as a form of *learning-to-learn*. Specifically, we propose the *PeFLL (Personalized Federated Learning by Learning to Learn)* algorithm, which consists of solving the following optimization problem, where $\|\cdot\|$ denotes the L^2-norm,

$$\min_{\eta_h, \eta_v} \quad \lambda_h \|\eta_h\|^2 + \lambda_v \|\eta_v\|^2 + \sum_{i=1}^{n} \left[\mathcal{L}\Big(h(v(S_i; \eta_v); \eta_h); S_i \Big) + \lambda_\theta \Big\| h(v(S_i; \eta_v); \eta_h) \Big\|^2 \right]. \quad (4.1)$$

PeFLL performs this optimization in a way that is compatible with the constraints imposed by the federated learning setup. Structurally, it splits the objective (4.1) into several parts: a *server objective*, which consist of the first two (regularization) terms, $f(\eta_h, \eta_v) = \lambda_h \|\eta_h\|^2 + \lambda_v \|\eta_v\|^2$, and multiple *per-client objectives*, each of which consists of the terms inside the summation $f_i(\theta_i; S_i) = \mathcal{L}(\theta_i; S_i) + \lambda_\theta \Big\| \theta_i \Big\|^2$. The terms are coupled through the identity $\theta_i = h(v(S_i; \eta_v); \eta_h)$. This split allows PeFLL to distribute the necessary computation efficiently, preserving the privacy of the client data, and minimizing the necessary communication overhead. Pseudocode of the specific steps is provided in Algorithms 7 and 8.

[1] This construction is motivated by the fact that v should be a permutation invariant (set) function [ZKR+17], but for the sake of conciseness, one can also take it simply as an efficient design choice.

Algorithm 7 PeFLL-predict

input target client with private dataset S

1: *Server* sends embedding network η_v to *client*
2: *Client* selects a data batch $B \subseteq S$
3: *Client* computes $v = v(B; \eta_v)$
4: *Client* sends descriptor v to *server*
5: *Server* computes $\theta = h(v; \eta_h)$
6: *Server* sends personalized model θ to *client*

Algorithm 8 PeFLL-train

input number of training steps, T
input number of clients to select per step, c

1: **for** $t = 1, \dots, T$ **do**
2: $P \leftarrow$ *Server* randomly samples c clients
3: *Server* broadcasts embedding network η_v to P
4: **for** client $i \in P$ in parallel **do**
5: *Client* selects a data batch $B \subseteq S_i$
6: $v_i \leftarrow v(B; \eta_v)$, *client* computes descriptor
7: *Client* sends v_i to *server*
8: $\theta_i \leftarrow h(v_i; \eta_h)$ *server* computes personalized model
9: *Server* sends θ_i to *client*
10: $\theta_i^{\text{new}} \leftarrow$ client runs k steps of local SGD on $f_i(\theta_i)$
11: *Client* sends $\Delta\theta_i := \theta_i^{\text{new}} - \theta_i$ to *server*
12: $(\Delta\eta_h^{(i)}, \Delta v_i) \leftarrow$ server runs `backprop` with error vector $\Delta\theta_i$
13: *Server* sends Δv_i to *client*
14: $\Delta\eta_v^{(i)} \leftarrow$ client runs `backprop` with error vector Δv_i
15: *Client* sends $\Delta\eta_v^{(i)}$ to *server*
16: **end for**
17: $\eta_h \leftarrow (1 - 2\beta\lambda_h)\eta_h + \frac{1}{|P|}\sum_{i \in P} \Delta\eta_h^{(i)}$
18: $\eta_v \leftarrow (1 - 2\beta\lambda_v)\eta_v + \frac{1}{|P|}\sum_{i \in P} \Delta\eta_v^{(i)}$
19: **end for**

output network parameters (η_h, η_v)

We start by describing the PeFLL-predict routine (Algorithm 7), which can predict a personalized model for any target client. First, the server sends the current embedding network, η_v, to the client (line 1), which evaluates it on all or a subset of its data to compute the client descriptor (line 3). Next, the client sends its descriptor to the server (line 4), who evaluates the hypernetwork on it (line 5). The resulting personalized model is sent back to the client (line 6), where it is ready for use. Overall, only two server-to-client and one client-to-server communication steps are required before the client has obtained a functioning personalized model.

The PeFLL-train routine (Algorithm 8) mostly adopts a standard stochastic optimization pattern in a federated setting. In each iteration the server selects a batch of available clients (line 2) and broadcasts the embedding model, η_v, to all of them (line 3). Then, each client in parallel computes its descriptor, v_i, (6), sends it to the server (line 7), and receives a personalized model from the server in return (line 8, 9). At this point the forward pass is over and backpropagation starts. To this end, each client performs local SGD for k steps on its

personalized model and personal data (line 10). It sends the resulting update, $\Delta\theta_i$, to the server (line 11), where it acts as a proxy for $\frac{\partial f_i}{\partial \theta_i}$. According to the chain rule, $\frac{\partial f_i}{\partial \eta_h} = \frac{\partial f_i}{\partial \theta_i}\frac{\partial \theta_i}{\partial \eta_h}$ and $\frac{\partial f_i}{\partial v_i} = \frac{\partial f_i}{\partial \theta_i}\frac{\partial \theta_i}{\partial v_i}$. The server can evaluate both expressions using backpropagation (line 12), because all required expressions are available to it now. Thereby, it obtains updates $\Delta\eta_h^{(i)}$ and Δv_i, the latter of which it sends to the client (line 13) as a proxy for $\frac{\partial f_i}{\partial v_i}$. Again based on the chain rule ($\frac{\partial f_i}{\partial \eta_v} = \frac{\partial f_i}{\partial v_i}\frac{\partial v_i}{\partial \eta_v}$), the client computes an update vector for the embedding network, $\Delta\eta_v^{(i)}$ (line 14), and sends it back to the server (line 15). Finally, the server updates all network parameters from the average of the per-client contributions as well as the contributions from the server objective (lines 17, 18) [2].

The above analysis shows that PeFLL has a number of desirable properties: **1) It distributes the necessary computation in a client-friendly way**. The hypernetwork is stored on the server and evaluated only there. It is also never sent via the communication channel. This is important, because hypernetworks can be quite large, as already their output layer must have as many neurons as the total number of parameters of the client model. Therefore, they tend to have high memory and computational requirements, which client devices might not be able to provide.

2) It has low latency and communication cost. Generating a model for any client requires communicating only few and small quantities: a) the parameters of the embedding network, which are typically small, b) the client descriptors, which are low dimensional vectors, and c) the personalized models, which typically also can be kept small, because they only have to solve client-specific rather than general-purpose tasks. For the backward pass in the training phase, three additional quantities need to be communicated: a) the clients' suggested parameter updates, which are of equal size as the model parameters, b) gradients with respect to the client descriptors, which are of equal size as the descriptors, and c) the embedding network's updates, which are of the same size as the embedding network. All of these quantities are rather small compared to, e.g., the size of the hypernetwork, which PeFLL avoids sending.

3) It respects the federated paradigm. Clients are not required to share their training data but only their descriptors and—during training—the gradient of the loss with respect to the model parameters. Both of these are computed locally on the client devices. Note that we do not aim for formal privacy guarantees in this work. These could, in principle, be provided by incorporating *multi-party computation* [EKR18] and/or *differential privacy* [DR14b]. Since a descriptor is an average of a batch of embeddings, this step can be made differentially private with some added noise using standard DP techniques such as Gaussian or Laplace mechanisms. This, however, could add the need for additional trade-offs, so we leave the analysis to future work.

4.3.1 Convergence Guarantees

We now establish the convergence of PeFLL's training procedure. Specifically, we give guarantees in the form of bounding the expected average gradient norm, as is common for deep stochastic optimization algorithms. The proof and full formulation can be found in Appendix B.3.

[2] Note that all of these operations are standard first-order derivatives and matrix multiplications. In contrast to some meta-learning algorithms, no second-order derivatives, such as gradients of gradients, are required.

Theorem 2. *Under standard smoothness and boundedness assumptions (see appendix), PeFLL's optimization after T steps fulfills*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left\|\nabla F(\eta_t)\right\|^2 \leq \frac{(F(\eta_0)-F_*)}{\sqrt{cT}}+\frac{L(6\sigma_1^2+4k\gamma_G^2)}{k\sqrt{cT}}+\frac{224cL_1^2b_1^2b_2^2}{T}+\frac{8b_1^2\sigma_2^2}{b}+\frac{14L_1^2b_2^2\sigma_3^2}{b}, \tag{4.2}$$

where F is the PeFLL objective (4.1) with lower bound F_*. η_0 are the parameter values at initialization, $\eta_1, \ldots, \eta_T$ are the intermediate parameter values. L, L_1 are smoothness parameters of F and the local models. b_1, b_2 are bounds on the norms of the gradients of the local model and the hypernetwork, respectively. σ_1 is a bound on the variance of stochastic gradients of local models, and σ_2, σ_3 are bounds on the variance due to the clients generating models with data batches of size b instead of their whole training set. γ_G is a bound on the dissimilarity of clients, c is the number of clients participating at each round, and k is the number of local SGD steps performed by the clients.

The proof resembles convergence proofs for FedAvg with non-i.i.d. clients, but differs in three aspects that we believe makes our result of independent interest: 1) due to the non-linearity of the hypernetwork, gradients computed from batches might not be unbiased; 2) the updates to the network parameters are not simple averages, but consist of multiple gradient steps on the clients, which the server processes further using the chain rule; 3) the objective includes regularization terms that only the server can compute.

Discussion Theorem 2 characterizes the convergence rate of PeFLL's optimization step in terms of the number of iterations, T, and some problem-specific constants. For illustration, we first discuss the case where the client descriptors are computed from the complete client dataset ($B = S_i$ in Algorithm 7, line 5). In that case, σ_2 and σ_3 vanish, such that only three terms remain in (4.2). The first two are of order $\frac{1}{\sqrt{T}}$, while the third one is of order $\frac{1}{T}$. For sufficiently large T, the first two terms dominate, resulting in the same order of convergence as FedAvg [KKM$^+$20]. If clients compute their descriptors from batches ($B \subsetneq S_i$), two additional variance terms emerge in the bound, which depend on the size of the batches used by the clients to compute their descriptors. It is always possible to control these terms, though: for large S_i, one can choose B sufficiently large to make the additional terms as small as desired, and for small S_i, setting $B = S_i$ is practical, which will make the additional terms disappear completely, see above.

4.3.2 Generalization Guarantees

In this section, we prove a generalization bound that justifies PeFLL's learning objective. Following prior work in learning-to-learn theory we adopt a PAC-Bayesian framework. For this, we assume that the embedding network, the hypernetwork and the personalized models are all learned stochastically. Specifically, the result of training the hypernetwork and embedding network are Gaussian probability distributions over their parameters, $\mathcal{Q}_h = \mathcal{N}(\eta_h; \alpha_h \mathrm{Id})$, and $\mathcal{Q}_v = \mathcal{N}(\eta_v; \alpha_v \mathrm{Id})$, for some fixed $\alpha_h, \alpha_v > 0$. The mean vectors η_h and η_v are the learnable parameters, and Id is the identity matrix of the respective dimensions. The (stochastic) prediction model for a client with dataset S is obtained by sampling from the Gaussian distribution $Q = \mathcal{N}(\theta; \alpha_\theta \mathrm{Id})$ for $\theta = h(v(S; \eta_v); \eta_h)$ and some $\alpha_\theta > 0$.

We formalize clients as tuples (D_i, S_i), where D_i is their data distribution, and $S_i \overset{i.i.d.}{\sim} D_i$ is training data. For simplicity, we assume that all data sets are of identical size, m. We assume

that both observed clients and new clients are sampled from a *client environment*, $\mathcal{T}$ [Bax00], that is, a probability distribution over clients. Then, we can prove the following guarantee of generalization from observed to future clients. For the proof and a more general form of the result, see Appendix B.2.

Theorem 3. *For all $\delta > 0$ the following statement holds with probability at least $1 - \delta$ over the randomness of the clients. For all parameter vectors, $\eta = (\eta_h, \eta_v)$:*

$$\mathop{\mathbb{E}}_{(D,S)\sim\mathcal{T}} \mathop{\mathbb{E}}_{(x,y)\sim D} \mathop{\mathbb{E}}_{\substack{\bar{\eta}_h\sim\mathcal{Q}_h \\ \bar{\eta}_v\sim\mathcal{Q}_v}} \ell\Big(x, y, h(v(S;\bar{\eta}_v);\bar{\eta}_h)\Big) \leq \frac{1}{n}\sum_{i=1}^{n} \frac{1}{m} \sum_{(x,y)\in S_i} \mathop{\mathbb{E}}_{\substack{\bar{\eta}_h\sim\mathcal{Q}_h \\ \bar{\eta}_v\sim\mathcal{Q}_v}} \ell\Big(x, y, h(v(S_i;\bar{\eta}_v);\bar{\eta}_h)\Big)$$
$$+\sqrt{\frac{\frac{1}{2\alpha_h}\|\eta_h\|^2+\frac{1}{2\alpha_v}\|\eta_v\|^2+\log(\frac{2\sqrt{n}}{\delta})}{2n}} + \mathop{\mathbb{E}}_{\substack{\bar{\eta}_h\sim\mathcal{Q}_h \\ \bar{\eta}_v\sim\mathcal{Q}_v}} \sqrt{\frac{\frac{1}{2\alpha_\theta}\sum_{i=1}^{n}\|h(v(S_i;\bar{\eta}_v);\bar{\eta}_h)\|^2+\log(\frac{8mn}{\delta})+1}{2mn}} \tag{4.3}$$

Discussion Theorem 3 states that the expected loss of the learned models on future clients (which is the actual quantity of interest) can be controlled by the empirical loss on the observed clients' data (which we can compute) plus two terms that resemble regularizers. The first term penalizes extreme values in the parameters of the hypernetwork and the embedding network. Thereby, it prevents overfitting for the part of the learning process that accumulates information across clients. The second term penalizes extreme values in the output of the hypernetwork, which are the parameters of the per-client models. By this, it prevents overfitting on each client. Because the guarantee holds uniformly over all parameter vectors, we can optimize the right hand side with respect to η and the guarantee will still be fulfilled for the minimizer. The PeFLL objective is modeled after the right hand side of (4.3) to mirror this optimization in simplified form: we drop constant terms and use just the mean vectors of the network parameters instead of sampling them stochastically. Also, we drop the square roots from the regularization terms to make them numerically better behaved.

Relation to previous work A generalization bound of similar form as the one underlying Theorem 3 appeared in [Rez22, Theorem 5.2]. That bound, however, is not well suited to the federated setting. First, it contains a term of order $\frac{(n+m)\,\log m\sqrt{n}}{nm}$, which is not necessarily small for large n (number of clients) but small m (samples per client). In contrast, the corresponding terms in our bound, $\frac{\log\sqrt{n}}{n}$ and $\frac{\log nm}{nm}$, are both small in this regime. Second, when instantiating the bound from [Rez22] an additional term of order $\frac{1}{m}\|\eta\|$ would appear, which can be large in the case where the dimensionality of the network parameters is large but m is small.

4.4 Experiments

In this section we report on our experimental evaluation. The values reported in every table and plot are given as the mean together with the standard deviation across three random seeds.

Datasets For our experiments, we use three datasets that are standard benchmarks for federated learning: CIFAR10/CIFAR100 [Kri09] and FEMNIST [CWL+18]. Following prior pFL works, for CIFAR10 and CIFAR100 we form statistically heterogeneous clients by randomly assigning a fixed fraction of the total number of C classes to each of n clients, for $n \in$

$\{100, 500, 1000\}$. The clients then receive test and train samples from only these classes. For CIFAR10 each client has 2 of the $C = 10$ classes and for CIFAR100 each client has 10 of the $C = 100$ classes. FEMNIST is a federated dataset for handwritten character recognition, with $C = 62$ classes (digits and lower/upper case letters) and 817,851 samples. We keep its predefined partition into 3597 clients based on writer identity. We randomly partition the clients into 90% seen and 10% unseen. The seen clients are used for training while the unseen clients do not contribute to the initial training and are only used to later assess each method's performance on new clients as described below.

Baselines We evaluate and report results for the following pFL methods, for which we build on the *FL-Bench* repository [3]: Per-FedAvg [FMO20a], which optimizes the MAML [FAL17] objective in a federated setting; FedRep [CHMS21], which trains a global feature extractor and per-client classifier heads; pFedMe [DTN20], which trains a personal model per client using a regularization term to penalize differences from a global model; kNN-Per [MNVK22], which trains a single global model which each client uses individually to extract features of their data for use in a k-nearest-neighbor-based classifier; pFedHN [SNFC21] which jointly trains a hypernetwork and per client embedding vectors to output a personalized model for each client. For reference, we also include results of (non-personalized) FedAvg [MMR+17] and of training a local model separately on each client.

Constructing models for unseen clients We are interested in the performance of PeFLL not just on the training clients but also on clients not seen at training time. As described in Algorithm 7 inference on new clients is simple and efficient for unseen clients as it does not require any model training, by either the client or the server. With the exception of kNN-Per all other methods require some form of finetuning in order to obtain personalized models for new clients. Per-FedAvg and pFedMe obtain personal models by finetuning the global model locally at each client for some small number of gradient steps. FedRep freezes the global feature extractor and optimizes a randomly initialized head locally at each new client. pFedHN freezes the trained hypernetwork and optimizes a new embedding vector for each new client, which requires not just local training but also several communication rounds with the server. The most efficient baseline for inference on a new client is kNN-Per, which requires only a single forward pass through the trained global model and the evaluation of a k-nearest-neighbor-based predictor.

Models Following prior works in pFL the personalized model used by each client is a LeNet-style model [LBBH98a] with two convolutional layers and three fully connected layers. For fair comparison we use this model for PeFLL as well as all reported baselines. PeFLL uses an embedding network and a hypernetwork to generate this personalized client model. For our experiments the hypernetwork is a three layer fully connected network which takes as input a client descriptor vector, $v \in \mathbb{R}^l$, and outputs the parameters of the client model, $\theta \in \mathbb{R}^d$. For further details, see Appendix A.3. Note that the final layer of the client model predicts across all classes in the dataset. In case that a client knows which classes it wishes to predict for (e.g. from its training data) it can select only the outputs for those classes. For the embedding network we tested two options, a single linear projection which takes as input a one-hot encoded label vector, and a LeNet-type ConvNet with the same architecture as the client models except that its input is extended by C extra channels that encode the label in one-hot form. The choice of such small models for our embedding network is consistent

[3] https://github.com/KarhouTam/FL-bench

	CIFAR10			CIFAR100			FEMNIST
#trn.clients	$n = 90$	450	900	90	450	900	3237
Local	82.2 ± 0.6	70.9 ± 0.5	65.5 ± 0.7	39.4 ± 0.2	19.7 ± 0.2	11.3 ± 0.1	62.2 ± 0.1
FedAvg	47.5 ± 0.5	50.4 ± 0.6	51.9 ± 0.7	16.4 ± 0.4	20.2 ± 0.1	20.2 ± 0.2	82.1 ± 0.2
Per-FedAvg	79.1 ± 2.1	76.9 ± 0.9	76.6 ± 0.3	40.0 ± 0.3	31.5 ± 0.2	20.5 ± 0.1	82.7 ± 0.9
FedRep	84.6 ± 0.8	79.2 ± 0.6	77.3 ± 0.3	42.7 ± 0.8	35.5 ± 0.7	30.8 ± 0.5	83.6 ± 0.8
pFedMe	83.9 ± 1.2	79.0 ± 2.0	80.4 ± 0.9	39.6 ± 0.5	40.5 ± 0.4	34.6 ± 0.2	85.9 ± 0.8
kNN-Per	84.0 ± 0.3	82.0 ± 0.5	79.9 ± 0.7	42.2 ± 0.9	38.1 ± 0.4	34.5 ± 0.5	85.2 ± 0.3
pFedHN	87.8 ± 0.5	77.4 ± 1.3	66.0 ± 1.1	53.6 ± 0.2	25.5 ± 0.3	20.0 ± 1.2	83.8 ± 0.3
PeFLL	$\mathbf{89.0 \pm 0.8}$	$\mathbf{88.9 \pm 0.4}$	$\mathbf{87.8 \pm 0.4}$	$\mathbf{56.0 \pm 0.3}$	$\mathbf{43.2 \pm 0.8}$	$\mathbf{40.9 \pm 0.6}$	$\mathbf{90.1 \pm 0.1}$

(a) Accuracy on clients observed at training time (best values per setup in bold)

	CIFAR10			CIFAR100			FEMNIST
#trn.clients	$n = 90$	450	900	90	450	900	3237
Local	81.6 ± 0.7	70.6 ± 1.8	65.5 ± 0.5	38.7 ± 0.9	19.9 ± 1.1	11.0 ± 0.5	62.1 ± 0.2
FedAvg	48.6 ± 0.3	49.6 ± 1.0	51.7 ± 0.7	17.1 ± 0.9	19.3 ± 2.0	20.5 ± 1.0	81.9 ± 0.4
Per-FedAvg	77.6 ± 1.9	69.0 ± 2.5	67.3 ± 0.8	35.6 ± 1.5	30.6 ± 0.4	20.5 ± 0.5	81.1 ± 1.5
FedRep	81.8 ± 1.0	77.4 ± 1.7	76.0 ± 0.3	38.6 ± 1.1	35.9 ± 1.5	31.8 ± 1.6	82.8 ± 0.7
pFedMe	78.4 ± 2.2	74.3 ± 1.4	72.3 ± 1.7	31.1 ± 1.4	32.3 ± 0.5	28.9 ± 0.2	86.1 ± 0.4
kNN-Per	82.4 ± 0.7	80.8 ± 1.5	78.4 ± 1.1	$\mathbf{41.8 \pm 1.2}$	37.2 ± 2.0	33.7 ± 0.7	84.6 ± 0.6
pFedHN	63.3 ± 3.7	60.9 ± 2.4	57.8 ± 2.0	24.1 ± 1.2	21.5 ± 1.4	20.8 ± 1.2	82.5 ± 0.1
PeFLL	$\mathbf{89.3 \pm 2.2}$	$\mathbf{88.9 \pm 0.6}$	$\mathbf{88.5 \pm 0.3}$	35.2 ± 1.4	$\mathbf{38.1 \pm 0.4}$	$\mathbf{38.4 \pm 0.9}$	$\mathbf{90.7 \pm 0.2}$

(b) Accuracy on clients not observed at training time (best values per setup in bold)

Table 4.1: Experimental results on standard pFL benchmarks. In all settings, except CIFAR100 with 100 clients, PeFLL achieves higher accuracy than previous methods, and the accuracy difference between training clients (top table) and unseen clients (bottom table) is small, if present at all.

with the fact that the embedding network must be transmitted to the client. We find that for CIFAR10 and FEMNIST the ConvNet embedding network produces the best results while for CIFAR100 the linear embedding is best, and these are the results we report.

Training Details We train all methods, except Local, for 5000 rounds with partial client participation. For CIFAR10 and CIFAR100 client participation is set to 5% per round. For FEMNIST we fix the number of clients participating per round to 5. The Local baseline trains on each client independently for 200 epochs. The hyperparameters for all methods are tuned using validation data that was held out from the training set (10,000 samples for CIFAR10 and CIFAR100, spread across the clients, and 10% of each client's data for FEMNIST). The optimizer used for training at the client is SGD with a batch size of 32, a learning rate chosen via grid search and momentum set to 0.9. The batch size used for computing the descriptor is also 32. More details of the hyperparameter selection for each method as well as ablation studies are provided in Appendix A.3.

4.4.1 Results

Table 4.1 shows the results for PeFLL and the baseline methods. In all cases, we report the test set accuracy on the clients that were used for training (Table 4.1a) and on new clients that were not part of the training process (Table 4.1b). **PeFLL achieves the best results in all cases, and often by a large margin.** The improvements over previous methods are most prominent for previously unseen clients, for which PeFLL produces model of almost identical accuracy as for the clients used for training. This is especially remarkable in light of

the fact that several of the other methods have computationally more expensive procedures for generating models in this setting than PeFLL, in particular requiring on-client or even federated training to produce the personalized models. We see this result as a strong evidence that PeFLL successful generalizes, as predicted by Theorem 3.

Comparing PeFLL's results to the most similar baseline, pFedHN, one observes that the latter's performance decreases noticeably when the number of clients increases and the number of samples per client decrease accordingly. We attribute this to the fact that pFedHN learns independent client descriptors for each client, which can become unreliable if only few training examples are available for each client. Similarly, for Per-FedAvg, pFedMe and FedRep, which construct personalized models by local finetuning, the model accuracy drops when the amount of data per client decreases, especially in the more challenging CIFAR100 setup. kNN-Per maintains good generalization from train to unseen clients, but its performance worsens when the number of samples per client drops and the kNN predictors have less client data available. In contrast, PeFLL's performance remains stable, which we attribute to the fact that it learns the embedding and hypernetwork jointly from all available data. The new client does not have to use its data for training or finetuning, but rather just to generate a client descriptor. As a pure feed-forward process, this step does not suffer from overfitting in the low-data regime.

Personalization for clients with only unlabeled data A scenario of particular interest is when some clients have only unlabeled data available. PeFLL can still provide personalized models for them in the following way. To train PeFLL, as before one uses only clients which do have labeled data. However, the embedding network is constructed to take only the unlabeled parts of the client data as input (i.e. it ignores the labels). Consequently, even clients that only have unlabeled data can evaluate the embedding network and obtain descriptors, so the hypernetwork can assign personalized models to them during the text phase. We report results for this in Table 4.2.

Note that none of the personalized baseline methods from Section 4.4 are able to handle this setting, because the train or finetune on the test clients. Therefore, the only alternative would be to train a non-personalized model, e.g. from FedAvg, and use this for clients with only labeled data. One can see that Personalized Federated Learning by Learning to Learn improves the accuracy over this baseline in all cases.

	CIFAR10			CIFAR100		
#trn.clients	$n = 90$	450	900	90	450	900
FedAvg	48.6 ± 0.3	49.6 ± 1.0	51.7 ± 0.7	17.1 ± 0.9	19.3 ± 2.0	20.5 ± 1.0
PeFLL	84.5 ± 0.6	83.6 ± 1.0	81.5 ± 0.7	26.3 ± 0.4	28.2 ± 0.6	30.3 ± 0.6

Table 4.2: Test accuracy on CIFAR10 and CIFAR100 for (test) clients that have only unlabeled data.

Analysis of Client Descriptors Our work relies on the hypothesis that clients with similar data distributions should obtain similar descriptors, such that the hypernetwork then produces similar models for them. To study this hypothesis empirically, we create clients of different similarities to each other in the following way. Let C denote the number of classes and n the number of clients. Then, for each client i we sample a vector of class proportions, $\pi_i \in \Delta^C$, from a Dirichlet distribution Dir($\boldsymbol{\alpha}$) with parameter vector $\boldsymbol{\alpha} = (0.1, \ldots, 0.1)$, where Δ^C is the unitary simplex of dimension C. Each client then receives a dataset, S_i, of samples from

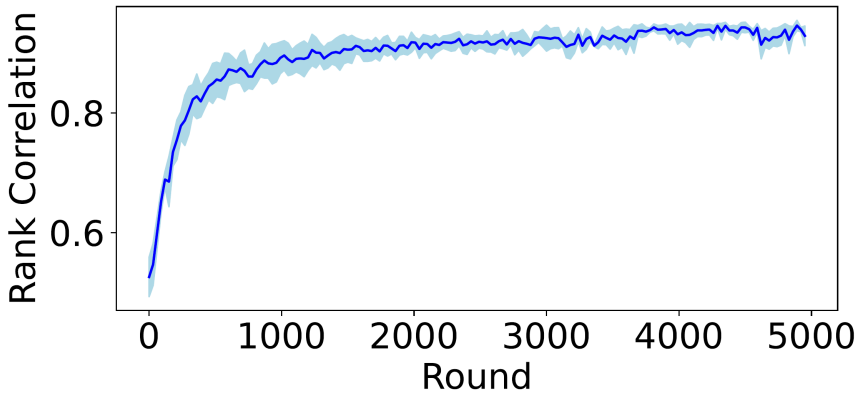

Figure 4.1: Correlation between *client descriptor similarity* obtained from the embedding network and *ground truth similarity* over the course of training.

each of the C classes according to its class proportion vector π_i. We randomly split the clients into train and unseen clients, and we run PeFLL on the train clients. Throughout training we periodically use the embedding network to compute all client descriptors in order to track how they evolve.

For any pair of such clients, we compute two similarity measures: $d^{\pi}_{ij} = \|\pi_i - \pi_j\|$, which provides a ground truth notion of similarity between client distributions, and $d^{v}_{ij} = \|v(S_i) - v(S_j)\|$, which measures the similarity between client descriptors. To measure the extent to which d^v reflects d^π, we use their average rank correlation in the following way. For any unseen client i we a form vector of similarities $d^v_i = (d^v_{i1}, \dots, d^v_{in}) \in \mathbb{R}^n$, and compute its Spearman's rank correlation coefficient [Spe04] to the corresponding ground truth vector $d^\pi_i = (d^\pi_{i1}, \dots, d^\pi_{in}) \in \mathbb{R}^n$. Figure 4.1 shows the average of this value across the unseen clients after different numbers of training steps (CIFAR 10 dataset, 1000 clients in total). One can see that the rank correlation increases over the course of training, reaching very high correlation values of approximately 0.93. Results for other numbers of clients are comparable, see Appendix A.3. This indicates that the embedding network indeed learns to organize the descriptor space according to client distributional similarity, with similar clients being mapped to similar descriptors. Moreover, because these results are obtained from unseen clients, it is clear that the effect does not reflect potential overfitting of the embedding network to the training clients, but that the learned similarity indeed generalizes well.

4.5 Conclusion

In this chapter, we presented PeFLL, an algorithm for personalized federated learning based on learning-to-learn. By means of an embedding network that creates inputs to a hypernetwork it efficiently generates personalized models, both for clients present during training and new clients that appear later. PeFLL has several desirable properties for real-world usability. It stands on solid theoretical foundation in terms of convergence and generalization. It is trained by stateless optimization and does not require additional training steps to obtain models for new clients. Overall, it has low latency and puts little computational burden on the clients, making it practical to use in large scale applications with many clients each possessing little data.

CHAPTER 5

Federated Learning with Unlabeled Clients: Personalization Can Happen in Low Dimensions

5.1 Motivation and Outline

In the previous chapter we touched on some of the disadvantages of personalization through some form of finetuning on the client local data. Our focus in that chapter was primarily on addressing efficiency disadvantages of such approaches, both in terms of computation and communication. An issue we briefly touched on, but did not focus on in much detail, is the fact that such finetuning based approaches require a client to have labeled data in order to obtain a personalized model. This poses a major challenge for personalization in applications where active participation from the client (i.e. user) is required to create labels for their local data. In such scenarios, even though some clients will likely be *labeled* (have data with labels), the majority of potentially participating clients will likely be *unlabeled* (i.e. have no labels for their data). Similarly, most future clients will potentially also be unlabeled.

In this chapter we address this issue with FLowDUP: **F**ederated **Low D**imensional **U**nlabeled **P**ersonalization. **FLowDUP learns to generate a personalized model for any client, even if it has only unlabeled data.** To do so, it trains a hypernetwork [HDL17] that takes as input an unlabeled dataset and outputs the parameters of a personalized predictive model. Crucially, rather than outputting all the parameters of a personalized model, the hypernetwork instead outputs a parametrization in a subspace of much smaller dimension than the underlying personalized model. The full model is obtained by multiplying the subspace parameters with a fixed (random) *expansion matrix*. The use of a low-dimensional subspace is a crucial contribution of our work. It allows for the generation of large model architectures and means that the hypernetwork is small and efficient enough to be transmitted to and run on the client devices. This allows FLowDUP to adhere strictly to the federated paradigm that clients data should not be transferred away from the device. Without it, hypernetwork-based methods are only able to generate very small models and are only practical if the hypernetwork runs on the central server [SNFC21, AEC24, SZL24], which is at odds with the FL principle that clients' data never leaves the clients' device.

We propose a learning objective for FLowDUP that consists of two parts. The first part

measures the quality of the generated client models, for which it exploits that some of the clients at training time do have labels. The second part prevents overfitting by penalizing large deviations between the generated models and a learned regularization term. Evaluating it requires only unlabeled data, so it can be computed from all clients available during training. FLowDUP and its learning objective are motivated by theoretical results derived in a multitask framework. Specifically, we prove a generalization bound that provides performance guarantees on unlabeled clients. Our learning objective is derived by optimizing terms that appear in this bound.

In addition to our theoretical contributions we carry out an experimental evaluation over a range of datasets, exhibiting various types of statistical heterogeneity, and at a range of proportions of labeled clients present during training. We conduct additional experiments and ablation studies to better understand the different components of FLowDUP.

5.2 Related work

Personalized federated learning for unlabeled clients First proposed by [SCST17a], personalized FL arose from the observation that statistical heterogeneity of client data distributions can make training a single global model suboptimal [KMA+21]. However, most works in personalized FL assume that all clients hold labeled data that they can use to obtain a personalized model, see Appendix C.2. A small number of works have looked at the problem of personalizing models to unlabeled clients. [YCW+24] apply a popular test time adaptation method to a FL setting. A global predictive model is personalized on a client using gradient updates obtained with forward passes through an adaptation model. Closest to our own work are [AEC24, SZL24] which use a combination of an embedding network and hypernetwork to generate personalized models using only unlabeled data. Their approaches differ from ours in two key aspects. Firstly, while they are able to produce personalized models for unlabeled clients, they are not able to use the unlabeled clients during training. In contrast, FLowDUP also leverages unlabeled clients during training, which leads to improved performance. Secondly, and more importantly, they generate all the parameters of the personalized model, rather than a low dimensional parameterization as FLowDUP does. Due to the high computational cost of this approach they are only able to generate very small personalized models for the clients. Moreover, even for such small models the hypernetwork is too large to send to the clients and instead remains on the server. This leads to complicated multi-step communication schemes that are inefficient and break the federated learning paradigm of working only with aggregate client statistics in order to maintain privacy. Finally, several works have explored Federated Representation Learning in the presence of statistically heterogeneous clients [ZKY+20, JHJY22]. These methods typically aim to learn a feature extractor(s) using unlabeled client data. However, obtaining a personalized predictive model on the client would still require the client to possess labels, and as such these methods are not applicable to the task we aim to solve.

Theory Beyond standard single-task learning, it has been shown that in *multitask learning (MTL)* [Car97, Bax95], when learning multiple related tasks, sharing information between them can provably improve the performance. Specifically, different works had studied the generalization behavior of multitask learning [Mau06, CM12, PM13, PL17, YLK+18, DHK+21] using notions like VC-dimensions [VC71] or Rademacher complexity [BM02]. Recently, with the success of PAC-Bayesian bounds for studying generalization behavior of neural networks [DR17],

and due to their strength in parameterizing multitask learning and meta-learning [Sch87], PAC-Bayes multitask learning has become an active line of research [PL14, AM18, RJFK23, GL22, ZBL24], However, the focus of these works is *inductive multitask learning*, where all tasks have access to labeled data. In contrast, in this work, we introduce *transductive multitask learning*, where only a subset of the tasks have labeled data.

5.3 Background

Notation Again n denotes the number of clients participating in training. Each client $i \in \{1, \dots, n\}$ possesses a data distribution D_i over the shared inputs and output sets, $\mathcal{X} \times \mathcal{Y}$. We call n_L the number of clients that hold labeled data and n_U the number that hold only unlabeled data, i.e. $n_L + n_U = n$. We denote by $\mathcal{I}_L$ and $\mathcal{I}_U$ the indices of the labeled and unlabeled clients respectively. So in this chapter we have datasets S_i, with $m_i = |S_i|$, and

$$S_i := \begin{cases} (X_i, Y_i) = \left(x_i^j, y_i^j\right)_{j=1}^{m_i} \sim D_i & \text{if } i \in \mathcal{I}_L, \\ X_i = \left(x_i^j\right)_{j=1}^{m_i} \sim D_{i|\mathcal{X}} & \text{if } i \in \mathcal{I}_U, \end{cases} \tag{5.1}$$

where $D_{i|\mathcal{X}}$ denotes the marginal distribution of D_i over $\mathcal{X}$. We denote by $f(\cdot\,;\theta) : \mathcal{X} \to \mathcal{Y}$ a predictive model parameterized by $\theta \in \mathbb{R}^d$, and by $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ a loss function, such that $\ell(y, f(x;\theta))$ measures the prediction quality of $f(\cdot\,;\theta)$.

Learning in a subspace Modern neural networks use models with many parameters, often more than the number of training examples. However, it has been shown that their *intrinsic dimensionality* is much smaller than their number of parameters [LFLY18], i.e. it is possible to learn strong models in a random subspace of much smaller dimension than the full dimension of the model parameter space. Formally, to learn model parameters $\theta \in \mathbb{R}^d$, given an initialization model $\theta_0 \in \mathbb{R}^d$, and a random expansion matrix $P \in \mathbb{R}^{d \times k}$ (which describes the basis of a random subspace), we can describe a model in the generated random subspace by learning a vector $v \in \mathbb{R}^k$ as

$$\theta = \theta_0 + Pv. \tag{5.2}$$

Prior work in standard learning [LFLY18] and recently multitask learning [ZGL25] showed that k can typically be chosen orders of magnitude smaller than d while still allowing high accuracy models to be trained.

In this work, we keep the matrix P and the initialization θ_0 fixed, such that θ is completely determined by v. With a slight abuse of notation, we then also write $f(\cdot\,; v)$ as shorthand for $f(\cdot\,; \theta_0 + Pv)$.

5.4 Personalized federated learning with unlabeled clients

In this section we present FLowDUP. In 5.4.1 we introduce the problem we are aiming to solve and propose FLowDUP's learning objective. In 5.4.2 we present the FLowDUP's training algorithm and in 5.4.3 we present how to use FLowDUP post training to generate personalized models for unlabeled clients.

5.4.1 Training objective

Our goal is to generate personalized models for clients that possess only unlabeled data. We assume we have access to labeled and potentially also unlabeled clients during training, and that future clients we will encounter might be unlabeled.

The primary object we work with is a hypernetwork $h : \mathcal{P}(\mathcal{X}) \to \mathbb{R}^k$, where $\mathcal{P}(\mathcal{X})$ denotes the power set of $\mathcal{X}$. The goal of h is to take in an unlabeled client dataset and output the (low dimensional subspace) parameters of a personalized model that works for the underlying client data distribution. Formally, if $X \subset \mathcal{X}$ then h is defined as

$$h(X) \coloneqq h_2\left(\frac{1}{|X|}\sum_{x\in X} h_1(x)\right), \tag{5.3}$$

where $h_1 : \mathcal{X} \to \mathbb{R}^e$ is feature extraction module, which is followed by an average across the (batch of) features, and a fully-connected module $h_2 : \mathbb{R}^e \to \mathbb{R}^k$.

To generate a client model f given X, we first compute $v = h(X)$, then the full-dimensional parameters for f are obtained by random expansion: $\theta = \theta_0 + Pv$ as defined in (5.2). The final personalized model is $f(\cdot\,;\theta) : \mathcal{X} \to \mathcal{Y}$. Note that P is a random matrix and the initialization θ_0 is typically random, and both are fixed before training. Therefore, these matrices can be generated on the client using an agreed-on random seed, and do not need to be transmitted via the network.

We denote the trainable parameters of h by ψ_h. Additionally, FLowDUP training also uses a learnable regularization, $\psi_r \in \mathbb{R}^k$, in the low dimensional model subspace to prevent overfitting. The learnable parameters of FLowDUP are therefore $\psi \coloneqq (\psi_h, \psi_r)$.

Training FLowDUP means to learn parameters ψ that, given unlabeled data X, are able to output personalized client model parameters that work on the distribution that X was drawn from. Our proposed training objective for doing this has two parts, a loss $\mathcal{L}$, and a (learnable) regularizer, Ω.

$$\min_{\psi}\ \mathcal{L}(\psi) + \lambda\Omega(\psi), \tag{5.4}$$

The loss measures the quality of a model that is generated for a client, and can therefore be evaluated only on clients that have at least some labeled data,

$$\mathcal{L}(\psi) \coloneqq \sum_{i\in\mathcal{C}} \sum_{(x',y')\in(X_i',Y_i')} \ell(y', f(x';\, h(X_i\,;\,\psi_h))), \tag{5.5}$$

where, $\mathcal{C}$ is a cohort (subset) of clients, X_i is a batch of data without labels from client i and (X_i', Y_i') is a batch of data with labels from client i. In case (X_i, Y_i) are empty, for example because client i has no labeled data, the value of the sum is simply taken to be 0. Intuitively, $\mathcal{L}$ penalizes the hypernetwork parameters ψ_h for outputting models that perform poorly on the clients' data distributions. Notice that different data batches are used to generate the client model and evaluate it. This is because we want f to be good on the client distribution, and not just on the actual batch used to generate f.

The regularizer ensures that the learned models do not diverge too much from each other. It prevents overfitting to individual clients by penalizing large deviations between the client model parameters and a global learned regularizer. It can be computed on all clients, because only unlabeled data is required for its evaluation,

$$\Omega(\psi) \coloneqq \sum_{i\in\mathcal{C}} \|h(X_i\,;\,\psi_h) - \psi_r\|^2, \tag{5.6}$$

Algorithm 9 FLowDUP- Server

Input: Client datasets $\{S_i\}_{i=1}^n$, number of rounds T, global learning rate η_g, labeled client sampling rate α
Initialize learnable parameters: ψ
for round $t = 1$ to T **do**
  Server selects a subset of clients $\mathcal{C}$, with fraction α labeled
  for each client $i \in \mathcal{C}$ **in parallel do**
    Client i receives current parameters ψ
    Client i performs local updates: $\Delta\psi_i \leftarrow ClientUpdate(S_i, \psi)$
    Client i sends update $\Delta\psi_i$ to server
  end for
  Server aggregates updates: $\Delta\psi \leftarrow \frac{1}{|\mathcal{C}|}\sum_{i\in\mathcal{C}} \Delta\psi_i$
  Server updates parameters: $\psi \leftarrow \texttt{GradientUpdate}(\psi, \Delta\psi\,;\, \eta_g)$
end for
Return: Final parameters ψ

Algorithm 10 FLowDUP- ClientUpdate

Input: client i, dataset S_i, learnable parameters ψ, number of local epochs E, local batch size B, local learning rate η_l, regularization strength λ
$\psi_{\text{start}} \leftarrow \psi$
for local epoch $j = 1$ to E **do**
  $\mathcal{B} \leftarrow$ client splits S_i into batches of size B
  for each batch in $\mathcal{B}$ **do**
    $X \leftarrow$ random half of batch of data (no labels required)
    $(X', Y') \leftarrow$ remaining half of batch data (with labels if available, else $Y' \leftarrow \emptyset$)
    $v \leftarrow h(X; \psi_h)$
    $\Omega \leftarrow \|v - \psi_r\|^2$
    $\theta \leftarrow \theta_0 + Pv$
    $\mathcal{L} \leftarrow \sum_{(x',y')\in(X',Y')} \ell(y', f(x', \theta))$ // interpreted as $\mathcal{L} \leftarrow 0$ if $Y' = \emptyset$
    $g \leftarrow \nabla_\psi(\mathcal{L} + \lambda\Omega)$
    $\psi \leftarrow \texttt{GradientUpdate}(\psi, g; \eta_l)$ // any suitable optimizers, e.g. SGD or Adam
  end for
end for
Return: parameter update: $\psi - \psi_{\text{start}}$

where, again, $\mathcal{C}$ is a cohort of clients and X_i is an unlabeled batch of data from client i. Note that, as required for federated learning, the participating clients and data batches can be different in every update step. The regularization strength, $\lambda \in \mathbb{R}$, is a hyperparameter. We provide a more in depth and formal motivation for our loss function $\mathcal{L}$, based on our generalization bound, in Section 5.5.

5.4.2 Training procedure

FLowDUP is designed to be trainable by standard federated learning frameworks. Algorithms 9 and 10 show the necessary steps. Algorithm 9 shows the server steps of FLowDUP that follow a standard Federated Averaging [MMR+17] pattern. Training proceeds over a number of rounds.

Each round, the server samples a subset of clients with the restriction that a certain fraction, α, are clients with labeled data. Each client in the subset receives the current parameters ψ from the server, computes an update to ψ and shares this update with the server. The server then aggregates the client updates and uses the aggregate to update the learnable parameters using an update rule of their choice.

The most important step, namely the ClientUpdate, is shown in Algorithm 10. The client receives the latest iteration of the parameters ψ from the server and computes an update direction to those parameters by training on their local data for E local epochs. At each epoch, the client samples a batch of data (line 5) and randomly splits the batch into two equal parts (lines 6, 7). The first part is used to generate the subspace parameter v (line 8), note that no labels are required for this step. The client then computes its contribution to the regularizer value Ω (line 9). Next, the labeled clients compute the labeled portion of the loss $\mathcal{L}$ (line 11) of the generated client model on the second half of the batch. The total loss is computed (line 12) and the client updates the learnable parameters ψ with a single gradient step (line 13). After all local epochs have been run, the client computes the difference between the final and initial parameters (line 16) and returns this update to the server.

5.4.3 Model generation

After training, inference with FLowDUP is simple and lightweight, in particular it requires no model training or finetuning, just a single forward pass through h. Given a client, with unlabeled data $X = (x)_{j=1}^{m}$, the client gets ψ from the server and generates a personalized model by passing X through the hypernetwork: $v = h(X\,;\,\psi_h)$ and expanding to full dimensionality: $\theta = \theta_0 + Pv$. The client then has a personalized classifier $f(\cdot\,;\,\theta)$ that it can use (either on X and/or on future data).

5.5 Theory

In this section, we provide a generalization bound for *transductive multitask learning* in the PAC-Bayesian framework, which provides a theoretical justification for our algorithm. In contrast to *inductive learning*, which uses a labeled training dataset to learn a model for inference on future samples, *transductive learning* [Vap98], involves access to a set of unlabeled data for which prediction are desired. Similarly, in transductive multitask learning, there are both labeled and unlabeled tasks, and the goal is to learn models for all tasks jointly. As our main theoretical contribution, we prove a PAC-Bayesian generalization bound for transductive multitask learning for stochastic algorithms that generate a model given an unlabeled dataset. The proof and the general results are provided in Appendix C.3. Here we provide a result tailored for our algorithm. Specifically, define a Gaussian distribution over the parameters of the hypernetwork, $\rho_h = \mathcal{N}(\psi_h\,;\,\alpha_h \mathsf{Id})$ for a fixed α_h, and n posterior models for n clients as $Q_i = \mathcal{N}(h(S_i\,;\,\psi_h)\,;\,\alpha_\theta \mathsf{Id})$, and a regularization distribution as $\mathcal{Q} = \mathcal{N}(\psi_r\,;\,\alpha_r \mathsf{Id})$. Our result bounds the gap between the true risk, er, of all clients

$$\mathsf{er}(\rho_h) = \mathbb{E}_{\psi'_h \sim \rho_h} \frac{1}{n} \sum_{i=1}^{n} \mathbb{E}_{(x',y')\sim\mathcal{D}_i} \ell\Big(y', f(x';\, h(S_i\,;\,\psi'_h))\Big), \tag{5.7}$$

and the training risk, $\widehat{\mathsf{er}}$, of the labeled clients

$$\widehat{\mathsf{er}}(\rho_h) = \mathbb{E}_{\psi'_h \sim \rho_h} \frac{1}{n_L} \sum_{i=1}^{n_L} \frac{1}{m} \sum_{j=1}^{m} \ell\Big(y_i^j, f(x_i^j;\, h(S_i\,;\,\psi'_h))\Big).$$

Theorem 4. *For all $\delta > 0$, and any loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, 1]$, the following statement holds with probability at least $1-\delta$ over the sampling of n_L clients of n clients and randomness of the dataset. For all parameter vectors, $\psi = (\psi_h, \psi_r)$:*

$$er(\rho_h) \leq \widehat{er}(\rho_h) \; + \; \sqrt{\left(1 - \frac{n_L}{n}\right) \frac{\frac{1}{2\alpha_h}\|\psi_h\|^2 + c_1}{2n_L}} \\ + \; \mathop{\mathbb{E}}_{\psi'_h \sim \rho} \sqrt{\frac{\frac{1}{2\alpha_\theta} \sum_{i=1}^{n} \mathbb{E}_{\psi'_r} \|h(S_i\,;\,\psi'_h) - \psi'_r\|^2 + \frac{1}{2\alpha_r}\|\psi_r\|^2 + c_2}{2mn}} \tag{5.8}$$

where c_1 and c_2 are logarithmic terms in n and n_L.

Discussion. Theorem 4 provides a direct mathematical justification for FLowDUP. Our overall goal is to achieve high accuracy across all tasks, i.e. minimize the left hand side of (5.8). That is not a computable quantity, though, so as a proxy we instead minimize its upper bound on the right hand side of (5.8). That consists of $\widehat{\text{er}}(\rho_h)$, which corresponds to the training loss across labeled tasks, i.e. FLowDUP's loss $\mathcal{L}$, and some complexity terms. Besides $\|\psi_h\|$ and $\|\psi_r\|$, which are automatically minimized when using optimization steps with weight decay, the latter in particular contains $\sum_{i=1}^{n} \mathbb{E}_{\psi'_r} \|h(S_i\,;\,\psi'_h) - \psi'_r\|^2$, which corresponds to FLowDUP's regularizer Ω. The difference between the theoretical viewpoint and the practical algorithm is that in practice we use deterministic neural networks instead of stochastic models, so no expected value operations are required. Also, we treat the regularization strength as a hyper-parameter that can be model-selected, instead of relying on the (often overly pessimistic) values provided by generalization theory.

On a technical level, Theorem 4 has a number of desirable properties. Firstly, it directly reflects the transductive setting, as it controls the risk across all clients in terms of the training loss of just the labeled ones. The complexity term, however, is computed from all clients and its denominator scales with the total number of available samples, which justifies the use of unlabeled clients during training, which contribute to the learning of a shared regularizer. Finally, the sample complexity of the first complexity term is improved by $\sqrt{1 - n_L/n}$ compared to standard inductive bounds, which holds the promise of better between-client generalization in transductive compared to inductive learning.

5.6 Experiments

Here we present our empirical results. In 5.6.1 we explain our experimental setup, in 5.6.2 we include our main results and in 5.6.3 we provide additional experiments and ablation studies to better understand FLowDUP. Implementation details and further ablation studies can be found in Appendix C.1. The values in all results tables are given as the mean and standard deviation of runs over 3 random seeds. We implement our experiments using the `pfl-research` library [GSC+24].

5.6.1 Experimental Details

Baselines We are interested in producing a personalized model for an unlabeled client and consider only baselines that are capable of doing this. The simplest to consider are those that train a single (full dimensional) global model f on just the labeled clients. This of course results in a predictive model that can be used by any unlabeled client. In this category we include

Federated Averaging (FedAvg) [MMR+17] and FedProx [LSZ+20]. We also include a baseline that we call LD-FedAvg that trains the low dimensional subspace parameterization of f, using federated averaging. This baseline therefore has the same client model f parameterization as the personalized models generated by FLowDUP. Finally, we include FedTTA [YCW+24] a federated test time adaption method. It personalizes a globally trained model f to an unlabeled client with a gradient update obtained by a forward pass with unlabeled data through an adaptation model.

Datasets We include a range of datasets with different types of statistically heterogeneous clients. Following prior work in personalized FL we simulate label heterogeneity with a non-IID partitioning of CIFAR-10 [Kri09], where each client receives 100 datapoints from only 2 classes. We simulate heterogeneity in the features $\mathcal{X}$ by partitioning and rotating Fashion-MNIST [XRV17] and MNIST [LBBH98b], as first done by [GCYR20a]. Specifically, we do this by creating an IID partitioning of the dataset into clients, where each client receives 100 randomly IID sampled datapoints. Then each client randomly samples a rotation from $\{0°, 90°, 180°, 270°\}$ and rotates all of their images by that amount. Finally, we also include results for FEMNIST [CWL+18], a federated dataset with 62 classes and an existing partition into around 3500 clients, holding in total ≈ 800000 datapoints. The clients in FEMNIST exhibit both feature (different writing styles) and label (different class frequencies per client) heterogeneity. For each of the above datasets we simulate a range of different levels of label prevalence in the clients. Specifically, for $p \in \{0.1, 0.2, 1.0\}$ we randomly choose pn of the n total clients to have labels and the rest are fully unlabeled.

Network architectures We run experiments using two different architectures for the client model f, a CNN following prior work [MMR+17] and a ResNet18 [HZRS16]. On partitioned CIFAR10 we present results for both the CNN and the ResNet18, for the remaining datasets we use a CNN. For FLowDUP recall the structure of h is: $h(X) = h_2(\frac{1}{|X|}\sum_{x \in X} h_1(x))$. For our main results in Section 5.6.2 we always implement h_1 using the same architecture (CNN or ResNet18) as f for that particular setting. In Section 5.6.3 we explore the case when they differ. We implement h_2 as a fully connected network with a single hidden layer. Both FLowDUP and LD-FedAvg work with subspace parameterizations of the client models f. For the results in Section 5.6.2 we set the subspace dimension to $k = 10^4$. This represents an approximately $15\times$ reduction in the number of client model parameters for the CNN and $1100\times$ for ResNet18. We examine the impact the choice of k has in 5.6.3. For FedTTA the global model architecture is the same as that used for FedAvg and the adaptation model is the same 3-layer network as used in [YCW+24].

5.6.2 Experimental results

Our main results are shown in Tables 5.1 and 5.2. Table 5.1 shows class-partitioned CIFAR10 as we vary the fraction of labeled clients present in the training set (p). The left part of the table shows the results for when the model architecture is a CNN and the right part for ResNet18. The results show that FLowDUP performs best across the board. As expected all methods improve as the fraction of labeled clients increases and when we switch from using a CNN to a ResNet18. Notably, even when using the CNN architectures, FLowDUP is able to outperform the non-personalized baselines using a ResNet18. We also observe that LD-FedAvg has significantly lower performance than FedAvg. This shows that in fact for a single global predictive model, lower subspace dimension reduces the power of the model.

Table 5.1: Class-partitioned CIFAR10 for varying fractions, p, of the training clients having labeled data. Accuracy on unlabeled test clients.

Model	**CNN**			**ResNet18**		
p	**0.1**	**0.2**	**1.0**	**0.1**	**0.2**	**1.0**
FedAvg	47.9 ± 0.1	53.5 ± 0.4	63.6 ± 0.4	63.6 ± 0.9	68.9 ± 0.7	75.9 ± 0.3
FedProx	47.6 ± 0.3	53.3 ± 0.3	63.7 ± 0.6	63.5 ± 0.8	68.8 ± 0.5	75.9 ± 0.3
LD-FedAvg	41.6 ± 0.6	47.8 ± 1.1	56.2 ± 0.7	54.6 ± 0.8	60.0 ± 0.5	65.4 ± 0.1
FedTTA	57.2 ± 1.0	60.2 ± 1.1	69.7 ± 3.8	69.3 ± 2.3	77.2 ± 1.5	81.9 ± 2.1
FLowDUP	66.1 ± 1.5	75.3 ± 1.8	86.6 ± 0.1	72.7 ± 1.3	82.9 ± 1.8	92.3 ± 0.4

Table 5.2: Rotated Fashion-MNIST (left) and FEMNIST (right) for varying fractions, p, of the training clients having labeled data. Accuracy on unlabeled test clients.

Dataset	**Rotated Fashion-MNIST**			**FEMNIST**		
p	**0.1**	**0.2**	**1.0**	**0.1**	**0.2**	**1.0**
FedAvg	77.7 ± 0.1	79.2 ± 0.2	81.4 ± 0.6	76.5 ± 0.3	78.0 ± 0.3	84.3 ± 0.1
FedProx	77.3 ± 0.7	78.7 ± 0.4	80.0 ± 0.6	72.3 ± 0.4	74.8 ± 0.4	83.3 ± 0.0
LD-FedAvg	76.1 ± 1.1	78.8 ± 0.3	81.3 ± 0.1	61.4 ± 1.4	74.4 ± 0.3	80.6 ± 0.4
FedTTA	78.3 ± 0.5	80.2 ± 0.1	81.2 ± 0.2	71.1 ± 2.4	72.6 ± 3.5	74.3 ± 3.1
FLowDUP	81.5 ± 0.3	83.3 ± 0.5	87.3 ± 0.2	84.5 ± 0.4	86.4 ± 0.3	89.3 ± 0.1

However, FLowDUP's strong performance shows that, given a well learned representation of client heterogeneity in h, a low dimensional subspace model can obtain good personalized performance.

Table 5.2 shows the results for Rotated Fashion-MNIST (left) and FEMNIST (right). The results for Rotated MNIST can be found in the Appendix in Table C.1. For these datasets all methods use the CNN architecture. When comparing to CIFAR10, for both these datasets FedTTA performs quite poorly. This is to be expected given that FedTTA adapts the global model based on the output logits of the unlabeled data. When statistical heterogeneity is present only in the labels (as is the case for class-partitioned CIFAR10), the logits are a good summary of the client personalization requirements. However, in the presence of feature heterogeneity the logits alone provide a poor adaptation signal. FLowDUP performs better as it personalizes based on all the clients feature data.

5.6.3 Additional Studies

Here we provide results from additional experiments investigating the components of FLowDUP. In addition to these we include in the appendix a study of the effect that training with unlabeled clients has (Tables C.2 and C.3) as well as the benefits of the learnable regularizer ψ_r (Table C.4).

Differing client model architectures We train h to output personalized model parameters in a subspace of dimension k. However, for any given k we are still free to choose the architecture of the client model f. Moreover, the choice of architecture of f does not affect the communication cost of the training procedure (which is critical and typically the primary

Table 5.3: Accuracy of FLowDUP for architecture combinations on class-partitioned CIFAR10.

Model Architecture		**Fraction Labeled (p)**			
h_1	f	**0.1**	**0.2**	**0.5**	**1.0**
CNN	CNN	66.1 ± 1.5	75.3 ± 1.8	83.3 ± 1.5	86.6 ± 0.1
CNN	ResNet18	71.4 ± 0.5	80.0 ± 3.3	88.0 ± 0.9	90.7 ± 0.2
ResNet18	CNN	67.0 ± 2.4	78.0 ± 3.1	87.3 ± 1.3	88.5 ± 1.1
ResNet18	ResNet18	72.7 ± 1.3	82.9 ± 1.8	91.0 ± 0.4	92.3 ± 0.4

Table 5.4: Accuracy of FLowDUP with different subspace dimensions, k, on class-partitioned CIFAR10.

Fraction Labeled (p)	**0.1**	**0.2**	**0.5**	**1.0**
FLowDUP ($k = 200$)	56.6 ± 1.9	65.7 ± 2.9	70.0 ± 1.8	71.2 ± 0.7
FLowDUP ($k = 500$)	60.0 ± 1.9	69.8 ± 2.8	75.9 ± 1.2	76.9 ± 0.8
FLowDUP ($k = 2000$)	64.5 ± 0.3	72.5 ± 1.7	80.3 ± 1.5	83.5 ± 0.9
FLowDUP ($k = 10000$)	66.1 ± 1.5	75.3 ± 1.8	83.3 ± 1.5	86.6 ± 0.1

bottleneck when running federated training in practice). It does, however, affect the compute cost on the client side. On the flip side, the choice of hypernetwork architecture does affect the communication cost as h is transmitted from server to client. This gives FLowDUP some flexibility when trading off communication vs computational cost. In Table 5.3 we show results for CIFAR10 for different combinations of hypernetwork and client model architectures. The results show that for a fixed subspace dimension k, changing the architecture of h_1 or f can have a substantial impact on performance. Most notably, when comparing row 1 with 2, and row 3 with 4, we see that changing f from a CNN to a ResNet18 gives a performance boost of 4-5% without incurring any additional communication cost.

Varying the subspace dimension While the choice of client model architecture affects only the computational cost of FLowDUP, the subspace dimension k affects both communication and computational cost. This is because the final layer of h has dimension k and the matrices in 5.2 scale with k. Table 5.4 shows results for FLowDUP on CIFAR10 as we vary k. In these experiments we fix the architectures of h_1 and f to be CNN. We observe that performance increases monotonically with k. Most notably we observe that, except for $p = 0.1$, FLowDUP outperforms the baselines, even for very low values of k which in turn incur low compute costs on the client.

Understanding dataset embeddings Recall the hypernetwork architecture from (5.3). We can interpret the input to h_2 as an embedding representation of the client dataset X, which we call

$$r(X) \coloneqq \frac{1}{|X|} \sum_{x \in X} h_1(x). \tag{5.9}$$

To better understand the inner workings of this mechanism we visualize the space of dataset embeddings. Concretely, for each validation client in the Rotated Fashion-MNIST dataset, we compute $r(X_i)$ for the h that was trained on the training clients. For our experiments these

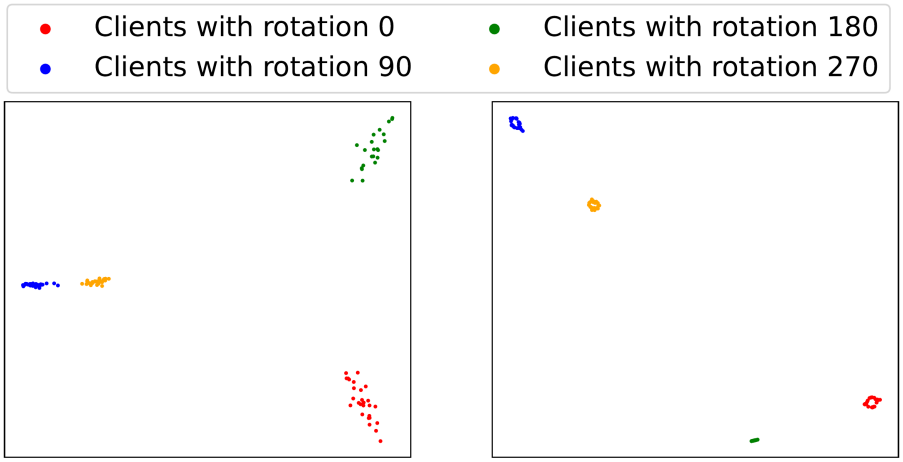

Figure 5.1: Visualization of the client embeddings on Rotated Fashion-MNIST. Projection to two dimensions using PCA (left) and t-SNE (right).

representations are 256-dimensional, therefore to create a visualization we apply dimensionality reduction. We separately apply both PCA and t-SNE [vdMH08] to project the representations into 2 dimensions. The scatter plots (Figure 5.1) show that FLowDUP learns to organize the space of client dataset representations according to the present statistical heterogeneity, namely that the representations are clustered according to which rotation the data has. Note that nowhere in the learning objective is this hard coded. Rather FLowDUP has learned the type of heterogeneity present in the client datasets, and organized the representation space so that in this case, roughly speaking, clients with the same rotation receive the same model.

5.7 Conclusion

In this chapter we presented FLowDUP, a method that can generate personalized models for clients using only a single forward pass with unlabeled data. FLowDUP's training objective is theoretically motivated by our new generalization bound in a transductive multitask framework and is able to make use of both labeled and unlabeled clients during training for improved performance. Our empirical evaluation demonstrated that FLowDUP can achieve strong performance in the presence of both feature and label heterogeneity. In additional studies, we investigated the effect of model architecture choices, subspace dimension, the benefit of using unlabeled clients and how FLowDUP is able to capture dataset heterogeneity.

FLowDUP's main strength is also its main limitation: because it uses only the unlabeled data to produce a personalized model, it will struggle in a setting where the conditional distributions, $(D_i)_{Y|X}$, differ across clients in a way such that knowledge of the marginals, $(D_i)_X$, are insufficient to infer good predictive models. Such situations could be possible, for example, in subjective prediction tasks, such as product recommendations or sentiment analysis. This limitation is not specific to FLowDUP though: all methods that rely only on unlabeled client data would fail in this setting, so additional information sources would be required, such as meta-data or user participation.

CHAPTER 6

Differentially Private Federated k-Means Clustering with Server-Side Data

6.1 Motivation and Outline

In previous chapters we have focused primarily on modeling data heterogeneity and directly adapting to it through training free personalization in a supervised setting. In this chapter we change the nature of our approach to data heterogeneity and instead focus on being unaffected by it. We do this in the context of unsupervised learning, more specifically clustering. Clustering has long been the technique of choice for understanding and identifying groups and structures in unlabeled data and effective algorithms to cluster non-private centralized data have been around for decades [Llo82, SM00, NJW01]. However, the major paradigm shift in how data are generated nowadays presents new challenges that often prevent the use of traditional methods. In this chapter we close this gap by introducing FedDP-KMeans, a fully federated and differentially private k-means clustering algorithm. Our main innovation is a new initialization method, FedDP-Init, that leverages server-side data to find good initial centers. Crucially, we do not require the server data to follow the same distribution as the client data, making FedDP-Init applicable to a wide range of practical FL scenarios. The initial centers serve as input to FedDP-Lloyds, a simple federated and differentially private variant of Lloyds algorithm [Llo82]. As we discussed already in 2.1.2 a good initialization is critical to obtaining a good final clustering. While this is already true for non-private, centralized clustering, it is especially the case in the differentially private, federated setting, where we are further limited by privacy and communication constraints in the number of times we can access client data and thereby refine our initialization. We report on experiments for synthetic as well as real datasets in two settings: when we wish to preserve individual *data point privacy*, as is common for cross-silo federated learning settings [LFTL20], and *client-level privacy*, as is typically used in cross-device learning settings [MMR$^+$17]. In both cases, FedDP-KMeans achieves clearly better results than all baseline techniques. We also provide a theoretical analysis, proving that under standard assumptions for the analysis of clustering algorithms (Gaussian mixture data with well-separated components), the cluster centers found by FedDP-KMeans converge exponentially fast to the true component means and the ground truth clusters are identified after only logarithmically many steps.

6.2 Related Work

For prior works relating to the theory of clustering, see Section D.1. Within FL, clustering appears primarily for the purpose of grouping clients together. Such *clustered FL* techniques find a clustering of the clients while training a separate ML model on each cluster [SMS20, GCYR20b, XHTZ20]. In contrast, in this work we are interested in the task of clustering the clients' data points, rather than the clients. In [DLS21], the one-shot scheme k-Fed is proposed for this task: first all clients cluster their data locally. Then, they share their cluster centers with the server, which clusters the set of client centers to obtain a global clustering of the data. However, due to the absence of aggregation of the quantities that clients share with the server, the method has no privacy guarantees. [LMY+20] propose using *federated averaging* [MMR+17] to minimize the k-means objective in combination with multi-party computation. Similarly, [MRT20] describe an efficient multi-party computation technique for distance computations. This will avoid the server seeing individual client contributions before aggregation, but the resulting clustering still exposes private information. For private clustering, many methods have been proposed based on DPLloyd's [BDMN05], i.e. Lloyd's algorithm with noise added to intermediate steps. The methods differ typically in the data representation and initialization. For example, [SCL+16] creates and clusters a proxy dataset by binning the data space. This is, however, tractable only in very low dimensions. [RXY+17] chooses initial centers by forming subsets of the original data and clustering those. [ZLH+22] initializes with randomly selected data points and uses multi-party computation to securely aggregate client contributions. None of the methods are compatible with the FL setting. On the other hand, algorithms designed for Local Differential Privacy [CGKM21, DITHS24] could directly work in FL; albeit with an additive error so large that it makes any implementation impractical [CJN22]. DP algorithms that work in parallel environment, such as in [CEM+22] could potentially be adapted to FL, however requiring a large number of communication rounds.

6.3 Method

Again we denote by n the number of clients. Each client, i, possesses a dataset $S_i = x_i^1, \ldots, x + i^{m_i} \sim D_i$. We denote by $P^i \in \mathbb{R}^{d \times m_i}$ the matrix obtained by stacking S_i so that each column is a data point. The server also has some data, Q, which can be freely shared with the clients, but that is potentially small and *out-of-distribution* (i.e. not following the client data distribution). The goal is to determine a k-means clustering of the joint clients' dataset $S := \bigcup_{i=1}^n S_i$, or equivalently of the stacked matrix $P \in \mathbb{R}^{d \times m}$, in a *federated* and *differentially private* way. We propose FedDP-KMeans, which solves this task in two stages. The first, FedDP-Init (Algorithm 11), is our main contribution: it constructs an initialization to k-means by exploiting server-side data. The second, FedDP-Lloyds (Algorithm 12), is a simple federated DP-Lloyds algorithm, which refines the initialization.

6.3.1 FedDP-Init

Sketch: FedDP-Init has three steps: **Step 1** computes a projection matrix onto the space spanned by the top k singular vectors of the client data matrix P. **Step 2** projects the server data onto that subspace, and computes a weight for each server point q that reflects how many client points have q as their nearest neighbor. **Step 3** computes initial cluster centers in the original data space by first clustering the weighted server data in the projected space and

then refining these centers by a step resembling one step of Lloyd's algorithm on the clients, but with the similarity computed in the projected space. To ensure the privacy of the client data all above computations are performed with sufficient amounts of additive noise, and the server only ever receives noised aggregates of the computed quantities across all clients. Consequently, FedDP-Init is differentially private and fully compatible with standard FL and secure aggregation setups, as described in Section 2.4.2. Intuitively, the goal of Step 1 is to project the data onto a lower-dimensional subspace that preserves the important variance (i.e. distance between the means) but reduces the variance in nuisance direction (in particular the intra-cluster variance). This construction is common for clustering algorithm that strive for theoretical guarantees, and was popularized by [KK10]. Our key novelty lies in Step 2 and 3: here, we exploit the server data, essentially turning it into a proxy dataset on which the server can operate without any privacy cost. After one more interaction with the clients, the resulting cluster centers are typically so close to the optimal ones, that only very few (sometimes none at all) steps of Lloyd's algorithm are required to refine them. Our theoretical analysis (Section 6.4) quantifies this effect: for suitably separated Gaussian Mixture data, the necessary number of steps to find the ground truth clusters is at most logarithmic in the total number of data points.

Algorithm 11 FedDP-Init

Input: Client data sets $P^1, \dots, P^n$, # of clusters k, privacy parameters $\varepsilon_1, \varepsilon_2, \varepsilon_{3G}, \varepsilon_{3L}, \delta$
Step 1: *// Projection onto top k singular vectors*
for client $i = 1, \dots, n$ **do**
  Client i computes outer product $P^i(P^i)^T$
end for
Server receives noisy aggregate $\widehat{PP^T} = \sum_{i=1}^n P^i(P^i)^T + \mathcal{N}_{d\times d}(0, \sigma^2(\varepsilon_1, \delta; \Delta^2))$
Server forms a projection matrix Π from top k eigenvectors of $\widehat{PP^T}$
Step 2: *// Determine importance weights*
for client $i = 1, \dots, n$ **do**
  Client i receives Π and ΠQ from server
  for every point $q \in \Pi Q$ **do**
    Client i computes weight $w_q(\Pi P^i) \coloneqq \left|\{p \in \Pi P^i \mid \forall q' \in \Pi Q,\ \|p - q\| \le \|p - q'\|\}\right|$
  end for
end for
Server receives noisy aggregate $\widehat{w_q(\Pi P)} = \sum_{i=1}^n w_q(\Pi P^i) + \mathrm{Lap}(0, \frac{1}{\varepsilon_2})$ for each $q \in \Pi Q$
Step 3: *// Cluster projected server points and initialize*
Server computes cluster centers $\xi_1, .., \xi_k$ by running k-means clustering of ΠQ with per-sample weights $\widehat{w_q(\Pi P)}$
for client $i = 1, \dots, n$ **do**
  Client i receives $\xi_1, .., \xi_k$ from server
  Client i computes $S_j^i = \{p \in P^i : \forall j', \|\Pi p - \xi_j\| \le \|\Pi p - \xi_{j'}\|\}$, for $j = 1, \dots, k$
  Client i computes $m_j^i = \sum_{p \in S_j^i} p$ and $n_i^j = |S_j^i|$
end for
Server receives noisy aggregates $\widehat{m_j} = \sum_{i=1}^n m_j^i + \mathcal{N}_d(0, \sigma^2(\varepsilon_{3G}, \delta; \Delta))$ and $\widehat{n_j} = \sum_{i=1}^n n_i^j + \mathrm{Lap}(0, \frac{1}{\varepsilon_{3L}})$
Server computes initial centers $\nu_j = \widehat{m_j}/\widehat{n_j}$ for $j = 1, \dots, k$
Output: Initial cluster centers $\nu_1, .., \nu_k$

In the rest of this section, we describe the individual steps in more detail. For the sake of simpler exposition, we describe only the setting of data-point-level DP. However, only minor changes are needed for client-level DP, see Section 6.5. As private budget, we treat δ as fixed for all steps, and denote the individual budgets of the three steps as ε_1, ε_2 and ε_3. We provide recommendations how to set these values given an overall privacy budget in Appendix D.8.1.

Algorithm details – Step 1: The server aims to compute the top k eigenvectors of the clients' data outer product matrix PP^T. However, in the federated setup, it cannot do so directly because it does not have access to the matrix P. Instead, the algorithm exploits that the overall outer product matrix can be decomposed as the sum of the outer products of each client data matrix, i.e. $PP^T = \sum_{i=1}^{n} P^i(P^i)^T$. Therefore, each client can locally compute their outer product matrix and the server only receives their noisy across-client aggregate, $\widehat{PP^T}$. We ensure the privacy of this computation by the Gaussian mechanism. The associated sensitivity is the maximum squared norm of any single data point, which is upper bounded by the square of the dataset radius, Δ. Consequently, a noise variance of $\sigma_G^2(\varepsilon_1, \delta; \Delta^2)$ ensures (ε_1, δ)-privacy, as shown by [DTTZ14].

The remaining operations the server can perform noise-free: it computes the top k eigenvectors of $\widehat{PP^T}$ and forms the matrix $\Pi \in \mathbb{R}^{d\times d}$ from them, which allows projecting to the k-dimensional subspace spanned by these vectors(which we call *data subspace*). The projection provides a data-adjusted way of reducing the dimension of data vectors from potentially large d to the much smaller k. This is an important ingredient to our algorithm, because in low dimension typically less noise is required to ensure privacy. The lower dimension also helps keep the communication between server and client small. The dimension k is chosen, because for sufficiently separated clusters, one can expect the subspace to align well with the subspace spanned by the cluster centers. In that case, the projection will preserve inter-cluster variance but reduce intra-cluster variance, which improves the signal-to-noise ratio of the data.

Step 2: Next, the server computes per-point weights for its own data such that it can serve as a proxy for the data of the clients. The server shares with the clients the computed projection matrix Π, and its own projected dataset ΠQ. Each client uses Π to project its own data to the data subspace. Then, it computes a weight for each server point $q \in \Pi Q$ as, $w_q(\Pi P^i) = \left|\{p \in \Pi P^i \mid \forall q' \in \Pi Q, \|p - q\| \leq \|p - q'\|\}\right|$, that is, the count of how many of the client's projected points are closer to q than to any other $q' \in \Pi Q$, breaking ties arbitrarily. The weights are sent to the server in aggregated and noised form. As an unnormalized histogram over the client data, the point weight has L^1-sensitivity 1. Therefore, the Laplace mechanism with noise scale $1/\varepsilon_2$ makes this step $(\varepsilon_2, 0)$-DP. The noisy total weights, $\widehat{w_q(\Pi P)}$ for $q \in \Pi Q$, provide the server with a (noisy) estimate of how many client data points each of its data points represents. It then runs k-means clustering on its projected data ΠQ, where each point q receives weight $\widehat{w_q(\Pi P)}$ in the k-means cost function, to obtain centers $\xi_1, \ldots, \xi_k$ in the data subspace.

Step 3: In the final step the server constructs centers in the original space. For this, it sends the projected centers $\xi_1, \ldots, \xi_k$ to the clients. For each projected cluster center ξ_j, each client i computes the set of all points $p \in P^i$ whose closest center in the projected space is ξ_j, i.e. $S_j^i := \{p \in P^i : \ \forall j', \|\Pi p - \xi_j\| \leq \|\Pi p - \xi_{j'}\|\}$. For any j, the union of these sets across all clients would form a cluster in the client data. We want the mean vector of this to constitute the j-th initialization center. For this, each client i computes the sum, and

the number, of their points in each cluster, $m_j^i = \sum_{p \in S_j^i} p$, $n_i^j = |S_j^i|$. Aggregated across all clients one obtains the global sum and count of the points in each cluster: $m_j = \sum_{i=1}^n m_j^i$ and $n_j = \sum_{i=1}^n n_i^j$. To make this step private, we first split $\varepsilon_3 = \varepsilon_{3G} + \varepsilon_{3L}$. For m_j^i, which has L^2-sensitivity Δ, we apply the Gaussian mechanism with variance $\sigma^2(\varepsilon_{3G}, \delta; \Delta)$. For n_j, which has the L^1-sensitivity is 1, we use the Laplace mechanisms with scale $1/\varepsilon_{3L}$. This ensures $(\varepsilon_{3G}, \delta)$ and $(\varepsilon_{3L}, 0)$ privacy, respectively, and therefore (at least) (ε, δ) privacy overall for this step. Finally, the server uses the noisy estimates of the total sums and counts, $\widehat{m_j}$ and $\widehat{n_j}$, to compute approximate means $\nu_j = \widehat{m_j}/\widehat{n_j}$, and outputs these as initial centers.

6.3.2 FedDP-Lloyds

The second step of FedDP-KMeans is a variant of Lloyd's algorithm that we adapt to a private FL setting. The basic observation here is that a step of Lloyd's algorithm can be expressed only as summations and counts of data points. Consequently, all quantities that the server requires can be expressed as aggregates over client statistics which allows us to preserve user privacy with secure aggregation and DP. Specifically, assume that we are given initial centers $\nu_1^0, \ldots, \nu_k^0$, and a privacy budget $(\varepsilon_4, \delta_4)$, which we split as $\varepsilon_4 = \varepsilon_{4G} + \varepsilon_{4L}$. For rounds $t = 1, \ldots, T$, we repeat the following steps. The server sends the latest estimate of the centers to the clients. Each client i computes, for $j = 1, \ldots, k$, $S_j^i := \{p \in P^i : \forall j', \|p - \nu_j^{t-1}\| \leq \|p - \nu_{j'}^{t-1}\|\}$, the set of points whose closest center is ν_j^{t-1}. Note that in contrast to the initialization, the distance is measured in the full data space here, not the data subspace. The remaining steps coincide with the end of Step 3 above. Each client i computes the summations and counts of their points in each cluster: $m_j^i = \sum_{p \in S_j^i} p$ and $n_i^j = |S_j^i|$. These are aggregated to $m_j = \sum_{i=1}^n m_j^i$ and $n_j = \sum_{i=1}^n n_i^j$, and made private by the Gaussian mechanisms with variance $\sigma^2(\varepsilon_{4G}/T, \delta/T, \Delta)$ and the Laplacian mechanism with scale T/ε_{4L}, respectively. The server receives the noisy total sums and counts $\widehat{m_j}$ and $\widehat{n_j}$, and it updates its estimate of the centers as $\nu_j^t = \widehat{m_j}/\widehat{n_j}$. Overall, the composition property of DP ensures that FedDP-Lloyds is at least (ε_4, δ)-private.

Algorithm 12 FedDP-Lloyds

Input: Initial centers $\nu_1^0, \ldots, \nu_k^0$, P, steps T, privacy parameters $\varepsilon_G, \varepsilon_L, \delta$
for $t = 1, \ldots, T$ **do**
  for client $i = 1, \ldots, n$ **do**
    Client i receives $\nu_1^{t-1}, \ldots, \nu_k^{t-1}$ from Server
    for $j = 1, \ldots, k$ **do**
      Client i computes $S_j^i := \{p \in P^i : \forall j', \|p - \nu_j^{t-1}\| \leq \|p - \nu_{j'}^{t-1}\|\}$
      Client i computes $m_j^i = \sum_{p \in S_j^i} p$, $n_i^j = |S_j^i|$
    end for
  end for
  Server gets $\widehat{m_j} = \sum_{i=1}^n m_j^i + \mathcal{N}_d(0, T\Delta^2\sigma^2(\varepsilon_G/T, \delta))$ and $\widehat{n_j} = \sum_{i=1}^n n_i^j + \mathrm{Lap}(0, \frac{T}{\varepsilon_L})$
  Server computes centers $\nu_j^t = \widehat{m_j}/\widehat{n_j}$, $j = 1, \ldots, k$
end for
Output: Final cluster centers $\nu_1^T, .., \nu_k^T$

The choice of noise parameters ensure that the combination of our two algorithms is differentially private:

Theorem 5. *FedDP-KMeans followed with FedDP-Lloyds is* (ε, δ)*-DP for* $\varepsilon = \varepsilon_1 + \varepsilon_2 + \varepsilon_{3G} + \varepsilon_{3L} + \varepsilon_{4G} + \varepsilon_{4L}$

6.4 Theoretical analysis

We analyze the theoretical properties of FedDP-KMeans in the standard setting of data from a k-component Gaussian mixture, i.e. the data P is sampled from a distribution $\mathcal{D}(x) = \sum_{j=1}^{k} w_j \mathcal{G}_j(x)$ with means μ_j, covariance matrix Σ_j and cluster weight w_j. The data is partitioned arbitrarily among the clients, i.e. each clients data is not necessarily distributed according to $\mathcal{D}$ itself. We denote by G_j the set of samples from the j-th component $\mathcal{G}_j$: the goal is to recover the clustering $G_1, ..., G_k$. The server data, $Q \subset \mathbb{R}^d$, can be small and not of the same distribution as P.

Our main result is Theorem 7, which states that FedDP-KMeans successfully clusters such data, in the sense that the cluster centers it computes converge to the ground truth centers, i.e. the means of the Gaussian parameters, and the induced clustering becomes the ground truth one. In doing so, the algorithm respects data-point differential privacy. For this result to hold, a *separation condition* is required (Definition 6). It ensures that the ground truth cluster centers are separated far enough from each other to be identifiable. We first introduce and discuss the separation condition and then state the theorem. The proof is in Appendix D.5 and D.6.

Definition 6 (Separation Condition)**.** *For constant* c*, a Gaussian mixture* $\big((\mu_j, \Sigma_j, w_j)\big)_{j=1,...,k}$ *with* m *samples is called* c-separated *if*

$$\forall j \neq j', \|\mu_j - \mu_{j'}\| \geq c\sqrt{\frac{k}{w_j}}\sigma_{max}\log(m),$$

where σ_{max} *is the maximum variance of any Gaussian along any direction. For some large enough constant* c *fixed independently of the input, we say that the mixture is separated*[1]

Note that the dependency in $\log(m)$ is unavoidable, because with growing m also the chance grows that outliers occur from the Gaussian distributions: assigning each data point to its nearest mean would not be identical to the ground truth clustering anymore.

To prove the main theorem, two additional assumptions on P are required: (1) the diameter of the dataset is bounded by $\Delta := O\Big(\frac{k\log^2(m)\sqrt{d}\sigma_{\max}}{\varepsilon w_{\min}}\Big)$ – so that the noise added to compute a private SVD preserves enough signal.[2] (2): there is not too many server data, namely $|Q| \leq \frac{\varepsilon m k \sigma_{\max}^2}{\Delta^2}$. This ensures the noise added Step 2 is not overwhelming compared to the signal. Note that conditions (1) and (2) can always be enforced by two preprocessing steps, which we present as part of the proof in Appendix D.5. In practice, however, they are typically satisfied automatically, see Appendix D.7.2. This allows use of the algorithm directly as stated.

[1] The constant c is determined by prior work: see [AS12]

[2] It may be surprising to see that the diameter is allowed to increase when the privacy budget ε gets smaller. However, Theorem 7 also requires the sample size to increase with $1/\varepsilon$, which counterbalances the growth of Δ.

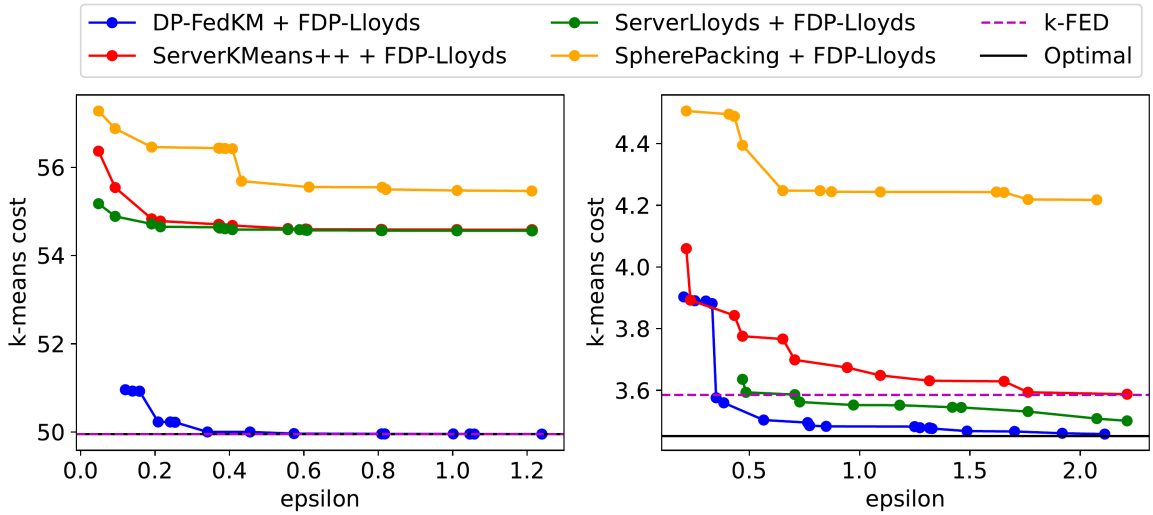

Figure 6.1: Results with data-point-level privacy ($k = 10$). Left: synthetic mixture of Gaussians data with 100 clients. Right: US census dataset. The 51 clients are US states, each client has the data of individuals with employment type "Federal government employee".

Theorem 7. *Suppose that the client dataset P is generated from a separated Gaussian mixtures with $m \geq \zeta_1 \frac{k \log^3 m \sqrt{d} \sigma_{max}}{\varepsilon^2 w_{min}^2}$ samples, where ζ_1 is a universal constant, and that Q contains a least one sample from each component of the mixture. Then, there is a constant ζ_2 such that, under assumptions (1) and (2), the centers $\nu_1, ..., \nu_k$ that are computed after T steps of FedDP-Lloyds satisfy with high probability*

$$\|\mu_j - \nu_j\| \leq \zeta_2 \cdot \left(2^{-T} \cdot \sqrt{\frac{m\sigma_{max}^2}{|G_j|}} + \frac{T\Delta \log(m)}{\varepsilon m w_{min}} \right). \tag{6.1}$$

Furthermore, there is a constant ζ_3 such that, after $\zeta_3 \log(m)$ rounds of communication, the clustering induced by $\nu_1, ..., \nu_k$ is the ground-truth clustering $G_1, ..., G_k$.

Note that assumption (1) implies that $\frac{\Delta \log(m)}{\varepsilon w_{\min}}$ is negligible compared to m. That means, the estimated centers converge exponentially fast towards the ground truth.

6.5 Experiments

We now present our empirical evaluation of FedDP-KMeans, which we implemented using the `pfl-research` framework [GSC+24].

To verify the broad applicability of our method we run experiments in both the setting of data-point-level privacy, see Section 6.5.1, and client-level privacy, see Section 6.5.2. The appropriate level of privacy in FL is typically determined by which data unit corresponds to a human. In *cross-silo* FL we typically have a smaller number of large clients, e.g. hospitals, with each data point corresponding to some individual, so data point-level privacy is appropriate. In *cross-device* FL, we typically have a large number of clients, where each client is a user device such as a smartphone, so client-level privacy is preferable. Our chosen evaluation datasets reflect these dynamics.

Appendix A.3 contains additional details regarding the experimental evaluation and Appendix D.8 contains additional experiments and ablation studies. In particular, in D.8.1, we investigate

how to set important hyperparameters of FedDP-KMeans, such as the privacy budgets of the individual steps. In D.8.2, we examine what happens when not all target clusters are present in the server data. Finally, in D.8.3, we discuss how to use existing methods for choosing k, based on the data, in the context of FedDP-KMeans.

Baselines As natural alternatives to FedDP-KMeans we consider different initializations of k-means combined with FedDP-Lloyds. Two baselines methods use the server data to produce initialization: *ServerKMeans++* runs k-means++ on the server data, while *ServerLloyds* runs a full k-means clustering of the server data. The baselines can be expected to work well when the server data is large and of the same distribution as the client data. This, however, is exactly the situation where the server data would suffice anyway, so any following FL would be wasteful. In the more realistic setting where the server data is small and/or out-of-distribution, the baselines might produce biased and therefore suboptimal results. As a third baseline, we include the *SpherePacking* initialization of [SCL+17]. This data-independent technique constructs initial centroids that are suitably spaced out and cover the data space, see Appendix D.7.3 for details. None of the above baselines use client data for initialization. Therefore, they consume none of their privacy budget for this step, leaving all of it for the subsequent FedDP-Lloyds. We also report results for two methods that do not actually adhere to the differentially-private federated paradigm. *k-FED* [DLS21] is the most popular federated k-means algorithm. As we will discuss in Section 6.2 it does not exploit server data and does not offer privacy guarantees. *Optimal* we call the method of transferring all client data to a central location and running non-private k-means clustering. This provides neither the guarantees of FL nor of DP, but is a lower bound on the k-means cost for all other methods.

Evaluation Procedure We compare FedDP-KMeans with the baselines over a range of privacy budgets. Specifically, if a method has s steps that are $(\varepsilon_1, \delta), \ldots, (\varepsilon_s, \delta)$ DP then the total privacy cost of the method is computed as $(\varepsilon_{\text{total}}, \delta)$ by strong composition using Google's `dp_accounting` library [3]. We fix $\delta = 10^{-6}$ for all privacy costs. We vary the ε_i of individual steps as well as other hyperparameters, e.g. the number of steps of FedDP-Lloyds, and measure the k-means cost of the final clustering. For each method we plot the Pareto front of the results in (k-means cost, $\varepsilon_{\text{total}}$) space. When plotting we scale the k-means cost by the dataset size, i.e. by $1/m$. This evaluation procedure gives us a good overview of the performance of each method at a range of privacy budgets. However, on its own it does not tell us how to set hyperparameters for FedDP-KMeans, such how much privacy budget to assign to each step. Knowing how to do this is important for using FedDP-KMeans in practice and we address this in Appendix D.8.1.

6.5.1 Data-point-level Privacy Experiments

Privacy Implementation details In our theoretical discussion we assumed that no individual data point has norm larger than Δ in order to compute the sensitivity of certain steps. As Δ is typically not known in practice, in our experiments we ensure the desired sensitivity by clipping the norm of each data point to be at most Δ, before using it in any computation. Δ is now a hyperparameter of the algorithm, which we set to be the radius of the server dataset.

Datasets We evaluate on synthetic and real federated datasets that resemble a cross-silo FL setting. Our synthetic data comes from a mixture of Gaussians, as assumed for our theoretical

[3] `https://github.com/google/differential-privacy/tree/main/python/dp_accounting`

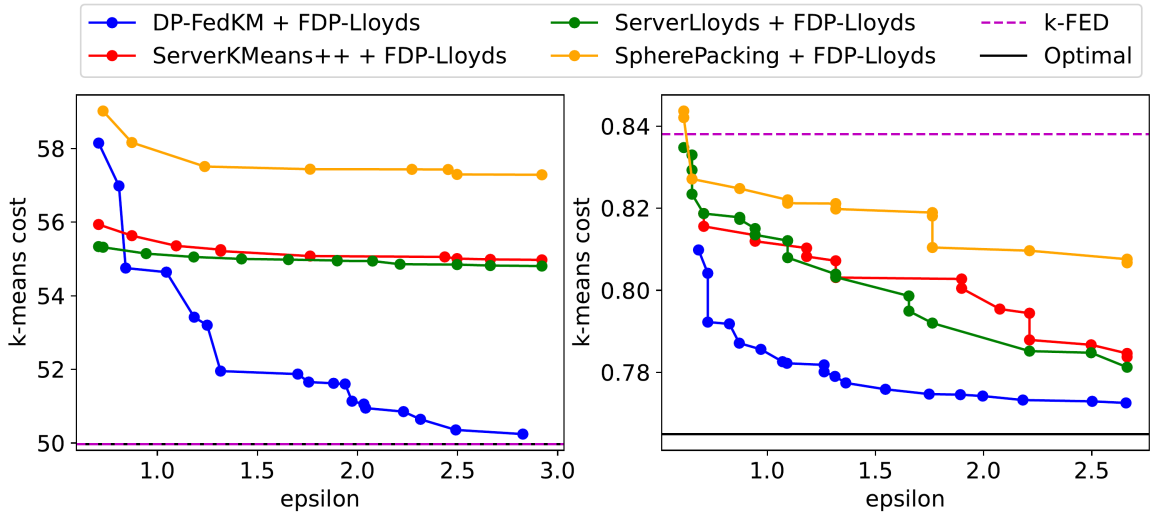

Figure 6.2: Results with client-level privacy $(k = 10)$. Left: synthetic mixture of Gaussians data with 2000 clients. Right: stackoverflow dataset with 9237 clients, topic tags github and pdf.

results in Section 6.4. The client data is of this mixture distribution. To simulate related, but OOD data, the server data consists of two thirds data from the true mixture and one third data that is uniformly distributed. We additionally evaluate on US census data using the folktables [DHMS21] package. The dataset has 51 clients, each corresponding to a US state. Each data point contains the information about a person in the census. We create a number of clustering tasks by filtering the client data to contain only those individuals with some chosen employment type. The server then recieves a small amount of data of individuals with a different employment type, to simulate related but OOD data. Full details on the datasets are in Appendix D.7.1.

Results See Figure 6.1. The left panel shows results for the Gaussian mixture and the right panel for the US census dataset when the clients hold the data of federal employees. The other two categories are shown in Figures D.6 and D.7 of Appendix D.9. On synthetic data, FedDP-KMeans outperforms all DP baselines by a wide margin. These baselines cannot overcome their poor initializations, with performance plateauing even as the privacy budget grows. In contrast FedDP-KMeans obtains optimal (non-private) performance at a low privacy budget of around $\varepsilon_{\text{total}} = 0.4$. The non-private k-FED also performs optimally in this setting as is to be expected given that the synthetic data fulfills the assumptions of [DLS21]. On the US census datasets we observe a more interesting picture. Over all three settings FedDP-KMeans outperforms the baselines, except in the very low privacy budget regime. The latter is to be expected since for sufficiently low privacy budgets a client-based initialization will become very noisy, whereas the initialization with only server data (which requires no privacy budget) stays reasonable. With a high enough privacy budget FedDP-KMeans recovers the optimal non-private clustering. Among the baselines we observe similar performance between the two methods that initialize using server data, with ServerLloyds performing slightly better overall. The data independent SpherePacking performs poorly, emphasizing the importance of leveraging related server data to initialize. We attribute FedDP-KMeans's good performance predominantly to the excellent quality of its initialization. As evidence, Table D.4 in Appendix A.3 shows how many steps of Lloyd's had to be performed for Pareto-optimal behavior: this is never more than 2, and often none at all.

6.5.2 Client-level Privacy Experiments

Privacy Implementation details Moving to client-level differential privacy changes the sensitivities of the steps of our algorithms, which now depend not only on the maximum norm of a client data point norm, but also on the maximum number of data points a client has. Rather than placing assumptions on this, and deriving corresponding bounds on the sensitivity of each step, we instead simply enforce sensitivity by clipping client statistics prior to aggregation. This is a standard technique to enforce a given sensitivity in private FL, where it is typically applied to model/gradient updates. For full details on our implementation in the client-level privacy setting see Appendix D.7.4

Datasets We evaluate on both synthetic and real federated datasets in a cross-device FL setting. For synthetic data we again use a mixture of Gaussians, but with more clients than in Section 6.5.1. We also use the Stack Overflow dataset from Tensorflow Federated. This is a large scale text dataset of questions posted by users on stackoverflow.com. We preprocess this dataset by embedding it with a pre-trainined sentence embedding model. Thus each client dataset consists of small number of text embedding vectors. The server data consists of embedding vectors from questions asked about different topics to the client data. See Appendix D.7.1.

Results In Figure 6.2 we report the outcomes. The left shows results for the synthetic Gaussian mixture dataset with 2000 clients, and the right for the stackoverflow dataset, with topics github and pdf. Further results are in Appendix D.9: synthetic data with 1000 and 5000 clients in Figures D.8 and D.9, and the other stackoverflow topics are shown in Figures D.10, D.11 and D.12. For the synthetic data we observe that the baselines that use only server data are unable to overcome their poor initialization, even with more generous privacy budgets. As the total number of clients grows, from 1000 to 2000 to 5000, FedDP-KMeans exhibits better performance for the same privacy budget and the budget at which FedDP-KMeans outperforms server initialization becomes smaller. This is to be expected since the more clients we have the better our amplification by sub-sampling becomes. For stackoverflow we observe that FedDP-KMeans exhibits the best performance, except for in a few cases in the low privacy budget regime. k-FED performs quite poorly overall, tending to be outperformed by the private baselines. As in data-point-level privacy, we find the quality of FedDP-Init's initialization to be excellent: few, if any, Lloyd's steps are required for optimality (see Table D.5, Appendix A.3).

[ZZL+25] also work on federated clustering but with a focus on an asynchronous and heterogeneous setting rather than on differential privacy. To our knowledge, only two prior works combine the advantages of DP and FL. [LWL23] is orthogonal to our work, as it targets *vertical FL* (all clients posses the same data points, but different subsets of the features). [DHK24] studies the same problem as we do, but they propose a custom aggregation scheme that does not fit standard security requirements of FL. It uses SpherePacking to initialize, which in our experiments led to poor results.

6.6 Conclusion

In this chapter we presented FedDP-KMeans, a federated and differentially private k-means clustering algorithm. FedDP-KMeans uses out-of-distribution server data to obtain a good initialization. Combined with a simple federated, DP, variant of Lloyd's algorithm we obtain an efficient and practical clustering algorithm. We show that FedDP-KMeans performs well

in practice with both data-point-level and client-level privacy. FedDP-KMeans comes with theoretical guarantees showing exponential convergence to the true cluster centers in the Gaussian mixture setting. A shortcoming of our method is the need to choose hyperparameters, which is difficult when privacy is meant to be ensured. While we provide heuristics for this in Appendix D.8.1, a more principled solution would be preferable. It would also be interesting to explore if the server data could be replaced with a private mechanism based on client data.

CHAPTER 7

Discussion and Future Work

In this thesis we presented several methods in federated learning, primarily with a focus on modeling and dealing with statistically heterogeneous clients. We now provide a summary of the thesis and its contributions before giving an outlook on potential future directions for research.

7.1 Thesis Summary

In chapter 3 we presented a method for explicitly modeling statistically heterogeneous clients. The key to the approach was to represent each client as a sample from a mixture of Dirichlet-multinomials (meta) distribution. The parameters of this distribution could then be inferred using maximum likelihood estimation over the training clients. This, now learned, distribution contains the information of what the seen clients typically "looked like" and one can simulate new clients by sampling from it. We presented an experimental evaluation showing that this approach is effective at modeling heterogeneous features in true federated learning datasets. We additionally observed that this approach can be used to simulate federated training on the central server in a way that better mimics true federated training dynamics than the alternative of i.i.d. simulated clients.

In chapter 4 we presented PeFLL, a personalized FL method that can generate personalized client models without additional finetuning or training. The approach uses an embedding network, which takes in client data and labels, and outputs an embedding vector that should summarize the client data distribution. This embedding vector is then fed into a hypernetwork, located on the server, which outputs the full personalized client model. PeFLL comes with theoretical guarantees in the form of a generalization bound, characterizing performance on unseen clients, as well as a proof of convergence.

In chapter 5 we again presented an approach to training free personalization, however, this time with a focus on personalization to unlabeled clients. FLowDUP also followed a hypernetwork based approach. This time the hypernetwork is trained to output a low dimensional parameterization of the client model, which is made full size by random expansion. This construction reduces the size of the hypernetwork significantly, making it tractable to send to, and run on, the client devices. FLowDUP has a training objective that is theoretically motivated by a PAC-Bayesian generalization bound for transductive multitask learning. In

addition to the labeled loss, the objective also contains a regularization term that is computable using only unlabeled data, allowing unlabeled clients to also contribute to the training process.

In chapter 6 we switched our focus to privacy, in the context federated clustering. We presented a fully federated and differentially private algorithm for k-means clustering. The algorithm consists of an initialization step, FedDP-Init, which can then be refined using a simple federated and differentially private adaptation of Lloyd's algorithm, which we called FedDP-Lloyds. FedDP-Init uses small and potentially out-of-distribution server-side data. It computes weights for these data, such that they better represent the true client data, and clusters these in order to obtain an initialization. We proved, in the setting of Gaussian-mixture data, that FedDP-Init followed by FedDP-Lloyds converges exponentially fast to the true Gaussian means. The method is applicable both when data-level and client-level differential privacy are desired, and our empirical evaluation tested both these settings.

7.2 Thesis Contributions

The primary contribution of this thesis is methodological and lies in the algorithms we proposed that solve previously unaddressed, or under-addressed, problems. Our approach to simulating heterogeneity was the first that is able to explicitly model clients as samples from a distribution, whose parameters we are able to infer, using an efficient federated MLE algorithm. We showed this was able to capture the data heterogeneity present in real federated datasets and could be used to run training simulations on the server that better captured the dynamics of actual federated training. This is useful in many practical applications since privacy and resource constraints prevent repeated runs of federated training on the actual clients, for instance in order to do hyperparameter tuning. The primary novel contribution of our algorithms in personalization lay in their ability to create personalized models without additional training. This has particular relevance in cross-device federated learning where limited client data or computational resources can prevent effective personalization via finetuning. FLowDUP is able to do this in the setting of personalizing to fully unlabeled clients, and is the first method to be able to use the unlabeled clients during the training process itself. It incorporated the additional novelty of generating low dimensional personalized parameters with significant implications for the practicality of the algorithm. The associated reduction in the size of the hypernetwork meant a simplified, "standard", federated communication protocol could be used, allowing for lower overhead, better privacy and easier application to practical settings. FLowDUP achieved state-of-the-art results in the setting of personalization to unlabeled clients. Finally, FedDP-KMeans is the first federated method for differentially private clustering. Previous approaches compromised on privacy in order to adapt to the other challenges of a federated setting. The main challenge and primary contribution of FedDP-KMeans is its initialization. Our approach is able to recover close to, and sometimes exactly, the optimal possible (non-private centralized clustering), using small numbers of iterations and low privacy budgets. Both data-level and client-level privacy can be handled, making FedDP-KMeans applicable in a wide range of practical federated settings.

7.3 Future Directions

Modeling more complex heterogeneity In chapter 3 our goal was to model client data heterogeneity in order to run more representative training simulations on the server. We took a probabilistic approach by explicitly modeling each client as a histogram and inferring a

distribution over these histograms. Such an approach is clearly best suited to modeling a single, discrete dimension of the client data, such as the label. We showed that continuous features can be handled through discretization, in the form of binning, though such an approach must trade off precision (fewer bins) with increasing sparsity (more bins) and finding a balance in practice might be challenging. Moreover, the data heterogeneity that occurs is most likely joint in many dimensions of the data. Indeed we saw in our training simulations that modeling just the label heterogeneity of the clients still led to a gap between the true clients and simulated clients, implying the existence of other heterogeneity in the data, that we did not capture. While in theory, our approach is still able to handle the setting of modeling multiple features simultaneously: with a histogram over the product of all possible values of each feature, this approach quickly breaks down in the face of the curse of dimensionality. Attempting to capture all aspects of the heterogeneity of a client likely requires a fundamentally different approach, e.g. via the application of generative models that have shown huge progress in recent years.

Our goal, of obtaining a dataset on the server that is similar to the client data, is closely related to the task of (DP) synthetic data generation, for which there has been growing interest recently in the context of federated learning. The goal is, given a private dataset D, to create a new (synthetic) dataset D' using a mechanism that is differentially private with respect to the original private data. This approach offers a number of practical advantages. Most notably, it decouples privacy and standard model training pipelines, since, by the DP post-processing guarantee, we can act on the synthetic data freely without incurring additional privacy costs. In particular, we can train on the synthetic data as many times as we like, without needing to add noise to the gradient updates. A number of different approaches exist to generate DP synthetic data from a private central dataset, for instance based on differentially private finetuning [YIL+23, KPS+23], private prediction [ABK+24] and private filtering [XLB+24]. In a federated setting, however, we are unable to run large models on-device which poses a challenge to applying finetuning and prediction methods. Existing works [HSZ+24, HWZ+25] focus on private filtering based approaches due to their need to only compute histogram queries over the private user data.

Future works in the federated setting have two potential directions to improve. Firstly, improving the quality of the DP synthetic data, which is the crucial factor determining usefulness for downstream applications. In particular the methods that produce the best quality synthetic data, finetuning and prediction, have not yet been made practical in a federated setting. Moreover, with current methods, the structure of the federated data is lost, in the sense that our synthetic data is a single central dataset with no notion of what a client is. This could be suboptimal when our end goal is, for instance, a personalized on-device model for each client. Here it might be useful to run a personalized FL algorithm in simulation on the server, and apply the learned model to the real clients in inference mode. Maintaining the notion of a client could be a challenge, especially when user-level privacy is required, and locally generating and sharing synthetic data would constitute a local DP setting and require much larger added noise. Most likely an approach based on aggregating client information, such as to learn a distribution over clients as in chapter 3, would be required.

This then ties into the second direction, namely improving how we leverage the synthetic data we have generated. As already mentioned it could be advantageous to train in a way that better reflects the downstream task of having per-client personalized models, rather than just training centrally on our synthetic data. Another possibility is using the synthetic data to guide future downstream training. For instance running model selection and hyperparameter tuning on the synthetic data in order to "one-shot" train on the true federated data down the

line. Investigating under what conditions HP tuning transfers to the true federated training would be an interesting direction to better understand the feasibility of such an approach.

Impact of Foundation Models The recent rise of large, pretrained foundation models has led to a paradigm shift in much of machine learning. In the context of federated learning, this has been impactful both in regard to the data utilized and the training methodologies employed. Firstly, as previously discussed, foundation models have strong potential in their ability to generate synthetic data that mimics the statistical properties of real data. This fundamentally changes the nature of FL by enabling a shift in the location of the training data, from clients to the server, while still maintaining data privacy. We refer the reader to the previous section for a more detailed overview of current trends in federated synthetic data.

Regarding training, the advent of foundation models has led to a major change in the main objective of federated systems. Initially, the goal of such systems was to train (typically small) models from scratch on-device. The focus has now shifted significantly toward the efficient adaptation of large, pretrained models. In the context of personalization (Chapters 4 and 5), this necessitates a move toward Parameter-Efficient Fine-Tuning (PEFT) techniques, such as Low-Rank Adaptation (LoRA) [HSW+22] or prompt tuning [LARC21]. While the hypernetwork-based approaches presented in this thesis were intended to generate small-scale personalized models for on-device use, we believe it would be an interesting direction to investigate their applicability within this new paradigm. For instance, rather than generating full model weights, the hypernetwork could be repurposed to generate client-specific LoRA adapters or soft-prompts. This would drastically reduce the dimensionality of the output space, providing an efficient, feed-forward approach to personalizing powerful foundation models to heterogeneous clients. We close by noting that while the methods may evolve, the problem of personalization remains highly relevant. Foundation models are powerful yet general, and the fundamental need to adapt such general models to heterogeneous users remains a critical requirement for many real-world applications.

More efficient personalization In chapters 4 and 5 we found hypernetworks to be a useful technique when applied in the context of personalized federated learning. A significant drawback still remains regarding model size due to the typically very large final layer size. While this was mitigated in part through the low dimensional parameterization introduced in chapter 5, the final layer size scales with k and was still much larger than the equivalent architecture when used for classification. We additionally observed that using much lower values of subspace dimension, table 5.4, led to decreased performance. There exist many techniques for reducing model size that could be combined with our approaches. Pruning [HPTD15], either in the final layer or all layers, could be used to reduce the total parameter counts. Quantization [JKC+18] reduces the memory footprint of each parameter. Low rank factorization [DSD+13], for instance in the final layer, would also allow for a reduction in total parameter count. Finding which techniques, most likely in combination with the low dimensional parameterization of FLowDUP, provide the best trade-off between accuracy and efficiency would be an interesting avenue of future work, with strong practical relevance.

Relaxing the need for server data Our major assumption in chapter 6, that was also crucial to the workings of FedDP-Init, was that of the existence of related server-side data. While this is a reasonable assumption, particularly given that we did not require the server data to follow the same distribution as the client data, it would of course be preferable if it were possible to do without this. As illustrated by the sphere packing initialization, naive attempts

at initializing, without using the client data, perform very poorly. Typically, the dimension of the space is large and the number of centers small, and we can end up with sparsely located centers, often very far from any client data at all. There is potential for improvement of this approach when combined with some of the techniques utilized by FedDP-Init. For instance, we could retain the projection step, thereby reducing the impact of high dimensionality. For low values of k it might then be reasonable to initialize via sphere packing alone, or through a combination of sphere packing, or grid initializing a large number of points, followed by weighting, as done by FedDP-Init, to find the k-most relevant. This is in fact essentially FedDP-Init if we assume the projected server data are the sphere packing points (or a point lattice) in the k-dimensional projected space. One issue with this approach is the grid size scales exponentially with k, so anything other than very small values of k quickly becomes intractable. Investigating if such approaches could work for initializing k-means without server data, or if a radically different approach is required, could be a fruitful and interesting future direction.

7.4 Conclusion

In conclusion, this thesis has presented a range of methods for performing federated learning in the presence of statistical heterogeneity. We view these technical contributions as part of a broader effort to improve the efficacy of privacy-preserving techniques, which currently face significant hurdles to wider adoption.

Presently, practitioners are often faced with a trade-off between model accuracy and privacy guarantees. For instance, in federated learning, accuracy is often lower than in centralized training due to the challenges discussed in this work, a gap that widens further when differential privacy is applied. In the current ecosystem, few practitioners are incentivized to prioritize privacy if it requires accepting a sometimes severe degradation in model performance.

While increased societal awareness and demand for privacy are helpful, we believe that minimizing the cost of adoption to be the most effective way to increase prevalence in real world applications. If the field can reach a point where integrating privacy incurs negligible downside, then privacy-preserving methods will shift from being a compromise to a standard best practice. Enabling a future where high-performance machine learning coexists with strong data privacy is essential for building societal trust and ensuring the long-term acceptance of machine learning technologies.

Bibliography

[AASC19] Manoj Ghuhan Arivazhagan, Vinay Aggarwal, Aaditya Kumar Singh, and Sunav Choudhary. Federated learning with personalization layers. 2019.

[ABDH+20] Hassan Ashtiani, Shai Ben-David, Nicholas JA Harvey, Christopher Liaw, Abbas Mehrabian, and Yaniv Plan. Near-optimal sample complexity bounds for robust learning of Gaussian mixtures via compression schemes. *Journal of the ACM (JACM)*, 67(6):1–42, 2020.

[ABK+24] Kareem Amin, Alex Bie, Weiwei Kong, Alexey Kurakin, Natalia Ponomareva, Umar Syed, Andreas Terzis, and Sergei Vassilvitskii. Private prediction for large-scale synthetic text generation. In *Findings of the Association for Computational Linguistics: EMNLP*, 2024.

[ADG+16] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[AEC24] Ohad Amosy, Gal Eyal, and Gal Chechik. Late to the party? On-demand unlabeled personalized federated learning. In *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2024.

[AGZ21] Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Association for Computational Linguistics (ACL)*, 2021.

[AM05] Dimitris Achlioptas and Frank McSherry. On spectral learning of mixtures of distributions. In *Conference on Computational Learning Theory (COLT)*, 2005.

[AM18] Ron Amit and Ron Meir. Meta-learning by adjusting priors based on extended PAC-Bayes theory. In *International Conference on Machine Learning (ICML)*, 2018.

[AS12] Pranjal Awasthi and Or Sheffet. Improved spectral-norm bounds for clustering. In *International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX)*, 2012.

[AV07] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Symposium on Discrete Algorithms (SODA)*, 2007.

[Bax95] Jonathan Baxter. Learning internal representations. In *Conference on Computational Learning Theory (COLT)*, 1995.

[Bax00] Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research (JAIR)*, 12:149–198, 2000.

[BDMN05] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: the SuLQ framework. In *Symposium on Principles of Database Systems (PODS)*, 2005.

[BEM+17] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnés, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Symposium on Operating Systems Principles*, 2017.

[BGLR14] Luc Bégin, Pascal Germain, François Laviolette, and Jean-Francis Roy. PAC-Bayesian Theory for Transductive Learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2014.

[BIK+16] K. A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[BIK+17] Kallista A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy preserving machine learning. *IACR Cryptol. ePrint Arch.*, page 281, 2017.

[BKS22] Alex Bie, Gautam Kamath, and Vikrant Singhal. Private estimation with public data. *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[BM02] Peter L Bartlett and Shahar Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research (JMLR)*, 2002.

[BMG+19] Duc Bui, Kshitiz Malik, Jack Goetz, Honglei Liu, Seungwhan Moon, Anuj Kumar, and Kang G. Shin. Federated user representation learning. 2019.

[BNJ01] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2001.

[Car97] Rich Caruana. Multitask learning. *Machine Learning*, 28:41–75, 1997.

[CEM+22] Vincent Cohen-Addad, Alessandro Epasto, Vahab Mirrokni, Shyam Narayanan, and Peilin Zhong. Near-optimal private and scalable k-clustering. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[CGKM21] Alisa Chang, Badih Ghazi, Ravi Kumar, and Pasin Manurangsi. Locally private k-means in one round. In *International Conference on Machine Learning (ICML)*, 2021.

[CHMS21] Liam Collins, Hamed Hassani, Aryan Mokhtari, and Sanjay Shakkottai. Exploiting shared representations for personalized federated learning. In *International Conference on Machine Learning (ICML)*, 2021.

[CJN22] Anamay Chaturvedi, Matthew Jones, and Huy Le Nguyen. Locally private k-means clustering with constant multiplicative approximation and near-optimal additive error. In *Conference on Artificial Intelligence (AAAI)*, 2022.

[CK21] Alisa Chang and Pritish Kamath. Practical differentially private clustering. `https://research.google/blog/practical-differentially-private-clustering/`, 2021. Accessed: 2024-09-23.

[CKM+21] Edith Cohen, Haim Kaplan, Yishay Mansour, Uri Stemmer, and Eliad Tsfadia. Differentially-private clustering of easy instances. In *International Conference on Machine Learning (ICML)*, 2021.

[CL18] Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 12(3), 2018.

[CM12] Koby Crammer and Yishay Mansour. Learning multiple tasks using shared hypotheses. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2012.

[CWL+18] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konečný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. LEAF: A benchmark for federated settings. 2018.

[CYG+23] Daoyuan Chen, Liuyi Yao, Dawei Gao, Bolin Ding, and Yaliang Li. Efficient personalized federated learning via sparse model-adaptation. In *International Conference on Machine Learning (ICML)*, 2023.

[Das08] Sanjoy Dasgupta. The hardness of k-means clustering. 2008.

[DDT22] Enmao Diao, Jie Ding, and Vahid Tarokh. Semifl: Semi-supervised federated learning for unlabeled clients with alternate training. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[DHK+21] Simon Shaolei Du, Wei Hu, Sham M. Kakade, Jason D. Lee, and Qi Lei. Few-shot learning via learning the representation, provably. In *International Conference on Learning Representations (ICLR)*, 2021.

[DHK24] Abdulrahman Diaa, Thomas Humphries, and Florian Kerschbaum. FastLloyd: Federated, accurate, secure, and tunable k-means clustering with differential privacy, 2024.

[DHMS21] Frances Ding, Moritz Hardt, John Miller, and Ludwig Schmidt. Retiring adult: New datasets for fair machine learning. *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[DKG+20] Dimitrios Dimitriadis, Kenichi Kumatani, Robert Gmyr, Yashesh Gaur, and Sefik Emre Eskimez. A federated approach in training acoustic models. In *Interspeech*, 2020.

[DKK+22] Ilias Diakonikolas, Daniel M. Kane, Daniel Kongsgaard, Jerry Li, and Kevin Tian. Clustering mixture models in almost-linear time via list-decodable mean estimation. In *Symposium on Theory of Computing (STOC)*, 2022.

[DLS21] Don Dennis, Tian Li, and Virginia Smith. Heterogeneity for the win: One-shot federated clustering. In *International Conference on Machine Learning (ICML)*, 2021.

[DlTHS24] Max Dupré la Tour, Monika Henzinger, and David Saulpic. Making old things new: A unified algorithm for differentially private clustering. In *International Conference on Machine Learning (ICML)*, 2024.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference (TTC)*, 2006.

[DR14a] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[DR14b] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Machine Learning*, 9(3–4), 2014.

[DR17] Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. In *Uncertainty in Artificial Intelligence (UAI)*, 2017.

[DSD+13] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2013.

[DTN20] Canh T. Dinh, Nguyen Hoang Tran, and Tuan Dung Nguyen. Personalized federated learning with moreau envelopes. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[DTTZ14] Cynthia Dwork, Kunal Talwar, Abhradeep Thakurta, and Li Zhang. Analyze Gauss: optimal bounds for privacy-preserving principal component analysis. In *Symposium on Theory of Computing (STOC)*, 2014.

[DVT+22] Canh T. Dinh, Tung Thanh Vu, Nguyen Hoang Tran, Minh N. Dao, and Hongyu Zhang. A new look and convergence rate of federated multi-task learning with Laplacian regularization. *IEEE Transactions on Neural Networks (TNN)*, 2022.

[Dwo06] Cynthia Dwork. Differential privacy. In *Automata, Languages and Programming*, pages 1–12, 2006.

[EKR18] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Foundations and Trends in Privacy and Security*, 2(2–3), 2018.

[FAL17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, 2017.

[FMO20a] Alireza Fallah, Aryan Mokhtari, and Asuman E. Ozdaglar. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[FMO20b] Alireza Fallah, Aryan Mokhtari, and Asuman E. Ozdaglar. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[GBDH20] Jonas Geiping, Hartmut Bauermeister, Michal Drozdzal, and Philipp Hennig. Inverting gradients – how easy is it to break privacy in federated learning? In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[GCYR20a] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[GCYR20b] Avishek Ghosh, Jichan Chung, Dong Yin, and Kannan Ramchandran. An efficient framework for clustered federated learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[GL22] Jiechao Guan and Zhiwu Lu. Fast-rate PAC-Bayesian generalization bounds for meta-learning. In *International Conference on Machine Learning (ICML)*, 2022.

[GPZ+22] Yan Gao, Titouan Parcollet, Salah Zaiem, Javier Fernández-Marqués, Pedro P. B. de Gusmao, Daniel J. Beutel, and Nicholas D. Lane. End-to-end speech recognition from federated acoustic models. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2022.

[GSC+24] Filip Granqvist, Congzheng Song, Áine Cahill, Rogier van Dalen, Martin Pelikan, Yi Sheng Chan, Xiaojun Feng, Natarajan Krishnaswami, Vojta Jina, and Mona Chitnis. pfl-research: simulation framework for accelerating research in private federated learning, 2024.

[HAMS21] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 44(9):5149–5169, 2021.

[HDL17] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations (ICLR)*, 2017.

[HHHR20] Filip Hanzely, Slavomír Hanzely, Samuel Horváth, and Peter Richtárik. Lower bounds and optimal algorithms for personalized federated learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[HHQ12] Ian Holmes, Keith Harris, and Christopher Quince. Dirichlet multinomial mixtures: Generative models for microbial metagenomics. *PLOS ONE*, 7:1–15, 02 2012.

[HL18] Samuel B. Hopkins and Jerry Li. Mixture models, robustness, and sum of squares proofs. In *Symposium on Theory of Computing (STOC)*, 2018.

[HNM+22] Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, Kaikai Wang, Anthony Shoumikhin, Jesik Min, and Mani Malek. PAPAYA: practical, private, and scalable federated learning. In *Conference on Machine Learning and Systems, MLSys,*, 2022.

[HPTD15] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural networks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[HQB19] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *CoRR*, abs/1909.06335, 2019.

[HR20] Filip Hanzely and Peter Richtárik. Federated learning of a mixture of global and local models. 2020.

[HRM+19] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction, 2019.

[HSW+22] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022.

[HSZ+24] Charlie Hou, Akshat Shrivastava, Hongyuan Zhan, Rylan Conway, Trang Le, Adithya Sagar, Giulia Fanti, and Daniel Lazar. Pre-text: Training language models on private federated data in the age of llms. In *International Conference on Machine Learning (ICML)*, 2024.

[Hua05] Jonathan Huang. Maximum likelihood estimation of dirichlet distribution parameters, 2005.

[HWZ+25] Charlie Hou, Mei-Yu Wang, Yige Zhu, Daniel Lazar, and Giulia Fanti. Popri: Private federated learning using preference-optimized synthetic data, 2025.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[JHJY22] Jaehee Jang, Heonseok Ha, Dahuin Jung, and Sungroh Yoon. FedClassAvg: Local representation learning for personalized federated learning on heterogeneous neural networks. In *International Conference on Parallel Processing*. ACM, 2022.

[JKC+18] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[JKRK19] Yihan Jiang, Jakub Konečný, Keith Rush, and Sreeram Kannan. Improving federated learning personalization via model agnostic meta learning. 2019.

[JYYH21] Wonyong Jeong, Jaehong Yoon, Eunho Yang, and Sung Ju Hwang. Federated semi-supervised learning with inter-client consistency & disjoint learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[KK10] Amit Kumar and Ravindran Kannan. Clustering with spectral norm and the k-means algorithm. In *Foundations of Computer Science (FOCS)*, 2010.

[KKM+20] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning (ICML)*, 2020.

[KMA+21] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14, 2021.

[KMRR16] Jakub Konečnỳ, H. Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated learning: Strategies for improving communication efficiency. In *arXiv preprint arXiv:1610.05492*, 2016.

[KMY+16] Jakub Konečnỳ, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated optimization: Distributed machine learning for on-device intelligence. In *arXiv preprint arXiv:1610.02527*, 2016.

[KOV15] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The composition theorem for differential privacy. In *International Conference on Machine Learning (ICML)*, 2015.

[KPS+23] Alexey Kurakin, Natalia Ponomareva, Umar Syed, Liam MacDermed, and Andreas Terzis. Harnessing large-language models to generate private synthetic text. *CoRR*, abs/2306.01684, 2023.

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[KSS18] Pravesh K. Kothari, Jacob Steinhardt, and David Steurer. Robust moment estimation and improved clustering via sum of squares. In *Symposium on Theory of Computing (STOC)*, 2018.

[KSSU19] Gautam Kamath, Or Sheffet, Vikrant Singhal, and Jonathan Ullman. Differentially private algorithms for learning mixtures of separated gaussians. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[LARC21] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, 2021.

[LBBH98a] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBBH98b] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 1998.

[LCW+23] Hongxia Li, Zhongyi Cai, Jingya Wang, Jiangnan Tang, Weiping Ding, Chin-Teng Lin, and Ye Shi. FedTP: Federated learning by transformer personalization. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[LDCH22] Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. Federated learning on non-iid data silos: An experimental study. In *IEEE International Conference on Data Engineering, ICDE*, 2022.

[LFK+22] Sanae Lotfi, Marc Finzi, Sanyam Kapoor, Andres Potapczynski, Micah Goldblum, and Andrew G Wilson. PAC-Bayes compression bounds so tight that they can explain generalization. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[LFLY18] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. In *International Conference on Learning Representations (ICLR)*, 2018.

[LFTL20] Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854, 2020.

[LHBS21] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning (ICML)*, 2021.

[LHLZ22] Shiyun Lin, Yuze Han, Xiang Li, and Zhihua Zhang. Personalized federated learning towards communication efficiency, robustness and fairness. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[LHY+20] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations (ICLR)*, 2020.

[LL22] Allen Liu and Jerry Li. Clustering mixtures with almost optimal separation in polynomial time. In *Symposium on Theory of Computing (STOC)*, 2022.

[LLL+20] Paul Pu Liang, Terrance Liu, Ziyin Liu, Ruslan Salakhutdinov, and Louis-Philippe Morency. Think locally, act globally: Federated learning with local and global representations. 2020.

[Llo82] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory (TIT)*, 28(2):129–136, 1982.

[LMY+20] Yang Liu, Zhuo Ma, Zheng Yan, Zhuzhu Wang, Ximeng Liu, and Jianfeng Ma. Privacy-preserving federated k-means for proactive caching in next generation cellular networks. *Information Sciences*, 521:14–31, 2020.

[LSTS20] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[LSZ+20] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *Conference on Machine Learning and Systems, MLSys*, 2020.

[LWL23] Zitao Li, Tianhao Wang, and Ninghui Li. Differentially private vertical federated clustering. *Proceedings of the VLDB Endowment*, 2023.

[Mau04] Andreas Maurer. A note on the PAC Bayesian theorem. *CoRR*, cs.LG/0411099, 2004.

[Mau06] Andreas Maurer. Bounds for linear multi-task learning. *Journal of Machine Learning Research (JMLR)*, 7:117–139, 2006.

[McA98] David A McAllester. Some PAC-Bayesian theorems. In *Conference on Computational Learning Theory (COLT)*, 1998.

[MCH+15] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. Never-ending learning. In *Conference on Artificial Intelligence (AAAI)*, 2015.

[McS14] Frank McSherry. Differential privacy for measure concentration. `https://windowsontheory.org/2014/02/04/differential-privacy-for-measure-concentration/`, 2014. Accessed: 2024-09-23.

[Min00] Thomas P. Minka. Estimating a dirichlet distribution, 2000.

[MLZ22] Van Sy Mai, Richard J. La, and Tao Zhang. Federated learning with server learning: Enhancing performance for non-iid data. *CoRR*, abs/2210.02614, 2022.

[MMR+17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2017.

[MMRS20a] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. 2020.

[MMRS20b] Yishay Mansour, Mehryar Mohri, Jae Ro, and Ananda Theertha Suresh. Three approaches for personalization with applications to federated learning. *CoRR*, abs/2002.10619, 2020.

[MNB+21] Othmane Marfoq, Giovanni Neglia, Aurélien Bellet, Laetitia Kameni, and Richard Vidal. Federated multi-task learning under a mixture of distributions. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[MNVK22] Othmane Marfoq, Giovanni Neglia, Richard Vidal, and Laetitia Kameni. Personalized federated learning through local memorization. In *International Conference on Machine Learning (ICML)*, 2022.

[MRT20] Payman Mohassel, Mike Rosulek, and Ni Trieu. Practical privacy-preserving k-means clustering. In *Privacy Enhancing Technologies Symposium*, 2020.

[MSDCS19] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2019.

[MV10] Ankur Moitra and Gregory Valiant. Settling the polynomial learnability of mixtures of Gaussians. In *Foundations of Computer Science (FOCS)*, 2010.

[MZGX22] Xiaosong Ma, Jie Zhang, Song Guo, and Wenchao Xu. Layer-wised model aggregation for personalized federated learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[NJW01] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2001.

[Pea94] Karl Pearson. Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, 185, 1894.

[PJK+25] Seonghwan Park, Jaehyeon Jeong, Yongjun Kim, Jaeho Lee, and Namhoon Lee. ZIP: An efficient zeroth-order prompt tuning for black-box vision-language models. In *International Conference on Learning Representations (ICLR)*, 2025.

[PKP+19] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural networks*, 113:54–71, 2019.

[PL14] Anastasia Pentina and Christoph H. Lampert. A PAC-Bayesian bound for lifelong learning. In *International Conference on Machine Learning (ICML)*, 2014.

[PL15] Anastasia Pentina and Christoph H. Lampert. Lifelong learning with non-iid tasks. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[PL17] Anastasia Pentina and Christoph H. Lampert. Multi-task learning with labeled and unlabeled tasks. In *International Conference on Machine Learning (ICML)*, 2017.

[PM13] Massimiliano Pontil and Andreas Maurer. Excess risk bounds for multitask learning with trace norm regularization. In *Conference on Computational Learning Theory (COLT)*, 2013.

[PORSTS21] María Pérez-Ortiz, Omar Rivasplata, John Shawe-Taylor, and Csaba Szepesvári. Tighter risk certificates for neural networks. *The Journal of Machine Learning Research*, 22(1):10326–10365, 2021.

[QYW+23] Zixuan Qin, Liu Yang, Qilong Wang, Yahong Han, and Qinghua Hu. Reliable and interpretable personalized federated learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[Rez22] Arezou Rezazadeh. A unified view on PAC-Bayes bounds for meta-learning. In *International Conference on Machine Learning (ICML)*, 2022.

[RFJK21] Jonas Rothfuss, Vincent Fortuin, Martin Josifoski, and Andreas Krause. PACOH: Bayes-optimal meta-learning with PAC-guarantees. In *International Conference on Machine Learning (ICML)*, 2021.

[RG19] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2019.

[Rin94] Mark Bishop Ring. *Continual Learning in Reinforcement Environments*. PhD thesis, University of Texas at Austin, Computer Science Department, 1994.

[RJFK23] Jonas Rothfuss, Martin Josifoski, Vincent Fortuin, and Andreas Krause. Scalable PAC-Bayesian meta-learning via the PAC-Optimal hyper-posterior: From theory to practice. *Journal of Machine Learning Research (JMLR)*, 2023.

[RV17] Oded Regev and Aravindan Vijayaraghavan. On learning mixtures of well-separated Gaussians. In *Foundations of Computer Science (FOCS)*, 2017.

[RXY+17] Jun Ren, Jinbo Xiong, Zhiqiang Yao, Rong Ma, and Mingwei Lin. DPLK-means: A novel differential privacy k-means mechanism. In *International Conference on Data Science in Cyberspace (DSC)*, 2017.

[SC24] Jonathan Scott and Áine Cahill. Improved modelling of federated datasets using mixtures-of-Dirichlet-multinomials. In *International Conference on Machine Learning (ICML)*, 2024.

[Sch87] Jürgen Schmidhuber. Evolutionary principles in self-referential learning. 1987.

[SCL+16] Dong Su, Jianneng Cao, Ninghui Li, Elisa Bertino, and Hongxia Jin. Differentially private k-means clustering. In *ACM Conference on Data and Application Security and Privacy*, 2016.

[SCL+17] Dong Su, Jianneng Cao, Ninghui Li, Elisa Bertino, Min Lyu, and Hongxia Jin. Differentially private k-means clustering and a hybrid approach to private optimization. *ACM Transactions of Privacy and Security (TOPS)*, 20:1–33, 2017.

[SCST17a] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. Federated multi-task learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[SCST17b] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. Federated multi-task learning. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[SGT22] Congzheng Song, Filip Granqvist, and Kunal Talwar. Flair: Federated learning annotated image repository. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2022.

[SLCB+12] Yevgeny Seldin, François Laviolette, Nicolo Cesa-Bianchi, John Shawe-Taylor, and Peter Auer. PAC-Bayesian inequalities for martingales. *IEEE Transactions on Information Theory*, 58(12):7086–7093, 2012.

[SLS25] Jonathan Scott, Christoph H. Lampert, and David Saulpic. Differentially private federated k-means clustering with server-side data. In *International Conference on Machine Learning (ICML)*, 2025.

[SM00] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 22(8):888–905, 2000.

[SMS20] Felix Sattler, Klaus-Robert Müller, and Wojciech Samek. Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)*, 2020.

[SNFC21] Aviv Shamsian, Aviv Navon, Ethan Fetaya, and Gal Chechik. Personalized federated learning using hypernetworks. In *International Conference on Machine Learning (ICML)*, 2021.

[SOK22] Jaehun Song, Min Hwan Oh, and Hyung-Sin Kim. Personalized federated learning with server-side information. *IEEE Access*, 10:120245–120255, 2022.

[Spe04] Charles Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.

[ST21] David Steurer and Stefan Tiegel. SoS degree reduction with applications to clustering and robust moment estimation. In *Symposium on Discrete Algorithms (SODA)*, 2021.

[SWMS19] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and communication-efficient federated learning from non-iid data. In *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

[SYL23] Jonathan Scott, Michelle Yeo, and Christoph H. Lampert. Cross-client label propagation for transductive and semi-supervised federated learning. *Transactions on Machine Learning Research*, 2023.

[SZL24] Jonathan Scott, Hossein Zakerinia, and Christoph H. Lampert. PeFLL: Personalized federated learning by learning to learn. In *The Twelfth International Conference on Learning Representations*, 2024.

[TCK+22] Eliad Tsfadia, Edith Cohen, Haim Kaplan, Yishay Mansour, and Uri Stemmer. FriendlyCore: Practical differentially private aggregation. In *International Conference on Machine Learning (ICML)*, 2022.

[TM95] Sebastian Thrun and Tom Mitchell. Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1):25–46, 1995.

[TP98] Sebastian Thrun and Lorien Pratt. *Learning to Learn*. Kluwer Academic Press, 1998.

[TWM+24] Kunal Talwar, Shan Wang, Audra McMillan, Vojta Jina, Vitaly Feldman, Pansy Bansal, Bailey Basile, Aine Cahill, Yi Sheng Chan, Mike Chatzidakis, Junye Chen, Oliver Chick, Mona Chitnis, Suman Ganta, Yusuf Goren, Filip Granqvist, Kristine Guo, Frederic Jacobs, Omid Javidbakht, Albert Liu, Richard Low, Dan Mascenik, Steve Myers, David Park, Wonhee Park, Gianni Parsa, Tommy Pauly, Christian Priebe, Rehan Rishi, Guy Rothblum, Michael Scaria, Linmao Song, Congzheng Song, Karl Tarbe, Sebastian Vogt, Luke Winstrom, and Shundong Zhou. Samplable anonymous aggregation for private federated data analysis, 2024.

[Vap98] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[VC71] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280, 1971.

[VD02] Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artifial Intelligence Reviews*, 18(2):77–95, 2002.

[vdMH08] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[WBAH23] Jun Wu, Wenxuan Bao, Elizabeth Ainsworth, and Jingrui He. Personalized federated learning with parameter propagation. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, 2023.

[WZY+23] Yue Wu, Shuaicheng Zhang, Wenchao Yu, Yanchi Liu, Quanquan Gu, Dawei Zhou, Haifeng Chen, and Wei Cheng. Personalized federated learning under mixture of distributions. In *International Conference on Machine Learning (ICML)*, 2023.

[XHTZ20] Chang Xia, Jingyu Hua, Wei Tong, and Sheng Zhong. Distributed k-means clustering guaranteeing local differential privacy. *Computers & Security*, 90:101699, 2020.

[XLB+24] Chulin Xie, Zinan Lin, Arturs Backurs, Sivakanth Gopi, Da Yu, Huseyin A. Inan, Harsha Nori, Haotian Jiang, Huishuai Zhang, Yin Tat Lee, Bo Li, and Sergey Yekhanin. Differentially private synthetic data via foundation model apis 2: Text. In *International Conference on Machine Learning (ICML)*, 2024.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 2017.

[YAG+19] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan H. Greenewald, Trong Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In *International Conference on Machine Learning (ICML)*, 2019.

[YCW+24] Tiandi Ye, Cen Chen, Yinggui Wang, Xiang Li, and Ming Gao. UPFL: unsupervised personalized federated learning towards new clients. In *Proceedings of the 2024 SIAM International Conference on Data Mining, SDM*, 2024.

[YIL+23] Xiang Yue, Huseyin A. Inan, Xuechen Li, Girish Kumar, Julia McAnallen, Hoda Shajari, Huan Sun, David Levitan, and Robert Sim. Synthetic text generation with differential privacy: A simple and practical recipe. In *Association for Computational Linguistics (ACL)*, 2023.

[YLK+18] Niloofar Yousefi, Yunwen Lei, Marius Kloft, Mansooreh Mollaghasemi, and Georgios C Anagnostopoulos. Local Rademacher complexity-based learning guarantees for multi-task learning. *Journal of Machine Learning Research (JMLR)*, 19(38):1–47, 2018.

[YNW+23] Rui Ye, Zhenyang Ni, Fangzhao Wu, Siheng Chen, and Yanfeng Wang. Personalized federated learning with inferred collaboration graphs. In *International Conference on Machine Learning (ICML)*, 2023.

[YSW+23] Liping Yi, Xiaorong Shi, Nan Wang, Ziyue Xu, Gang Wang, and Xiaoguang Liu. pFedLHNs: Personalized federated learning via local hypernetworks. In *International Conference on Artificial Neural Networks (ICANN)*, 2023.

[ZBL24] Hossein Zakerinia, Amin Behjati, and Christoph Lampert. More flexible PAC-Bayesian meta-learning by learning learning algorithms. In *International Conference on Machine Learning (ICML)*, 2024.

[ZGL25] Hossein Zakerinia, Dorsa Ghobadi, and Christoph H Lampert. From low intrinsic dimensionality to non-vacuous generalization bounds in deep multi-task learning. *arXiv preprint arXiv:2501.19067*, 2025.

[ZKR+17] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[ZKY+20] Fengda Zhang, Kun Kuang, Zhaoyang You, Tao Shen, Jun Xiao, Yin Zhang, Chao Wu, Yueting Zhuang, and Xiaolin Li. Federated unsupervised representation learning. 2020.

[ZLD+23] Hao Zhang, Chenglin Li, Wenrui Dai, Junni Zou, and Hongkai Xiong. FedCR: Personalized federated learning based on across-client common representation with conditional mutual information regularization. In *International Conference on Machine Learning (ICML)*, 2023.

[ZLH19] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019.

[ZLH+22] En Zhang, Huimin Li, Yuchen Huang, Shuangxi Hong, Le Zhao, and Congmin Ji. Practical multi-party private collaborative k-means clustering. *Neurocomputing*, 467:256–265, 2022.

[ZLL+18] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *CoRR*, abs/1806.00582, 2018.

[ZSL25] Hossein Zakerenia, Jonathan Scott, and Christoph H. Lampert. Federated learning with unlabeled clients: Personalization can happen in low dimensions, 2025. Preprint.

[ZZL+25] Yunfan Zhang, Yiqun Zhang, Yang Lu, Mengke Li, Xi Chen, and Yiu-ming Cheung. Asynchronous federated clustering with unknown number of clusters. In *Conference on Artificial Intelligence (AAAI)*, 2025.

APPENDIX A

Appendix for Chapter 3

A.1 How to select the number of mixture components

We detail here a strategy for choosing which K to use in Algorithms 5 and 6 based on selecting the value that maximizes the log likelihood of a validation cohort of clients.

A.1.1 Choosing K by validation log likelihood

1. Run Algorithms 1 and 2 until convergence on multiple choices of K in parallel.
2. Sample a new cohort of clients that we have not yet seen and for each choice of K evaluate the log likelihood, Equation 3.4, on this cohort of clients.
3. Use the K that gave the highest log likelihood on this validation cohort of clients.
4. In the case of (approximate) ties choose the smallest K.

The crucial point here is that this procedure to choose K can be done one shot and does not require multiple federated training runs which would be highly inefficient. This is because practically it is possible to run inference using multiple values of K in parallel due to the very low computational and communication related overheads of Algorithms 5 and 6.

A.1.2 Experimental evaluation

We provide here an experimental evaluation of the proposed method. Our evaluation procedure is as follows. We first fix the values of the ground truth MDM parameters. Specifically, we set the ground truth number of mixture components to $K = 3$, the mixture weights to

$$\boldsymbol{\tau} = [0.2, 0.5, 0.3],$$

the Dirichlet parameters to

$$\mathbf{A} = \begin{bmatrix} 0.1 & 0.2 & 0.1 & 0.3 & 0.1 \\ 1 & 4 & 1 & 2 & 0.5 \\ 10 & 5 & 3 & 2 & 30 \end{bmatrix},$$

and we set the number of samples of each component to 100, with $\mathbf{\Pi}$ defined accordingly. We then draw n clients (histograms) from this ground truth MDM distribution, and for $K \in \{1, \dots, 6\}$ we infer MDM parameters using Algorithms 1 and 2 from the paper, using this sample of clients. Finally, we sample a new validation cohort of 1000 clients from the true distribution and we compute the mean log likelihood of this validation cohort, using the parameters inferred for each K. We do this for $n = 100, 200$ and 1000 and plot the results in Figure A.1.

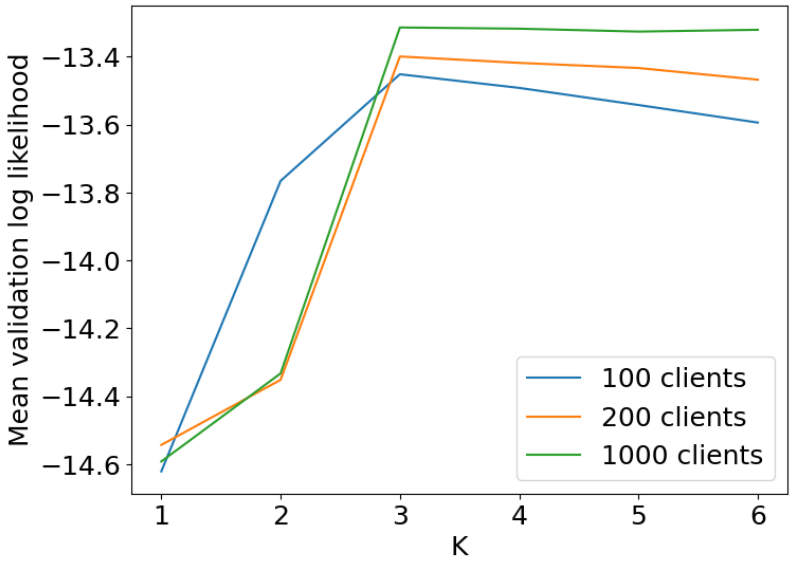

Figure A.1: Mean log likelihood on the validation cohort of clients with the parameters inferred using different values of K.

As we can see in the figure, for each n, step 3 of the given procedure (choose the smallest value of K that gives the maximum log likelihood), leads to us choosing to use the correct ground truth value of $K = 3$. Interestingly, when we infer using a smaller number of clients (100 and 200) we observe some overfitting when K is chosen to be too large. This effect is largely mitigated by inferring on a greater number of clients.

A.2 Proofs

A.2.1 Derivation of moment matching estimate

Let $\mathbf{p} \sim \text{Dir}(\boldsymbol{\alpha})$ and recall that $\alpha_0 := \sum_{j=1}^{C} \alpha_j$. Then for all $j \in [C]$ we have that the first two moments of the Dirichlet are:

$$\mathbb{E}\, p_j = \frac{\alpha_j}{\alpha_0}, \tag{A.1}$$

$$\mathbb{E}\, p_j^2 = \frac{\alpha_j(1+\alpha_j)}{\alpha_0(1+\alpha_0)}. \tag{A.2}$$

Moreover, from these two equations it can be checked that

$$\alpha_0 = \frac{\mathbb{E}\, p_1 - \mathbb{E}\, p_1^2}{\mathbb{E}\, p_1^2 - (\mathbb{E}\, p_1)^2}. \tag{A.3}$$

Therefore, combining equations (A.1) and (A.3) we obtain:

$$\alpha_j = \frac{\mathbb{E}\, p_1 - \mathbb{E}\, p_1^2}{\mathbb{E}\, p_1^2 - (\mathbb{E}\, p_1)^2}\, \mathbb{E}\, p_j. \tag{A.4}$$

This is precisely the moment matching estimate used in algorithm 5 where each client normalizes their count vector $\mathbf{c}_i$ to a probability vector $\mathbf{p}_i = \frac{1}{m_i}\mathbf{c}_i$ which we then assume is drawn from a Dirichlet distribution. Equation (A.4) is then used to initialize $\boldsymbol{\alpha}$ where the true expectations are replaced by the empirical mean over the sampled clients.

A.2.2 Proof of Theorem 1

We let $\boldsymbol{\theta} = (\boldsymbol{\tau}, \mathbf{A}, \boldsymbol{\Pi})$ denote the parameter variables and $\boldsymbol{\theta}^{(t)} = (\boldsymbol{\tau}^{(t)}, \mathbf{A}^{(t)}, \boldsymbol{\Pi}^{(t)})$ denote the current values of the parameters after t steps. Let Z_i be the latent (unobserved) variable denoting which component the ith observation was sampled from. Following standard expectation maximization our goal will be to improve Q, which is guaranteed to lead to improvements in the log likelihood L.

$$\begin{aligned}
Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)}) &= \mathop{\mathbb{E}}_{Z \sim p(\cdot \mid \mathbf{C}, \mathbf{n}, \boldsymbol{\theta}^{(t)})} \log p(\mathbf{C}, \mathbf{n}, Z \mid \boldsymbol{\theta}) && \text{(A.5)}\\
&= \sum_{i=1}^{n} \sum_{k=1}^{K} \omega_k^i \log p(\mathbf{c}_i, m_i, Z_i = k \mid \boldsymbol{\theta}) && \text{(A.6)}\\
&= \sum_{i=1}^{n} \sum_{k=1}^{K} \omega_k^i (\log p(\mathbf{c}_i, m_i \mid \boldsymbol{\alpha}_k, \boldsymbol{\pi}_k) + \log \tau_k) && \text{(A.7)}\\
&= \sum_{i=1}^{n} \sum_{k=1}^{K} \omega_k^i (\log p(\mathbf{c}_i \mid m_i, \boldsymbol{\alpha}_k) + \log \boldsymbol{\pi}_{k m_i} + \log \tau_k) && \text{(A.8)}\\
&= \sum_{i=1}^{n} \sum_{k=1}^{K} \omega_k^i \log p(\mathbf{c}_i \mid m_i, \boldsymbol{\alpha}_k) + \sum_{i=1}^{n} \sum_{k=1}^{K} \omega_k^i \log \boldsymbol{\pi}_{k m_i} + \sum_{i=1}^{n} \sum_{k=1}^{K} \omega_k^i \log \tau_k && \text{(A.9)}
\end{aligned}$$

where $\omega_k^i = p(Z_i = k \mid \mathbf{c}_i, m_i, \boldsymbol{\theta}^{(t)})$ is the probability that the ith observation came from the kth mixture component. This can be computed via Bayes rule:

$$\begin{aligned}
\omega_k^i &= \frac{p(\mathbf{c}_i, m_i \mid Z_i = k, \boldsymbol{\theta}^{(t)}) p(Z_i = k \mid \boldsymbol{\theta}^{(t)})}{\sum_{k=1}^{K} p(\mathbf{c}_i, m_i \mid Z_i = k, \boldsymbol{\theta}^{(t)}) p(Z_i = k \mid \boldsymbol{\theta}^{(t)})} && \text{(A.10)}\\
&= \frac{p(\mathbf{c}_i \mid m_i, \boldsymbol{\alpha}_k^{(t)}) \pi_{m_i}^{(t)} \tau_k^{(t)}}{\sum_{k=1}^{K} p(\mathbf{c}_i \mid m_i, \boldsymbol{\alpha}_k^{(t)}) \pi_{m_i}^{(t)} \tau_k^{(t)}}. && \text{(A.11)}
\end{aligned}$$

Now that we have computed Q we seek to find new parameter values at step $t+1$ that increase the value of Q compared to at step t. Note that in standard EM we would be looking to compute $\boldsymbol{\theta}^{(t+1)} = \text{argmax}_{\boldsymbol{\theta}}\, Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)})$, however, in order to make the later application to federated learning practical we are satisfied here with a weaker condition, namely we aim to find $\boldsymbol{\theta}^{(t+1)}$ such that $Q(\boldsymbol{\theta}^{(t+1)} \mid \boldsymbol{\theta}^{(t)}) \geq Q(\boldsymbol{\theta}^{(t)} \mid \boldsymbol{\theta}^{(t)})$. This is still sufficient to guarantee improvements in the log likelihood L, which follows from the standard proof of correctness of EM. Given that in (A.9) our variables appear in a decoupled form we can handle each separately. For $\boldsymbol{\Pi}$ and $\boldsymbol{\tau}$ this is quite simple and we can find a closed form solution that in fact maximizes each term. For $\mathbf{A}$ this is trickier as the DM log likelihood does not have a closed form solution. Instead we derive a fixed point update based on the one in [Min00], that

leads to an improvement in the term. Firstly, for $\mathbf{\Pi}$. We compute the Laplacian

$$F(\mathbf{\Pi}, \Lambda) = \sum_{i=1}^{n}\sum_{k=1}^{K} \omega_k^i \log \boldsymbol{\pi}_{km_i} - \sum_{k=1}^{K} \lambda_k \sum_{j=1}^{M} (\pi_{kj} - 1), \tag{A.12}$$

$$= \sum_{j=1}^{M}\sum_{k=1}^{K} (\sum_{i=1}^{n} \omega_k^i \mathbf{1}_{m_i=j}) \log \boldsymbol{\pi}_{kj} - \sum_{k=1}^{K} \lambda_k \sum_{j=1}^{M} (\pi_{kj} - 1). \tag{A.13}$$

Taking derivatives we obtain

$$\frac{\partial F}{\partial \pi_{kj}} = \frac{1}{\pi_{kj}} \sum_{i=1}^{n} \omega_k^i \mathbf{1}_{m_i=j} - \lambda_k, \tag{A.14}$$

$$\frac{\partial F}{\partial \lambda_k} = \sum_{j=1}^{M} (\pi_{kj} - 1). \tag{A.15}$$

Setting to 0 and solving for π_{kj} we obtain

$$\pi_{kj} = \frac{1}{\sum_{i=1}^{n} \omega_k^i} \sum_{i=1}^{n} \omega_k^i \mathbf{1}_{m_i=j}, \tag{A.16}$$

where $\mathbf{1}_{m_i=j} = 1$ if $m_i = j$ and 0 otherwise. Secondly, for $\boldsymbol{\tau}$. We again define the Laplacian as

$$G(\boldsymbol{\tau}, \lambda) = \sum_{i=1}^{n}\sum_{k=1}^{K} \omega_k^i \log \tau_k - \lambda(\sum_{k=1}^{K} \tau_k - 1). \tag{A.17}$$

Taking derivatives we obtain

$$\frac{\partial G}{\partial \tau_k} = \frac{1}{\tau_k} \sum_{i=1}^{n} \omega_k^i - \lambda, \tag{A.18}$$

$$\frac{\partial G}{\partial \lambda} = \sum_{k=1}^{K} \tau_k - 1. \tag{A.19}$$

Setting to 0 and solving for τ_k we obtain

$$\tau_k = \frac{1}{n} \sum_{i=1}^{n} \omega_k^i. \tag{A.20}$$

Finally, we deal with $\mathbf{A}$. Let

$$H(\mathbf{A}) = \sum_{i=1}^{n}\sum_{k=1}^{K} \omega_k^i \log p(\mathbf{c}_i \mid m_i, \boldsymbol{\alpha}_k) \tag{A.21}$$

$$= \sum_{k=1}^{K}\sum_{i=1}^{n} \omega_k^i \log \frac{\Gamma((\boldsymbol{\alpha}_k)_0)\Gamma(m_i + 1)}{\Gamma(m_i + (\boldsymbol{\alpha}_k)_0)} \prod_{j=1}^{C} \frac{\Gamma(\mathbf{c}_{ij} + \boldsymbol{\alpha}_{kj})}{\Gamma(\boldsymbol{\alpha}_{kj})\Gamma(\mathbf{c}_{ij} + 1)} \tag{A.22}$$

$$= \sum_{k=1}^{K}\sum_{i=1}^{n} \omega_k^i \left(\log \frac{\Gamma((\boldsymbol{\alpha}_k)_0)}{\Gamma(m_i + (\boldsymbol{\alpha}_k)_0)} + \sum_{j=1}^{C} \log \frac{\Gamma(\mathbf{c}_{ij} + \boldsymbol{\alpha}_{kj})}{\Gamma(\boldsymbol{\alpha}_{kj})} \right) + D \tag{A.23}$$

where D is constant w.r.t $\mathbf{A}$. We now recall the following two bounds from [Min00]

$$\frac{\Gamma(x)}{\Gamma(m + x)} \geq \frac{\Gamma(\hat{x})}{\Gamma(m + \hat{x})} \exp\left((\hat{x} - x)b\right) \tag{A.24}$$

where $b = \psi(m + \hat{x}) - \psi(\hat{x})$, and ψ is the digamma function, and

$$\frac{\Gamma(m+x)}{\Gamma(x)} \geq cx^a \tag{A.25}$$

where $a = (\psi(m + \hat{x}) - \psi(\hat{x}))\hat{x}$ and $c = \frac{\Gamma(m+\hat{x})}{\Gamma(\hat{x})}\hat{x}^{-a}$. These bounds hold for all $x, \hat{x} \geq 0$ and all $m \geq 1$. Moreover, both bounds are tight when $x = \hat{x}$. We apply (A.24) with $x = (\boldsymbol{\alpha}_k)_0$ and $\hat{x} = (\boldsymbol{\alpha}_k^{(t)})_0$ and (A.25) with $x = \boldsymbol{\alpha}_{kj}$ and $\hat{x} = \boldsymbol{\alpha}_{kj}^{(t)}$. Plugging these into (A.23) and collecting everything that does not depend on $\mathbf{A}$ into the D' term we obtain

$$H(\mathbf{A}) \geq \sum_{k=1}^{K}\sum_{i=1}^{n}\omega_k^i\left((1-(\boldsymbol{\alpha}_k)_0)b_{ki} + \sum_{j=1}^{C} a_{kij}\log\boldsymbol{\alpha}_{kj}\right) + D' = H'(\mathbf{A}). \tag{A.26}$$

Now if we maximize H' and set $\mathbf{A}^{(t+1)} = \text{argmax}_{\mathbf{A}}\, H'(\mathbf{A})$ and use the fact that the bounds are tight at $\mathbf{A} = \mathbf{A}^{(t)}$ we obtain

$$H(\mathbf{A}^{(t)}) = H'(\mathbf{A}^{(t)}) \leq H'(\mathbf{A}^{(t+1)}) \leq H(\mathbf{A}^{(t+1)}) \tag{A.27}$$

as was our initial goal. So all that remains is to maximize $H'(\mathbf{A})$. Taking partial derivatives w.r.t $\boldsymbol{\alpha}_{kj}$ and setting to 0 we obtain

$$\sum_{i=1}^{n}\omega_k^i\left(-b_{ki} + a_{kij}\frac{1}{\boldsymbol{\alpha}_{kj}}\right) = 0 \tag{A.28}$$

hence

$$\boldsymbol{\alpha}_{kj} = \frac{\sum_{i=1}^{n}\omega_k^i a_{kij}}{\sum_{i=1}^{n}\omega_k^i b_{ki}}. \tag{A.29}$$

Therefore, we obtain the update rule

$$\boldsymbol{\alpha}_{kj}^{(t+1)} = \boldsymbol{\alpha}_{kj}^{(t)}\frac{\sum_{i=1}^{n}\omega_k^i\left(\psi(\mathbf{c}_{ij} + \boldsymbol{\alpha}_{kj}^{(t)}) - \psi(\boldsymbol{\alpha}_{kj}^{(t)})\right)}{\sum_{i=1}^{n}\omega_k^i\left(\psi(m_i + (\boldsymbol{\alpha}_k^{(t)})_0) - \psi((\boldsymbol{\alpha}_k^{(t)})_0)\right)}. \tag{A.30}$$

A.3 Experiment Details

A.3.1 Normalized MSE

In Section 3.4.1 we use normalized mean squared as a metric to measure how accurately we recover the ground truth MDM distribution parameters. Here we define normalized MSE as:

$$NMSE(\mathbf{x}, \mathbf{y}) \coloneqq \sqrt{||\frac{\mathbf{x}-\mathbf{y}}{\mathbf{y}}||^2} \tag{A.31}$$

where the division is entry-wise. Thus we are measuring the MSE but normalized by the size of $\mathbf{y}$, which in our case will be the ground truth parameters. This simply has the effect of ensuring that our metric is invariant to the size of the parameters we are trying to recover.

A.3.2 Folktables Dataset

We provide here details of the Folktables dataset used in our experimental evaluation. Folktables, [DHMS21], is a US census dataset from the year 2018 with a natural partitioning into heterogeneous federated clients. Each datapoint corresponds to an individual, with many features describing the individual, including age, gender, race, employment details, etc. The dataset comes with a number of different possible prediction tasks, such as predicting employment, commute time, health etc. given user features. Full details on the dataset can be found at the GitHub page: `https://github.com/socialfoundations/folktables/tree/main`. The data contains a partitioning into federated clients based on location. Specifically, a feature of the data is the PUMA code, which is a code specifying the area the individual is registered to live in. Splitting the data based on PUMA code gives 2373 clients, each client holding the data of all individuals that live in the corresponding region. Due to the natural geographical population heterogeneity within the United States this partitioning leads to clients that are statistically heterogeneous in a multitude of factors. Figure A.2 shows the histogram of the number of samples per client, which shows heterogeneity in the amount of data per client.

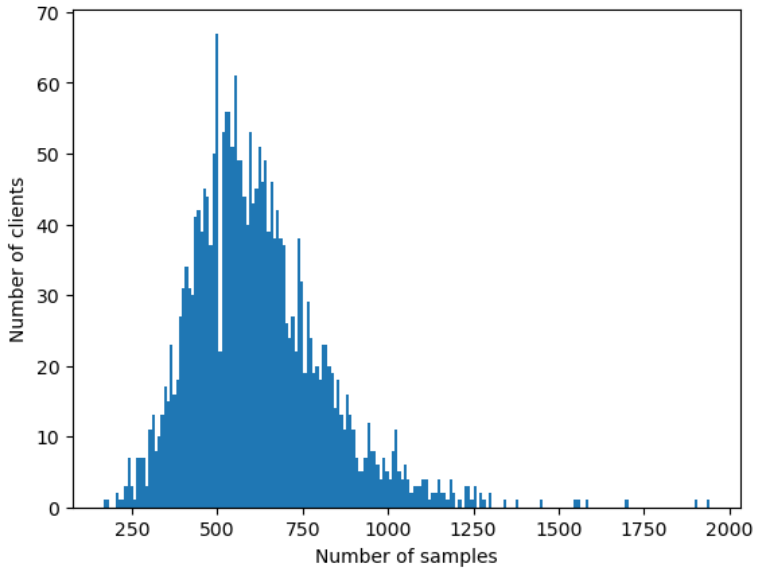

Figure A.2: Histogram of the number of samples per client in the federated partition of the Folktables dataset.

Modeling the Race Feature In the Folktables dataset this feature is categorical with $C = 9$ different possible values. Given the sensitivity of this feature, as well as it's importance in many associated questions of fairness in downstream tasks, this is a good example of a feature one would want to ensure can be both privately learned and accurately represented in our server-side simulations.

Modeling the Income Feature As discussed in Section 3.4 income is a non-categorical feature of the data that describes an individual's annual income. These are real values, which in the 2018 census data range from 0 to 1423000. We bin the values using intervals of length 5000 up to 200000, with the final bin being all values greater than this. That is to say the

bins are defined by the following 41 semi-closed intervals:

$$[0, 5000),\ [5000, 10000),\ \ldots\ ,\ [195000, 200000),\ [200000, \infty).$$

Thus our binned income is now a categorical feature with $C = 41$ possible values.

A.3.3 Hyperparameters for inference

Heterogeneity and number of components	**Parameter Values**
Low heterogeneity 1 mixture component	$\mathbf{A} = \begin{bmatrix} 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix}$ $\boldsymbol{\tau} = \begin{bmatrix} 1.0 \end{bmatrix}$
Low heterogeneity 2 mixture components	$\mathbf{A} = \begin{bmatrix} 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \\ 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 \end{bmatrix}$ $\boldsymbol{\tau} = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$
Low heterogeneity 3 mixture components	$\mathbf{A} = \begin{bmatrix} 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 & 2.5 \\ 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 & 0.5 \end{bmatrix}$ $\boldsymbol{\tau} = \begin{bmatrix} 0.333 & 0.334 & 0.333 \end{bmatrix}$
Medium heterogeneity 1 mixture component	$\mathbf{A} = \begin{bmatrix} 0.1 & 0.2 & 0.6 & 1.0 & 2.0 & 0.1 & 1.0 & 2.0 & 0.5 & 0.5 \end{bmatrix}$ $\boldsymbol{\tau} = \begin{bmatrix} 1.0 \end{bmatrix}$
Medium heterogeneity 2 mixture components	$\mathbf{A} = \begin{bmatrix} 0.1 & 0.2 & 0.6 & 1.0 & 2.0 & 0.1 & 1.0 & 2.0 & 0.5 & 0.5 \\ 2.5 & 2.6 & 2.7 & 2.8 & 3.0 & 2.5 & 2.0 & 3.0 & 1.0 & 0.9 \end{bmatrix}$ $\boldsymbol{\tau} = \begin{bmatrix} 0.4 & 0.6 \end{bmatrix}$
Medium heterogeneity 3 mixture components	$\mathbf{A} = \begin{bmatrix} 5.0 & 4.0 & 5.0 & 1.0 & 1.0 & 1.0 & 5.0 & 4.0 & 5.0 & 1.0 \\ 0.1 & 0.2 & 0.6 & 1.0 & 2.0 & 0.1 & 1.0 & 2.0 & 0.5 & 0.5 \\ 2.5 & 2.6 & 2.7 & 2.8 & 3.0 & 2.5 & 2.0 & 3.0 & 1.0 & 0.9 \end{bmatrix}$ $\boldsymbol{\tau} = \begin{bmatrix} 0.5 & 0.2 & 0.3 \end{bmatrix}$
High heterogeneity 1 mixture component	$\mathbf{A} = \begin{bmatrix} 0.1 & 0.2 & 0.15 & 0.18 & 0.1 & 0.05 & 0.08 & 0.4 & 0.2 & 0.12 \end{bmatrix}$ $\boldsymbol{\tau} = \begin{bmatrix} 1.0 \end{bmatrix}$
High heterogeneity 2 mixture components	$\mathbf{A} = \begin{bmatrix} 0.1 & 0.2 & 0.15 & 0.18 & 0.1 & 0.05 & 0.08 & 0.4 & 0.2 & 0.12 \\ 2.5 & 2.6 & 2.0 & 3.2 & 1.5 & 0.9 & 0.8 & 1.3 & 3.1 & 2.4 \end{bmatrix}$ $\boldsymbol{\tau} = \begin{bmatrix} 0.1 & 0.9 \end{bmatrix}$
High heterogeneity 3 mixture components	$\mathbf{A} = \begin{bmatrix} 5.0 & 5.0 & 0.2 & 0.2 & 3.1 & 3.0 & 3.2 & 0.8 & 0.9 & 5.0 \\ 0.1 & 0.2 & 0.15 & 0.18 & 0.1 & 0.05 & 0.08 & 0.4 & 0.2 & 0.12 \\ 2.5 & 2.6 & 2.0 & 3.2 & 1.5 & 0.9 & 0.8 & 1.3 & 3.1 & 2.4 \end{bmatrix}$ $\boldsymbol{\tau} = \begin{bmatrix} 0.8 & 0.05 & 0.15 \end{bmatrix}$

Table A.1: Mixture-of-Dirichlet-Multinomial distribution parameter values.

In Section 3.4.1 we evaluate how well Algorithms 5 and 6 are able to recover the ground truth distribution parameters, when they exist, and infer meaningful values for the parameters when run on real federated data. For the synthetic data which follows a MDM distribution we ran experiments using 1, 2 or 3 ground truth mixture components with three different settings

of ground truth distribution parameters in each case. These settings corresponded to low, medium and high levels of client statistical heterogeneity. The exact values are given in Table A.1. Algorithm 5 was run using a client cohort size of 1000 followed by Algorithm 6 which was run for 100 global rounds with a client cohort size of 1000.

For inference on FEMNIST we there are no ground truth distribution parameters for us to set, we use the existing partition of the dataset into the true clients for inference. We run inference using 2 and 3 mixture components. In both cases we run algorithm 5 using a client cohort size of 3400 followed by Algorithm 6 for 50 global rounds with a client cohort size of 3400.

A.3.4 Hyperparameters for model training

In Section 3.4.2 we train a model using Federated Averaging on the true clients and on various types of simulated clients over both CIFAR10 and FEMNIST. For both datasets the model used is a CNN with 2 convolutional layers, one dense hidden layer and ReLU activations. In our experiments we compare the performance on the true clients against the simulated clients while varying the certain important hyperparameters. For CIFAR10 we vary the local batch size over $[10, 15, 20, 25]$, the local number of epochs over $[1, 2, 5, 10]$ and the local learning rate over $[0.005, 0.01, 0.05, 0.1, 0.5]$. We report over all combinations of these HPs. The remaining hyperparameters are fixed and equal across true client and all simulated client training. Global learning rate for FedAvg is 1.0, client cohort size is 50, and the number of global training rounds is 1500.

For FEMNIST we vary the local number of epochs over $[1, 2, 5, 10]$ and the local learning rate over $[0.005, 0.01, 0.05]$. We report over all combinations of these HPs. The remaining hyperparameters are fixed and equal across true client and all simulated client training. Global learning rate for FedAvg is 1.0, client cohort size is 50, the number of global training rounds is 1500 and the local batch size is 10.

A.4 Additional Experiments

Here we include additional experiments and figures relating to the empirical evaluation in Section 3.4.

A.4.1 Distribution Parameter Inference

Inference with known parameters In Section 3.4.1 we first investigated how well we are able to recover ground truth parameters when the clients follow the assumed MDM distribution. We reported the MSE over time for clients corresponding the medium levels of heterogeneity. Figures A.3 and A.4 show the results for when running inference on clients with low and high levels of client statistical heterogeneity respectively. Recall the exact values are given in Table A.1.

Inference without known parameters In Section 3.4.1 we additionally evaluated inferring the MDM distribution on federated datasets with a natural partitioning into federated clients. For FEMNIST we showed results when inferring using 3 mixture components. Here we show results when inferring with different numbers of components. The experimental setup is exactly as described in Section 3.4.1. We plot t-SNE visualisations of the simulated clients when we infer using both 2 and 3 mixture components. The results are shown in Figure A.5. We see

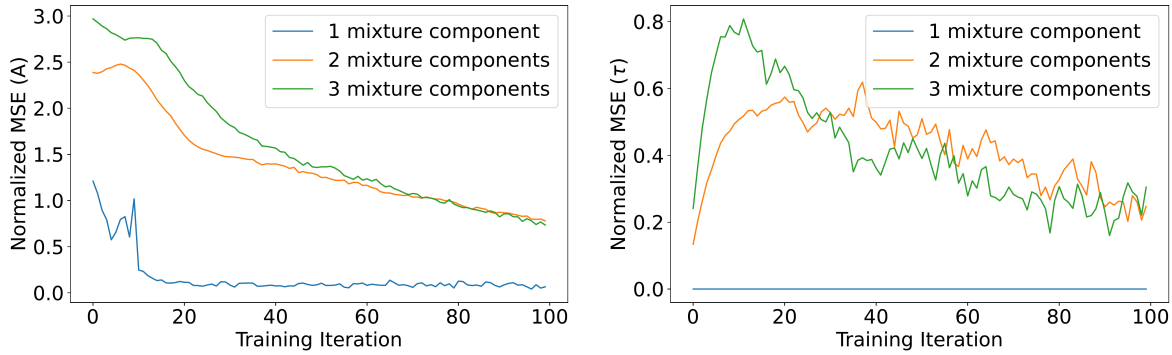

Figure A.3: Normalized mean squared error (MSE) between the ground truth distribution parameter value and the inferred parameter value over time. Ground truth corresponds to low levels of client statistical heterogeneity. On the left for $\mathbf{A}$, on the right for $\boldsymbol{\tau}$.

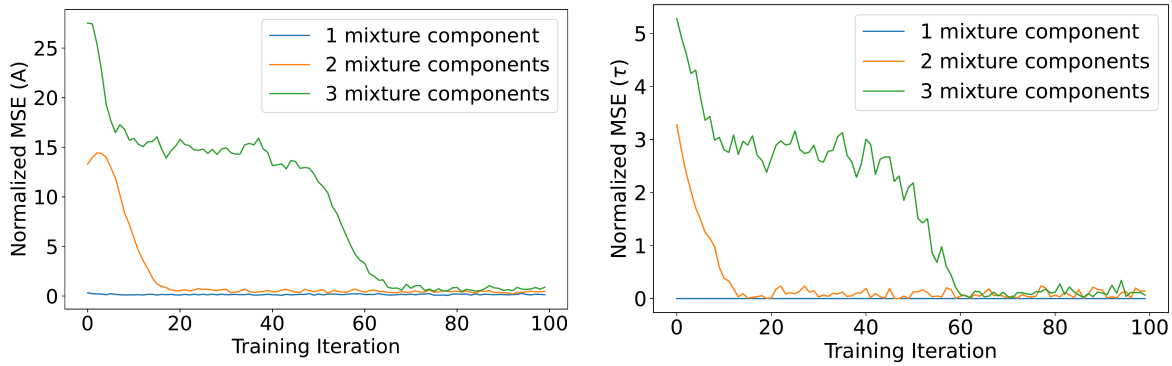

Figure A.4: Normalized mean squared error (MSE) between the ground truth distribution parameter value and the inferred parameter value over time. Ground truth corresponds to high levels of client statistical heterogeneity. On the left for $\mathbf{A}$, on the right for $\boldsymbol{\tau}$.

Figure A.5: t-SNE visualisation of FEMNIST clients, each point corresponds to a single client's class histogram. True clients (green), fully IID simulated clients (blue) and MDM clients (red). On the left we infer using 2 mixture components and on the right we use 3 components.

that in both cases the inferred MDM distribution does a superior job of capturing the class heterogeneity of the true clients compared to the fully IID baseline. We see in the case of 2 components that the inference is dominated by the large left and lower right clusters of true clients while adding the flexibility of an extra component gives better coverage of the small upper right hand cluster. For the Folktables dataset we also infer over a range of different values of K, namely $K = 1, 3, 5, 7$. The results when modeling the race feature are shown in Figure A.6. As we can see in all cases the MDM distribution is able to well model the true federated clients while the fully IID baseline performs poorly. We observe qualitatively

that $K \geq 3$ leads to a better simulation of the true federated clients, with more complete coverage of the tails. The results for modelling the binned income feature are shown in Figure A.7. As we can see, with the exception of $K = 1$, the simulated MDM clients exhibit strong similarity to the true clients with larger values of K being better. It is interesting to observe that for the case $K = 1$ the MDM clients fail quite badly at simulating the heterogeneity of the true clients, again confirming the importance of the mixture model beyond just using a single Dirichlet-multinomial.

A.4.2 Federated Training Simulations

We provide here additional experiments when running training simulations on MDM simulated clients as described in Section 3.4.2.

Disjoint Server and Client Data We provide here additional experiments in the setting that the server-side data and client data are different and non-overlapping. We do this in the setting of the hyperparameter sweep experiments described in Section 4.2 of the paper. We again use the FEMNIST dataset. Previously our server side data was the whole FEMNIST dataset, i.e. data of all clients shuffled together with client identifiers removed. We now create disjoint server and client datasets by assigning roughly half of the data to the server and leaving the other half as our true clients. Concretely we do this split as follows:

We take the first 1302 FEMNIST clients (in the original ordering of clients in the dataset). This corresponds to roughly half of the data, and we leave these clients unchanged. These are our true federated clients. The data of the remaining clients, client indices [1302:], is shuffled together into a single dataset and is used as our server-side data. This splitting introduces a domain shift between the server data and the true client data. That is because the clients in FEMNIST are not randomly ordered and there is in fact a difference between the first roughly 1300 clients and the remaining clients (both in terms of the number of samples they possess and the statistical heterogeneity of the clients). We refer you to figure 3 in the paper, this difference is in fact exactly shown by the left and right clusters of the true clients (green) in the t-SNE visualisation. We now proceed identically to the experimental evaluation of Section 4.2. We infer our MDM distribution parameters using Algorithms 1 and 2 on the true clients. We use these inferred parameters to partition the server-side data into simulated MDM clients. We then train on these simulated clients using a range of different HP settings. We compare the obtained accuracy to the accuracy of training on the true clients as well as our Fully IID and Conditionally IID simulated client baselines. The results are shown in Figure A.8. As we can see there is a much greater similarity between the MDM simulations and true clients than between the IID simulations and the true clients.

CIFAR10 with MDM partitioning Here we include the additional results for training on MDM partitioned clients of CIFAR10. There correspond to the settings of the parameters given in Table A.1. In Section 3.4.1 we showed results for 2 components and high heterogeneity. The results for the remaining settings are shown in Figures A.9 - A.16. In general these results exhibit very similar properties to what we observed in Section 3.4.1. We do see in the cases of Medium and High heterogeneity with 3 components some slightly larger deviations between the MDM simulations, and the true clients, although they are still very similar. This is a reflection of the fact that the parameters we inferred in Section 3.4.1 for these settings differed more than for the other settings.

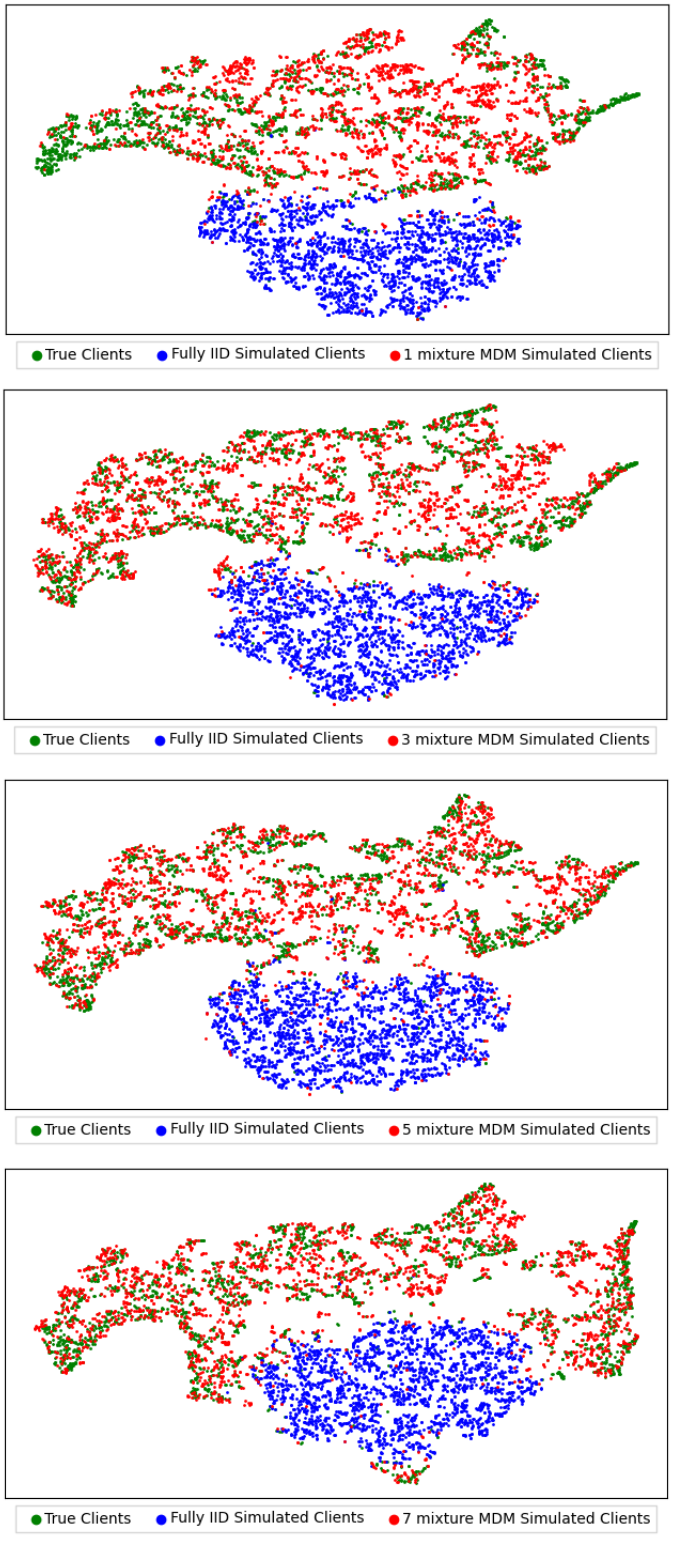

Figure A.6: t-SNE visualisations of Folktables clients, each point corresponds to a single client's histogram of the race feature. True clients (green), fully IID simulated clients (blue) and MDM clients (red). Inferred using (from top to bottom) $K = 1, 3, 5, 7$.

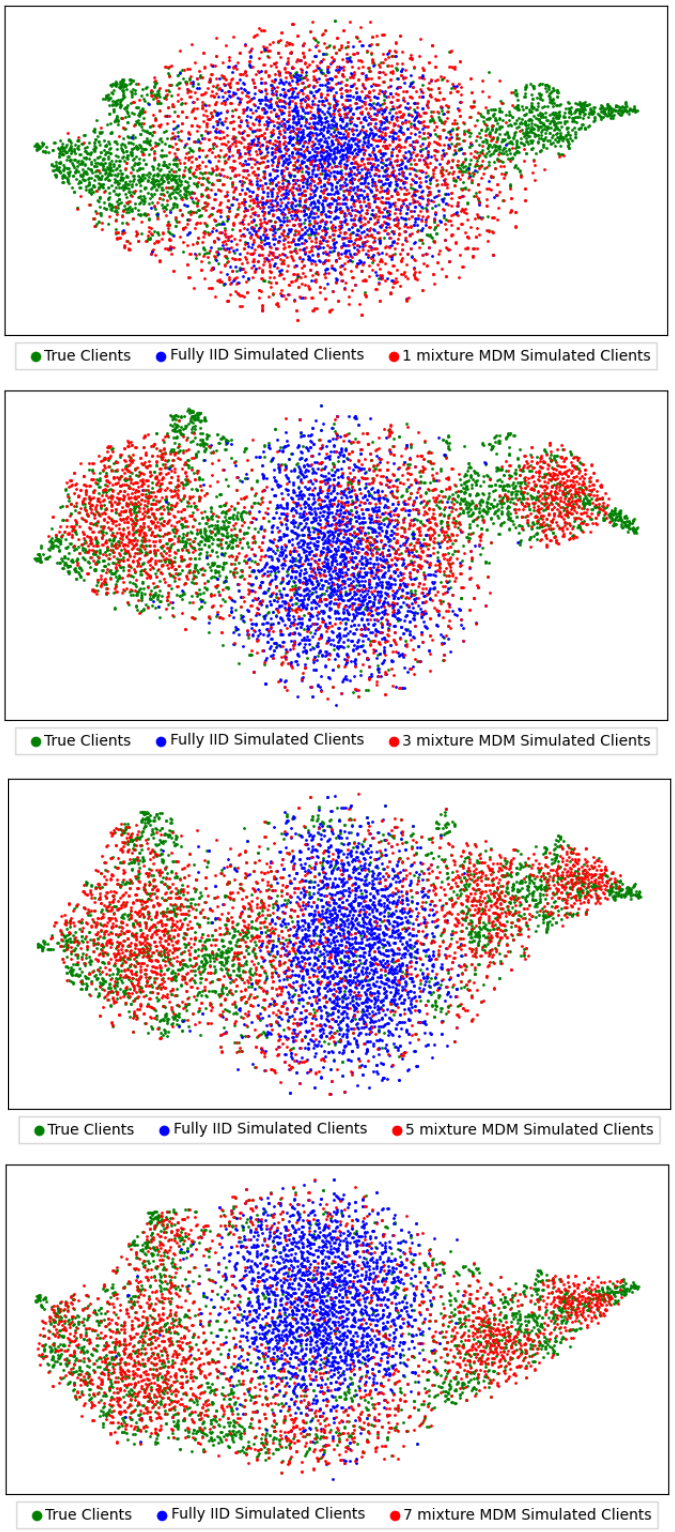

Figure A.7: t-SNE visualisations of Folktables clients, each point corresponds to a single client's histogram of the binned income feature. True clients (green), fully IID simulated clients (blue) and MDM clients (red). Inferred using (from top to bottom) $K = 1, 3, 5, 7$.

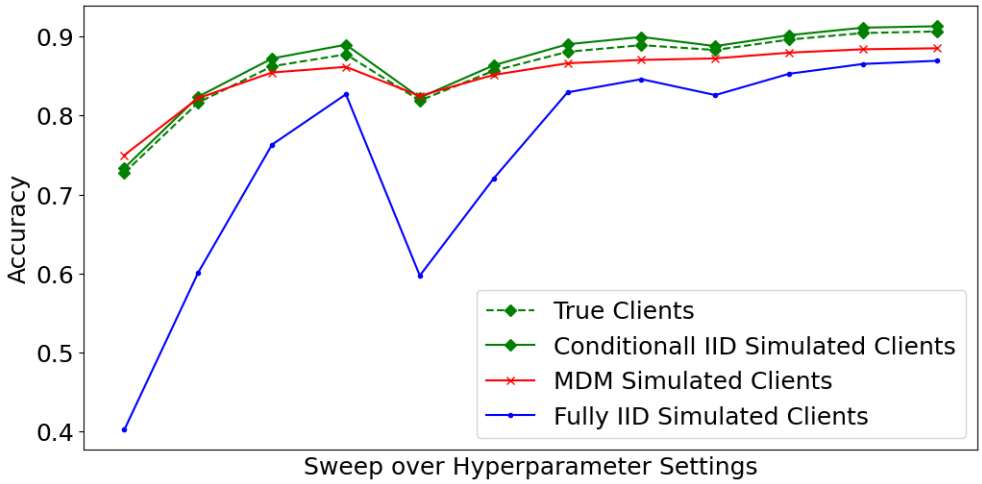

Figure A.8: FEMNIST test accuracy when training with FedAvg for different settings of local learning rate and local epochs. True clients (dotted green), conditionally IID simulated clients (green), learned MDM simulated clients (red) and fully IID simulated clients (blue).

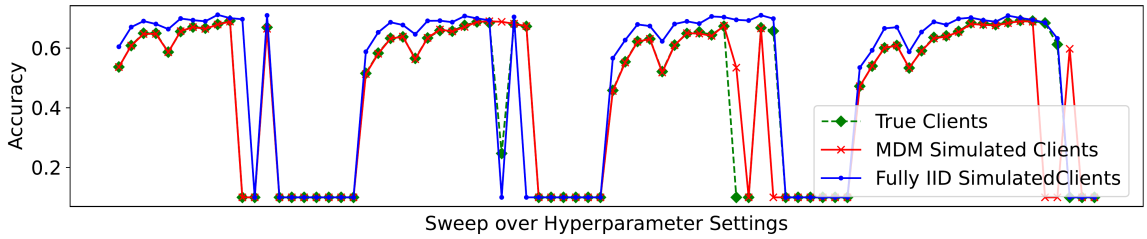

Figure A.9: CIFAR10 test accuracy when training with FedAvg for different settings of local batch size, local learning rate and local epochs. True clients (dotted green), learned MDM simulated clients (red) and fully IID simulated clients (blue). Ground truth: Low Heterogeneity and 1 mixture component.

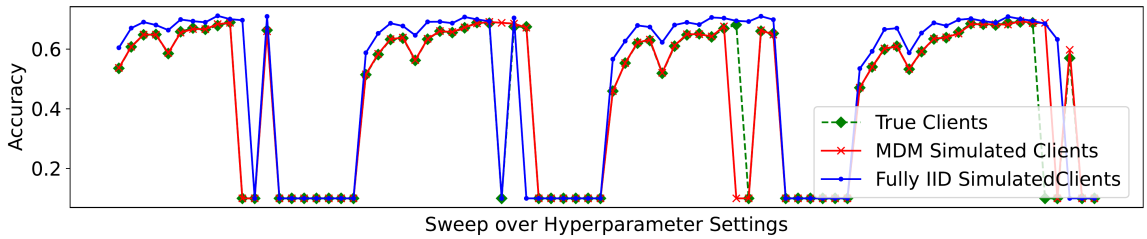

Figure A.10: CIFAR10 test accuracy when training with FedAvg for different settings of local batch size, local learning rate and local epochs. True clients (dotted green), learned MDM simulated clients (red) and fully IID simulated clients (blue). Ground truth: Low Heterogeneity and 2 mixture component.

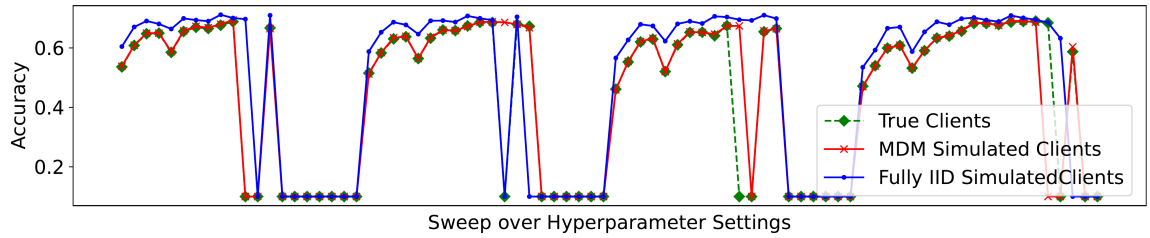

Figure A.11: CIFAR10 test accuracy when training with FedAvg for different settings of local batch size, local learning rate and local epochs. True clients (dotted green), learned MDM simulated clients (red) and fully IID simulated clients (blue). Ground truth: Low Heterogeneity and 3 mixture component.

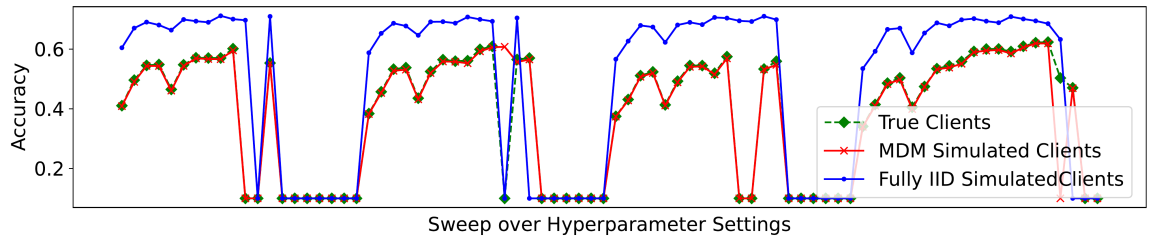

Figure A.12: CIFAR10 test accuracy when training with FedAvg for different settings of local batch size, local learning rate and local epochs. True clients (dotted green), learned MDM simulated clients (red) and fully IID simulated clients (blue). Ground truth: Medium Heterogeneity and 1 mixture component.

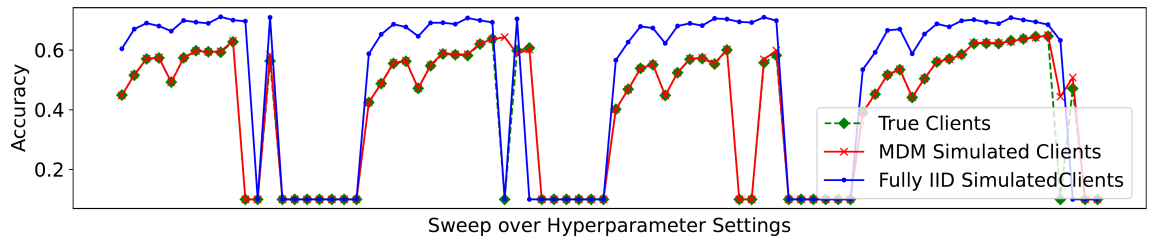

Figure A.13: CIFAR10 test accuracy when training with FedAvg for different settings of local batch size, local learning rate and local epochs. True clients (dotted green), learned MDM simulated clients (red) and fully IID simulated clients (blue). Ground truth: Medium Heterogeneity and 2 mixture component.

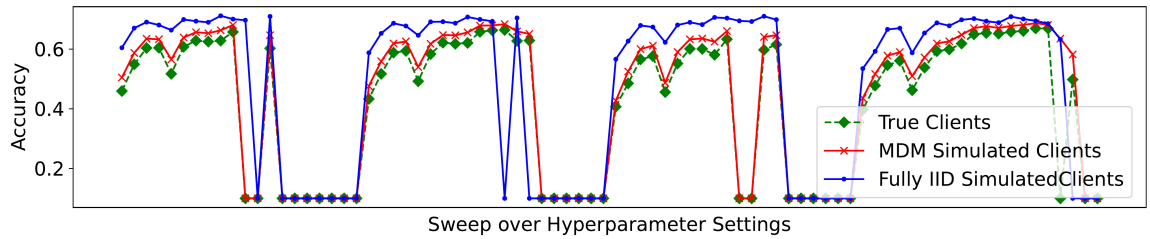

Figure A.14: CIFAR10 test accuracy when training with FedAvg for different settings of local batch size, local learning rate and local epochs. True clients (dotted green), learned MDM simulated clients (red) and fully IID simulated clients (blue). Ground truth: Medium Heterogeneity and 3 mixture component.

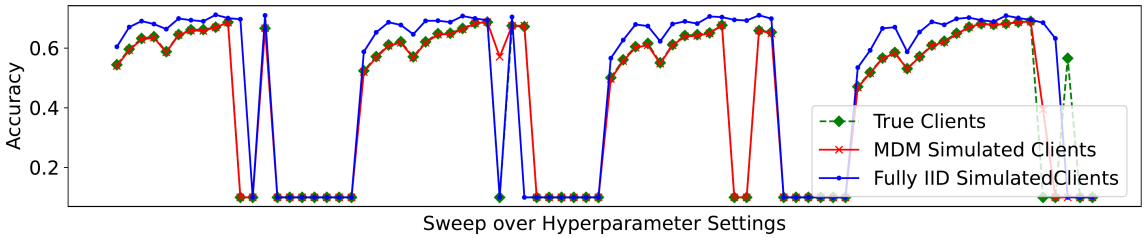

Figure A.15: CIFAR10 test accuracy when training with FedAvg for different settings of local batch size, local learning rate and local epochs. True clients (dotted green), learned MDM simulated clients (red) and fully IID simulated clients (blue). Ground truth: High Heterogeneity and 1 mixture component.

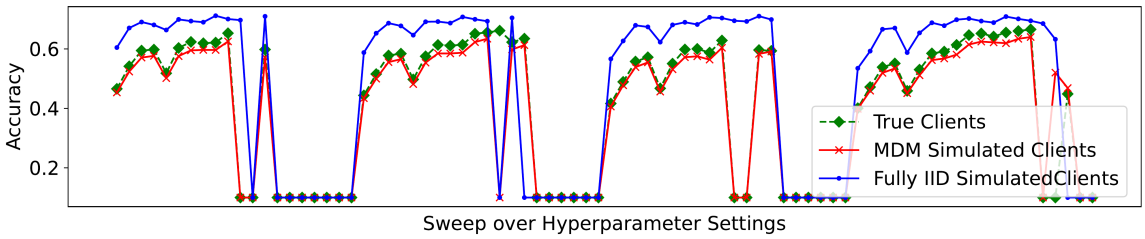

Figure A.16: CIFAR10 test accuracy when training with FedAvg for different settings of local batch size, local learning rate and local epochs. True clients (dotted green), learned MDM simulated clients (red) and fully IID simulated clients (blue). Ground truth: High Heterogeneity and 3 mixture component.

APPENDIX B

Appendix for Chapter 4

B.1 Experiments

B.1.1 Model Architectures

We used a LeNet-based architecture for the personalized model on each client. The exact architecture is shown in Table B.1. After each convolutional and fully connected layer we apply a ReLU non-linearity except for the final layer which uses a softmax for input into the cross-entropy loss. The embedding network uses the same architecture except that the final layer has dimension $84 \times l$ and we do not apply a non-linearity (Table B.2). For the hypernetwork we use a fully connected network. The exact architecture is shown in Table B.3. Again we apply a ReLU after each layer except the last one, which does not have a non-linearity and simply has output of size D, where D denotes the number of parameters in the client personalized model.

Layer	Shape	Nonlinearity
Conv1	$3 \times 5 \times 5 \times 16$	ReLU
MaxPool	2×2	–
Conv2	$16 \times 5 \times 5 \times 32$	ReLU
MaxPool	2×2	flatten
FC1	800×120	ReLU
FC2	120×84	ReLU
FC3	$84 \times C$	softmax

Table B.1: LeNet-based ConvNet of personalized client models. Convolutions are without padding. C is the number of classes.

Layer	Shape	Nonlinearity
Conv1	$3 \times 5 \times 5 \times 16$	ReLU
MaxPool	2×2	–
Conv2	$16 \times 5 \times 5 \times 32$	ReLU
MaxPool	2×2	flatten
FC1	800×120	ReLU
FC2	120×84	ReLU
FC3	$84 \times l$	none

Table B.2: LeNet-based embedding network. Convolutions are without padding. l is the dimensionality of the client descriptor.

Layer	Shape	Nonlinearity
FC1	$l \times 100$	ReLU
FC2	100×100	ReLU
FC3	100×100	ReLU
FC4	100×100	ReLU
FC5	$100 \times D$	none

Table B.3: Fully connected hypernetwork. D is the dimensionality of the model to be created. Note that because of FC5 alone the hypernetwork is at least 100 bigger than the client models.

B.1.2 Hyperparameter Settings

For PeFLL, each client uses a single batch of data (batch size 32) as input to the embedding network. For the hypernetwork parameter settings we use the recomended values given in [SNFC21] as we found these to work well in our setting as well. Specifically, the dimension of the embedding vectors is $l = \frac{n}{4}$ and the number of client SGD steps is $k = 50$. The regularization parameters for the embedding network and hypernetwork are set to $\lambda_h = \lambda_v = 10^{-3}$, while the output regularization is $\lambda_\theta = 0$.

For Per-FedAvg we used the first order (FO) approximation detailed in [FMO20a], and following the recomendations we set the number of client local steps is set to be a single epoch. For FedRep we set the number of client updates to the head and body to 5 and 1 epochs respectively as recommended in [CHMS21]. To infer on a new client we we randomly initialized a head and trained only the head locally for 20 epochs. For pFedMe we follow the recomendations in [DTN20] and set the regularization parameter to $\lambda = 15$, the computation parameter to $K = 3$ and $\beta = 1$. To infer for a new client we first initialize the clients personalized model to the global model and finetuned this for 20 epochs using the pFedMe loss function. For kNN-Per, for the scale parameter in the kNN classifier we searched over the values recommended in [MNVK22], namely $\{0.1, 1, 10, 100\}$, we found that setting the scale parameter to 100 to worked best. The k parameter for the kNN search and the weight parameter for how much to weight the client local vs global model were chosen individually on each client by cross-validation searching over $\{5, 10\}$ and $\{0.0, 0.1, 0.2, \ldots, 1.0\}$ respectively. For pFedHN we set the hyperparameters following the recommendations in [SNFC21]. For inferring on a new client we followed the procedure given in [SNFC21] and we randomly initialized a new embedding vector and optimized only this using 20 rounds of client server communication.

B.1.3 Additional Experiments

Comparison of communication and computation cost with FedAvg We compare the performance of PeFLL to FedAvg with respect to communication costs of training. At each global round, the communication cost of PeFLL between the server and c clients is $2c(\|\theta\| + \|\eta_v\| + \|v\|)$ while for FedAvg is $2c\|\theta\|$. In our setting, the embedding network and client models are of comparable size and the size of client descriptors (v) is negligibly small compared to them. Therefore, the per round communication cost of PeFLL is about twice that of FedAvg. In Figures B.5–B.11 we plot the accuracies during training for a given communication cost (measured in GB of information transferred between server and clients) as well computational cost for the clients (measured in number of global rounds). As shown in the figures, for any given communication or computational budget PeFLL outperforms FedAvg on these datasets.

Shakespeare dataset In order to test the effectiveness of PeFLL in non-image tasks, we also provide results for the Shakespeare dataset [CWL$^+$18], which is a next character prediction task. We use the LSTM model from [CWL$^+$18] as the client model, and as the embedding network. The hypernetwork is the same as in all other experiments. We compare the performance in comparison to a number of baselines. Figure B.2 shows the comparison of the accuracies for train and test clients during training with respect to the number of global rounds. PeFLL outperforms all baselines when comparing performance with respect to the number of global rounds. We additionally compare the performance of PeFLL with FedAvg with respect to accuracy for a given communication cost, see Figure B.3. We observe similar performances for a fixed communication budget.

ResNet Experiments In order to test the effectiveness of PeFLL for more modern architectures, we also performed experiments using a ResNet20 model[1] for CIFAR10 dataset. We compared the performance of PeFLL with FedAvg, and the results are shown in Figure B.4 for 100, 500, and 1000 clients, respectively. These figures show PeFLL outperforms FedAvg also for this model, and support the feasibility of generating modern models as part of our pipeline.

Analysis of Client Extrapolation For all experiments so far, clients at training time and new clients were related in the sense that they come from the same client environment. In this section, we examine how well PeFLL is able to generalize beyond this, by studying its *extrapolation* performance, where the new clients come from a different client environment than the train clients. We follow the same procedure as in the previous section to simulate heterogeneous clients and use sampled Dirichlet class proportions to assign classes. For the train clients we again use $\boldsymbol{\alpha}_{\text{train}} = (0.1, \dots, 0.1)$ to generate each clients class proportion vector. However, for the new clients we use a different Dirichlet parameter $\boldsymbol{\alpha}_{\text{new}} = (\alpha, \dots, \alpha)$. For each $\alpha \in \{0.1, 0.2, 0.3, \dots, 1.0\}$ we generate a group of new clients using this parameter. We run PeFLL on the train clients then use the trained embedding and hypernetworks to generate a model for each of the new clients.

[1] We use the implementation from `https://github.com/chenyaofo/pytorch-cifar-models`, except without the batch normalization layers.

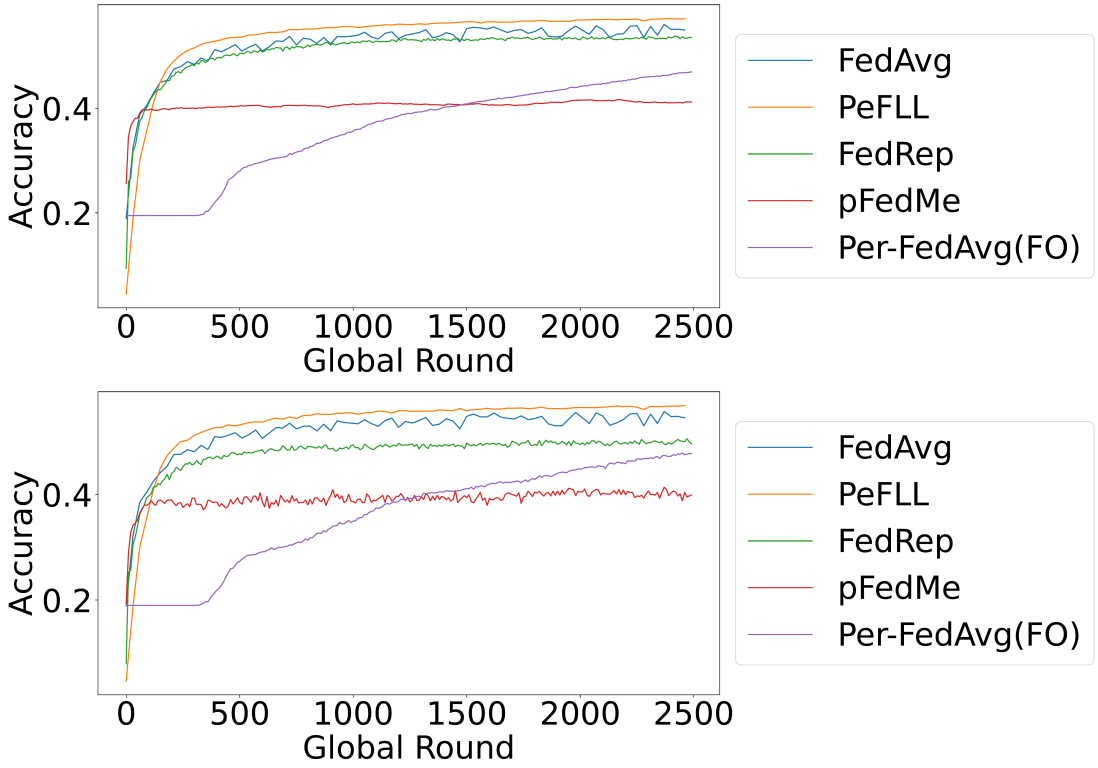

Figure B.2: Test Accuracies for *train* clients (top row) and *test* clients (bottom row) during different steps of the training for PeFLL and baselines, for the Shakespeare dataset.

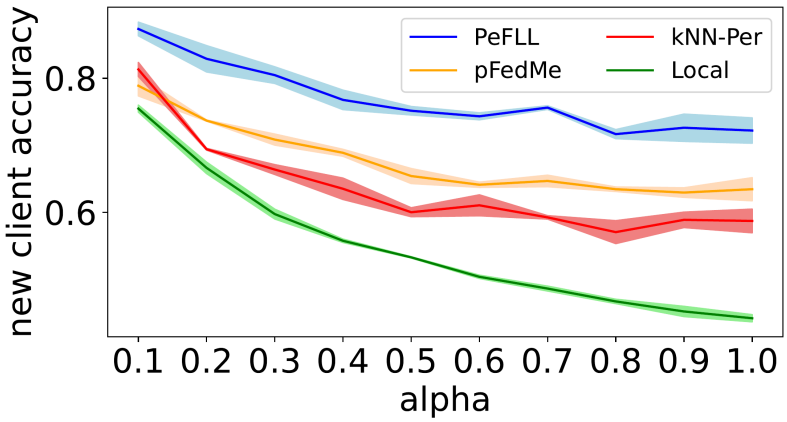

Figure B.1: Accuracy in *client extrapolation*. Larger values of α indicate new clients that are more dissimilar to the train clients.

Figure B.1 shows the resulting accuracy values for PeFLL and the best performing baselines of Table 4.1, as well the baseline of purely local per-client training. Note that as α increases so does the difficulty of the client's problem, as illustrated by the fact that the accuracy of purely local training decreases. Despite this increased task difficulty and distributional difference PeFLL still obtains strong results. Even at $\alpha = 1$, PeFLL produces models that are more accurate than those learned by the other methods for smaller values of α, and far superior to purely local training on the new clients.

Integrating known client label sets In the vanilla setting of Section 3.4, only the client training data is used to create a model. By design, each model was a multi-class classifier

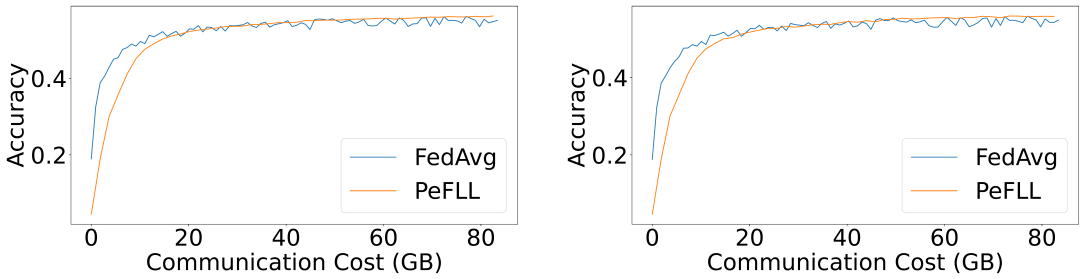

Figure B.3: Test Accuracies for *train* clients (left) and *test* clients (right) over the course of training measured with respect to total communication cost for PeFLL and FedAvg, for the Shakespeare dataset.

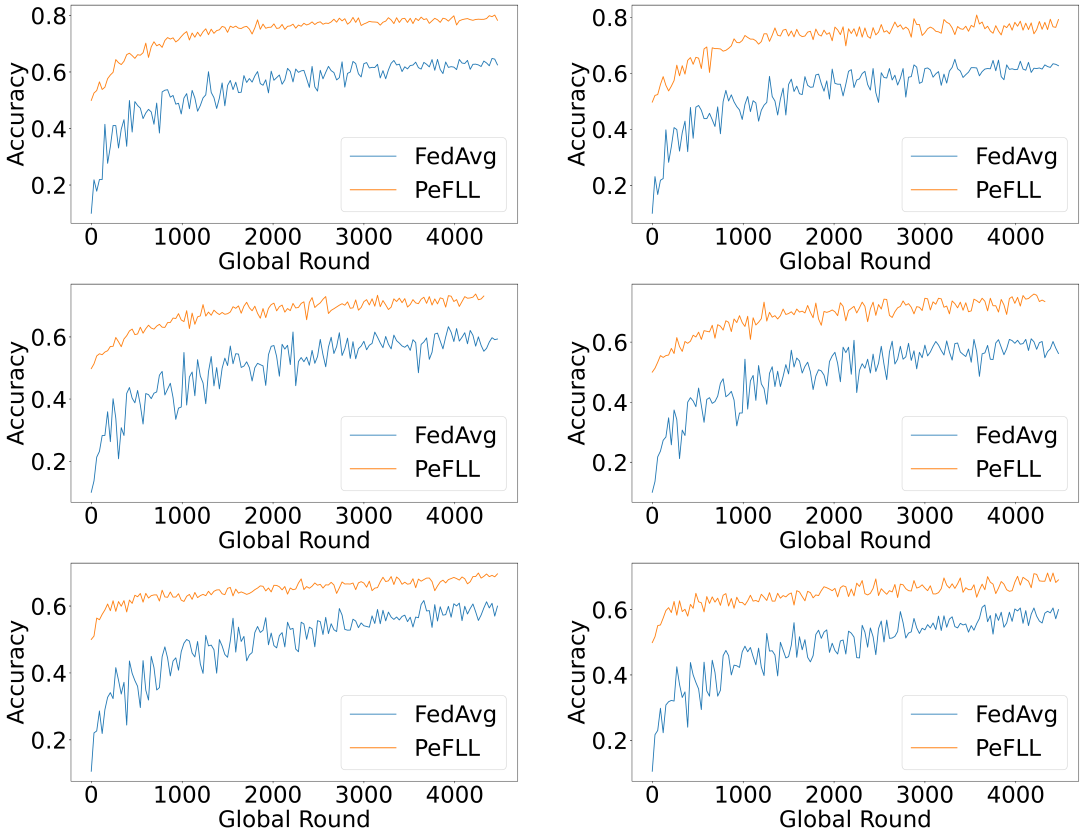

Figure B.4: Test Accuracies for *train* clients (left) and *test* clients (right) during different steps of the training for PeFLL and FedAvg for the CIFAR10 dataset with 100 clients (top row), 500 clients (middle row) and 1000 clients (bottom row). PeFLL consistently outperforms FedAvg.

across all possible classes, even if only a subset of those are relevant for the client. However, it is also reasonable to assume that clients might have prior knowledge which of the output classes they actually want to predict. This information can be included by *label masking (LM)*, in which the client overwrites the output probabilities of classes that cannot occur for it by 0, such that they will never be predicted. We report results for this in Table B.4. One can see that this has minor effects for CIFAR10, but the results for CIFAR100 are clear improved. We attribute this to the fact that the larger label set and the smaller number of examples per class increase the chances of spurious classes being predicted.

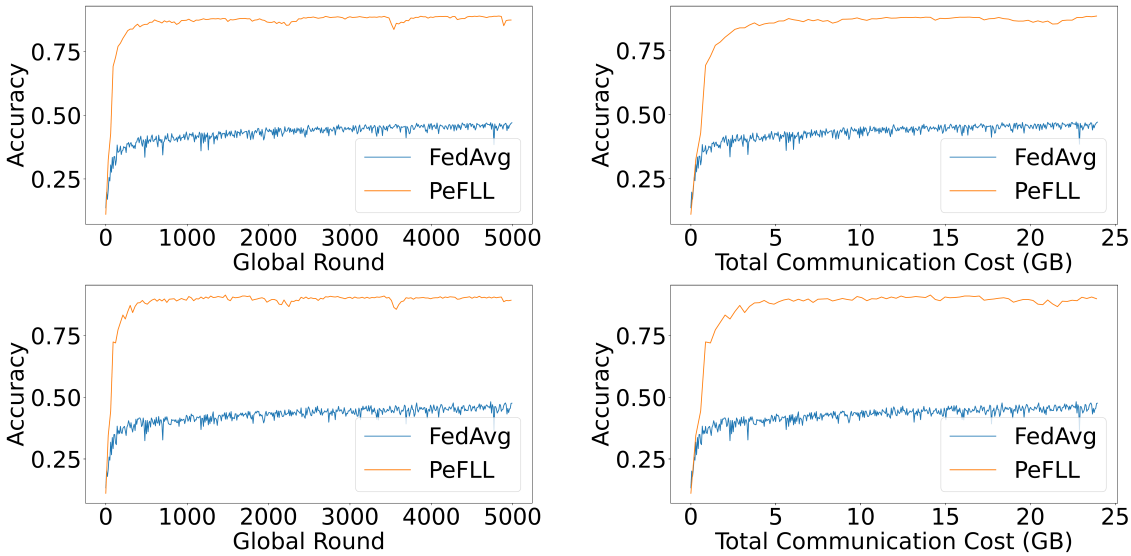

Figure B.5: Test Accuracies for *train* clients (top) and *test* clients (bottom) during different steps of the training for PeFLL and FedAvg, for the CIFAR10 dataset, 100 clients. PeFLL's accuracy is higher than FedAvg's with respect to the number of rounds (which roughly reflect the computational cost for the clients) as well as the communication cost.

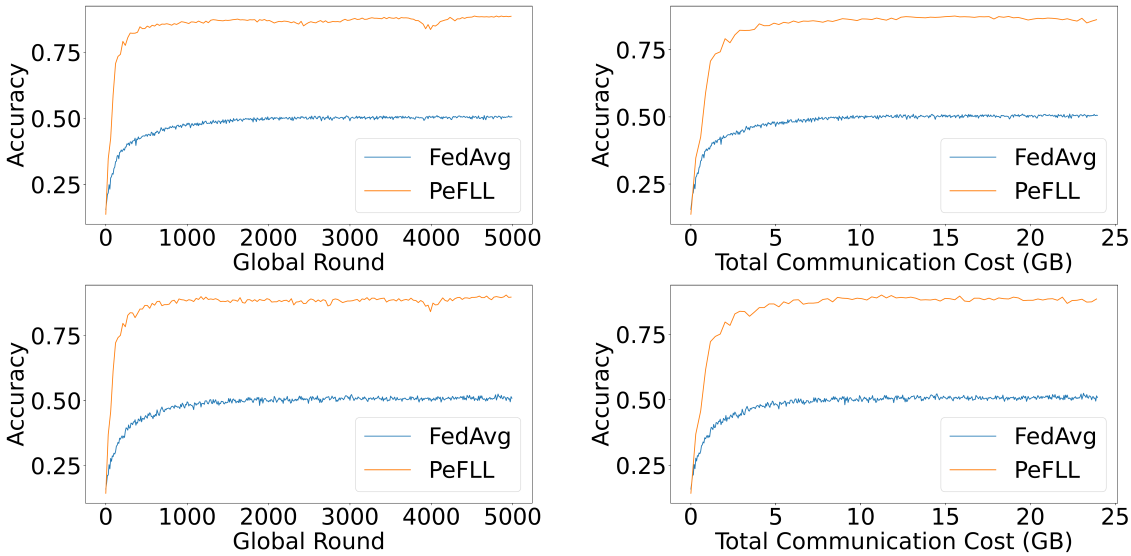

Figure B.6: Test Accuracies for *train* clients (top) and *test* clients (bottom) during different steps of the training for PeFLL and FedAvg, for the CIFAR10 dataset, 500 clients. PeFLL's accuracy is higher than FedAvg's with respect to the number of rounds (which roughly reflect the computational cost for the clients) as well as the communication cost.

B.1.4 Additional figures

We include additional figures for the correlation between the distances between client descriptors and the true client distributional distances as described in Section 3.4. In Figures B.12 - B.14 we show the results of the experiment for different numbers of clients, 100, 500 and 1000. As we can see in all cases high correlation is achieved. With more clients we obtain higher levels of correlation and less variability in the results.

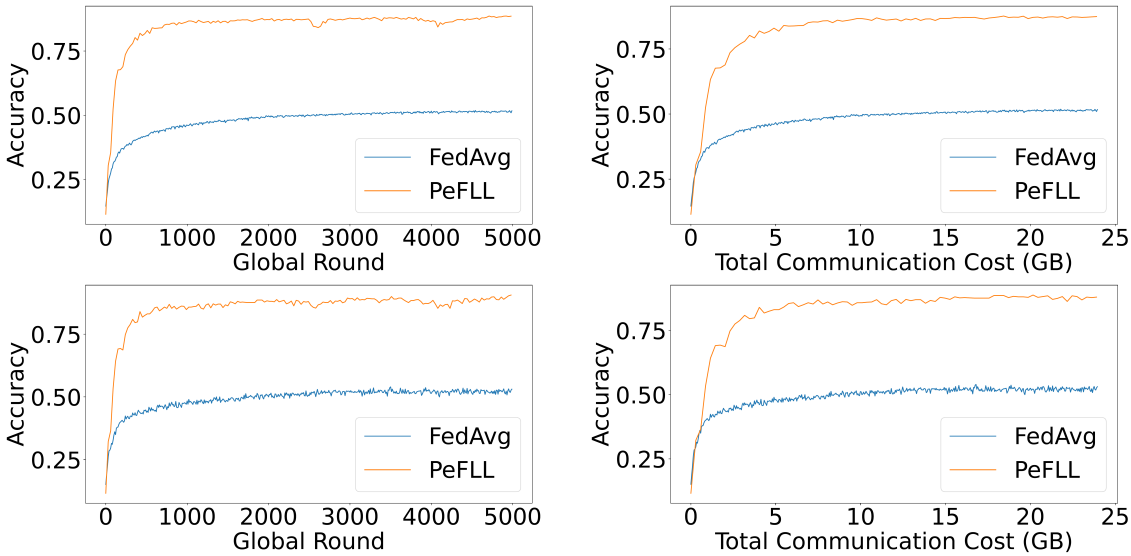

Figure B.7: Test Accuracies for *train* clients (top) and *test* clients (bottom) during different steps of the training for PeFLL and FedAvg, for the CIFAR10 dataset, 1000 clients. PeFLL's accuracy is higher than FedAvg's with respect to the number of rounds (which roughly reflect the computational cost for the clients) as well as the communication cost.

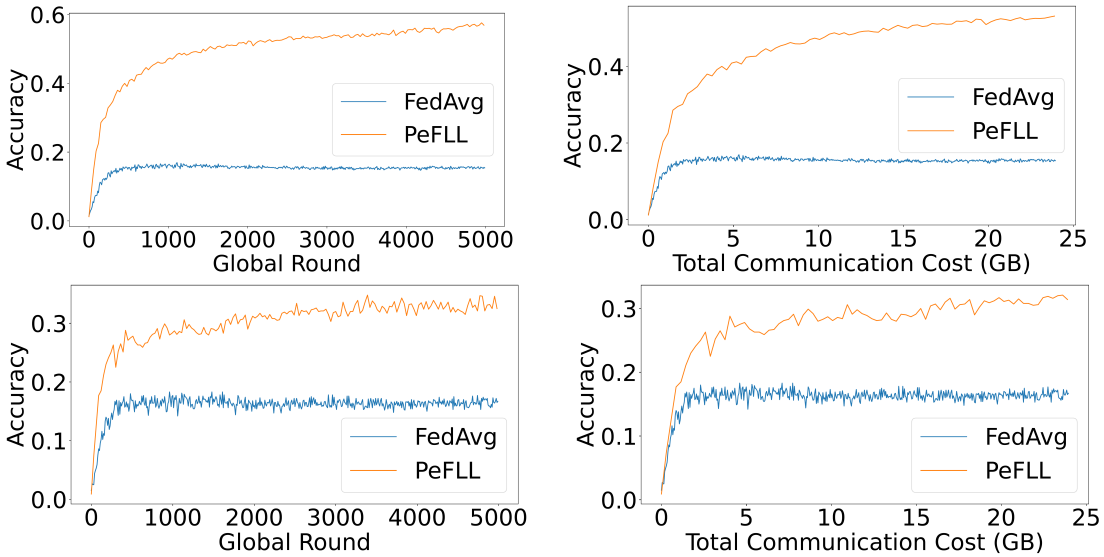

Figure B.8: Test Accuracies for *train* clients (top) and *test* clients (bottom) during different steps of the training for PeFLL and FedAvg, for the CIFAR100 dataset, 100 clients. PeFLL's accuracy is higher than FedAvg's with respect to the number of rounds (which roughly reflect the computational cost for the clients) as well as the communication cost.

B.1.5 Ablation Studies

We include several ablation studies to explore how different design choices impact PeFLL's performance.

Hypernetwork Size. We investigate how the size of the hypernetwork impacts performance. We keep the same hypernetwork architecture as described in Table B.3 but vary the number of hidden layers. We have a small network (S) with only a single hidden layer, a medium network (M) with 3 hidden layers and a large network (L) with 5 hidden layers. We evaluate

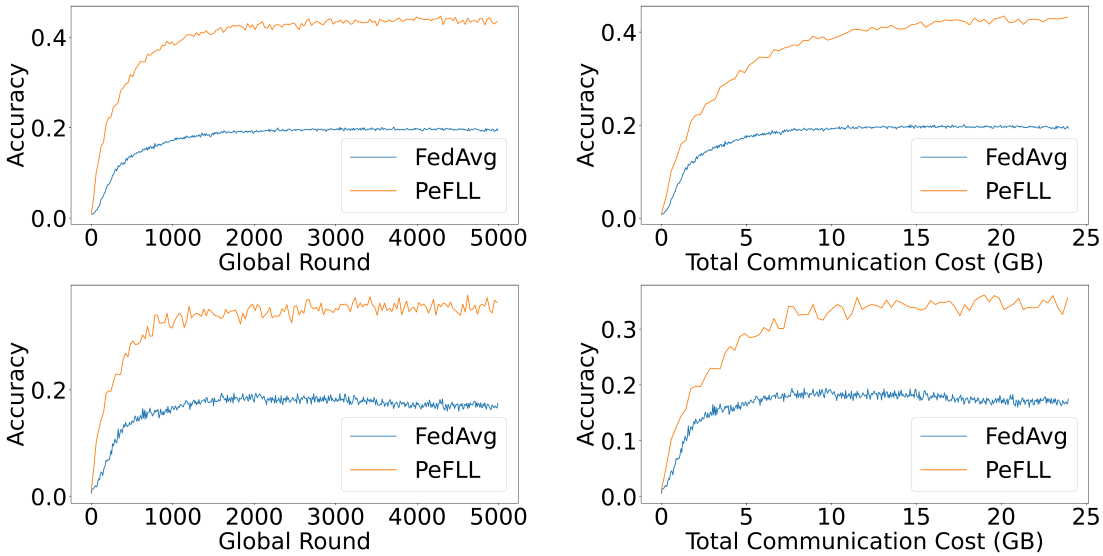

Figure B.9: Test Accuracies for *train* clients (top) and *test* clients (bottom) during different steps of the training for PeFLL and FedAvg, for the CIFAR100 dataset, 500 clients. PeFLL's accuracy is higher than FedAvg's with respect to the number of rounds (which roughly reflect the computational cost for the clients) as well as the communication cost.

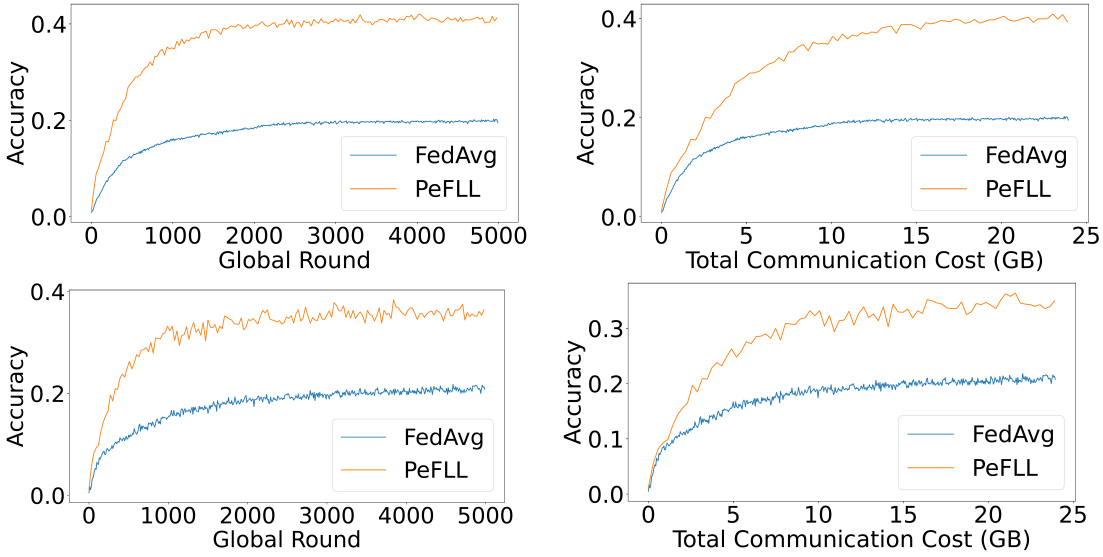

Figure B.10: Test Accuracies for *train* clients (top) and *test* clients (bottom) during different steps of the training for PeFLL and FedAvg, for the CIFAR100 dataset, 1000 clients. PeFLL's accuracy is higher than FedAvg's with respect to the number of rounds (which roughly reflect the computational cost for the clients) as well as the communication cost.

on CIFAR10 and CIFAR100 using the same experimental setup as for previous experiments, with 100, 500 and 1000 clients. The results are shown in Table B.5. As we can see in the results, for CIFAR10 all 3 networks are reasonably similar with a slight drop in performance observed for the S network. For CIFAR100 it is more pronounced, with a modest increase in performance for the L network and a noticeable drop for the S network.

Regularization Parameters. We examine the impact of varying the regularization parameters of the objective (4.1) on the performance of PeFLL. We consider different values for

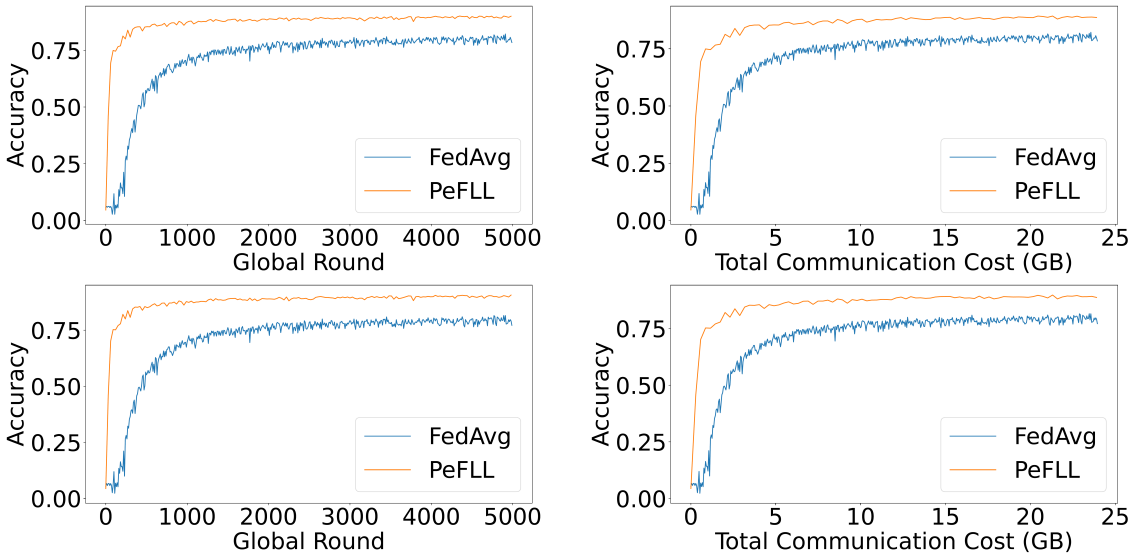

Figure B.11: Test Accuracies for *train* clients (top) and *test* clients (bottom) during different steps of the training for PeFLL and FedAvg, for the FEMNIST dataset. PeFLL's accuracy is higher than FedAvg's with respect to the number of rounds (which roughly reflect the computational cost for the clients) as well as the communication cost.

	CIFAR10			CIFAR100		
#trn.clients	$n = 90$	450	900	90	450	900
PeFLL	89.0 ± 0.8	88.9 ± 0.4	87.8 ± 0.4	56.0 ± 0.3	43.2 ± 0.8	40.9 ± 0.6
PeFLL + LM	88.9 ± 0.7	88.9 ± 0.3	88.4 ± 0.4	59.5 ± 0.7	53.2 ± 0.6	51.4 ± 0.3
	CIFAR10			CIFAR100		
#trn.clients	$n = 90$	450	900	90	450	900
PeFLL	89.3 ± 2.2	88.9 ± 0.6	88.5 ± 0.3	35.2 ± 1.4	38.1 ± 0.4	38.4 ± 0.9
PeFLL + LM	88.9 ± 2.4	88.2 ± 0.9	87.4 ± 1.3	55.6 ± 1.6	53.4 ± 0.1	51.1 ± 0.2

Table B.4: Test accuracy on CIFAR10 and CIFAR100 for training clients (top) and test clients (bottom) without and with label masking (LM).

	CIFAR10			CIFAR100		
#Clients	100	500	1000	100	500	1000
S	88.7 ± 0.3	88.2 ± 0.4	87.7 ± 0.4	57.9 ± 0.3	51.9 ± 0.6	47.4 ± 1.4
M	88.9 ± 0.7	88.9 ± 0.3	88.1 ± 0.3	59.4 ± 0.2	53.1 ± 0.4	50.8 ± 0.5
L	88.7 ± 0.8	88.8 ± 0.3	88.2 ± 0.6	59.8 ± 0.1	53.8 ± 0.2	52.6 ± 1.0

Table B.5: Test accuracy on CIFAR10 and CIFAR100 using different sizes of hypernetworks, small (S), medium (M) and large (L).

the regularization parameters and study the values in $\{0, 5 \times 10^{-5}, 5 \times 10^{-3}, 5 \times 10^{-1}\}$ for client regularization parameter, and values in $\{0, 5 \times 10^{-5}, 5 \times 10^{-3}\}$ for embedding and hypernetwork regularization. The results are shown in Table B.6. As one can see, PeFLL's performance is rather robust over the regularization strengthh. However, consistent with the theory, modest amounts of hypernetwork regularization parameter help preventing overfitting on training clients and improve client-level generalization. Similarly, modest amounts of client regularization improve sample-level generalization. As can be expected, very large values of this parameter leads to underfitting.

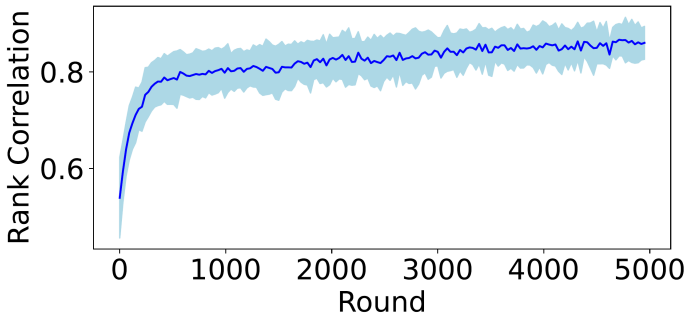

Figure B.12: Correlation between *client descriptor similarity* obtained from the embedding network and *ground truth similarity* over the course of training. CIFAR10, 100 Clients.

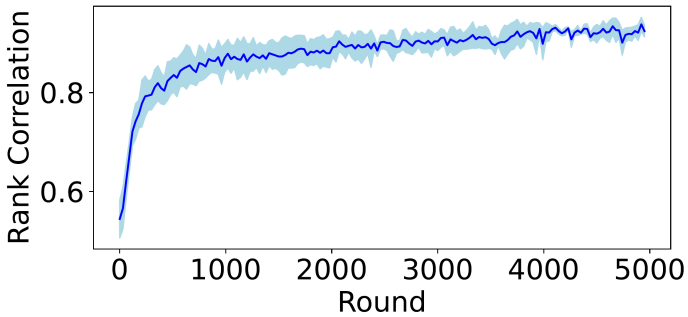

Figure B.13: Correlation between *client descriptor similarity* obtained from the embedding network and *ground truth similarity* over the course of training. CIFAR10, 500 clients.

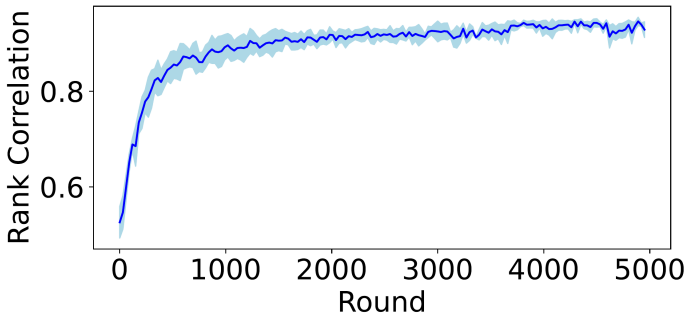

Figure B.14: Correlation between *client descriptor similarity* obtained from the embedding network and *ground truth similarity* over the course of training. CIFAR10, 1000 clients.

Embedding network architecture. We also study the effect of different architecture choices for the embedding network. We compare the performance of a single-layer MLP network layer and a LeNet-style ConvNet for CIFAR10 and CIFAR100. As we can see in Table B.7 the CNN embedding outperforms the MLP in all experiments.

	$\lambda_h = \lambda_v$		
λ_θ	0	10^{-5}	10^{-3}
0	87.4 ± 1.1	87.4 ± 1.0	88.4 ± 0.6
$5 \cdot 10^{-5}$	88.7 ± 0.7	88.7 ± 0.6	88.8 ± 0.3
$5 \cdot 10^{-3}$	87.8 ± 0.8	87.6 ± 0.8	87.5 ± 0.5
$5 \cdot 10^{-1}$	82.5 ± 1.5	83.8 ± 1.1	86.2 ± 0.4

Table B.6: Test accuracy on CIFAR10 with 100 clients for different settings of the regularization parameters. On the left is the regularization of the client network, and along the top is the regularization of the hypernetwork and embedding network.

	CIFAR10			CIFAR100		
#Clients	100	500	1000	100	500	1000
MLP	88.5 ± 1.2	87.8 ± 1.1	87.5 ± 2.1	59.1 ± 0.2	53.4 ± 0.4	51.2 ± 0.5
CNN	88.9 ± 0.7	88.9 ± 0.3	88.4 ± 0.4	59.4 ± 0.2	54.4 ± 0.3	53.2 ± 0.3

Table B.7: Test accuracy on CIFAR10 and CIFAR100 using different embedding network architectures. MLP is a single hidden layer fully connected network and CNN is a LeNet-style ConvNet.

B.2 Generalization

In this section we prove a new generalization bound for the generic learning-to-learn setting that, in particular, will imply Theorem 3 as a special case. Before formulating and proving our main results, we remind the reader of the PAC-Bayesian learning framework [McA98] and its use for obtaining guarantees for learning-to-learn.

PAC-Bayesian learning and learning-to-learn In standard PAC-Bayesian learning, we are given a set of possible models, $\mathcal{H}$, and a prior distribution over these $P \in \mathcal{M}(\mathcal{H})$, where $\mathcal{M}(\cdot)$ denotes the set of probability measures over a base set. Given a dataset, S, *learning a model* means constructing a new (posterior) distribution, $Q \in \mathcal{M}(\mathcal{H})$, which is meant to give high probability to models with small loss. The posterior distribution, Q, induces a stochastic predictor: for any input, one samples a specific model, $f \sim Q$, and outputs the result of this model applied to the input. Note that Q can in principle be a Dirac delta distribution at a single model, resulting in a deterministic predictor. However, for large (uncountably infinite) model such a choice typically does not lead to strong generalization guarantees. For conciseness of notation, in the following we do not distinguish between the distributions over models and their stochastic predictors.

The quality of a stochastic predictor, Q, on a data point (x, y) is quantified by its expected loss, $\ell_{(x,y)}(Q) = \mathbb{E}_{f\sim Q}\, \ell(x, y, f)$. From this, we define the empirical error on a dataset, S, as $\frac{1}{|S|}\sum_{(x,y)\in S} \ell_{(x,y)}(Q)$, and its expected loss with respect to a data distribution, D, as $\mathbb{E}_{(x,y)\sim D}\, \ell_{(x,y)}(Q)$. Ordinary PAC-Bayesian generalization bounds provide high-probability upper bounds to the expected loss of a stochastic predictors by the corresponding empirical loss as well as some complexity terms, which typically include the Kullback-Leibler divergence between the chosen posterior distribution and the original (training-data independent) prior, $\mathbf{KL}(Q||P)$ [McA98].

Typically, the posterior distribution is not chosen arbitrarily, but it is the result of a *learning algorithm*, $A : (\mathcal{X} \times \mathcal{Y})^m \to \mathcal{M}(\mathcal{H})$, which takes as input the training data and, potentially, the prior distribution. The idea of *learning-to-learn* (also called *meta-learning* or *lifelong learning* in the literature) is to *learn the learning algorithm* [Bax00, PL15].

To study this theoretically, we adopt the setting where n learning tasks is available, which we write as tuples, (S_i, D_i), for $i \in \{1, \ldots, n\}$, each with a data set $S_i \subset \mathcal{X} \times \mathcal{Y}$ that is sampled from a corresponding data distribution $D_i \in \mathcal{M}(\mathcal{X} \times \mathcal{Y})$. For simplicity, we assume that all datasets are of the same size, m. We assume tasks are sampled i.i.d. from a *task environment*, $\mathcal{T}$, which is simply a data distribution over such tuples.

Again adopting the PAC-Bayesian framework, we assume that a data-independent (meta-)prior distribution over learning algorithms is available, $\mathcal{P} \in \mathcal{M}(\mathcal{A})$, where $\mathcal{A}$ is the set of possible algorithms, and the goal is use the observed task data to construct a (meta-)posterior distribution, $\mathcal{Q} \in \mathcal{M}(\mathcal{A})$. As before, the resulting procedure is stochastic: at every invocation, the system samples an algorithm $A \sim \mathcal{Q}$. It applies this to the training data, obtaining a posterior distribution $A(S)$, and it makes predictions by sampling models accordingly, $f \sim A(S)$.

Analogously to above situation, we define two measures of quality for such a stochastic

algorithms. Its *empirical loss on the data of the observed clients*:

$$\widehat{\text{er}}(\mathcal{Q}) := \frac{1}{n}\sum_{i=1}^{n} \mathop{\mathbb{E}}_{A\sim\mathcal{Q}} \frac{1}{m}\sum_{j=1}^{m} \ell(x_j^i, y_j^i, A(S_i)), \tag{B.1}$$

and its *expected loss on future clients,*

$$\text{er}(\mathcal{Q}) := \mathop{\mathbb{E}}_{(D,S)\sim\mathcal{T}} \mathop{\mathbb{E}}_{A\sim\mathcal{Q}} \mathop{\mathbb{E}}_{(x,y)\sim D} \ell(x, y, A(S)). \tag{B.2}$$

The following theorem provides a connection between both quantities.

Theorem 8. *Let $\mathcal{P} \in \mathcal{M}(\mathcal{A})$ and $P \in \mathcal{M}(\mathcal{H})$ be a meta-prior and a prior distribution, respectively, which are chosen independently of the observed training data, $S_1, \ldots, S_n$. Assume that the loss function is bounded in $[0, M]$. Then, for all $\delta \geq 0$ it holds with probability at least $1-\delta$ over the sampling of the datasets, that for all distributions $\mathcal{Q} \in \mathcal{M}(\mathcal{A})$ over algorithms,*

$$\textit{er}(\mathcal{Q}) \leq \widehat{\textit{er}}(\mathcal{Q}) + M\sqrt{\frac{\mathbf{KL}(\mathcal{Q}||\mathcal{P}) + \log(\frac{4\sqrt{n}}{\delta})}{2n}} + M\mathop{\mathbb{E}}_{A\sim\mathcal{Q}}\sqrt{\frac{\sum_{i=1}^{n}\mathbf{KL}(A(S_i)||P) + \log(\frac{8mn}{\delta}) + 1}{2mn}} \tag{B.3}$$

where $\mathbf{KL}$ denotes the Kullback-Leibler divergence.

Proof. The proof resembles previous proofs for PAC-Bayesian learning-to-learn learning, but it differs in some relevant steps. We discuss this in a *Discussion* section after the proof itself.

First, we observe that (B.3) depends linearly in M. Therefore, if suffices to prove the case of $M = 1$, and the general case follows by applying the theorem to ℓ/M with subsequent rescaling. Next, we define the *expected loss for the n training clients*, $\widetilde{\text{er}}(\mathcal{Q})$, as an intermediate object:

$$\widetilde{\text{er}}(\mathcal{Q}) = \frac{1}{n}\sum_{i=1}^{n} \mathop{\mathbb{E}}_{A\sim\mathcal{Q}} \mathop{\mathbb{E}}_{(x,y)\sim D_i} \ell(x, y, A(S_i)). \tag{B.4}$$

To bound $\text{er}(\mathcal{Q}) - \widehat{\text{er}}(\mathcal{Q})$, we divide it in two parts: $\text{er}(\mathcal{Q}) - \widetilde{\text{er}}(\mathcal{Q})$ and $\widetilde{\text{er}}(\mathcal{Q}) - \widehat{\text{er}}(\mathcal{Q})$. We then bound each part separately and combine the results.

Part I For the former, for any $h \in \mathcal{H}$ let $\Delta_i(h) = \frac{1}{m}\sum_{j=1}^{m} \ell(x_j^i, y_j^i, h) - \mathbb{E}_{(x,y)\sim D_i}\, \ell(x, y, h)$, and $\Delta_i(Q) = \mathbb{E}_{h\in Q}\, \Delta_i(h)$, such that $\widetilde{\text{er}}(\mathcal{Q}) - \widehat{\text{er}}(\mathcal{Q}) = \mathbb{E}_{A\sim\mathcal{Q}}\, \frac{1}{n}\sum_{i=1}^{n} \Delta_i(A(S_i))$. Then, for any $\lambda > 0$,

$$\widetilde{\text{er}}(\mathcal{Q}) - \widehat{\text{er}}(\mathcal{Q}) - \frac{1}{\lambda}\mathop{\mathbb{E}}_{A\sim\mathcal{Q}} \mathbf{KL}(A(S_1) \times \cdots \times A(S_n)||P \times \cdots \times P) \tag{B.5}$$

$$\leq \sup_{Q_1,\ldots,Q_n} \frac{1}{n}\sum_{i=1}^{n} \Delta_i(Q_i) - \frac{1}{\lambda}\mathbf{KL}(Q_1 \times \cdots \times Q_n||P \times \ldots \times P) \tag{B.6}$$

$$\leq \frac{1}{\lambda}\log \mathop{\mathbb{E}}_{h_1\sim P} \cdots \mathop{\mathbb{E}}_{h_n\sim P} \prod_{i=1}^{n} e^{\frac{\lambda}{n}\Delta_i(h_i)}, \tag{B.7}$$

where the second inequality is due to the *change of measure inequality* [SLCB+12].

Because for each $i = 1, \ldots, n$, P is independent of $S_i, ..., S_n$, we have

$$\mathop{\mathbb{E}}_{S_1,...,S_n} \mathop{\mathbb{E}}_{h_1 \sim P} \cdots \mathop{\mathbb{E}}_{h_n \sim P} \prod_{i=1}^{n} e^{\frac{\lambda}{n}\Delta_i(h_i)} = \mathop{\mathbb{E}}_{S_1} \mathop{\mathbb{E}}_{h_1 \sim P} e^{\frac{\lambda}{n}\Delta_1(h_1)} \cdots \mathop{\mathbb{E}}_{S_n} \mathop{\mathbb{E}}_{h_n \sim P} e^{\frac{\lambda}{n}\Delta_n(h_n)} \tag{B.8}$$

Each $\Delta_i(h_i)$ is a bounded random variable with support in an interval of size 1. By Hoeffding's lemma we have

$$\mathop{\mathbb{E}}_{S_i} \mathop{\mathbb{E}}_{h_i \sim P} e^{\frac{\lambda}{n}\Delta_i(h_i)} \leq e^{\frac{\lambda^2}{8n^2 m}}. \tag{B.9}$$

Therefore,

$$\mathop{\mathbb{E}}_{S_1,...,S_n} \mathop{\mathbb{E}}_{h_1 \sim P} \cdots \mathop{\mathbb{E}}_{h_n \sim P} \prod_{i=1}^{n} e^{\frac{\lambda}{n}\Delta_i(h_i)} \leq e^{\frac{\lambda^2}{8nm}}, \tag{B.10}$$

and by Markov's inequality, for any $\epsilon \in \mathbb{R}$,

$$\mathbb{P}_{S_1,...,S_n}\Big(\mathop{\mathbb{E}}_{h_1 \sim P} \cdots \mathop{\mathbb{E}}_{h_n \sim P} \prod_{i=1}^{n} e^{\frac{\lambda}{n}\Delta_i(h_i)} \geq e^{\epsilon} \Big) \leq e^{\frac{\lambda^2}{8nm} - \epsilon} \tag{B.11}$$

Hence

$$\mathbb{P}_{S_1,...,S_n}\Big(\exists Q_1, ..., Q_n : \frac{1}{n}\sum_{i=1}^{n} \Delta_i(Q_i) - \frac{1}{\lambda}\mathbf{KL}(Q_1 \times \cdots \times Q_n || P \times ... \times P) \geq \frac{1}{\lambda}\epsilon \Big) \leq e^{\frac{\lambda^2}{8nm} - \epsilon}, \tag{B.12}$$

or equivalently,

$$\mathbb{P}_{S_1,...,S_n}\Big(\forall Q_1, \ldots, Q_n : \frac{1}{n}\sum_{i=1}^{n} \Delta_i(Q_i) \tag{B.13}$$

$$\leq \frac{1}{\lambda}\sum_{i=1}^{n} \mathbf{KL}(Q_i||P) + \frac{1}{\lambda}\log(\frac{2}{\delta}) + \frac{\lambda}{8nm} \Big) \geq 1 - \frac{\delta}{2}. \tag{B.14}$$

By applying a union bound for all the values of $\lambda \in \Lambda$ with $\Lambda = \{1, \ldots, 4mn\}$ we get that

$$\mathbb{P}_{S_1,...,S_n}\Big(\forall Q_1, \ldots, Q_n, \forall \lambda \in \Lambda : \frac{1}{n}\sum_{i=1}^{n} \Delta_i(Q_i) \tag{B.15}$$

$$\leq \frac{1}{\lambda}\sum_{i=1}^{n} \mathbf{KL}(Q_i||P) + \frac{1}{\lambda}\log(\frac{8mn}{\delta}) + \frac{\lambda}{8nm} \Big) \geq 1 - \frac{\delta}{2} \tag{B.16}$$

Note that $\lfloor \lambda \rfloor \leq \lambda$ and $\frac{1}{\lfloor \lambda \rfloor} \leq \frac{1}{\lambda - 1}$. Therefore, for all values of $\lambda \in (1, 4mn)$ we have

$$\mathbb{P}_{S_1,...,S_n}\Big(\forall Q_1, \ldots, Q_n, \forall \lambda \in (1, 4mn) : \frac{1}{n}\sum_{i=1}^{n} \Delta_i(Q_i) \tag{B.17}$$

$$\leq \underbrace{\frac{1}{\lambda - 1}\Big(\sum_{i=1}^{n} \mathbf{KL}(Q_i||P) + \log(\frac{8mn}{\delta}) \Big) + \frac{\lambda}{8mn}}_{=:\Gamma(\lambda)} \Big) \geq 1 - \frac{\delta}{2} \tag{B.18}$$

For any choice of $Q_1, \dots, Q_n$, let $\lambda^* = \sqrt{8mn(\sum_{i=1}^n \mathbf{KL}(Q_i||P) + \log(\frac{8mn}{\delta})) + 1}$. If $\lambda^* \geq 4mn$, that implies $\sum_{i=1}^n (\mathbf{KL}(Q_i||P) + \log(\frac{8mn}{\delta}) + 1) \geq 2mn$ and $\Gamma(\lambda^*) > 1$, so Inequality (B.18) holds trivially. Otherwise, $\lambda^* \in (0, 4mn)$, so Inequality (B.18) also holds. Therefore, we

$$\mathbb{P}_{S_1,\dots,S_n}\Big(\forall Q_1, \dots, Q_n, : \frac{1}{n}\sum_{i=1}^n \Delta_i(Q_i) \leq \sqrt{\frac{\sum_{i=1}^n \mathbf{KL}(Q_i||P) + \log(\frac{8mn}{\delta}) + 1}{2mn}}\Big) \geq 1 - \frac{\delta}{2}. \tag{B.19}$$

In combination with (B.7), we obtain:

$$\mathbb{P}_{S_1,\dots,S_n}\Big(\forall \mathcal{Q} : \widetilde{\mathsf{er}}(\mathcal{Q}) - \widehat{\mathsf{er}}(\mathcal{Q}) \leq \mathop{\mathbb{E}}_{A\sim\mathcal{Q}} \sqrt{\frac{\sum_{i=1}^n \mathbf{KL}(A(S_i)||P) + \log(\frac{8mn}{\delta}) + 1}{2mn}}\Big) \geq 1 - \frac{\delta}{2}. \tag{B.20}$$

Part II For upper bounding $\mathsf{er}(\mathcal{Q}) - \widetilde{\mathsf{er}}(\mathcal{Q})$ we have by a standard PAC-Bayesian bound [Mau04, PORSTS21],

$$\mathbb{P}_{S_1,\dots,S_n}\Big(\forall \mathcal{Q} : \mathsf{er}(\mathcal{Q}) - \widetilde{\mathsf{er}}(\mathcal{Q}) \leq \sqrt{\frac{\mathbf{KL}(\mathcal{Q}||\mathcal{P}) + \log(\frac{4\sqrt{n}}{\delta})}{2n}}\Big) \geq 1 - \frac{\delta}{2}. \tag{B.21}$$

Combination We combine (B.20) and (B.21) by a union bound to obtain

$$\mathbb{P}_{S_1,\dots,S_n}\Big(\forall \mathcal{Q} : \mathsf{er}(\mathcal{Q}) - \widehat{\mathsf{er}}(\mathcal{Q}) \leq \sqrt{\frac{\mathbf{KL}(\mathcal{Q}||\mathcal{P}) + \log(\frac{4\sqrt{n}}{\delta})}{2n}} \tag{B.22}$$

$$+ \mathop{\mathbb{E}}_{A\sim\mathcal{Q}} \sqrt{\frac{\sum_{i=1}^n \mathbf{KL}(A(S_i)||P) + \log(\frac{8mn}{\delta}) + 1}{2mn}}\Big) \geq 1 - \delta. \tag{B.23}$$

This concludes the proof of the theorem. □

Discussion The difference in our proof and previous bounds in the similar settings [PL15, Rez22] is in the upper bound for $\widetilde{\mathsf{er}}(\mathcal{Q}) - \widehat{\mathsf{er}}(\mathcal{Q})$ which is the average of the generalization gaps for the n training tasks that we observe. In our case we prove a generalization bound for arbitrary posteriors to be used by the tasks, using the change-of-measure inequality (CMI) for the transition from $Q_1 \times \cdots \times Q_n$ to $P \times \cdots \times P$. Because our bound holds uniformly in $Q_1, \dots Q_n$, it also holds for the posteriors produced by a deterministic learning algorithm (a fixed A) or any distributions of algorithms (a hyperposterior $\mathcal{Q}$ over algorithms). This way we do not have to include the transition from $\mathcal{Q}$ to $\mathcal{P}$ in the CMI, as all prior work did. Consequently, in part of the proof we avoid getting a $KL(\mathcal{Q}||\mathcal{P})$ term which otherwise would be divided only by a $\frac{1}{m}$ factor, which is undesirable in the federated setting. Additionally compared to [Rez22] we have better logarithmic dependency. They have a term of order $\frac{\log m\sqrt{n}}{m}$ which is not negligible in the case that we have large number of tasks and small number of samples per task, which is usual in the federated setting. In contrast, our logarithmic terms are divided by n.

Proof of Theorem 3 We now instantiate the situation of Theorem 3 by specific choice of prior and posterior distributions. Let the learning algorithm be parameterized by the hypernetwork weights, η_h, and the embedding networks weights, η_v. As meta-posterior we use a Gaussian distribution, $\mathcal{Q} = \mathcal{Q}_h \times \mathcal{Q}_v$ for $\mathcal{Q}_h = \mathcal{N}(\eta_h; \alpha_h \mathsf{Id})$, and $\mathcal{Q}_v = \mathcal{N}(\eta_v; \alpha_v \mathsf{Id})$, where η_h and η_v are learnable and α_v and α_h are fixed. For any $(\bar{\eta}_h, \bar{\eta}_v) \sim \mathcal{Q}$ and training set S, the learning algorithm produces a posterior distribution $Q = \mathcal{N}(\theta; \alpha_\theta \mathsf{Id})$, where $\theta = h(v; \eta_h)$ with $v = \frac{1}{|S|} \sum_{(x,y)} \phi(x, y; \eta_v)$. As prior, we use $\mathcal{N}(0; \alpha_\theta \mathsf{Id})$. With these choices, we have $\mathbf{KL}(\mathcal{Q}, \mathcal{P}) = \alpha_h \|\eta_h\|^2 + \alpha_v |\eta_v\|^2$ and $\mathbf{KL}(Q_i, P) = \alpha_\theta \|\theta\|^2$. Inserting these into Theorem 8, we recover Theorem 3.

B.3 Optimization

In this section, we formulate the technical assumptions for Theorem 2 and prove it. For the convenience of reading we repeat some notation from Section 4.3. For each client $i = 1, \ldots, n$ and parameter θ_i, we note the client objective as $f_i(\theta_i)$ and $F_i(\eta) = f_i(h(S_i, \eta))$, where $h(S, \eta) = h(v(S, \eta_v), \eta_h)$ is a shorthand notation for the combination of hypernetwork and embedding network. The server objective we denote by $f(\eta)$, which in our setting consists only of regularization terms. The overall objective is $F(\eta) = \frac{1}{n} \sum_{i=1}^n F_i(\eta) + f(\eta)$. For the rest of this section, by η_t we mean the parameters at iteration t, and $g_{t,i,j}$ is the stochastic gradient for client i at global step t, local step l.

B.3.1 Analytical Assumptions

We make the following assumptions, which are common in the literature [KKM+20, FMO20a]. Since our algorithm consists of multiple parts, we have to make the assumptions for different parts separately.

- Bounded Gradients: there exist constants, b_1, b_2, such that

$$\|\nabla_\theta f_i(\theta)\| \leq b_1, \quad \|\nabla_\eta h(S, \eta)\| \leq b_2 \tag{B.24}$$

  which, in particular, implies

$$\|h(S, \eta) - h(S, \eta')\| \leq b_2 \|\eta - \eta'\| \tag{B.25}$$

- Smoothness: there exist constants, L_1, L_2, L_3, such that

$$\|\nabla_\theta f_i(\theta) - \nabla_\theta f_i(\theta')\| \leq L_1 \|\theta - \theta'\| \tag{B.26}$$
$$\|\nabla_\eta h(S, \eta) - \nabla_\eta h(S, \eta')\| \leq L_2 \|\eta - \eta'\| \tag{B.27}$$
$$\|\nabla_\eta f(\eta) - \nabla_\eta f(\eta')\| \leq L_3 \|\eta - \eta'\| \tag{B.28}$$

- Bounded Dissimilarity: there exists a constant γ_G with, such that with $\bar{F}(\eta) = \frac{1}{n} \sum_{i=1}^n F_i(\eta)$:

$$\frac{1}{n} \sum_{i=1}^n \mathbb{E}[\|\nabla F_i(\eta) - \nabla \bar{F}(\eta)\|^2] \leq \gamma_G^2 \tag{B.29}$$

- Bounded Variance: there exists constants σ_1, σ_2, σ_3, such that, for batches $B_i \subset S_i$ of size $b < |S_i|$:

$$\mathbb{E}\,\|\nabla_\theta(f_i(\theta)) - \widetilde{\nabla}_\theta(f_i(\theta))\|^2 \leq \sigma_1^2 \tag{B.30}$$

$$\mathbb{E}\,\|\nabla h(B_i,\eta) - \nabla h(S_i,\eta)\|^2 \leq \frac{\sigma_2^2}{b} \tag{B.31}$$

$$\mathbb{E}\,\|h(B_i,\eta) - h(S_i,\eta)\|^2 \leq \frac{\sigma_3^2}{b} \tag{B.32}$$

B.3.2 Convergence

Lemma 9. *There exists a constant L which $\nabla F(\eta)$ is L-smooth.*

Proof. For each F_i we have

$$\|\nabla_\eta F_i(\eta) - \nabla_\eta F_i(\eta')\| = \|\nabla_\theta f_i(h(S_i,\eta))\nabla_\eta h(S_i,\eta)^T - \nabla_\theta f_i(h(S_i,\eta'))\nabla_\eta h(S_i,\eta')^T\| \tag{B.33}$$

$$= \|\nabla_\theta f_i(h(S_i,\eta))\nabla_\eta h(S_i,\eta)^T - \nabla_\theta f_i(h(S_i,\eta))\nabla_\eta h(S_i,\eta')^T \tag{B.34}$$

$$+ \nabla_\theta f_i(h(S_i,\eta))\nabla_\eta h(S_i,\eta')^T - \nabla_\theta f_i(h(S_i,\eta'))\nabla_\eta h(S_i,\eta')^T\| \tag{B.35}$$

$$\leq \|\nabla_\theta f_i(h(S_i,\eta))\nabla_\eta h(S_i,\eta)^T - \nabla_\theta f_i(h(S_i,\eta))\nabla_\eta h(S_i,\eta')^T\| \tag{B.36}$$

$$+ \|\nabla_\theta f_i(h(S_i,\eta))\nabla_\eta h(S_i,\eta')^T - \nabla_\theta f_i(h(S_i,\eta'))\nabla_\eta h(S_i,\eta')^T\| \tag{B.37}$$

$$\leq b_1 L_2\|\eta - \eta'\| + b_2 L_1 b_2\|\eta - \eta'\| = (b_1 L_2 + b_2 L_1 b_2)\|\eta - \eta'\| = L'\|\eta - \eta'\| \tag{B.38}$$

Therefore for F we have

$$\|\nabla_\eta F(\eta) - \nabla_\eta F(\eta')\| \leq \|\frac{1}{n}\sum_{i=1}^n \nabla_\eta F_i(\eta) + \nabla_\eta f(\eta) - \frac{1}{n}\sum_{i=1}^n \nabla_\eta F_i(\eta') - \nabla_\eta f(\eta')\| \tag{B.39}$$

$$\leq \frac{1}{n}\sum_{i=1}^n \|\nabla_\eta F_i(\eta) - \nabla_\eta F_i(\eta')\| + \|\nabla_\eta f(\eta) - \nabla_\eta f(\eta')\| \tag{B.40}$$

$$\leq (L' + L_3)\|\eta - \eta'\| = L\|\eta - \eta'\| \tag{B.41}$$

□

Lemma 10. *For independent random vectors $a_1, ..., a_n$we have*

$$\mathbb{E}[\|\sum_{i\in C} a_i - \mathbb{E}[a_i]\|^2] \leq \frac{c}{n}\sum_i \mathbb{E}[\|a_i - \mathbb{E}[a_i]\|^2] \tag{B.42}$$

Also if $\mathbb{E}[a_i] \leq a$ for each i, we have

$$\mathbb{E}[\|\sum_{i\in C} a_i - \mathbb{E}[a_i]\|^2] \leq \frac{c}{n}\sum_i \mathbb{E}[\|a_i - \mathbb{E}[a_i]\|^2] + c^2 a^2 \tag{B.43}$$

Proof. By expanding the power 2 we get

$$\mathbb{E}[\|\sum_{i\in C} a_i - \mathbb{E}[a_i]\|^2] = \mathbb{E}[\sum_{i\in C} \|a_i - \mathbb{E}[a_i]\|^2] + \mathbb{E}[\sum_{i\neq j\in C} \langle a_i - \mathbb{E}[a_i], a_j - \mathbb{E}[a_j]\rangle] \tag{B.44}$$

$$\leq \frac{\binom{n-1}{c-1}}{\binom{n}{c}} \sum_i \mathbb{E}[\|a_i - \mathbb{E}[a_i]\|^2] + \frac{\binom{n-2}{c-2}}{\binom{n}{c}} \sum_{i\neq j} \mathbb{E}[\langle a_i - \mathbb{E}[a_i], a_j - \mathbb{E}[a_j]\rangle] \tag{B.45}$$

$$= \frac{\binom{n-1}{c-1}}{\binom{n}{c}} \sum_i \mathbb{E}[\|a_i - \mathbb{E}[a_i]\|^2] - \frac{2\binom{n-2}{c-2}}{\binom{n}{c}} \sum_i \mathbb{E}[\|a_i - \mathbb{E}[a_i]\|^2] \tag{B.46}$$

$$\leq \frac{c}{n} \sum_i \mathbb{E}[\|a_i - \mathbb{E}[a_i]\|^2] \tag{B.47}$$

For the second part we have

$$\mathbb{E}[\|\sum_{i\in C} a_i\|^2] = \mathbb{E}[\|\sum_{i\in C} a_i - \mathbb{E}[a_i] + \mathbb{E}[a_i]\|^2] \tag{B.48}$$

$$= \mathbb{E}[\|\sum_{i\in C} a_i - \mathbb{E}[a_i]\|^2] + \mathbb{E}[\|\sum_{i\in C} \mathbb{E}[a_i]\|^2] + 2\,\mathbb{E}[\langle \sum_{i\in C} a_i - \mathbb{E}[a_i], \sum_{i\in C} \mathbb{E}[a_i]\rangle] \tag{B.49}$$

$$= \mathbb{E}[\|\sum_{i\in C} a_i - \mathbb{E}[a_i]\|^2] + \mathbb{E}[\|\sum_{i\in C} \mathbb{E}[a_i]\|^2] \leq \frac{c}{n} \sum_i \mathbb{E}[\|a_i - \mathbb{E}[a_i]\|^2] + c^2 a^2 \tag{B.50}$$

□

Lemma 11. *For any step t and client i the following inequality holds:*

$$\mathbb{E}\,\|\nabla F_i(\eta_t) - \nabla_\theta f_i(h(B_i, \eta_t) - \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i, \eta_t)))\nabla h(B_i, \eta_t)^T\|^2 \tag{B.51}$$

$$\leq 27 L_1^2 b_2^2 \beta^2 l^2 (b_1^2 + \sigma_1^2) + 3 b_1^2 \sigma_2^2 + 6 L_1^2 b_2^2 \sigma_3^2 \tag{B.52}$$

Proof. By adding and subtracting some immediate terms we get

$$\mathbb{E}\,\|\nabla F_i(\eta_t) - \nabla_\theta f_i(h(B_i, \eta_t) - \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i, \eta_t)))\nabla h(B_i, \eta_t)^T\|^2 \tag{B.53}$$

$$= \mathbb{E}\,\|\nabla F_i(\eta_t) - \nabla_\theta f_i(h(S_i, \eta_t) - \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(S_i, \eta_t)))\nabla h(S_i, \eta_t)^T \tag{B.54}$$

$$+ \nabla_\theta f_i(h(S_i, \eta_t) - \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(S_i, \eta_t)))(\nabla h(S_i, \eta_t)^T - \nabla h(B_i, \eta_t)^T) \tag{B.55}$$

$$+ (\nabla_\theta f_i(h(S_i, \eta_t) - \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(S_i, \eta_t))) - \nabla_\theta f_i(h(B_i, \eta_t) \tag{B.56}$$

$$- \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i, \eta_t))))\nabla h(B_i, \eta_t)^T\|^2 \tag{B.57}$$

And by Cauchy–Schwarz inequality we get

$$\mathbb{E}\,\|\nabla F_i(\eta_t) - \nabla_\theta f_i(h(B_i,\eta_t) - \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i,\eta_t)))\nabla h(B_i,\eta_t)^T\|^2 \tag{B.58}$$

$$\leq 3\,\mathbb{E}\,\|\nabla_\theta f_i(h(S_i,\eta_t))\nabla h(S_i,\eta_t)^T - \nabla_\theta f_i(h(S_i,\eta_t) \tag{B.59}$$

$$- \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(S_i,\eta_t)))\nabla h(S_i,\eta_t)^T\|^2 \tag{B.60}$$

$$+ 3\,\mathbb{E}\,\|\nabla_\theta f_i(h(S_i,\eta_t) - \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(S_i,\eta_t)))(\nabla h(B_i,\eta_t)^T - \nabla h(S_i,\eta_t)^T)\|^2 \tag{B.61}$$

$$+ 3\,\mathbb{E}\,\|(\nabla_\theta f_i(h(B_i,\eta_t) - \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i,\eta_t))) - \nabla_\theta f_i(h(S_i,\eta_t) \tag{B.62}$$

$$- \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(S_i,\eta_t))))\nabla h(B_i,\eta_t)^T\|^2 \tag{B.63}$$

$$\leq 3\,\mathbb{E}(\|\nabla_\theta f_i(h(S_i,\eta_t)) - \nabla_\theta f_i(h(S_i,\eta_t) - \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(S_i,\eta_t)))\|^2 \|\nabla h(S_i,\eta_t)\|^2) \tag{B.64}$$

$$+ 3\,\mathbb{E}(\|\nabla_\theta f_i(h(S_i,\eta_t) - \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(S_i,\eta_t)))\|^2 \|\nabla h(B_i,\eta_t) - \nabla h(S_i,\eta_t)\|^2) \tag{B.65}$$

$$+ 3\,\mathbb{E}(\|\nabla_\theta f_i(h(B_i,\eta_t) - \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i,\eta_t))) - \nabla_\theta f_i(h(S_i,\eta_t) \tag{B.66}$$

$$- \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(S_i,\eta_t)))\|^2 \|\nabla h(B_i,\eta_t)\|^2) \tag{B.67}$$

$$\leq 3L_1^2 b_2^2 \beta^2\,\mathbb{E}\,\|\sum_{j=0}^{l-1} g_{t,i,j}(h(S_i,\eta_t))\|^2 + 3b_1^2\,\mathbb{E}\,\|\nabla h(B_i,\eta_t) - \nabla h(S_i,\eta_t)\|^2 \tag{B.68}$$

$$+ 6L_1^2 b_2^2\,\mathbb{E}\,\|h(B_i,\eta_t) - h(S_i,\eta_t)\|^2 \tag{B.69}$$

$$+ 6L_1^2 b_2^2 \beta^2\,\mathbb{E}\,\|\sum_{j=0}^{l-1} g_{t,i,j}(h(S_i,\eta_t)) - \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i,\eta_t))\|^2 \tag{B.70}$$

$$\leq 3L_1^2 b_2^2 \beta^2 l^2 (b_1^2 + \sigma_1^2) + 3b_1^2\sigma_2^2 + 6L_1^2 b_2^2 \sigma_3^2 + 24 L_1^2 b_2^2 \beta^2 l^2 (b_1^2 + \sigma_1^2) \tag{B.71}$$

$$= 27 L_1^2 b_2^2 \beta^2 l^2 (b_1^2 + \sigma_1^2) + 3b_1^2\sigma_2^2 + 6L_1^2 b_2^2 \sigma_3^2 \tag{B.72}$$

which we used the assumptions on the bounded gradients and bounded variance and Lemma 10. □

Lemma 12. *For any step t the following inequality holds:*

$$\mathbb{E}[\langle \nabla F(\eta_t), \eta_{t+1} - \eta_t \rangle] \leq \frac{-3\beta k}{4}\|\nabla F(\eta_t)\|^2 + 27 L_1^2 b_2^2 \beta^3 k^3 (b_1^2 + \sigma_1^2) + 3\beta k b_1^2 \sigma_2^2 + 6\beta k L_1^2 b_2^2 \sigma_3^2 \tag{B.73}$$

Proof.

$$\mathop{\mathbb{E}}_{t}[\langle \nabla F(\eta_t), \eta_{t+1} - \eta_t \rangle] = \mathop{\mathbb{E}}_{t}[\langle \nabla F(\eta_t), -\frac{\beta}{c} \sum_{i \in C} \sum_{l=1}^{k} g_{t,i,l}(h(B_i, \eta_t)) \nabla h(B_i, \eta_t)^T - \beta k \nabla f(\eta_t) \rangle] \tag{B.74}$$

$$= \mathop{\mathbb{E}}_{t}[\langle \nabla F(\eta_t), -\frac{\beta}{n} \sum_{i=1}^{n} \sum_{l=1}^{k} \nabla_\theta f_i(h(B_i, \eta_t) - \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i, \eta_t))) \nabla h(B_i, \eta_t)^T \rangle] \tag{B.75}$$

$$+ \mathop{\mathbb{E}}_{t}[\langle \nabla F(\eta_t), -\beta k \nabla f(\eta_t) \rangle] \tag{B.76}$$

$$= \frac{\beta}{n} \sum_{i=1}^{n} \sum_{l=1}^{k} \mathop{\mathbb{E}}_{t}[\langle \nabla F(\eta_t), \nabla F_i(\eta_t) - \nabla_\theta f_i(h(B_i, \eta_t) - \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i, \eta_t))) \nabla h(B_i, \eta_t)^T \rangle] \tag{B.77}$$

$$+ \frac{\beta}{n} \sum_{i=1}^{n} \sum_{l=1}^{k} \mathop{\mathbb{E}}_{t}[\langle \nabla F(\eta_t), -\nabla F_i(\eta_t) \rangle] + \mathop{\mathbb{E}}_{t}[\langle \nabla F(\eta_t), -\beta k \nabla f(\eta_t) \rangle] \tag{B.78}$$

By definition $F(\eta) = \frac{1}{n} \sum_{i=1}^{n} F_i(\eta) + f(\eta)$, and by Young's inequality and lemma 11 we have

$$\mathop{\mathbb{E}}_{t}[\langle \nabla F(\eta_t), \eta_{t+1} - \eta_t \rangle] \tag{B.79}$$

$$\leq \frac{\beta k}{4} \|\nabla F(\eta_t)\|^2 \tag{B.80}$$

$$+ \frac{\beta}{n} \sum_{i=1}^{n} \sum_{l=1}^{k} \mathop{\mathbb{E}}_{t} \|\nabla F_i(\eta_t) - \nabla_\theta f_i(h(B_i, \eta_t) - \beta \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i, \eta_t))) \nabla h(B_i, \eta_t)^T\|^2 \tag{B.81}$$

$$- \beta k \|\nabla F(\eta_t)\|^2 \tag{B.82}$$

$$\leq \frac{-3\beta k}{4} \|\nabla F(\eta_t)\|^2 + \frac{\beta}{n} \sum_{i=1}^{n} \sum_{l=1}^{k} (27 L_1^2 b_2^2 \beta^2 l^2 (b_1^2 + \sigma_1^2) + 3 b_1^2 \sigma_2^2 + 6 L_1^2 b_2^2 \sigma_3^2) \tag{B.83}$$

$$= \frac{-3\beta k}{4} \|\nabla F(\eta_t)\|^2 + 27 L_1^2 b_2^2 \beta^3 k^3 (b_1^2 + \sigma_1^2) + 3\beta k b_1^2 \sigma_2^2 + 6\beta k L_1^2 b_2^2 \sigma_3^2 \tag{B.84}$$

□

Lemma 13. *For any step t the following inequality holds:*

$$\mathbb{E}[\|\eta_{t+1} - \eta_t\|^2] \leq \frac{4\beta^2 k}{c} \sigma_1^2 + \frac{4\beta^2 k^2}{c} \gamma_G^2 + 4\beta^2 k^2 \, \mathbb{E}[\|\nabla F(\eta_t)\|^2] \tag{B.85}$$

$$+ 108 L_1^2 b_2^2 \beta^4 k^4 (b_1^2 + \sigma_1^2) + 12 b_1^2 k^2 \beta^2 \sigma_2^2 + 24 L_1^2 b_2^2 k^2 \beta^2 \sigma_3^2 \tag{B.86}$$

Proof. The change of η in one step of optimization is the average of the change caused by selected clients plus function $f(\eta)$ at the server.

$$\mathbb{E}[\|\eta_{t+1} - \eta_t\|^2] = \mathbb{E}[\| - \frac{\beta}{c} \sum_{i \in C} \sum_{l=1}^{k} g_{t,i,l}(h(B_i, \eta_t)) \nabla h(B_i, \eta_t)^T - \beta k \nabla f(\eta_t)\|^2] \tag{B.87}$$

Which we can write as

$$\mathbb{E}[\|\eta_{t+1}-\eta_t\|^2] = \frac{\beta^2}{c^2}\,\mathbb{E}[\|\sum_{i\in C}\sum_{l=1}^{k} g_{t,i,l}(h(B_i,\eta_t))\nabla h(B_i,\eta_t)^T + ck\nabla f(\eta_t)\|^2] \tag{B.88}$$

$$= \frac{\beta^2}{c^2}\,\mathbb{E}[\|\sum_{i\in C}\sum_{l=1}^{k} g_{t,i,l}(h(B_i,\eta_t))\nabla h(B_i,\eta_t)^T \tag{B.89}$$

$$- \nabla_\theta f_i(h(B_i,\eta_t) - \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i,\eta_t)))\nabla h(B_i,\eta_t)^T \tag{B.90}$$

$$+ \sum_{i\in C}\sum_{l=1}^{k} \nabla_\theta f_i(h(B_i,\eta_t) - \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i,\eta_t)))\nabla h(B_i,\eta_t)^T - \nabla F_i(\eta_t) \tag{B.91}$$

$$+ \sum_{i\in C}\sum_{l=1}^{k}(\nabla F_i(\eta_t) + \nabla f(\eta_t) - \nabla F(\eta_t)) + \sum_{i\in C}\sum_{l=1}^{k}\nabla F(\eta_t)\|^2] \tag{B.92}$$

And by Cauchy–Schwarz inequality we get

$$\mathbb{E}[\|\eta_{t+1}-\eta_t\|^2] \tag{B.93}$$

$$\leq \frac{4\beta^2}{c^2}\,\mathbb{E}[\|\sum_{i\in C}\sum_{l=1}^{k} g_{t,i,l}(h(B_i,\eta_t))\nabla h(B_i,\eta_t)^T \tag{B.94}$$

$$- \nabla_\theta f_i(h(B_i,\eta_t) - \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i,\eta_t)))\nabla h(B_i,\eta_t)^T\|^2] \tag{B.95}$$

$$+ \frac{4\beta^2}{c^2}\,\mathbb{E}[\|\sum_{i\in C}\sum_{l=1}^{k} \nabla_\theta f_i(h(B_i,\eta_t) - \sum_{j=0}^{l-1} g_{t,i,j}(h(B_i,\eta_t)))\nabla h(B_i,\eta_t)^T - \nabla F_i(\eta_t)\|^2] \tag{B.96}$$

$$+ \frac{4\beta^2}{c^2}\,\mathbb{E}[\|\sum_{i\in C}\sum_{l=1}^{k} \nabla F_i(\eta_t) + \nabla f(\eta_t) - \nabla F(\eta_t)\|^2] + \frac{4\beta^2}{c^2}\,\mathbb{E}[\|\sum_{i\in C}\sum_{l=1}^{k}\nabla F(\eta_t)\|^2] \tag{B.97}$$

Note that $\nabla F(\eta) - \nabla f(\eta) = \frac{1}{n}\sum_{i=1}^{n}\nabla F_i(\eta)$. By lemma 10, 11, the bounded dissimilarity assumption and simplifying terms we get

$$\mathbb{E}[\|\eta_{t+1}-\eta_t\|^2] \tag{B.98}$$

$$\leq \frac{4\beta^2 k}{c}\sigma_1^2 + \frac{4\beta^2 k^2}{cn}\sum_i \mathbb{E}[\|\nabla F_i(\eta_t) - (\frac{1}{n}\sum_{i=1}^{n}\nabla F_i(\eta_t))\|^2] + 4\beta^2k^2\,\mathbb{E}[\|\nabla F(\eta_t)\|^2] \tag{B.99}$$

$$+ 108L_1^2b_2^2\beta^4k^4(b_1^2+\sigma_1^2) + 12b_1^2k^2\beta^2\sigma_2^2 + 24L_1^2b_2^2k^2\beta^2\sigma_3^2 \tag{B.100}$$

$$\leq \frac{4\beta^2 k}{c}\sigma_1^2 + \frac{4\beta^2 k^2}{c}\gamma_G^2 + 4\beta^2k^2\,\mathbb{E}[\|\nabla F(\eta_t)\|^2] \tag{B.101}$$

$$+ 108L_1^2b_2^2\beta^4k^4(b_1^2+\sigma_1^2) + 12b_1^2k^2\beta^2\sigma_2^2 + 24L_1^2b_2^2k^2\beta^2\sigma_3^2 \tag{B.102}$$

□

We are now ready to prove Theorem 2, which we restate for the convenience of the reader.

Theorem 2. *Under standard smoothness and boundedness assumptions (see appendix), PeFLL's optimization after T steps fulfills*

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left\|\nabla F(\eta_t)\right\|^2 \leq \frac{(F(\eta_0)-F_*)}{\sqrt{cT}}+\frac{L(6\sigma_1^2+4k\gamma_G^2)}{k\sqrt{cT}}+\frac{224cL_1^2b_1^2b_2^2}{T}+\frac{8b_1^2\sigma_2^2}{b}+\frac{14L_1^2b_2^2\sigma_3^2}{b}, \tag{4.2}$$

where F is the PeFLL objective (4.1) with lower bound F_*. η_0 are the parameter values at initialization, $\eta_1, \ldots, \eta_T$ are the intermediate parameter values. L, L_1 are smoothness parameters of F and the local models. b_1, b_2 are bounds on the norms of the gradients of the local model and the hypernetwork, respectively. σ_1 is a bound on the variance of stochastic gradients of local models, and σ_2, σ_3 are bounds on the variance due to the clients generating models with data batches of size b instead of their whole training set. γ_G is a bound on the dissimilarity of clients, c is the number of clients participating at each round, and k is the number of local SGD steps performed by the clients.

Proof. By smoothness of F we have

$$\mathbb{E}[F(\eta_{t+1}) - F(\eta_t)] \leq \mathbb{E}\langle \nabla F(\eta_t), \eta_{t+1} - \eta_t\rangle + \frac{L}{2}\mathbb{E}\left\|\eta_{t+1} - \eta_t\right\|^2 \tag{B.103}$$

And by combining it with Lemma 12 and 13 and simplifying terms we get

$$\mathbb{E}[F(\eta_{t+1}) - F(\eta_t)] \leq \mathbb{E}\langle \nabla F(\eta_t), \eta_{t+1} - \eta_t\rangle + \frac{L}{2}\mathbb{E}\left\|\eta_{t+1} - \eta_t\right\|^2 \tag{B.104}$$

$$\leq \frac{-3\beta k}{4}\mathbb{E}\left\|\nabla F(\eta_t)\right\|^2 + 27L_1^2b_2^2\beta^3k^3(b_1^2+\sigma_1^2) + 3\beta k b_1^2\sigma_2^2 + 6\beta k L_1^2b_2^2\sigma_3^2 \tag{B.105}$$

$$+ \frac{2L\beta^2k}{c}\sigma_1^2 + \frac{2L\beta^2k^2}{c}\gamma_G^2 + 2L\beta^2k^2\,\mathbb{E}[\|\nabla F(\eta_t)\|^2] \tag{B.106}$$

$$+ 56LL_1^2b_2^2\beta^4k^4(b_1^2+\sigma_1^2) + 6Lb_1^2k^2\beta^2\sigma_2^2 + 12LL_1^2b_2^2k^2\beta^2\sigma_3^2 \tag{B.107}$$

$$= (\frac{-3\beta k}{4} + 2L\beta^2k^2)\,\mathbb{E}\left\|\nabla F(\eta_t)\right\|^2 + (27L_1^2b_2^2\beta^3k^3 + 56LL_1^2b_2^2\beta^4k^4)b_1^2 \tag{B.108}$$

$$+ 3\beta k b_1^2\sigma_2^2 + 6Lb_1^2k^2\beta^2\sigma_2^2 + 6\beta k L_1^2b_2^2\sigma_3^2 + 12LL_1^2b_2^2k^2\beta^2\sigma_3^2 \tag{B.109}$$

$$+ (\frac{L\beta^2k}{c} + 27L_1^2b_2^2\beta^3k^3 + 56LL_1^2b_2^2\beta^4k^4)\sigma_1^2 + \frac{L\beta^2k^2}{c}\gamma_G^2 \tag{B.110}$$

$$\leq \frac{-\beta k}{2}\mathbb{E}\left\|\nabla F(\eta_t)\right\|^2 + 28L_1^2b_1^2b_2^2\beta^3k^3 + 4\beta k b_1^2\sigma_2^2 + 7\beta k L_1^2b_2^2\sigma_3^2 \tag{B.111}$$

$$+ \frac{3L\beta^2k}{c}\sigma_1^2 + \frac{2L\beta^2k^2}{c}\gamma_G^2 \tag{B.112}$$

Therefore

$$\mathbb{E}[F(\eta_{t+1}) - F(\eta_t)] \leq \frac{-\beta k}{2}\mathbb{E}\left\|\nabla F(\eta_t)\right\|^2 + 28L_1^2b_1^2b_2^2\beta^3k^3 + 4\beta k b_1^2\sigma_2^2 + 7\beta k L_1^2b_2^2\sigma_3^2 \tag{B.113}$$

$$+ \frac{3L\beta^2k}{c}\sigma_1^2 + \frac{2L\beta^2k^2}{c}\gamma_G^2 \tag{B.114}$$

Hence

$$\mathbb{E}\left\|\nabla F(\eta_t)\right\|^2 \leq \frac{2}{\beta k}\mathbb{E}[F(\eta_t) - F(\eta_{t+1})] + 56L_1^2 b_1^2 b_2^2 \beta^2 k^2 + 8b_1^2\sigma_2^2 + 14L_1^2 b_2^2 \sigma_3^2 \tag{B.115}$$

$$+ \frac{6L\beta}{c}\sigma_1^2 + \frac{4L\beta k}{c}\gamma_G^2 \tag{B.116}$$

And

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left\|\nabla F(\eta_t)\right\|^2 \leq \frac{2(F(\eta_0) - F_*)}{\beta k T} + 56L_1^2 b_1^2 b_2^2 \beta^2 k^2 + 8b_1^2\sigma_2^2 + 14L_1^2 b_2^2 \sigma_3^2 \tag{B.117}$$

$$+ \frac{6L\beta}{c}\sigma_1^2 + \frac{4L\beta k}{c}\gamma_G^2 \tag{B.118}$$

by setting $\beta = \frac{2\sqrt{c}}{k\sqrt{T}}$ we get

$$\frac{1}{T}\sum_{t=1}^{T}\mathbb{E}\left\|\nabla F(\eta_t)\right\|^2 \leq \frac{(F(\eta_0) - F_*)}{\sqrt{cT}} + \frac{224cL_1^2 b_1^2 b_2^2}{T} + 8b_1^2\sigma_2^2 + 14L_1^2 b_2^2 \sigma_3^2 \tag{B.119}$$

$$+ \frac{L(6\sigma_1^2 + 4k\gamma_G^2)}{k\sqrt{cT}} \tag{B.120}$$

□

APPENDIX C

Appendix for Chapter 5

C.1 Experiments

Here we include additional experiments (Section C.1.1) and implementation details (Section C.1.2).

C.1.1 Additional experiments

Rotated MNIST We include here results for Rotated MNIST The observations here are broadly similar to those discussed in Section 5.6.2 with FLowDUP exhibiting the best performance overall.

Table C.1: Accuracy on Rotated MNIST for varying fractions (p) of the training clients having labeled data.

Fraction Labeled (p)	**0.1**	**0.2**	**0.5**	**1.0**
FedAvg	92.9 $\pm$ 0.2	94.8 $\pm$ 0.2	96.0 $\pm$ 0.2	96.4 $\pm$ 0.1
FedProx	92.8 $\pm$ 0.4	94.7 $\pm$ 0.1	95.8 $\pm$ 0.2	96.3 $\pm$ 0.1
LD-FedAvg	90.5 $\pm$ 0.3	92.4 $\pm$ 0.3	94.1 $\pm$ 0.0	95.0 $\pm$ 0.1
FedTTA	93.2 $\pm$ 0.3	94.8 $\pm$ 0.1	96.6 $\pm$ 0.1	97.3 $\pm$ 0.1
FLowDUP	94.0 $\pm$ 1.4	96.6 $\pm$ 0.4	97.8 $\pm$ 0.1	98.5 $\pm$ 0.1

The effect of unlabeled clients One of the advantages of FLowDUP is that it's learning objective is structured in such a way that it allows unlabeled clients that are present during training to contribute to regularizing h. Specifically, unlabeled clients are able to compute the regularizer Ω in 5.6 and obtain from this a gradient update for ψ. Here we investigate the effect that unlabeled clients have. To do this we run FLowDUP on partitioned CIFAR10 both with and without using the unlabeled clients. For FLowDUP without unlabeled clients we set the cohort size to 100 and sample only clients with labeled data. For FLowDUP with labeled clients we set the cohort size to 200 and the labeled client sampling rate to $\alpha = 0.5$ so that the number of labeled clients present during each round is the same in both cases. We vary the total fraction of labeled clients, with $p \in \{0.05, 0.1, 0.2, 0.5\}$. As always we set $k = 10^4$. Tables C.2 and C.3 show the results. As we can see, overall using unlabeled clients leads to

an increase in performance. This is most pronounced for lower values of p (in particular at $p = 0.5$ the effect is small or negligible) and occurs for both architectures.

Table C.2: Partitioned CIFAR10 Accuracy for CNN.

Fraction Labeled (p)	**0.05**	**0.1**	**0.2**	**0.5**
FLowDUP (without unlabeled)	51.5 ± 1.8	65.3 ± 1.3	74.5 ± 1.3	82.5 ± 1.0
FLowDUP (with unlabeled)	52.3 ± 1.4	67.5 ± 0.4	76.6 ± 2.3	83.8 ± 0.7

Table C.3: Partitioned CIFAR10 Accuracy for ResNet18.

Fraction Labeled (p)	**0.05**	**0.1**	**0.2**	**0.5**
FLowDUP (without unlabeled)	54.8 ± 1.3	71.7 ± 1.2	83.3 ± 3.5	90.5 ± 0.4
FLowDUP (with unlabeled)	57.5 ± 1.6	72.6 ± 0.6	83.0 ± 2.7	90.6 ± 0.7

The learnable regularizer FLowDUP prevents overfitting by penalizing large deviations between the generated client subspace parameters and a learned regularizer ψ_r. Intuitively, we can think of ψ_r as a global subspace model that clients should not deviate too much from. Through the lens of our theoretical results, 4, ψ_r is the mean of a learnable prior distribution on the client models. Here we investigate the efficacy of *learning* the regularizer. To do this we replace ψ_r by 0, so that the regularization term in the loss becomes instead

$$\Omega = \sum_{i \in \mathcal{C}} \|h(X\,;\,\psi_h)\|^2. \tag{C.1}$$

Note, this is of course still a reasonable regularizer more in line with classic ℓ_2 regularization. We train FLowDUP using this new non-learnable regularizer and compare it our proposed version using the learnable regularizer 5.6. As in the previous section we again take a cohort size of 100 labeled clients and 100 unlabeled clients each round, i.e. $\alpha = 0.5$. We set $k = 10^4$ and vary $p \in \{0.1, 0.2, 0.5, 1.0\}$. The results can be seen in Table C.4. They indeed show that the extra flexibility afforded by learning the regularizer leads to a modest boost in performance for all values of p.

C.1.2 Implementation details

Hyperparameters There are number of general federated learning hyperparameters that are shared across methods. We train all methods for the same number of global rounds T. For class partitioned CIFAR10 and FEMNIST we set $T = 1000$ while for Rotated Fashion-MNIST and Rotated MNIST we set $T = 500$. All methods use a client cohort size each round of size

Table C.4: The effect of learning the regularizer ψ_r. Accuracy on partitioned CIFAR10 for CNN.

Fraction Labeled (p)	**0.1**	**0.2**	**0.5**	**1.0**
FLowDUP (Without Learnable Reg)	65.5 ± 0.4	75.2 ± 0.8	83.3 ± 1.7	85.6 ± 0.8
FLowDUP (With Learnable Reg)	67.5 ± 0.4	76.6 ± 0.3	83.8 ± 0.7	86.6 ± 0.1

100, a local number of epochs set to $E = 1$ and a local batch size of $B = 20$ for FEMNIST and $B = 50$ on all other datasets. For all methods we tune the local learning rate η_l on validation clients.

Regarding method specific hyperparameters. For FedProx, we set μ following [LSZ+20], that is $\mu = 1$ initially and is updated over the course of training as described in [LSZ+20]. For FedTTA we tune the adaptive learning rate. For FLowDUP, unless otherwise stated, we set the labeled client sampling rate to $\alpha = 0.9$ and the subspace dimension to $k = 10^4$. We tune the regularization strength λ.

Model architectures When used for prediction the CNN follows the architecture used in prior work [MMR+17] while the ResNet18 follows the standard architecture with the final classification layer having output dimension 10 for the 10 classes present in CIFAR10. When used for h_1, we instead replace the final linear layers of the CNN and ResNet18 with another with output dimension 256. For h_2 we always use a fully connected network with a single hidden layer and ReLU non-linearity.

C.2 Additional Related Work

Personalized federated learning Approaches to personalized FL typically fall into one of the following categories: Meta-learning based approaches [JKRK19, FMO20a] which learn a single global model that can be easily personalized using a small number of gradient steps on the client. Parameter decomposition-approaches [AASC19, CHMS21, MNVK22, CYG+23, WZY+23] which divide the learnable parameters into some that are shared across clients (such as a feature extractor) and some that are specific individual to each client (such as a classification head). Federated multi-task approaches [SCST17a, DTN20, HHHR20, MNB+21, LHBS21, LHLZ22, YNW+23, ZLD+23] learn separate models for each client while still sharing some information across clients, for instance by regularizing towards some global model. All of the above approaches, however, require a client to have labeled data in order to obtain a personalized model.

Learning in a subspace This formulation of intrinsic dimensionality and learning in a subspace has been studied in different contexts in the literature. [LFLY18] introduced the formulation and showed that for real-world problems the intrinsic dimension is much smaller than the total number of parameters $k \ll d$. [AGZ21] showed that pretraining reduces the intrinsic dimensionality of fine-tuning. [LFK+22] used this parametrization to achieve non-vacuous generalization bounds for neural networks. [ZGL25] extended the definition of the intrinsic dimension to multi-task learning. [PJK+25] used the formulation for black-box prompt tuning. [LHLZ22] used it to reduce the communication of the global model in personalized federated learning.

C.3 Transductive multi-task learning

In this section, we provide our general theoretical contributions and their proof.

PAC-Bayesian theory [McA98] studies the generalization behavior of stochastic models (a distribution over a set of models $f \in \mathcal{F}$). Formally, when training a posterior distribution $Q \in \mathcal{M}(\mathcal{F})$ using a dataset S consisting of m_L i.i.d. labeled samples from a distribution D over $\mathcal{X} \times$

$\mathcal{Y}$, PAC-Bayes bounds guarantee upper-bounds for the true risk of a posterior Q i.e. $\text{er}(Q) = \mathbb{E}_{f\sim Q}\,\mathbb{E}_{(x,y)\sim D}\,\ell(f,x,y)$ based on its training risk ($\widehat{\text{er}}(Q) = \mathbb{E}_{f\sim Q}\,\frac{1}{m_L}\sum_{i=1}^{m_L}\ell(f,x_i,y_i)$), and a complexity term based on $\mathbf{KL}(Q\|P)$ where $P \in \mathcal{M}(\mathcal{F})$ is a data-independent prior over the space of models, and $\mathbf{KL}$ is the Kullback-Leibler divergence. As an example, from [Mau04] we have that for any prior P, and any $\delta > 0$, over the sampling of the dataset S for all posteriors Q it holds:

$$\text{er}(Q)) \leq \widehat{\text{er}}(Q) + \sqrt{\frac{\mathbf{KL}(Q\|P) + \log(\frac{2\sqrt{m_L}}{\delta})}{2m_L}}, \tag{C.2}$$

This bound holds in an inductive learning setting, i.e. we have a distribution D, and the goal is to generalize from training data to the unseen data. On the other hand, in transductive learning [Vap98], there is a set of labeled data, and a set of unlabeled data, and the goal is to generalize from labeled to unlabeled data. Transductive learning has also been studied in the PAC-Bayes literature. Therefore, if we have m samples which m_L of them is labeled we want to generalize from $\widehat{\text{er}}(Q)$ to $\mathbb{E}_{f\sim Q}\,\frac{1}{m}\sum_{i=1}^{m}\ell(f,x_i,y_i)$. For this problem, [BGLR14] have proved for any prior P, and any $\delta > 0$, over the sampling of m_L labeled samples out of m samples (without replacement) for all posteriors Q it holds:

$$\mathop{\mathbb{E}}_{f\sim Q}\frac{1}{m}\sum_{i=1}^{m}\ell(f,x_i,y_i) \leq \widehat{\text{er}}_L(Q) + \sqrt{(1-\frac{m_L}{m})\frac{\mathbf{KL}(Q\|P) + \log(\frac{t(m_L,m)}{\delta})}{2m_L}}, \tag{C.3}$$

where $t(m_L,m) = 3\log(m_L)\sqrt{m_L(1-\frac{m_L}{m})}$. Since the dominant term in both bounds is the $\mathbf{KL}$ terms, the transductive setting has an improved generalization guarantee by a factor of $\sqrt{1-\frac{m_L}{m}}$.

PAC-Bayes multi-task learning Beyond standard single-task learning, it has been shown theoretically that when learning multiple related tasks, sharing information between them can improve performance. Formally, in *multi-task learning (MTL)* [Car97], there are n tasks with different distributions $D_1,\ldots,D_n$ and respective datasets $S_1,\ldots,S_n$, and the goal is to learn individual models (or Posteriors) jointly. *Meta-learning* [Sch87], or *learning to learn* [TP98], extends multi-task learning to the setting where there will also be future tasks that are not observed yet, with the assumption that the observed tasks and future tasks are all i.i.d. samples from a distribution τ over an environment of tasks [Bax00]. In multi-task learning, the goal is to generalize from the average training error to the average true risk of observed clients, and in meta-learning, the goal is to generalize to the expected true risk over τ. Given that in the training step, we can not learn a model for future tasks, meta-learning is formalized as learning an algorithm to apply to a future task. However, past works focused on multi-task learning and meta-learning in an inductive way, i.e. the model suggested by [Bax00]. In this work, we focus on a transductive version where there are n tasks, out of which n_L tasks are selected randomly to have a labeled dataset, and the remaining $n - n_L$ tasks have only unlabeled data.

Following [PL14], PAC-Bayes is a popular framework to study multi-task learning and meta-learning, given its ability to share information between tasks through the concept of a prior. In this work, we follow the framework introduced in [ZBL24]. As our theoretical contribution, we prove new bounds for the transductive multi-task learning scenario we described.

Formally, we have access to n_L labeled datasets and $n - n_L$ unlabeled datasets, and our goal is to generate models with good performance on all tasks. Since we have unlabeled tasks,

we work with a family of algorithms $\mathcal{A}$ that can create posteriors from unlabeled data i.e. $A : \mathcal{P}(\mathcal{X}) \to \mathcal{M}(\mathcal{F})$ is a mapping from the set of unlabeled datasets to models. We aim to learn a stochastic algorithm i.e. a *meta-posterior* over a set of algorithms ($\rho \in \mathcal{M}(\mathcal{A})$) that have good performance on all tasks. For each algorithm, $A(X_1), \ldots, A(X_n)$ are the task posteriors, and we denote to $\mathcal{Q}(A)$ as a hyper-posterior, a distribution over priors to capture the similarities between the outputs of the algorithm, and to have a data-dependent prior for our PAC-Bayes bounds. For a detailed explanation of the role of hyper-posterior, we refer the reader to [ZBL24].

For each meta-posterior, our goal is to minimize the average true risk of all tasks:

$$\mathsf{er}(\rho) = \mathbb{E}_{A\sim\rho}\frac{1}{n}\sum_{i=1}^{n}\mathbb{E}_{(x_i,y_i)\sim\mathcal{D}_i}\mathop{\mathbb{E}}_{f\sim A(X_i)}\ell(y_i, f(x_i)) \tag{C.4}$$

However, the true distribution of tasks is unknown, and we can not compute the training risk of unlabeled clients; therefore, the computable object is the average training risk of labeled clients:

$$\widehat{\mathsf{er}}(\rho) = \mathbb{E}_{A\sim\rho}\frac{1}{n_L}\sum_{i=1}^{n_L}\frac{1}{m}\sum_{j=1}^{m}\mathop{\mathbb{E}}_{f\sim A(X_i)}\ell(y_{i,j}, f(x_{i,j})) \tag{C.5}$$

We now state our main theoretical results:

Theorem 14. *For any fixed meta-prior π, fixed hyper-prior $\mathcal{P}$ and any $\delta > 0$ with probability at least $1-\delta$ over the sampling of the datasets, for all distributions $\rho \in \mathcal{M}(\mathcal{A})$ over algorithms, and for all hyper-posterior functions $\mathcal{Q} : \mathcal{A} \to \mathcal{M}(\mathcal{M}(\mathcal{F}))$ it holds*

$$\mathit{er}(\rho) \le \widehat{\mathit{er}}(\rho) + \sqrt{(1-\frac{n_L}{n})\frac{\mathbf{KL}(\rho||\pi) + \log(\frac{t(n_L,n)}{\delta})}{2n_L}} + \mathop{\mathbb{E}}_{A\sim\rho}\sqrt{\frac{C(A,\mathcal{Q},\mathcal{P}) + \log(\frac{8mn}{\delta}) + 1}{2mn}} \tag{C.6}$$

where,

$$C(A,\mathcal{Q},\mathcal{P}) = \mathbf{KL}(\mathcal{Q}(A)\|\mathcal{P}) \quad + \mathop{\mathbb{E}}_{P\sim\mathcal{Q}(A)}\sum_{i=1}^{n}\mathbf{KL}(A(S_i)||P) \tag{C.7}$$

Proof. For the proof, we define the following intermediate objective, which quantifies the training risk of all clients. Note that this object exists (There is a labeling for the unlabeled data, however, we do not have access to them, i.e. the loss function for unlabeled clients is well-defined, but we can not compute it.)

$$\widetilde{\mathsf{er}}(\rho) = \mathbb{E}_{A\sim\rho}\frac{1}{n}\sum_{i=1}^{n}\frac{1}{m}\sum_{j=1}^{m}\mathop{\mathbb{E}}_{f\sim A(X_i)}\ell(y_{i,j}, f(x_{i,j})) \tag{C.8}$$

We divide the proof into bounding $\mathsf{er}(\rho) - \widetilde{\mathsf{er}}(\rho)$, and bounding $\widetilde{\mathsf{er}}(\rho) - \widehat{\mathsf{er}}(\rho)$ separately. For stating our results, we use the following notations as in [ZBL24]:

- *Posterior* $\mathfrak{Q}(A, \mathcal{Q}(A))$*:* given as input an algorithm $A \in \mathcal{A}$ and a hyper-posterior mapping $\mathcal{Q} : \mathcal{A} \to \mathcal{M}(\mathcal{F})$ as input, it outputs the distribution over $\mathcal{M}(\mathcal{F}) \times \mathcal{F}^{\otimes n}$ with the following generating process: *i)* sample a prior $P \sim \mathcal{Q}(A)$, *ii)* for each task, $i = 1, \ldots, n$, sample a model $f_i \sim A(X_i)$.
- *Prior* $\mathfrak{P}$*:* For the hyper-prior $\mathcal{P} \in \mathcal{M}(\mathcal{F})$ as input, it outputs the distribution over $\mathcal{M}(\mathcal{F}) \times \mathcal{F}^{\otimes n}$ with the following generating process: *i)* sample a prior $P \sim \mathcal{P}$, *ii)* for each task, $i = 1, \ldots, n$, sample a model $f_i \sim P$.

First part is bounding $\mathsf{er}(\rho) - \widetilde{\mathsf{er}}(\rho)$ i.e. multi-task generalization bound for all n tasks. The change of objective and intermediate step we have here compared to [ZBL24] allows us to consider the generalization behavior of all tasks (labeled and unlabeled) in contrast to only the labeled tasks. For the proof, for any task i and any model f_i we define:

$$\Delta_i(f_i) = \mathop{\mathbb{E}}_{x \sim D_i} \ell(y, f_i(x)) - \frac{1}{m} \sum_{j=1}^{m} \ell(y_{i,j}, f_i(x_{i,j})) \tag{C.9}$$

By this definition and the definitions of $\widetilde{\mathsf{er}}$ and er we have:

$$\mathop{\mathbb{E}}_{(P, f_1, f_2, \ldots, f_n) \sim \mathfrak{Q}(A, \mathcal{Q}(A))} \left[\frac{1}{n} \sum_{i=1}^{n} \Delta_i(f_i) \right] = \mathsf{er}(A) - \widetilde{\mathsf{er}}(A) \tag{C.10}$$

By *change of measure inequality* [SLCB+12], for any $\lambda > 0$, any $A \in \mathcal{A}$ and any $\mathcal{Q} : \mathcal{A} \to \mathcal{M}(\mathcal{M}(\mathcal{F}))$, we have:

$$\mathsf{er}(A) - \widetilde{\mathsf{er}}(A) - \frac{1}{\lambda} \mathbf{KL}\left(\mathfrak{Q}(A, \mathcal{Q}(A)) || \mathfrak{P}\right) \leq \frac{1}{\lambda} \log \mathop{\mathbb{E}}_{(P, f_1, f_2, \ldots, f_n) \sim \mathfrak{P}} \prod_{i=1}^{n} e^{\frac{\lambda}{n} \Delta_i(f_i)} \tag{C.11}$$

Because $\mathfrak{P}$ is independent of $S_1, ..., S_n$, we have

$$\mathop{\mathbb{E}}_{S_1, \ldots, S_n} \mathop{\mathbb{E}}_{(P, f_1, f_2, \ldots, f_n) \sim \mathfrak{P}} \prod_{i=1}^{n} e^{\frac{\lambda}{n} \Delta_i(f_i)} = \mathop{\mathbb{E}}_{P \sim \mathcal{P}} \prod_{i=1}^{n} \mathop{\mathbb{E}}_{S_i} \mathop{\mathbb{E}}_{f_i \sim P} e^{\frac{\lambda}{n} \Delta_i(f_i)} \tag{C.12}$$

By Hoeffding's lemma for $\Delta_i(f_i) \in [0, 1]$, we have

$$\mathop{\mathbb{E}}_{S_i} \mathop{\mathbb{E}}_{f_i \sim P} e^{\frac{\lambda}{n} \Delta_i(f_i)} \leq e^{\frac{\lambda^2}{8n^2 m}}. \tag{C.13}$$

Therefore by combining (C.12) and (C.13) we have:

$$\mathop{\mathbb{E}}_{S_1, \ldots, S_n} \mathop{\mathbb{E}}_{(P, f_1, f_2, \ldots, f_n) \sim \mathfrak{P}} \prod_{i=1}^{n} e^{\frac{\lambda}{n} \Delta_i(f_i)} \leq e^{\frac{\lambda^2}{8nm}}. \tag{C.14}$$

By Markov's inequality, for any $\epsilon > 0$ we have

$$\mathbb{P}_{S_1, \ldots, S_n}\left(\mathop{\mathbb{E}}_{(P, f_1, f_2, \ldots, f_n) \sim \mathfrak{P}} \prod_{i=1}^{n} e^{\frac{\lambda}{n} \Delta_i(f_i)} \geq e^{\epsilon} \right) \leq e^{\frac{\lambda^2}{8nm} - \epsilon} \tag{C.15}$$

Hence by combining (C.11) and (C.15) we get

$$\mathbb{P}_{S_1,\dots,S_n}\Big(\exists A, \mathcal{Q} : \mathsf{er}(A) - \widetilde{\mathsf{er}}(A) - \frac{1}{\lambda}\,\mathbf{KL}(\mathfrak{Q}(A,\mathcal{Q}(A))||\mathfrak{P}(\mathcal{P})) \geq \frac{1}{\lambda}\epsilon\Big) \leq e^{\frac{\lambda^2}{8nm}-\epsilon}, \quad \text{(C.16)}$$

or, equivalently, it holds for any $\delta > 0$ with probability at least $1 - \frac{\delta}{2}$:

$$\forall A, \mathcal{Q} : \quad \mathsf{er}(A) - \widetilde{\mathsf{er}}(A) \leq \frac{1}{\lambda}\,\mathbf{KL}(\mathfrak{Q}(A,\mathcal{Q}(A))||\mathfrak{P}) + \frac{1}{\lambda}\log(\frac{2}{\delta}) + \frac{\lambda}{8nm} \quad \text{(C.17)}$$

With a union bound for $\lambda \in \{1,\dots,4mn\}$, and choosing the best λ we get:

$$\mathbb{P}_{S_1,\dots,S_n}\Big(\forall A, \mathcal{Q} : \mathsf{er}(A) - \widetilde{\mathsf{er}}(A) \leq \sqrt{\frac{\mathbf{KL}\left(\mathfrak{Q}(A,\mathcal{Q}(A))||\mathfrak{P}\right) + \log(\frac{8mn}{\delta}) + 1}{2mn}}\Big) \geq 1 - \frac{\delta}{2} \quad \text{(C.18)}$$

With probability at least $1 - \frac{\delta}{2}$, the bound holds for all $A \in \mathcal{A}$, therefore, we can take the expectation for $A \in \rho$. Given that $\mathbb{E}_{A\sim\rho}[\mathsf{er}(A) - \widetilde{\mathsf{er}}(A)] = \mathsf{er}(\rho) - \widetilde{\mathsf{er}}(\rho)]$, the following holds with probability at least $1 - \frac{\delta}{2}$, for all ρ, and $\mathfrak{Q}$:

$$\forall \rho, \mathcal{Q} : \widetilde{\mathsf{er}}(\rho) - \widehat{\mathsf{er}}(\rho) \leq \mathop{\mathbb{E}}_{A\sim\rho}\sqrt{\frac{\mathbf{KL}(\mathfrak{Q}(A,\mathcal{Q}(A))||\mathfrak{P}) + \log(\frac{8mn}{\delta}) + 1}{2mn}}. \quad \text{(C.19)}$$

For $\mathbf{KL}(\mathfrak{Q}(A,\mathcal{Q}(A))||\mathfrak{P})$, we have:

$$\begin{aligned}
\mathbf{KL}(\mathfrak{Q}(A,\mathcal{Q}(A))||\mathfrak{P}) &= \mathop{\mathbb{E}}_{P\sim\mathcal{Q}(A)}\Big[\mathop{\mathbb{E}}_{f_i\sim A(X_i)} \ln \frac{\mathcal{Q}(A)(P)\prod_{i=1}^n A(X_i)(f_i)}{\mathcal{P}(P)\prod_{i=1}^n P(f_i)}\Big] \\
&= \mathop{\mathbb{E}}_{P\sim\mathcal{Q}(A)}\Big[\ln\frac{\mathcal{Q}(A)(P)}{\mathcal{P}(P)}\Big] + \mathop{\mathbb{E}}_{P\sim\mathcal{Q}(A)}\Big[\sum_{i=1}^n \mathop{\mathbb{E}}_{f_i\sim A(S_i)} \ln\frac{A(X_i)(f_i)}{P(f_i)}\Big]\Big] \\
&= \mathbf{KL}(\mathcal{Q}(A)\|\mathcal{P}) + \mathop{\mathbb{E}}_{P\sim\mathcal{Q}(A)}\sum_{i=1}^n \mathbf{KL}(A(S_i)||P) \quad \text{(C.20)}
\end{aligned}$$

Combining equations (C.19) and (C.20) gives that with probability $1-\frac{\delta}{2}$, for all $\rho \in \mathcal{M}(\mathcal{A})$, $\mathcal{Q} : \mathcal{A} \to \mathcal{M}(\mathcal{M}(\mathcal{F}))$:

$$\widetilde{\mathsf{er}}(\rho) - \widehat{\mathsf{er}}(\rho) \leq \mathop{\mathbb{E}}_{A\sim\rho}\sqrt{\frac{C(A,\mathcal{Q},\mathcal{P}) + \log(\frac{8mn}{\delta}) + 1}{2mn}} \quad \text{(C.21)}$$

For the second part, we apply the transductive bound (C.3) to generalization between $\widetilde{\mathsf{er}}(\rho)$ and $\widehat{\mathsf{er}}(\rho)$, we get:

$$\mathsf{er}(\rho) \leq \widehat{\mathsf{er}}(\rho) + \sqrt{(1-\frac{n_L}{n})\frac{\mathbf{KL}(\rho||\pi) + \log(\frac{t(n_L,n)}{\delta})}{2n_L}} \quad \text{(C.22)}$$

Combining (C.21) and (C.22) proves the theorem. □

From this general theorem, we can now prove the generalization bound for our algorithm FLowDUP.

Theorem 4. *For all $\delta > 0$, and any loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, 1]$, the following statement holds with probability at least $1-\delta$ over the sampling of n_L clients of n clients and randomness of the dataset. For all parameter vectors, $\psi = (\psi_h, \psi_r)$:*

$$\begin{aligned} \mathit{er}(\rho_h) \leq \widehat{\mathit{er}}(\rho_h) \; + \; & \sqrt{\left(1 - \frac{n_L}{n}\right) \frac{\frac{1}{2\alpha_h}\|\psi_h\|^2 + c_1}{2n_L}} \\ + \; & \mathop{\mathbb{E}}_{\psi'_h \sim \rho} \sqrt{\frac{\frac{1}{2\alpha_\theta} \sum_{i=1}^{n} \mathbb{E}_{\psi'_r} \|h(S_i\,;\,\psi'_h) - \psi'_r\|^2 + \frac{1}{2\alpha_r}\|\psi_r\|^2 + c_2}{2mn}} \end{aligned} \tag{5.8}$$

where c_1 and c_2 are logarithmic terms in n and n_L.

Proof. For each hypernetwork with trainable parameters ψ_h, consider an algorithm A that for task i generates $A(S_i) = \mathcal{N}(h(S_i\,;\,\psi_h)\,;\,\alpha_\theta \mathsf{Id})$.

For each $v \in \mathbb{R}^k$, the corresponding prior is $P = \mathcal{N}(v; \alpha_\theta \mathsf{Id})$, and $\mathcal{P} = \mathcal{N}(0; \alpha_r \mathsf{Id})$ is the hyper-prior over priors i.e. over their mean. Define the meta-prior as $\pi = \mathcal{N}(0; \alpha_h \mathsf{Id})$ over $\mathcal{A}$ i.e. over hypernetwork parameters ψ_h.

Define the meta-posterior as a Gaussian distribution over the parameters of the hypernetwork, $\rho_h = \mathcal{N}(\psi_h\,;\,\alpha_h \mathsf{Id})$ for a fixed α_h. Define the hyper-posterior as $\mathcal{Q} = \mathcal{N}(\psi_r\,;\,\alpha_r \mathsf{Id})$.

By applying the theorem 14 to the defined distributions, we complete the proof. □

APPENDIX D

Appendix for Chapter 6

D.1 Extended related work

Clustering Gaussian mixture The problem of clustering Gaussian mixtures is a fundamental of statistics, perhaps dating back from the work of [Pea94]. Estimating the parameters of the mixture, as we are trying to in this paper, has a rich history. [MV10] showed that, even non-privately, the sample complexity has to be exponential in k; the standard way to bypass this hardness is to require some separation between the means of the different components. If this separation is $o(\sqrt{\log k})$, then any algorithm still requires a non-polynomial number of samples [RV17]. When the separation is just above this threshold, namely $O(\log(k)^{1/2+c})$, [LL22] present a polynomial-time algorithm based on Sum-of-Squares to recover the means of *spherical* Gaussians. For clustering general Gaussians, the historical approach is based solely on statistical properties of the data, and requires a separation $\Omega(\sqrt{k})$ times the maximal variance of each component [AM05, AS12]. This separation is necessary for accurate clustering, namely, if one aims at determining from which component each samples is from [DKK+22]. This approach has been implemented privately by [KSSU19] (with the additional assumption that the input is in a bounded area): this is the one we follow, as the simplicity of the algorithms allows to have efficient implementation in a Federated Learning environment. [BKS22] studied how public data can improve performances of this private algorithm: they assume access to a small set of samples from the distribution, which improves the sample complexity and allows them to remove the assumption that the input lies in a bounded area. We note that both private works of [KSSU19] and [BKS22] have a separation condition that grows with $\log m$, as ours. To only recover the means of the Gaussians, and not the full clustering, a separation of k^α (for any $\alpha > 0$) is enough [HL18, KSS18, ST21]. This is also doable privately (when additionally the input has bounded diameter) using the approach of [CKM+21] and [TCK+22]. Those works are hard to implement efficiently in our FL framework for two reasons: first, they rely on Sum-of-Square mechanisms, which does not appear easy to implement efficiently. Second, they use Single Linkage as a subroutine: this does not seem possible to implement in FL. Therefore, some new ideas would be necessary to get efficient algorithm for FL based on this approach. A different and orthogonal way of approaching the problem of clustering Gaussian mixtures is to recover a distribution that is ε-close to the mixture in total variation distance, in which case the algorithm of [ABDH+20] has optimal sample complexity $\tilde{O}(kd^2/\varepsilon^2)$ – albeit with a running time $\omega(\exp(kd^2))$.

On private k-means clustering The private k-means algorithm of [DITHS24], implemented in our FL setting, would require either $\Omega(k)$ rounds of communication with the clients (for simulating their algorithm for central DP algorithm), or a a very large amount of additive noise $k^{O(1)}$ (for their local DP algorithm, with an unspecified exponent in k). Furthermore, the algorithm requires to compute a net of the underlying Euclidean space, which has size exponential in the dimension, and does not seem implementable. To the best of our knowledge, the state-of-the-art implementation of k-means clustering is from [CK21]: however, it has no theoretical guarantee, and is not tailored to FL.

D.2 Technical preliminaries

D.2.1 Differential Privacy definitions and basics

As mentioned in introduction, one of the most important properties of Differential Privacy is the ability to compose mechanisms. There are two ways of doing so. First, parallel composition: if an (ε, δ)-DP algorithms is applied on two distinct datasets, then the union of the two output is also (ε, δ)-DP. Formally, the mechanism that takes as input two elements $P_1, P_2 \in \mathcal{P}$ and outputs $(\mathcal{A}(P_1), \mathcal{A}(P_2))$ is (ε, δ)-DP. The second property is sequential composition: applying an (ε, δ)-DP algorithm to the output of another (ε, δ)-DP algorithm is $(2\varepsilon, 2\delta)$-DP. Formally: if $\mathcal{A} : \mathcal{P} \to \mathcal{S}_A$ is $(\varepsilon_A, \delta_A)$-DP and $\mathcal{B} : \mathcal{P} \times \mathcal{S}_A \to \mathcal{S}_B$ is $(\varepsilon_B, \delta_B)$-DP, then $\mathcal{B}(\mathcal{A}(\cdot), \cdot) : \mathcal{P} \to \mathcal{S}_B$ is $(\varepsilon_A + \varepsilon_B, \delta_A + \delta_B)$-DP. Those are the composition theorem that we use for the theoretical analysis. We chose for simplicity not to use advanced composition: the number of steps in our final algorithms Algorithm 14 and Algorithm 15 is logarithmic, hence the improvement would be marginal. However, in practice, better bounds can be computed – although they don't have closed-form expression. We use a standard algorithm to estimate more precise upper-bounds on the privacy parameters of our algorithms [KOV15]. The sensitivity of a function is a key element to know how much noise is needed to add in order to make the function DP. Informally, the sensitivity measures how much the function can change between two neighboring datasets. Formally, we have the following definition.

Definition 15 (Sensitivity)**.** *Given a norm $\ell : \mathbb{R}^d \to \mathbb{R}$, the ℓ-sensitivity of a function $f : \mathcal{X}^m \to \mathbb{R}^d$ is defined as*

$$\sup_{x \sim X' \in \mathcal{X}^m} \ell(f(X) - f(X')),$$

where $X \sim X'$ means that X and X' are neighboring datasets.

The two most basic private mechanism are the Laplace and Gaussian mechanism, that make a query private by adding a simple noise. We use the Laplace mechanism for counting:

Lemma 16 (Laplace Mechanism for Counting.)**.** *Let X be a dataset. Then, the mechanism $M(X) = |X| + \mathrm{Lap}(1/\varepsilon)$ is $(\varepsilon, 0)$-DP, where $\mathrm{Lap}(1/\varepsilon)$ is a variable following a Laplace distribution with variance $1/\varepsilon$.*

We use the Gaussian mechanism for more general purposes (e.g., the PCA step). It is defined as follows:

Lemma 17. *Gaussian Mechanism Let $f : \mathcal{X} \to \mathbb{R}^n$ be a function with ℓ_2-sensitivity $\Delta_{f,2}$. Then, for $\sigma(\varepsilon, \delta) = \frac{\sqrt{2\log(2/\delta)}}{\varepsilon}$ the Gaussian mechanism $M(X) = f(X) + \mathcal{N}_d\left(0, \Delta_{f,2}^2 \sigma(\varepsilon, \delta)^2\right)$ is (ε, δ)-DP, where $\mathcal{N}_d(0, \sigma^2)$ is a d-dimensional Gaussian random variable, where each dimension is independent with mean 0 and variance σ^2.*

Combining those two mechanisms gives a private and accurate estimate for the average of a dataset

Lemma 18 (Private averaging). *For dataset X in the ball $B(0,\Delta)$, the mechanism $M(X) := \frac{\sum_{x\in X} X+\mathcal{N}_d(0,\Delta_{f,2}\sigma^2(\varepsilon/2,\delta))}{|X|+\mathrm{Lap}(2/\varepsilon)}$ is (ε,δ)-DP. Additionally, $|X|\geq$, then it holds with probability $1-\beta$ that $\|M(X)-\mu(X)\|_2 \leq \frac{\Delta\ln(2/\beta)}{|X|\varepsilon} + \frac{\Delta\sigma(\varepsilon/2,\delta)\sqrt{\ln(2/\beta)}}{|X|}$.*

D.2.2 Differentially Private tools for Gaussian mixtures

First, we review some properties of the private rank-k approximation: this algorithm was analyzed by [DTTZ14], and its properties when applied on Gaussian mixtures by [KSSU19]. The guarantee that is verified by the projection onto the noisy eigenvectors is the following:

Definition 19. *Fix a matrix $X\in\mathbb{R}^{d\times m}$, and let Π_k be the projection matrix onto the principal rank-k subspace of XX^T. For some $B\geq 0$, we say that a matrix Π is a B-almost k-PCA of X if Π is a projection such that:*

- $\|XX^T-(\Pi X)(\Pi X)^T\|_2 \leq \|XX^T-(\Pi_k X)(\Pi_k X)^T\|_2 + B$, *and*
- $\|XX^T-(\Pi X)(\Pi X)^T\|_F \leq \|XX^T-(\Pi_k X)(\Pi_k X)^T\|_F + kB$.

[DTTZ14] shows how to compute a B-almost k-PCA, with a guarantee on B that depends on the diameter of the dataset:

Theorem 20 (Theorem 9 of [DTTZ14]). *Let $X\in\mathbb{R}^{d\times m}$ such that $\|X_i\|_2\leq 1$, and fix $\sigma(\varepsilon,\delta)=\sqrt{2\ln(2/\delta)}/\varepsilon$. Let $E\in R^{d\times d}$ be a symmetric matrix, where each entry $E_{i,j}$ with $j\geq i$ is an independent draw from $\mathcal{N}(0,\sigma(\varepsilon,\delta)^2)$. Let Π_k be the rank-k approximation of XX^T+E. Then, Π_k is a $O(\sqrt{d}\cdot\sigma(\varepsilon,\delta))$-almost k-PCA of X, and is (ε,δ)-DP.*

[KSSU19] shows crucial properties of Gaussian mixtures: first, the projection of each empirical mean with a B-almost k-PCA is close to the empirical mean:

Lemma 21 (Lemma 3.1 in [KSSU19]). *Let $X\in\mathbb{R}^{d\times m}$ be a collection of points from k clusters centered at $\mu_1,...,\mu_k$. Let C be the cluster matrix, namely $C_p=\mu_j$ if X_p belongs to the j-th cluster, and G_j be the j-th cluster. Let Π_k be a B-almost k-PCA, and denote $\bar{\mu_1},...,\bar{\mu_k}$ the empirical means of each cluster, and $\tilde{\mu_1},...,\tilde{\mu_k}$ the projected empirical means. Then, $\|\bar{\mu_j}-\tilde{\mu_j}\|\leq\frac{1}{\sqrt{|G_j|}}\|X-C\|_2+\sqrt{\frac{B}{|G_j|}}$.*

Second – and this helps bounding the above – they provide bounds on the spectral norm of the clustering matrix $X-C$:

Lemma 22 (Lemma 3.2 in [KSSU19]). *Let $X\in\mathbb{R}^{d\times m}$ be a set of m samples from a mixture of k Gaussians. Let σ_j be the maximal unidirectional variance of the j-th Gaussian, and $\sigma_{max}=\max\sigma_j$. Let C be the cluster matrix, namely $C_p=\mu_j$ if X_p is sampled from $\mathcal{N}(\mu_j,\Sigma_j)$. If $m\geq\frac{1}{w_{min}}(\zeta_1 d+\zeta_2\log_2(k/\beta))$, where ζ_1,ζ_2 are some universal constants, then with probability $1-\beta$ it holds that*

$$\frac{\sqrt{mw_{min}}\sigma_{max}}{4}\leq\|X-C\|_2\leq 4\sqrt{m\sum_{j=1}^{k}w_j\sigma_j^2}.$$

D.2.3 Properties of Gaussian mixtures

Lemma 23. *Consider a set P of m samples from a Gaussian mixtures $\{(\mu_j, \Sigma_j, w_j)\}_{j\in[k]}$. Let G_j be the set of points sampled from the j-th component. If $m \geq \frac{24\log(k)}{w_{min}}$, then with probability 0.99 it holds that $\forall j, |G_j| \geq mw_j/2$*

Proof. This is a direct application of Chernoff bounds: each sample s is in G_j with probability w_j. Therefore, the expected size of G_j is mw_j, and with probability at least $1 - 2\exp(-mw_j/12)$ it holds that $\left||G_j| - mw_j\right| \leq mw_j/2$: for $m \geq 24\log(k)/w_{\min}$, the probability is at least $1 - 2/k^2$. A union-bound over all j concludes. □

D.2.4 Clustering preliminaries

Our algorithm first replaces the full dataset P with a weighted version of Q, and then computes a k-means solution on this dataset. The next lemma shows that, if $\mathrm{cost}(P, Q)$ is small, then the k-means solution on the weighted Q is a good solution for P:

Lemma 24. *Let $P, C_1 \subset \mathbb{R}^d$, and $f : P \to C_1$ be a mapping with $\Gamma := \sum_{p\in P} \|p - f(p)\|^2$. Let w_ν be such that $|w_\nu - |f^{-1}(\nu)|| \leq |f^{-1}(\nu)|/2$. Let $\tilde{P}$ be the multiset where each $\nu \in C_1$ appears w_ν many times, . Let C_2 be such that $\mathrm{cost}(\tilde{P}, C_2) \leq \alpha OPT(\tilde{P})$. Then,*

$$\mathrm{cost}(P, C_2) \leq (2 + 12\alpha)\Gamma + 12\alpha OPT(P).$$

Proof. Recall that $C_2(p)$ is the closest point in C_2 to p. We have, using triangle inequality:

$$\begin{aligned}
\mathrm{cost}(P, C_2) &= \sum_{p\in P} \|p - C_2(p)\|^2 \\
&\leq \sum_{p\in P} \|p - C_2(f(p))\|^2 \\
&\leq \sum_{p\in P} \left(\|p - f(p)\| + \|f(p) - C_2(f(p))\|\right)^2 \\
&\leq \sum_{p\in P} 2\|p - f(p)\|^2 + 2\|f(p) - C_2(f(p))\|^2 \\
&\leq 2\Gamma + 2\sum_{\nu\in C_1} |f^{-1}(\nu)|\|\nu - C_2(\nu)\|^2 \\
&\leq 2\Gamma + 4\sum_{\nu\in C_1} w_\nu\|\nu - C_2(\nu)\|^2 \\
&\leq 2\Gamma + 4\alpha\mathsf{OPT}(\tilde{P}).
\end{aligned}$$

A similar argument bounds $\mathsf{OPT}(\tilde{P})$: let C^* be the optimal solution for P, then, for any point p we have $\|f(p) - C^*(f(p))\| \leq \|f(p) - C^*(p)\| \leq \|f(p) - p\| + \|p - C^*(p)\|$. Therefore,

$$\begin{aligned}
\mathsf{OPT}(\tilde{P}) &\leq \sum_{\nu\in C_1} w_\nu\|\nu - C*(\nu)\|^2 \\
&\leq \frac{3}{2}\sum_{\nu\in C_1} |f^{-1}(\nu)|\|\nu - C*(\nu)\|^2 \\
&\leq 3\sum_{p\in P} 2\|C_1(p) - p\|^2 + 2\|p - C^*(p)\|^2 \\
&\leq 3\Gamma + 3\mathsf{OPT}(P).
\end{aligned}$$

Combining those two inequalities concludes the lemma. □

D.3 The non-private, non-federated algorithm of [AS12]

The algorithm we take inspiration from is the following, from [AS12] and inspired by [KK10]: first, project the dataset onto the top-k eigenvectors of the dataset, and compute a constant-factor approximation to k-means (e.g., using local search). Then, improve iteratively the solution with Lloyd's steps. The pseudo-code of this algorithm is given in Algorithm 13, and the main result of [AS12] is the following theorem:

Theorem 25 ([AS12])**.** *For a separated Gaussian mixture, Algorithm 13 correctly classifies all point w.h.p.*

Their result is more general, as they do not require the input to be randomly generated, and only requires a strict separation between the clusters. In this paper, we focus specifically on Gaussian mixtures.

Algorithm 13 Cluster(P)

1: **Part 1:** find initial Clusters

a) Compute $\hat{P}$ the projection of P onto the subspace spanned by the top k singular vectors of P.

b) Run a c-approximation algorithm for the k-means problem on $\hat{P}$ to obtain centers $\nu_1, ..., \nu_k$.

2: **Part 2:** For $j = 1, ...k$, set $S_j \leftarrow \{p : \forall s, \|\hat{P}_p - \nu_j\| \leq \frac{1}{3}\|\hat{P}_p - \nu_s\|\}$ and $\theta_j \leftarrow \mu(S_j)$

3: **Part 3:** Repeat Lloyd's steps until convergence:
for $j = 1, ...k$, set $C(\nu_j) \leftarrow \{p : \forall s, \|P_p - \nu_j\| < \|P_p - \nu_s\|\}$, and $\theta_j \leftarrow \mu(C(\nu_j))$

D.4 Our result

Our main theoretical results is to adapt Algorithm 13 to a private and federated setting. As mention in the main body, we show a stronger version of Theorem 7, without assumptions on diameter and server data. More precisely, we show the following theorem:

Theorem 26. *There is an (ε, δ)-DP algorithm with the following accuracy guarantee. Suppose that the client dataset P is generated from a separated Gaussian mixtures with $m \geq \zeta_1 \frac{kdT \log^2 m \cdot \sqrt{\ln(1/\delta)}}{\varepsilon^2 w_{min}^2}$ samples, where ζ_1 is some universal constant, and that Q contains a least one sample from component of the mixture. Then, the algorithm computes centers $\nu_1, ..., \nu_k$ such that, for some universal constants ζ_2, ζ_3, after $T + \zeta_2 \log \frac{\sigma_{max} \log |Q|}{\varepsilon w_{min}}$ rounds of communications, it holds with high probability that:*

$$\|\mu_j - \nu_j\| \leq \zeta_3 \max\left(\frac{1}{2^T}, \frac{kdT \log^2 m \sigma_{max} \sqrt{\ln(T/\delta)}}{m\varepsilon^2 w_{min}^2}\right).$$

Note that the precision increases with the number of samples: if m is larger than

$$\frac{2^T \log(\sigma_{\max}/w_{\min}) kd \log^2 m\sigma_{\max}}{\varepsilon^2 w_{\min}^2},$$

then the dominating term is $1/2^T$.

Corollary 27. *There is an (ε, δ)-DP algorithm with $O(\log(m))$ rounds of communications with the following accuracy guarantee. Suppose that the client dataset P is generated from a separated Gaussian mixtures with $m = \Omega\left(\frac{k \log^2 m\sqrt{d}\sigma_{max}}{\varepsilon^2 w_{min}^2}\right)$ samples, that Q contains a least one sample from each component of the mixture and at most m data points. Suppose that $m = \Omega\left(\frac{k \log^3 m\sqrt{d}\sigma_{max}}{\varepsilon^2 w_{min}^2}\right)$, and that $m = \Omega\left(\frac{\log(m)^6 \cdot kd^2}{\varepsilon^4 w_{min}^2}\right)$. Then, the algorithm computes centers $\nu_1, ..., \nu_k$ such that, with high probability, the clustering induced by $\nu_1, ..., \nu_k$ is the partition $G_1, ..., G_k$.*

Proof. Theorem 5.4 of [KK10] (applied to Gaussian mixtures) bounds the number of misclassified points in a given cluster in terms of the distance between ν_j and μ_j. Define, for any j, S_j as the cluster of ν_j, and $\delta_j = \|\mu_j - \nu_j\|$. Then, for $j' \neq j$, [KK10] show that, for some constant c':

$$|G_j \cap S_{j'}| \leq \frac{c' m w_{\min}(\delta_j^2 + \delta_{j'}^2)}{\|\mu_j - \mu_{j'}\|^2} 1$$

Since $\|\mu_j - \mu_{j'}\|^2 \geq c^2 \frac{k\sigma_{\max}^2 \log(m)^2}{w_{\min}}$, we get that the number of points from G_j assigned to cluster j' is at most $\frac{c' m w_{\min}^2(\delta_j^2+\delta_{j'}^2)}{k\sigma_{\max}^2 \log(m)^2}$. We aim at bounding δ_j and $\delta_{j'}$ using Theorem 26. For $T = \log\left(\frac{10c' m w_{\min}}{k\sigma_{\max}}\right)$, it holds that $\frac{1}{2^T} \leq \frac{\sqrt{k}\sigma_{\max}}{10c'\sqrt{m}w_{\min}}$. In addition, for this value of T and a number of samples m at least $m \geq \frac{100c'^2 \log(m)^2 \cdot kd^2 \log(m)^4}{\varepsilon^4 w_{\min}^2}$, we also have $\frac{kdT \log^2 m\sigma_{\max}\sqrt{\ln(T/\delta)}}{m\varepsilon^2 w_{\min}^2} \leq \frac{\sqrt{k}\sigma_{\max}}{10c'\sqrt{m}w_{\min}}$. Therefore, the upper bound on δ_j and $\delta_{j'}$ from Theorem 26 after $T + \log(\sigma_{\max} \log|Q|/w_{\min}) = O(\log(m))$ rounds of communications ensure that there is no point misclassified. This which concludes the statement. □

In the case where the assumption of Theorem 7 are satisfied, namely, (1) the diameter is bounded and (2) the server data are well spread, then the algorithm of Theorem 26 reduces directly to Algorithm 11 followed with T steps of Algorithm 12, with only T rounds of communication. Indeed, the first $O\left(\log \frac{\sigma_{\max} \log|Q|}{\varepsilon w_{\min}}\right)$ rounds of the algorithm from Theorem 26 are dedicated to enforcing condition (1) and (2): if they are given, there is no need for those steps. The organization of the proof is as follows. First, we give some standard technical preliminary tools about differential privacy and Gaussian mixtures. Then, we show how to implement Algorithm 13: the bulk of the work is in the implementation of its Part 1, computing a good solution for ΠP. The second part to iteratively improve the solution is very similar to the non-private part.

[1] We simplified the original statement of [KK10] to directly adapt it to separated Gaussian mixtures: in this case, $\|P - C\|_2^2 \leq 4m\sigma_{\max}^2$, and $\Delta_{i,j}$ (defined in the original statement) is our separation value, $c\sqrt{k/w_{\min}}\sigma_{\max} \log(m)$.

D.5 Part 1: Computing centers close to the means

D.5.1 Reducing the diameter

Lemma 28. *There is an ε-DP algorithm with one communication round that, given w_{min} and σ_{max}, reduces the diameter of the input to $O\left(\frac{\log|Q|\log m\sqrt{d}\sigma_{max}}{\varepsilon w_{min}}\right)$.*

Proof. We fix a distance $D = 4\log m\sqrt{d}\sigma_{\max}$. First, the server identifies regions that contains many server points: if q is such that $|Q \cap B(q, D)| \geq \frac{\varepsilon m w_{\min}}{200\log|Q|}$, then q is marked *frozen*. Then, each client assigns its points to their closest server point in Q, breaking ties arbitrarily. In one round of communication, the server learns, for each server point $q \in Q$, the noisy number of points assigned to q, namely $\hat{w}_q(P) = w_q(P) + \mathrm{Lap}(1/\varepsilon)$. For privacy, the noise added to each count follows a Laplace distribution with parameter $1/\varepsilon$. Hence, with high probability, the noise is at most $O\left(\frac{\log|Q|}{\varepsilon}\right)$ on each server data q. With high probability on the samples, for all j the $B(\mu_j, D)$ contains all the $w_j m$ samples from $\mathcal{G}_j$. Therefore, any server point q sampled from $\mathcal{G}_j$ is either frozen, or the noisy count in $B(q, D)$ ball is at least $mw_{\min}/2 - |Q \cap B(\mu_j, D)| \cdot \frac{\log|Q|}{\varepsilon} \geq mw_{\min}/3$, using Lemma 23.

Consider now an arbitrary point $p \in \mathbb{R}^d$. Since G_j is fully contained in $B(\mu_j, D/2)$, either the ball $B(p, D/2)$ doesn't intersect with G_j, or $B(p, D)$ contains entirely G_j. Furthermore, by triangle inequality, for any $q \in G_j \cap Q$ the ball $B(q, D)$ contains entirely G_j: if q is not frozen, it has noisy count at least $mw_{\min}/3$, and therefore true count at least $mw_{\min}/6$. To reduce the diameter, we first remove all points from Q that are not frozen and for which the ball $B(q, D)$ has noisy count less than $mw_{\min}/3$: by the previous discussion, those points are not sampled from any $\mathcal{G}_j$ and are part of the noise. In addition, connect any pair of points that are at distance less than D. We claim that each connected component has diameter at most $O\left(\frac{\log^2 m\sqrt{d}\sigma_{\max}}{\varepsilon w_{\min}}\right)$. To prove this claim, we fix such a component, and consider the following iterative procedure. Pick an arbitrary point from the component, and remove all points that are at distance $2D$. Repeat those two steps until there are no more points. Let q be a point selected at some step of this procedure. First, note that $B(q, D)$ is disjoint from any ball $B(q', D)$, for q' previously selected – as $B(q', 2D)$ has been removed. Furthermore, either q is frozen and the ball contains $\frac{\varepsilon m w_{\min}}{200\log|Q|}$ many points of Q, or q is not frozen and $B(q, D)$ contains at least $mw_{\min}/6$ points of P. Therefore, there are at most $t_{max} := \frac{6}{w_{\min}} + \frac{200\log|Q|}{\varepsilon w_{\min}}$ iterations. So the connected component can be covered with t_{max} balls of radius $2D$. Additionally, since each edge has length at most D, the component has diameter at most $O(t_{max}D) = O\left(\frac{\log|Q|\log m\sqrt{d}\sigma_{\max}}{\varepsilon w_{\min}}\right)$. This concludes the claim.

The other key property of the connected component is that each G_j is fully contained in a single connected component, as all points of G_j are at distance at most D of each other. Therefore, we can transform the space such that the connected components get closer but do not interact, so that the diameter reduces while the centers of Gaussians are still far apart. Formally, let D' be the maximum diameter of the connected components. Select an arbitrary representative in Q from each connected component, and apply a translation to the connected component such that its representative has coordinate $(100D' \cdot i, 0, 0, ..., 0)$. This affine transformation ensures that (1) within each connected component, all means are still separated and the points are still drawn from Gaussian with the same covariance matrix and (2) the separation between centers of different component is at least $50D'$. Therefore, the

instance constructed still satisfy the separation conditions of Definition 6, and has diameter at most $O(kD') = O\left(\frac{k \log m \log |Q| \sqrt{d} \sigma_{\max}}{\varepsilon w_{\min}}\right)$ □

D.5.2 A relaxation of Awathi-Sheffet's conditions

The result of [AS12], applied to Gaussian, requires a slightly weaker separation between the centers than what we enforce. They consider a dataset P sampled from a Gaussian mixtures, and with cluster matrix C (namely, $C_p = \mu_j$ if P_p is sampled from the j-th component). They define for each cluster $\Delta_j^{AS} := \frac{1}{\sqrt{|G_j|}} \min(\sqrt{k}\|P - C\|_2), \|P - C\|_F)$, and require $\|\mu_j - \mu_{j'}\| \geq c(\Delta_j^{AS} + \Delta_{j'}^{AS})$ for some large constant c.

In the Gaussian setting, we have $|G_j| \approx m w_j$ (Lemma 23), $\|P - C\|_2 = O(\sigma_{\max}\sqrt{m})$ Lemma 22 and $\|A - C\|_F = \Theta(\sqrt{md}\sigma_{\max})$. Thus, in most cases, $\min\left(\sqrt{k}\|P - C\|_2, \|P - C\|_F\right) = \sqrt{mk}\sigma_{\max} \operatorname{polylog}(d/w_{\min})$, except in some degenerate cases – and we keep the minimum only to fit with the proof of [AS12]. We can define $\Delta_j = \frac{\sigma_{\max}\sqrt{m}}{\sqrt{|G_j|}} \min\left(\sqrt{k} \operatorname{polylog}(d/w_{\min}), \sqrt{d}\right)$: our separation condition Definition 6 ensures that $\|\mu_j - \mu_{j'}\| \geq c(\Delta_j + \Delta_{j'})$, for some large c. We now show the two key lemmas from [AS12], adapted to our private algorithm.

Fact 29 (Fact 1.1 in [AS12])**.** *Let $P \in \mathbb{R}^{d \times m}$ be a set of m points sampled from a Gaussian mixtures, and let C be the cluster matrix, namely the p-th column is $C_p = \mu_j$ if X_p is sampled from $\mathcal{N}(\mu_j, \Sigma_j)$. Let Π be a B-almost k-PCA for $P_1, ..., P_m$. Suppose that B satisfies $B \leq \frac{\sqrt{m w_{min}}\sigma_{max}}{4k}$. Then:*

$$\|\Pi P - C\|_F^2 \leq 20 \min(k\|A - C\|_2^2, \|A - C\|_F^2) (= m w_j \Delta_j^2).$$

Proof. First, since both ΠP and C have rank k, it holds that $\|\Pi P - C\|_F^2 \leq 2k\|\Pi P - C\|_2^2$. By triangle inequality, this is at most $2k\left(\|\Pi P - P\|_2 + \|P - C\|_2\right)^2$. Now, we have that $\|\Pi P - P\|_2^2 = \|(\Pi P - P)^T(\Pi P - P)\|_2$: since Π is an orthogonal projection, $\Pi = \Pi^T = \Pi^2$ and therefore $\|\Pi P - P\|_2^2 = \|PP^T - (\Pi P)(\Pi P)^T\|_2$. Using that Π is a B-almost k-PCA of P, this is at most $\|(\Pi_k P - P)(\Pi_k P - P)^T\|_2 + B$, where $\Pi_k P$ is the best rank-k approximation to P. By definition of Π_k, this is equal to $\|P - \Pi_k P\|_2^2 + B \leq \|P - C\|_2^2 + B$. Overall, we get using $\sqrt{a+b} \leq \sqrt{a} + \sqrt{b}$:

$$\begin{aligned}
\|\Pi P - C\|_F^2 &\leq 2k\left(\|\Pi P - P\|_2 + \|P - C\|_2\right)^2 \\
&\leq 2k\left(2\|P - C\|_2 + \sqrt{B}\right)^2 \\
&\leq 16k\|P - C\|_2^2 + 4kB.
\end{aligned}$$

Using Lemma 22 and the assumption that $4kB \leq \sqrt{m w_{\min}}\sigma_{\max}$ concludes the first part of the lemma. For the other term, we have $\|\Pi P - C\|_F \leq \|\Pi P - P\|_F + \|P - C\|_F$. The fact that Π is a B-almost k-PCA ensures that $\|\Pi P - P\|_F^2 \leq \|P - C\|_F^2 + kB$; and the fact that $\|P - C\|_F^2 \geq \|P - C\|_2^2 \geq \frac{m w_{\min}\sigma_{\max}^2}{16} \geq B$ concludes (where the second inequality is from Lemma 22). □

Fact 30. *[Analogous to Fact 1.2 in [AS12]] Let $P \in \mathbb{R}^{d \times m}$ be a Gaussian mixtures, and Π be a B-almost k-PCA for $P_1, ..., P_m$. Suppose that B satisfies $B^2 \leq m w_{min}\sigma_{max}^2$. Let $S = \{\nu_1, ..., \nu_k\}$ be centers such that $\operatorname{cost}(\Pi P, S) \leq mk\sigma_{max}^2 \cdot \log^2 m$. Then, for each μ_j, there exists j' such that $\|\mu_j - \nu_{j'}\| \leq 6\Delta_j$, so that we can match each μ_j to a unique $\nu_{j'}$.*

Proof. The proof closely follows the one in [AS12]. Assume by contradiction that there is a j such that $\forall j', \|\mu_j - \nu_{j'}\| > 6\Delta_j$. For any point $p \in P$, let ν_p be its closest center. Then, the contribution of the points in G_j to the cost is at least

$$\sum_{p\in G_j} \|\mu_j - \nu_p + \Pi p - \mu_j\|^2 > \frac{|G_j|}{2}(6\Delta_j)^2 - \sum_{p\in G_j} \|\Pi p - \mu_j\|^2 \geq 18|G_j|\Delta_j^2 - \|\Pi P - C\|_F^2,$$

where the first inequality follows from $(a-b)^2 \geq \frac{a^2}{2} - b^2$. Using first that $|G_j|\Delta_j^2 = 100mk\sigma_{\mathsf{max}}^2 \log^2(m)$, then Fact 29 combined with Lemma 22 yields that $\sum_{p\in G_j} \|\Pi p - \nu_p\|^2 > 1800mk\sigma_{\mathsf{max}}^2 \log^2(m) - 16mk\sigma_{\mathsf{max}}^2$ This contradicts the assumption on the clustering cost. □

Assuming there is a matching as in Fact 30, the proof of [AS12] directly goes through (when the Lloyd steps in Parts 2 and 3 of the algorithm are implemented non-privately), and we can conclude in that case that the clustering computed by Algorithm 11 is correct. Therefore, we first show that our algorithm computes a set of centers satisfying the conditions of Fact 30; and will show afterwards that the remaining of the proof works even with the addition of private noise.

D.5.3 Computing a good k-means solution for ΠP

The goal of this section is to show the following lemma:

Lemma 31. *There is an ε-DP algorithm with $10\log\frac{4\log|Q|}{\varepsilon w_{min}}$ rounds of communications that computes a k-means solution S with*

$$\mathrm{cost}(\Pi P, S) = O\left(m \cdot \log^2\left(\frac{1}{\varepsilon w_{min}}\right) \cdot k\sigma_{max}^2 \log m\right).$$

The proof of this lemma is divided into several parts: first, we show that the means of the projected Gaussians $\Pi\mu_1, ..., \Pi\mu_k$ would be a satisfactory clustering. As points in Q are drawn independently from Π, there are points ΠQ close to each center $\Pi\mu_j$: our second step is therefore an algorithm that finds those points, in few communications rounds.

Lemma 32. *Let Π be the private projection computed by the algorithm. With high probability, clustering the projected set ΠG_j to the projected mean $\Pi\mu_j$ has cost $|G_j|\log m \cdot k\sigma_{max}^2$.*

Proof. We focus on a single Gaussian $\mathcal{G}_j$, and denote for simplicity $\mu := \mu_j$ its center and $\hat{\Sigma} := \Pi\Sigma_j$ the covariance matrix of $\Pi\mathcal{G}_j$. Standard arguments (see Proof of Corollary 5.15 in [KSSU19], or the blog post from [McS14]) show that, with high probability, for all point it holds that $\|\Pi(p-\mu)\|_2^2 \leq \sqrt{k\log(m)}\sigma_{\mathsf{max}}$. For a sketch of that argument, notice that if the projection Π was fixed independently of the samples, this inequality is direct from the concentration of Gaussians around their means, as the projection of $\mathcal{G}_j$ via Π is still a Gaussian, with maximal unidirectional variance at most σ_{max}. This does not stay true when Π depends on the sample; however, since Π is computed via a private mechanism, the dependency between Π and any fixed sample is limited, and we can show the concentration. Combined with the fact that there are $|G_j|$ samples from $\mathcal{G}_j$, this concludes. □

Lemma 31 in particular ensures that clustering ΠP to the full set ΠQ yields a cost $mk\sigma^2_{\max} \cdot \log^2 m$. Therefore, if we could compute for each $q \in Q$ the size $w_q(\Pi P)$ of Πq's cluster in ΠP, namely, the number of points in ΠP closer to Πq than to any other point in ΠQ (breaking ties arbitrarily), then Lemma 24 would ensure that computing an $O(1)$-approximation to k-means on this weighted set yields a solution to k-means on ΠP with cost $O\left(mk\sigma^2_{\max} \cdot \log^2 m\right)$. However, the privacy constraint forbids to compute $w_q(\Pi P)$ exactly, and the server only receives a noisy version $\widehat{w_q(\Pi P)}$ – with a noise following a Laplace noise with parameter $1/\varepsilon$. Hence, for all points $q \in Q$, the noise added is at most $\frac{\log m}{\varepsilon}$ with high probability.

D.5.4 If assumption (2) is satisfied: the noise is negligible

Assumption (2) can be used to bound the total amount of noise added to the server data: we can show that the total contribution of the noise is small compared to the actual k-means cost, in which case solving k-means on the noisy data set yields a valid solution. We show the next lemma:

Lemma 33. *For any set of k centers S, it holds that*

$$\left|\sum_q w_q(\Pi P)\,\mathrm{cost}(p,S) - \sum_q \widehat{w_q(\Pi P)}\,\mathrm{cost}(p,S)\right| \leq \frac{|Q|\log|Q|\Delta^2}{\varepsilon}$$

Proof.

$$\left|\sum_q w_q(\Pi P)\,\mathrm{cost}(q,S) - \sum_q \widehat{w_q(\Pi P)}\,\mathrm{cost}(q,S)\right| = \left|\sum_q \mathrm{Lap}(1/\varepsilon)\,\mathrm{cost}(q,S)\right|$$

With high probability, each of the $|Q|$ Laplace law is smaller than $\frac{\log|Q|}{\varepsilon}$. In this case, we get $\left|\sum_q \mathrm{Lap}(1/\varepsilon)\,\mathrm{cost}(q,S)\right| \leq \frac{|Q|\log|Q|\Delta^2}{\varepsilon}$ Therefore, the gap between the solution evaluated with true weight $w_q(\Pi P)$ and noisy weight $\widehat{w_q(\Pi P)}$ is at most $\frac{|Q|\log|Q|\Delta^2}{\varepsilon}$ □

Using $|Q| \leq m$, the assumption $|Q| \leq \frac{\varepsilon mk\sigma^2_{\max}}{\Delta^2}$ therefore ensures that the upper bound of the previous lemma is at most $mk\log(m)\sigma^2_{\max}$. Hence, if S is a solution that has cost $O(1)$ times optimal on the noisy projected server data, it has cost $O(mk\sigma^2_{\max}\log(m))$ on the projected server data. Combining this result with Lemma 24 concludes: $\mathrm{cost}(\Pi P, S) = O\left(mk\sigma^2_{\max}\log(m)\right)$.

D.5.5 Enforcing Assumption (2)

In order to get rid of Assumption (2), we view the problem slightly differently: we will not try to reduce the number of points in Q to the precise upper-bound, but will nonetheless manage to control the noise and show Lemma 31. Indeed, if all points of Q get assigned more than $2\log m/\varepsilon$ many input points, then the estimates of w_q are correct up to a factor 2, and Lemma 24 shows that a k-means solution S for the dataset consisting of ΠQ with the noisy weights satisfies $\mathrm{cost}(\Pi P, S) = O\left(mk\sigma^2_{\max} \cdot \log^2 m\right)$. However, it may be that some points of Q get assigned less than $2\log m/\varepsilon$ points, in which case the noise would dominate the signal and Lemma 24 becomes inapplicable. Our first goal is therefore to preprocess the set of hings Q to get $\hat{Q}$ such that :

1. for each cluster, $\Pi\hat{Q}$ still contains one good center, and
2. $\forall q \in \hat{Q}, \hat{w}_q \geq 2\log m/\varepsilon$ (where the weight $\hat{w}$ is computed by assigning each data point to its closest center of $\hat{Q}$)

The first item ensures that $\mathrm{cost}(\Pi P, \Pi\hat{Q}) = O\left(mk\sigma_{\mathsf{max}}^2 \cdot \log^2 m\right)$; the second one that the size of each cluster is well approximated, even after adding noise. Our intuition is the following. Removing all points $q \in Q$ with estimated weight less than $2\log m/\varepsilon$ is too brutal: indeed, it may be that one cluster is so over-represented in Q that all its points get assigned less than $2\log m/\varepsilon$ points from P. However, in that case, there are many points in the cluster and in ΠQ: we can therefore remove each point with probability $1/2$ and preserve (roughly) the property that there is a good center in ΠQ. Repeating this intuition, we obtain the algorithm described in Algorithm 14.

Algorithm 14 SimplifyServerData

1: **Input:** Server data Q, client datasets $P^1, ..., P^n$, a projection matrix Π computed from $P^1, ..., P^n$, and privacy parameter ε
2: Let $F \leftarrow \emptyset, Q_0 \leftarrow Q$, $T = 10\log\left(\frac{4\log|Q|}{\varepsilon w_{\mathsf{min}}}\right)$
3: **for** $t = 0$ to T **do**
4: Let $C = F \cup Q_t$
5: for each $q \in C$, the server receives a noisy estimate $\hat{w}_q^{(t)}$ of $w_{\Pi q}(\Pi Q_t)$, with noise $\mathrm{Lap}(T/\varepsilon)$.
6: Server computes $L := \{q \in C : \hat{w}_q^{(t)} \leq 2\log m/\varepsilon\}$.
7: $F \leftarrow F \cup (Q_t \setminus L)$.
8: Server computes Q_{t+1}, a subset of L where each point is sampled with probability $1/2$.
9: **end for**
10: **Return:** F

We sketch briefly the properties of algorithm 14, before diving into details of the proof. First, the algorithm is ε-DP, as each of the T steps is ε/T-DP. Then, points in F are *frozen*: even after adding noise, their weight is well approximated. We will show by induction on the time t that, for any cluster j that does not contain any frozen point at time t, then $Q_t \cap B(\mu_j, 2t\cdot\sqrt{k\log m}\sigma_{\mathsf{max}})$ contains many points: more precisely, $|Q_t \cap B(\mu_j, 2t\cdot\sqrt{k\log m}\sigma_{\mathsf{max}})| \geq \varepsilon|G_j|/2$. Since at each time step only half of the points in L are preserved in Q_{t+1} (line 7 of the algorithm), it implies that, at the beginning, $|Q \cap B(\mu_j, 2t\cdot\sqrt{k\log m}\sigma_{\mathsf{max}})| \gtrsim 2^t\varepsilon/T|G_j|$. Therefore, for $t = \log(1/(\varepsilon w_{\mathsf{min}}))$, we have for each cluster that either it contains a frozen point, or $|Q \cap B(\mu_j, 2t\cdot\sqrt{k\log m}\sigma_{\mathsf{max}})| \geq \frac{|G_j|}{w_{\mathsf{min}}} > m$: as the second option is not possible, all clusters contains a frozen point, which is a good center for that cluster. Our next goal is to formalize the argument above, and show:

Lemma 34. *Let F be the output of Algorithm 14. Then, for each cluster j, there is a point $\nu_j \in F$ such that $\|\Pi(\mu_j - \nu_j)\| \leq \log\left(\frac{4\log|Q|}{\varepsilon w_{min}}\right)\cdot\sqrt{k\log m}\sigma_{max}$. Furthermore, for each $q \in F$, define w_q as the number of points closest to q than any other point in F : it holds that $w_q \geq 2\log m/\varepsilon$.*

For simplicity, we define $\Delta_C := \sqrt{k\log m}\sigma_{\mathsf{max}}$. To prove this lemma, we show inductively that after t iterations of the loop in the algorithm, then either $B(\Pi\mu_j, 2t\Delta_C)$ contains a frozen point, or $|B(\Pi\mu_j, (t+1)\Delta_C) \cap \Pi Q_t| \geq \varepsilon|G_j|/2$. Since the number of points in ΠQ_t is divided

by roughly 2 at every time step, the latter condition implies that there was initially at least $\approx 2^t \varepsilon |G_j|$ points in $B(\Pi\mu_j, (t+1)\Delta_C) \cap \Pi Q$. For $t \approx \log(1/(\varepsilon w_{\min})$, this is bigger than m and we get a contradiction: the ball contains therefore a frozen point.

Our first observation to show this claim is that many points of P are close to μ_j:

Fact 35. *With high probability on the samples, it holds that* $\left|B(\Pi\mu_j, \sqrt{k \log m}\sigma_{max}) \cap \Pi G_j\right| \geq |G_j|$

Proof. As in the proof of Lemma 32, the fact that Π is computed privately ensures that, with high probability, all points $p \in G_j$ satisfy $\|\Pi(p - \mu_j)\| \leq \sqrt{k \log m}\sigma_{\max}$. Thus, $\left|B(\Pi\mu_j, \sqrt{k \log m}\sigma_{\max}) \cap \Pi G_j\right| \geq |G_j|$. □

For the initial time step $t = 0$ we actually provide a weaker statement to initialize the induction, and show that there is at least one point in $B(\Pi\mu_j, \sqrt{k \log m}\sigma_{\max}) \cap \Pi Q_t$. This will be enough for the induction step.

Fact 36 (Initialization of the induction)**.** *With high probability,* $\exists q \in Q, \|\Pi(\mu_j - q)\| \leq \sqrt{k \log m}\sigma_{max}$.

Proof. This directly stems from the fact that there is some point $q \in Q$ that is sampled according to $\mathcal{G}_j$, and that Π is independent of that point. Therefore, Πq follows the Gaussian law $\Pi\mathcal{G}_j$, which is in a k dimensional space and has maximal unidirectional variance $\sigma_{\max}$. Concentration of Gaussian random variables conclude. □

To show our induction, the key lemma is the following:

Lemma 37. *After* t *iteration of the loop, either* $B\left(\Pi\mu_j, (t+1)\sqrt{k \log m}\sigma_{max}\right)$ *contains a frozen point, or* $\left|\Pi Q_t \cap B(\Pi\mu_j, 2(t+1)\sqrt{k \log m}\sigma_{max})\right| \geq \frac{\varepsilon|G_j|}{4T \log(T|Q|)}$.

Proof. Let $\Delta_C := \sqrt{k \log m}\sigma_{\max}$. First, it holds with high probability that all the noise added Line 5 satisfy $\left|\hat{w}_q^{(t)} - w_{\Pi q}(\Pi Q_t)\right| \leq T \log(T|Q|)/\varepsilon$. This directly stems from concentration of Laplace random variables, and the fact that there are $T|Q|$ many of them. We prove the claim by induction. Fix a $t \geq 0$. The induction statement at time t ensures that either there is a point frozen in $B(\Pi\mu_j, (t+1)\Delta_C)$, in which case we are done, or there is at least one point in $\Pi Q_t \cap B(\Pi\mu_j, (t+1)\Delta_C)$ (note that this statement holds for $t = 0$ by Fact 36). By triangle inequality, this means that all points of $\Pi G_j \cap B(\Pi\mu_j, \Delta_C)$ are assigned at time $t+1$ to a point in $B(\Pi\mu_j, (t+2)\Delta_C)$ (in line 4 of Algorithm 14). Therefore, by Fact 35, $\sum_{q:\Pi q \in \Pi Q_t \cap B(\Pi\mu_j, (t+2)\Delta_C)} w_q^t \geq |G_j|$. Then, either ΠQ_t contains less than $\frac{\varepsilon|G_j|}{2T \log(T|Q|)}$ many points from $B\left(\Pi\mu_j, (t+2)\Delta_C\right)$, and we are done, as one of them must have $w_{\Pi q}(\Pi Q_{t+1}) \geq 2T \log(|Q|T)/\varepsilon$ and will be frozen – as in this case $\hat{w}_q^{(t)} \geq T \log(|Q|T)/\varepsilon$. Or, there are more than $\frac{\varepsilon|G_j|}{2T \log(T|Q|)}$ points, and they all have $w_q^{t+1} \leq 2T \log(T|Q|)/\varepsilon$: Chernoff bounds ensure that, with high probability, at least $\frac{\varepsilon|G_j|}{4T \log(T|Q|)}$ will be sampled in the set Q_{t+1}, which concludes the lemma. □

Lemma 34 is a mere corollary of those results:

Proof of Lemma 34. Again, we define $\Delta_C := \sqrt{k \log m}\sigma_{\max}$. At the end of Lemma 34, all points in F are frozen: let $f : P \to F$ such that $f(p) = \arg\min_{q \in F} \|\Pi(p-q)\|$, breaking ties arbitrarily. Since all points are frozen, it holds that for all q, $|f^{-1}(q)| \geq 2 \log m/\varepsilon$: therefore, their noisy weight $\hat{w}_q$ satisfy $|\hat{w}_q - |f^{-1}(q)| \leq \frac{|f^{-1}(q)|}{2}$. Furthermore, for T large enough it holds that $T \geq \log\left(\frac{4T\log(T|Q|)}{\varepsilon w_{\min}}\right)$: this holds e.g. for $T = 10\log\left(\frac{4\log(|Q|)}{\varepsilon w_{\min}}\right)$.

Lemma 37 ensures that either $B(\Pi\mu_j, (T+1)\Delta_C)$ contains a frozen point, or $|\Pi Q_T \cap B(\Pi\mu_j, 2(T+1)\Delta_C)| \geq \frac{\varepsilon|G_j|}{4T\log(T|Q|)}$.

Suppose by contradiction that we are in the latter case. Since, at each time step, every point in Q is preserved with probability $1/2$, it holds with high probability that $|\Pi Q \cap B(\Pi\mu_j, 2(T+1)\Delta_C)| \geq \varepsilon 2^T \cdot |G_j|$. Indeed, all points of that ball are still present in Q_T with probability $1/T^t$: Chernoff bounds ensure that there must be initially at least $2^T \cdot \frac{\varepsilon|G_j|}{4T\log(T|Q|)}$ points in that ball in order to preserve $\frac{\varepsilon|G_j|}{4T\log(T|Q|)}$ of them after the sampling. With our choice of T, this means $|\Pi Q \cap B(\Pi\mu_j, 2(t+1)\Delta_C)| > |Q|$, which is impossible. Therefore, it must be that $B(\Pi\mu_j, (T+1)\Delta_C)$ contains a frozen point, which concludes the proof. □

Proof of Lemma 31 We now have all the ingredients necessary to the proof of Lemma 31. The algorithm is a mere combination of the previous results:

- Use Algorithm 14 to compute a set F.
- Server sends F to the clients, who define $f : P \to F$ such that $f(p) = \arg\min_{q \in F} \|\Pi(p-q)\|$, breaking ties arbitrarily.
- Client i sends $w_{\Pi q}(\Pi P^i) := |\{p \in P^i : f(p) = q\}|$.
- Server receives $\hat{w}_q$, a noisy version of $w_q := \sum_i w_q^i$.
- Server computes an $O(1)$-approximation S to k-means on the dataset ΠF with weights $\hat{w}_q$.

To show that S has the desired clustering cost, we aim at applying Lemma 24. For this, we first bound $\sum_p \|\Pi(p - f(p))\|^2$. For each cluster j, let ν_j be the point from F as defined in Lemma 34. We have, using the definition of f and triangle inequality:

$$\sum_p \|\Pi(p-f(p))\|^2 \leq \sum_j \sum_{p \in G_j} \|\Pi(p-\nu_j)\|^2 \leq 2\sum_j \sum_{p \in G_j} \|\Pi(p-\mu_j)\|^2 + \|\Pi(\mu_j - \nu_j)\|^2.$$

From Lemma 32, we know that $\sum_j \sum_{p \in G_j} \|\Pi(p-\mu_j)\|^2 = O(m \log m \cdot k\sigma_{\max}^2)$. The guarantee of ν_j in Lemma 34 ensures $\sum_j \sum_{p \in G_j} \|\Pi(\mu_j - \nu_j)\|^2 = O\left(m \cdot \log^2\left(\frac{1}{\varepsilon w_{\min}}\right) \cdot k\sigma_{\max}^2 \log m\right)$. Thus, $\sum_p \|\Pi(p - f(p))\|^2 = O\left(m \cdot \log^2\left(\frac{1}{\varepsilon w_{\min}}\right) \cdot k\sigma_{\max}^2 \log m\right)$. Since all points in F have an estimated that satisfies $|\hat{w}_q - |f^{-1}(q)|| \leq \frac{|f^{-1}(q)|}{2}$, we can apply Lemma 24: the solution computed by the above algorithm on the dataset ΠF with weights $\hat{w}_q$ has cost at most $O\left(m \cdot \log^2\left(\frac{1}{\varepsilon w_{\min}}\right) \cdot k\sigma_{\max}^2 \log m\right) + O(\mathsf{OPT}(\Pi P)) = O\left(m \cdot \log^2\left(\frac{1}{\varepsilon w_{\min}}\right) \cdot k\sigma_{\max}^2 \log m\right)$. This concludes the proof of Lemma 31.

D.6 Part 2: Improving iteratively the solution

Our global algorithm is described in Algorithm 15: first, we use Lemma 28 to reduce the diameter of the input; then, we compute a good initial solution using Lemma 31. Then, we implement privately Part 2 and Part 3 of Algorithm 13, using private mean estimation. We note that this algorithm, when assumptions (1) and (2) are satisfied, is exactly Algorithm 11 followed with Algorithm 12. Hence, Theorem 7 follows directly from the proof of Theorem 26.

Algorithm 15 Cluster

1: **Input:** Server data Q, client datasets $P^1, ..., P^n$, and privacy parameters ε, δ
2: Process the input to reduce the diameter to Δ using Lemma 28, with privacy parameter $\varepsilon/4$.
3: In one round of communication, compute a $O(\sqrt{d}\Delta \cdot \sigma(\varepsilon/4, \delta))$-almost k-PCA using Theorem 20.
4: **Part 1:** find initial centers $\nu_1^{(1)}, ..., \nu_k^{(1)}$ using Lemma 31, with privacy parameter $\varepsilon/4$
5: **Part 2:**

a) Server sends $\nu_1^{(1)}, .., \nu_k^{(1)}$ to clients, and client i computes $S_j^i := \{p \in P^i : \forall s, \|\hat{p} - \nu_j^{(1)}\| \leq \frac{1}{3}\|\hat{p} - \nu_s^{(1)}\|\}$.

b) Server receives, for all cluster j, $\nu_j^{(2)} := \frac{1}{\sum_{i=1}^n |S_j^i| + \mathrm{Lap}(T/\varepsilon)} \left(\sum_{i=1}^n \sum_{p \in S_j^i} p + \mathcal{N}_d\left(0, \frac{2T^2\Delta \log(2T/\delta)}{\varepsilon^2}\right)\right)$

6: **Part 3:** Repeat Lloyd's steps for T steps, with privacy parameter $(\varepsilon/T, \delta/T)$:

1. Server sends $\nu_1^{(t)}, .., \nu_k^{(t)}$ to clients, and client i computes $S_j^i := \{p \in P^i : \forall s, \|\hat{p} - \nu_j^{(t)}\| \leq \|\hat{p} - \nu_s^{(t)}\|\}$.
2. Server receives, for all cluster j, $\nu_j^{(t+1)} := \frac{1}{\sum_{i=1}^n |S_j^i| + \mathrm{Lap}(T/\varepsilon)} \left(\sum_{i=1}^n \sum_{p \in S_j^i} p + \mathcal{N}_d\left(0, \frac{2T^2\Delta \log(2T/\delta)}{\varepsilon^2}\right)\right)$

Given the mapping of Fact 30, the main result of [AS12] is that step 2 of the algorithm computes centers that are very close to the μ_j:[2]

Theorem 38 (Theorem 4.1 in [AS12])**.** *Suppose that the solution $\nu_1, ..., \nu_k$ is as in Fact 30, namely, for each μ_j, it holds that $\|\mu_j - \nu_j\| \leq 6\Delta_j$. Denote $S_j = \{p : \forall r \neq j, \|\Pi P_p - \nu_j\| \leq \frac{1}{3}\|\Pi P_p - \nu_r\|\}$. Then, for every $j \in [k]$ it holds that*

$$\|\mu(S_j) - \mu_j\| = O\left(\frac{1}{c\sqrt{|G_j|}} \cdot \|P - C\|_2\right),$$

where c is the separation constant from Definition 6.

Finally, the next result from [KK10] shows that the Lloyd's steps converge towards the true means:

[2]Note that the original theorem of [AS12] is stated slightly differently: however, their proof only requires Fact 29 and the matching provided by Fact 30, and we modified the statement to fit our purposes.

Theorem 39 (theorem 5.5 in [KK10]). *If, for all j and a parameter $\gamma \leq ck/50$,*

$$\|\mu_j - \nu_j\| \leq \frac{\gamma \|P - C\|_2}{\sqrt{|G_j|}},$$

then

$$\|\mu_j - \mu(C(\nu_j))\| \leq \frac{\gamma \|P - C\|}{2\sqrt{|G_j|}},$$

where $C(\nu_j)$ is the set of points closer to ν_j than to any other $\nu_{j'}$.

This allows us to conclude the accuracy proof of Theorem 26

Proof of Theorem 26. The algorithm is (ε, δ)-DP: each of the 4 steps step – reducing the diameter, computing a PCA, finding a good initial solution and running T Lloyd's steps – is $(\varepsilon/4, \delta/4)$-DP, and private composition concludes. The first three steps require a total of $2+10\log\frac{4\log|Q|}{\varepsilon w_{\min}}$ many rounds of communication, the last one requires $T+\log\frac{\sigma_{\max}^2}{w_{\min}}$ rounds. This simplifies to $T + \zeta_2 \log\frac{\sigma_{\max}\log|Q|}{\varepsilon w_{\min}}$, for some constant ζ_2. The first step reduces the diameter to $\Delta = O\left(\frac{k\log^2 m\sqrt{d}\sigma_{\max}}{\varepsilon w_{\min}}\right)$; therefore, Lemma 31 combined with Fact 30 ensures that $\nu_1^{(1)}, ..., \nu_k^{(1)}$ satisfies the condition of Theorem 38. In addition, Lemma 4.2 of [AS12] ensures that the size of each cluster $|S_j|$ is at least $\frac{|G_j|}{2}$ at every time step. Therefore, the private noise $\frac{\mathcal{N}_d\left(\Delta^2\sigma^2(\varepsilon',\delta')\right)}{|S_j^i|}$ is bounded with high probability by $\eta := O\left(\frac{\Delta\sqrt{d}\sigma(\varepsilon/T,\delta/T)}{|S_j^i|}\right) = O\left(\frac{kdT\log^2 m\sigma_{\max}\cdot\sqrt{\ln(1/\delta)}}{m\varepsilon^2 w_{\min}^2}\right)$, which for and $m = \Omega\left(\frac{kdT\log^2 m\cdot\sqrt{\ln(1/\delta)}}{\varepsilon^2 w_{\min}^2}\right)$ is smaller than $\Delta_j = \frac{\sigma_{\max}}{\sqrt{w_j}}\min\left(\sqrt{k}\,\mathrm{polylog}(d/w_{\min}), d\right)$. Hence, the conditions of Theorem 38 and Theorem 39 are still satisfied after adding noise, and the latter implies that the noisy Lloyd steps converge exponentially fast towards $B(\mu_j, \eta)$. More precisely, it holds with probability $1 - 1/k^2$ that $\left\|\mu_j - \nu_j^{T+\log\frac{\sigma_{\max}^2}{w_{\min}}}\right\| = O\left(\frac{1}{c2^{T+\log\frac{\sigma_{\max}^2}{w_{\min}}}}\right)\cdot \frac{\|P-C\|_2}{\sqrt{|G_j|}} + \eta$.

From Lemma 22 ensures $\|P - C\| \leq O(\sqrt{m}\sigma_{\max})$. Since $|G_j| \geq mw_{\min}/2$, the first term is at most $O\left(\frac{1}{2^T}\right)$.

Therefore,

$$\left\|\mu_j - \nu_j^{T+\log\frac{\sigma_{\max}^2}{w_{\min}}}\right\| = O\left(\max\left(\frac{1}{2^T}, \frac{kdT\log^2 m\sigma_{\max}\sqrt{\ln(T/\delta)}}{m\varepsilon^2 w_{\min}^2}\right)\right).$$

□

D.7 Experiment Details

D.7.1 Dataset details

Mixture of Gaussians Datasets We generate a mixture of Gaussians in the following way. We set the data dimension to $d = 100$ and we generate $k = 10$ mixtures by uniformly

randomly sampling k means $\{\mu_1, \dots \mu_k\}$ from $[0,1]^d$. Each mixture has diagonal covariance matrix $\Sigma_j = 0.5 I_d$ and equal mixture weights $w_j = 1/k$. The server data is generated by combining samples from the true mixture distribution together with additional data sampled uniformly randomly from $[0,1]^d$ representing related but out-of-distribution data. We sample 20 points from each mixture component, for a total of $20 \times k = 200$ in distribution points and sample an additional 100 uniform points. For Section 6.5.1 we simulate a cross-silo setting with 100 clients, with each client having 1000 datapoints sampled i.i.d from the Gaussian mixture. For Section 6.5.2 we simulate a cross-device setting with 1000, 2000 and 5000 clients, each client having 50 points i.i.d sampled from the Gaussian mixture distribution. The server data is identical in both cases.

US Census Datasets We create individual datapoints coming from the ACSIncome task in folktables. Thus each datapoint consists of $d = 819$ binary features describing an individual in the census, including details such as employment type, sex, race etc. In order to create a realistic server dataset (of related but not not in-distribution data) we filter the client datasets to contain only individuals of a given employment type. The server then receives a small amount (20) of datapoints with the chosen employment type, and a larger amount (1000) of datapoints sampled i.i.d from the set of individuals with a different employment type. We do this for 3 different employment types, namely "Employee of a private not-for-profit, tax-exempt, or charitable organization", "Federal government employee" and "Self-employed in own not incorporated business, professional practice, or farm". These give us three different federated datasets, each with 51 clients, with total dataset sizes of 127491, 44720 and 98475 points respectively.

Stack Overflow Datasets Each client in the dataset is a stackoverflow user, with the data of each user being the questions they posted. Each question also has a number of tags associated with it, describing the broad topic area under which the question falls. We first preprocess the user questions by embedding them using a pre-trained sentence embedding model [RG19]. Thus a user datapoint is now a $d = 384$ text embedding. Now we again wish to create a scenario where the server can receive related but out of distribution data. We follow a similar approach to the creation of the US census datasets. We select two tag topics and filter our clients to consist of only those users that have at least one question that was tagged with one of the selected topics. For those clients we retain only the questions tagged with one of the chosen topics. The server then receives 1000 randomly sampled questions with topic tags that do not overlap with the selected client tags as well as 20 questions with the selected tags, 10 of each one. For our experiments we use the following topic tag pairs to create clients [(machine-learning, math), (github, pdf), (facebook, hibernate), (plotting, cookies)]. These result in federated clustering problems with $[10394, 9237, 23266, 2720]$ clients respectively.

D.7.2 Verifying our assumptions

On each of the datasets used in our data-point-level experiments we compute the radius of the dataset Δ, shown in Table D.1.

Assumption (1) requires $\Delta = O\left(\frac{k \log^2(m) \sigma_{\max} \sqrt{d}}{\varepsilon w_{\min}}\right)$. For the Gaussian mixture, $k = 10, d = 100, w_{\min} = 1/10, m = 10^6$ and $\sigma_{\max} = 0.5$: thus Δ clearly satisfies the condition. For the US Census datasets, $k = 10, d = 819, m \in \{127491, 44720, 98475\}$. As we cannot estimate

Dataset	Δ
Gaussian Mixture (100 clients)	10.57
US Census (Not-for-profit Employees)	2.65
US Census (Federal Employees)	2.65
US Census (Self-Employed)	2.65

Table D.1: Radius of each dataset.

$\sigma_{\max}$ and $w_{\min}$(since the dataset is not Gaussian), we use an upper-bound $w_{\min} = 1$, and replace $\sigma_{\max}$ with a proxy based on the optimal k-means cost, $\sqrt{\mathsf{OPT}/m}$: this is a priori a large upper-bound on the value of $\sigma_{\max}$, but it still gives an indication on the geometry of each cluster. As can be seen in Figure 6.1, Figure D.6, the average optimal cost is about 3.5 : thus, $\sqrt{\mathsf{OPT}/m} \approx 1.87$, and we estimate $\frac{k\log^2(m)\sigma_{\max}\sqrt{d}}{\varepsilon w_{\min}} \approx \frac{10\cdot\log^2(10^5)\cdot 0.005\cdot\sqrt{819}}{0.5} \approx 123000$. This indicates that Condition (1) is satisfied as well for this dataset. Assumption 2 requires that the size of the server data is not too large: $|Q| \leq \frac{\varepsilon m k \log(m)\sigma_{\max}^2}{\Delta^2}$. In the Gaussian case, we have $|Q| = 300$, and the right-hand-side is about 29000.

In the US Census Dataset, we again upper-bound $\sigma_{\max}^2 = \frac{\mathsf{OPT}}{m}$. In that case, the right-hand-side is about 620000, while there are 1020 server point. Although our estimate of $\sigma_{\max}$ is only an upper-bound, this indicates that assumption (2) is also satisfied.

D.7.3 Baseline implementation details

SpherePacking We implement the data independent initialization described in [SCL+17] as follows. We estimate the data radius Δ using the server dataset. We set $a = \Delta\sqrt{d}$, for $j = 1, \ldots, k$, we randomly sample a center ν_j in $[-\Delta, \Delta]^d$. If ν_j is at least distance a from the corners of the hypercube $[-\Delta, \Delta]^d$ and at least distance $2a$ away from all previously sampled centers $\nu_1, \ldots, \nu_{j-1}$, then we keep it. If not we resample ν_j. We allow 1000 attempts to sample ν_j, if we succeed with sampling all k centers then we call the given a feasible. If not then a is infeasible. We find the largest feasible a by binary search and use the corresponding centers as the initialization.

D.7.4 Adapting FedDP-KMeans to client-level privacy

As discussed in Section 6.5.2, moving to client-level DP changes the sensitivities of the algorithm steps that use client data. To calibrate the noise correctly we enforce the sensitivity of each step by clipping the quantities sent by each client to the server, prior to them being aggregated. Concretely, suppose v_i is a vector quantity owned by client i, and the server wishes to compute the aggregate $v = \sum_i v_i$. Then prior to aggregation the client vector is clipped to have maximum norm B so that

$$\hat{v}_i = \begin{cases} \frac{B}{\|v_i\|} v_i, & \text{if } \|v_i\| > B \\ v_i, & \text{otherwise.} \end{cases}$$

The aggregate is then computed as $\hat{v} = \sum_i \hat{v}_i$. This query now has sensitivity B, and noise can be added accordingly. Each step of our algorithms can be expressed as such an aggregation over client statistics, the value of B for each step becomes a hyperparameter of the algorithm. We make one additional modification to Step 3 of FedDP-Init to make it better suited to

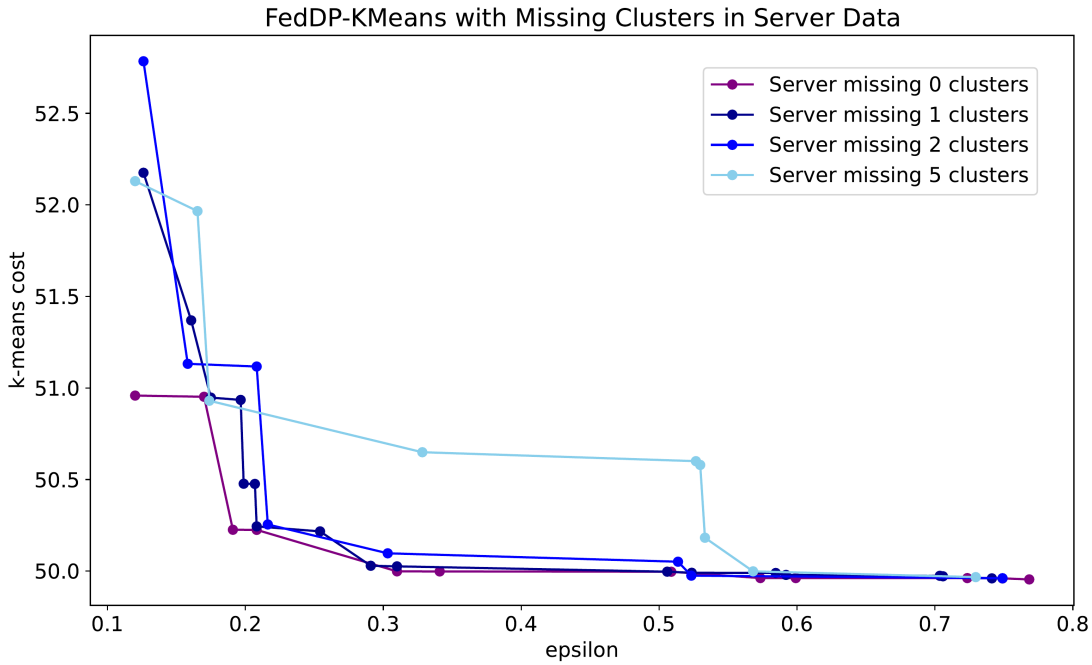

Figure D.1: Mixture of Gaussians data with $k = 10$ clusters and 100 clients. Performance of FedDP-KMeans when the server is missing 0, 1, 2 and 5 of the 10 total clusters.

the client-level DP setting. In Algorithm 11 during Step 3 the clients compute the sum m_j^i and count n_j^i of the vectors in each cluster S_j^i. Rather than send these to the server to be aggregated the client instead computes their cluster means locally as

$$u_j^i = \begin{cases} \frac{m_j^i}{n_j^i}, & \text{if } n_j^i > 0 \\ 0, & \text{otherwise,} \end{cases}$$

as well as a histogram counting how many non-empty clusters the client has:

$$c_j^i = \begin{cases} 1, & \text{if } n_j^i > 0 \\ 0, & \text{otherwise.} \end{cases}$$

The server then receives the noised aggregates $\widehat{u_j}$ and $\widehat{c_j}$ and computes the initial cluster centers as $\nu_j = \widehat{u_j}/\widehat{c_j}$. In other words we use a mean of the means estimate of the true cluster mean.

D.8 Additional Experiments

D.8.1 Setting hyperparameters of FedDP-KMeans

In this section we analyze the hyperparameter settings of FedDP-KMeans that produced the Pareto optimal results shown in the figures in Sections 6.5.1 and 6.5.2. These analyses give us some insights on the optimal ways to set the hyperparameters when using FedDP-KMeans in practice.

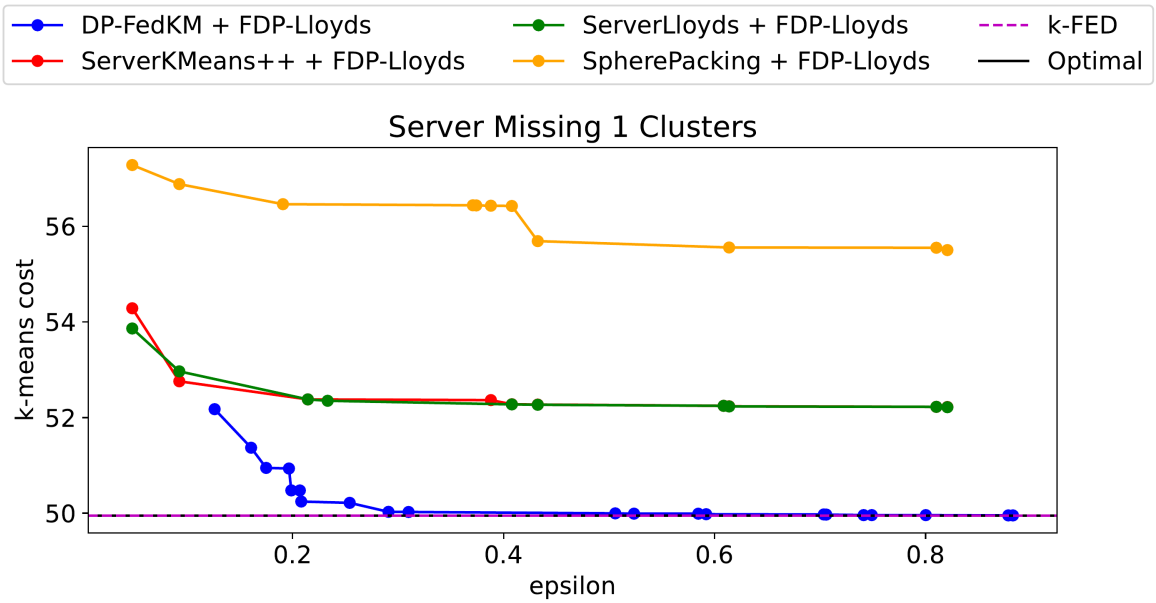

Figure D.2: Mixture of Gaussians data with 100 clients. Server missing 1 of the $k = 10$ clusters.

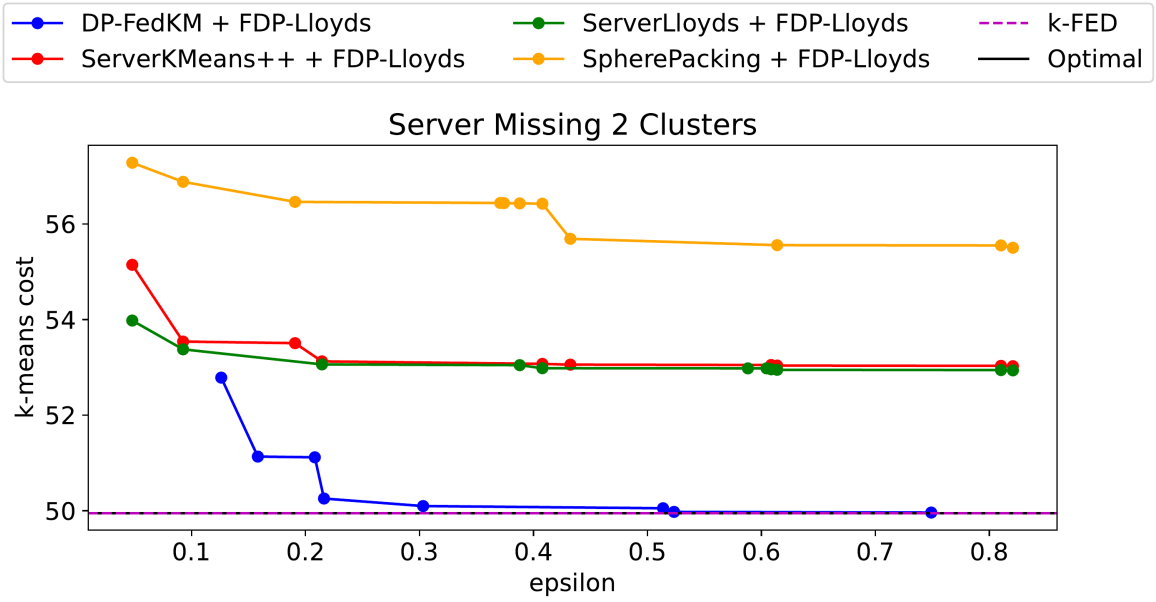

Figure D.3: Mixture of Gaussians data with 100 clients. Server missing 2 of the $k = 10$ clusters.

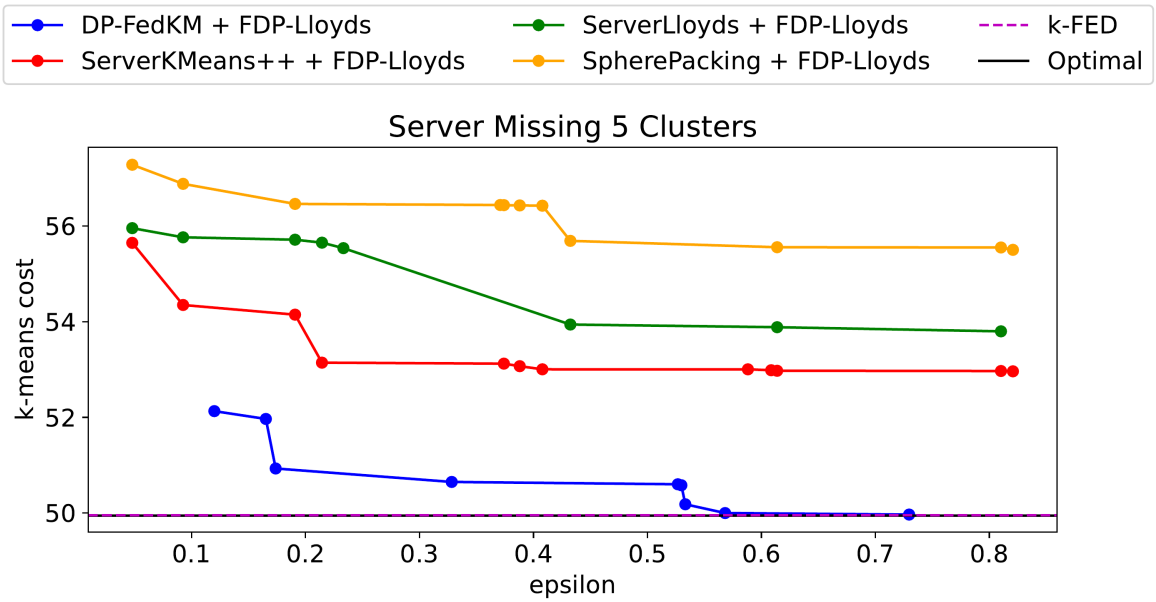

Figure D.4: Mixture of Gaussians data with 100 clients. Server missing 5 of the $k = 10$ clusters.

Distributing the privacy budget The most important parameters to set are the values of epsilon in Parts 1-3 of Algorithm 11. Here we discuss how to set these.

Let ε_1, ε_2, ε_{3G} and ε_{3L} denote the epsilon we allow for part 1, part 2, the Gaussian query in part 3 and the Laplace query in part 3 respectively. We let $\varepsilon_{\text{init}} = \varepsilon_1 + \varepsilon_2 + \varepsilon_{3G} + \varepsilon_{3L}$. By strong composition the initialization will have a lower overall budget than $\varepsilon_{\text{init}}$, however, it serves as a useful proxy to the overall budget as we can think of what proportion of $\varepsilon_{\text{init}}$ we are assigning to each step. Shown in Tables D.2 and D.3 are the values from our experiments. Specifically, for each dataset we take the mean across the Pareto optimal results that we plotted of the ε values used for each step. We then express this as a fraction of $\varepsilon_{\text{init}}$. Loosely speaking, we interpret these values as answering "What fraction of our overall privacy budget should we assign to each step?"

The results paint a consistent picture when comparing values with the same unit-level of privacy with slight differences between the two levels. For datapoint level privacy, clearly the most important step in terms of assigning budget is to the Gaussian mechanism in Step 3 with the other steps being roughly even in term of importance. Therefore, as a rule of thumb we would recommend assigning budget using the following approximate proportions $[0.2, 0.2, 0.45, 0.15]$. For user level privacy we observe the same level of importance being placed on the Gaussian mechanism in Step 3 but additionally on the Gaussian mechanism in Step 1. Based on these results we would assign budget following approximate proportions $[0.35, 0.1, 0.45, 0.1]$. Clearly these are recommendations based only on the datasets we have experimented with and the optimal settings will vary from dataset to dataset, most notably based on the number of clients and the number of datapoints per client.

Number of steps of FedDP-Lloyds The other important parameter to set in FedDP-KMeans is the number of steps of FedDP-Lloyds to run following the initialization obtained by FedDP-Init. As discussed already, this comes with the inherent trade-off of number of iterations vs accuracy of each iteration. For a fixed overall budget, if we run many iterations, then each iteration will have a lower privacy budget and will therefore be noisier. Not only

Dataset	$\epsilon_1/\epsilon_{\text{init}}$	$\epsilon_2/\epsilon_{\text{init}}$	$\epsilon_{3\text{G}}/\epsilon_{\text{init}}$	$\epsilon_{3\text{L}}/\epsilon_{\text{init}}$
Gaussian Mixture (100 clients)	0.18	0.23	0.43	0.17
US Census (Not-for-profit Employees)	0.24	0.17	0.41	0.18
US Census (Federal Employees)	0.15	0.16	0.52	0.17
US Census (Self-Employed)	0.20	0.23	0.47	0.10

Table D.2: Amount of privacy budget, as a fraction of $\varepsilon_{\text{init}}$, that is assigned to each step of FedDP-Init. Results shown are the mean of the Pareto optimal results plotted for each of the data-point-level experiments in Figures 6.1, D.6 and D.7.

Dataset	$\varepsilon_1/\varepsilon_{\text{init}}$	$\varepsilon_2/\varepsilon_{\text{init}}$	$\varepsilon_{3\text{G}}/\varepsilon_{\text{init}}$	$\varepsilon_{3\text{L}}/\varepsilon_{\text{init}}$
Gaussian Mixture (1000 clients)	0.38	0.09	0.42	0.10
Gaussian Mixture (2000 clients)	0.43	0.10	0.36	0.11
Gaussian Mixture (5000 clients)	0.43	0.09	0.37	0.11
Stack Overflow (facebook, hibernate)	0.29	0.15	0.42	0.15
Stack Overflow (github, pdf)	0.37	0.12	0.40	0.10
Stack Overflow (machine-learning, math)	0.29	0.14	0.45	0.13
Stack Overflow (plotting, cookies)	0.33	0.11	0.47	0.09

Table D.3: Amount of privacy budget, as a fraction of $\varepsilon_{\text{init}}$, that is assigned to each step of FedDP-Init. Results shown are the mean of the Pareto optimal results plotted for each of the client-level experiments in Figures 6.2, D.8, D.9, D.10 and D.11.

that, but in fact the question of whether we even want to run any iterations has the same trade-off. If we run no iterations of FedDP-Lloyds, then we use none of our privacy budget here, and we have more available for FedDP-Init. To investigate this we do the following: for each dataset we compute, for each number of steps T of FedDP-Lloyds, the fraction of the Pareto optimal runs that used T steps.

Dataset	0 steps	1 step	2 steps
Gaussian Mixture (100 clients)	0.61	0.39	0
US Census (Not-for-profit Employees)	0.8	0.1	0.1
US Census (Federal Employees)	0.91	0.09	0
US Census (Self-Employed)	0.92	0.08	0

Table D.4: Fraction of the Pareto optimal results that used a given number of steps of FedDP-Lloyds for the data-point-level experiments.

Dataset	0 steps	1 step	2 steps
Gaussian Mixture (1000 clients)	0.86	0.11	0.04
Gaussian Mixture (2000 clients)	0.8	0.17	0.03
Gaussian Mixture (5000 clients)	0.81	0.1	0.1
Stack Overflow (facebook, hibernate)	1.0	0	0
Stack Overflow (github, pdf)	1.0	0	0
Stack Overflow (machine-learning, math)	0.94	0.06	0
Stack Overflow (plotting, cookies)	0.96	0.04	0

Table D.5: Fraction of the Pareto optimal results that used a given number of steps of FedDP-Lloyds for the client-level experiments.

The results, shown in Tables D.4 and D.5, are interesting. In all but one dataset more than 80% of the optimal runs used no steps of FedDP-Lloyds, with many of the datasets being over 90%. The preference was to instead use all the budget for the initialization. The reason for this is again the inherent trade-off between number of steps and accuracy of each step, with it clearly here being the case that fewer more accurate steps were better. One point to note here is that FedDP-Init essentially already has a step of Lloyds built into it, Step 3 is nearly identical to a Lloyds step but with points assigned by distance in the projected space. Running this step once and to a higher degree of accuracy tended to outperform using more steps. This in fact highlights the point made in our motivation, about the importance of finding an initialization that is already very good, and does not require many follow up steps of Lloyds.

D.8.2 Missing clusters in the server data

In order for our theoretical guarantees to hold we required the assumption that the server data include at least one point sampled from each of the components of the Gaussian mixture and this assumption was reflected in the experimental setup of Sections 6.5.1 and 6.5.2. This is, however, not a requirement for FedDP-KMeans to run or work in practice. To test this we run experiments in the setting that certain clusters are missing from the server data in our Gaussian mixtures setting. Specifically, the server data is constructed by sampling from only a subset of the $k = 10$ Gaussian components of the true distribution. Figure D.1 shows the performance of FedDP-KMeans as we increase the number of clusters missing on the server. As we can see performance deteriorates modestly as the number of clusters missing from the server dataset increases. Figures D.2-D.4 show that this also occurs in the other baselines that make use of the server data and that FedDP-KMeans is still the best performing method in this scenario.

D.8.3 Choosing k using Weighted Server Data

While in practice, k-means is often used with a value of k determined by external factors, such as computational or memory demands, k can also be chosen based on the data at hand. Existing methods can be incorporated into our setting quite simply, by using the method on the weighted and projected server dataset ΠQ, with weights $w_q(\hat{\Pi}P)$. This dataset serves as a proxy for the client data and we can operate on it without incurring any additional privacy costs. We illustrate this using the popular elbow method. Concretely, we run lines 1-16 of Algorithm 11 using some large value k', then we run line 17 for $k = 1, 2, 3, \ldots$ and plot the k-means costs of the resulting clusterings. This is shown in Figure D.5. Clearly, the elbow of the curve occurs at $k = 10$ which is indeed the number of clusters in the true data distribution (we used the same Gaussian mixture dataset as in the original experiments).

D.9 Additional figures

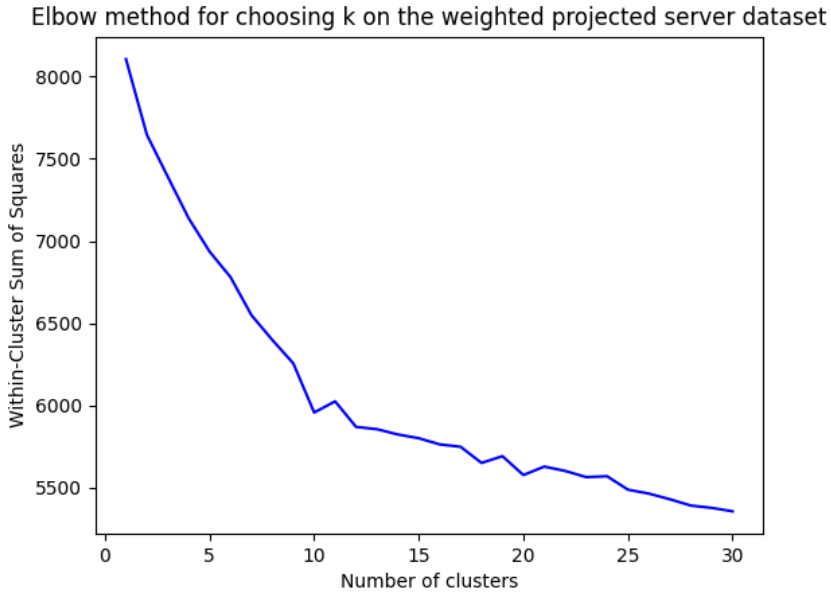

Figure D.5: Plotting the Within-Cluster Sum of Squares (aka k-means cost), against the number of clusters, when clustering the weighted projected server data points. The true number of clusters in the data is $k = 10$, the prior steps of FedDP-Init were run with $k' = 20$. The "elbow" of the curve indeed occurs at $k = 10$.

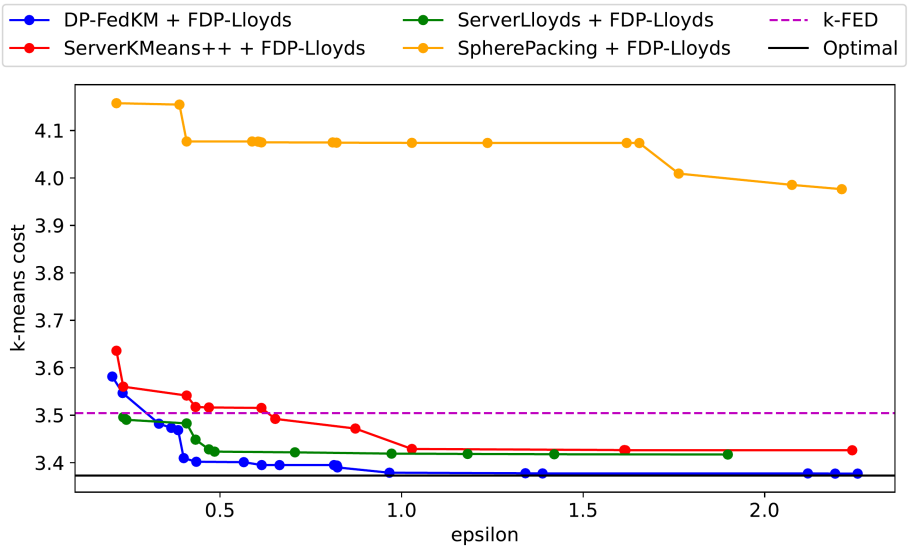

Figure D.6: Results with data-point-level privacy on US census data. The 51 clients are US states, each client has the data of individuals with employment type "Employee of a private not-for-profit, tax-exempt, or charitable organization".

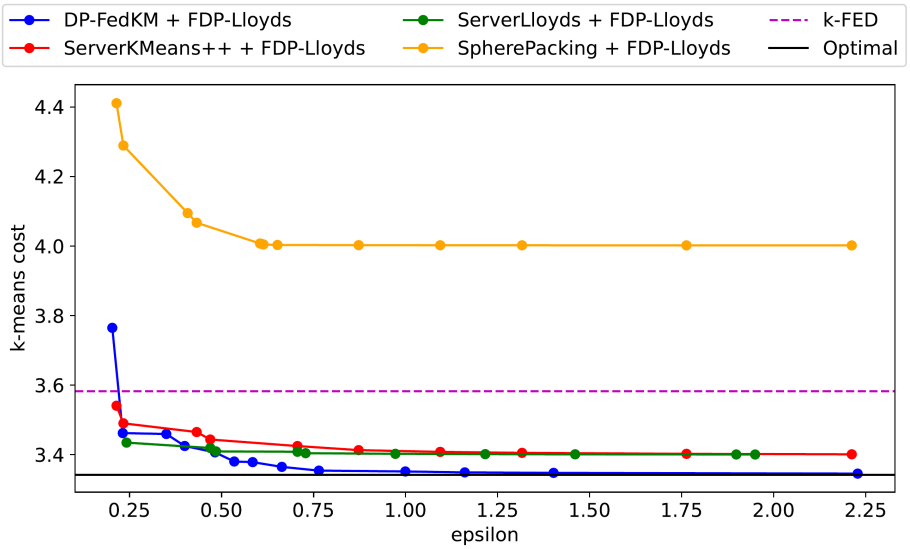

Figure D.7: Results with data-point-level privacy on US census data. The 51 clients are US states, each client has the data of individuals with employment type "Self-employed in own not incorporated business, professional practice, or farm".

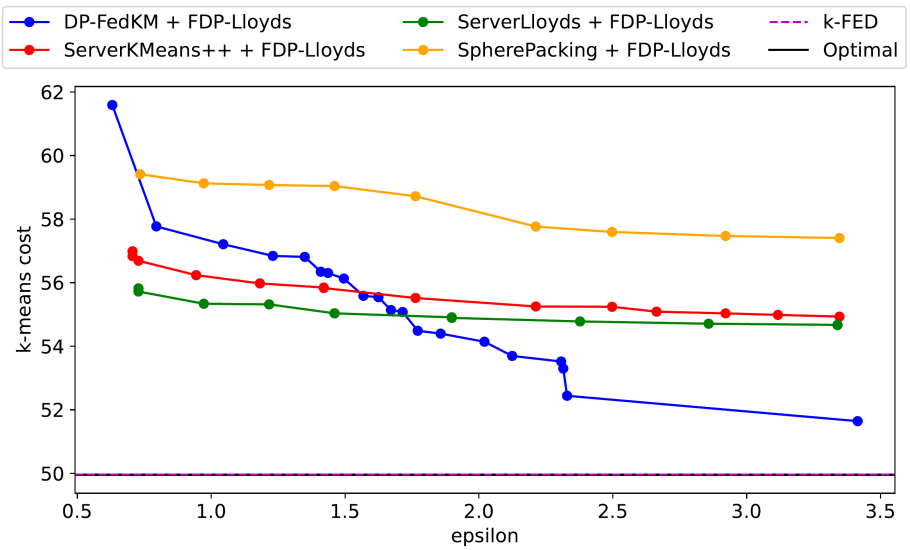

Figure D.8: Results with client-level privacy on Synthetic mixture of Gaussians data with 1000 clients in total.

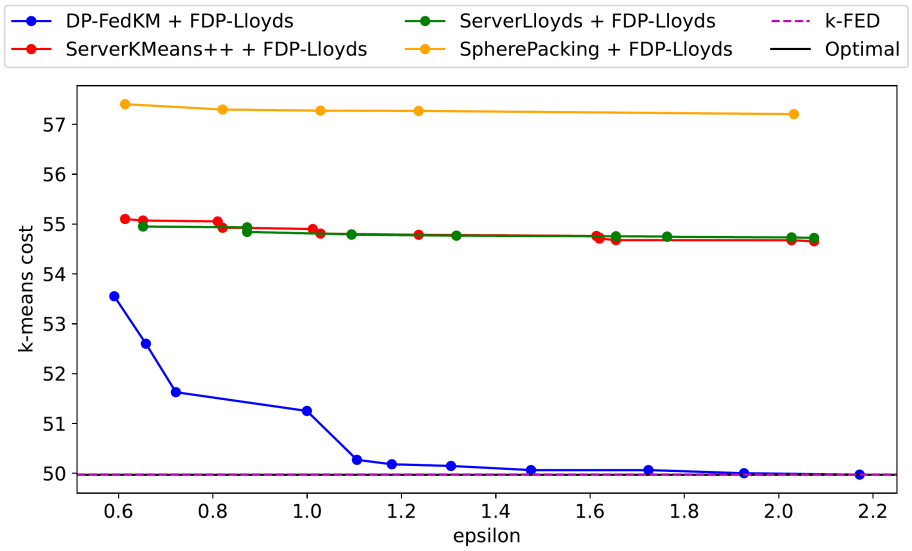

Figure D.9: Results with client-level privacy on Synthetic mixture of Gaussians data with 5000 clients in total.

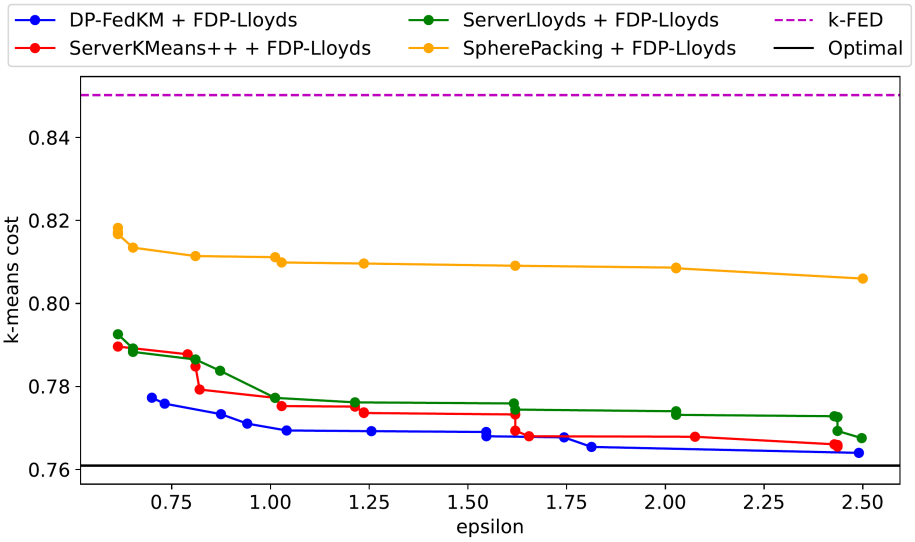

Figure D.10: Results with client-level privacy on the stackoverflow dataset with 23266 clients with topic tags facebook and hibernate.

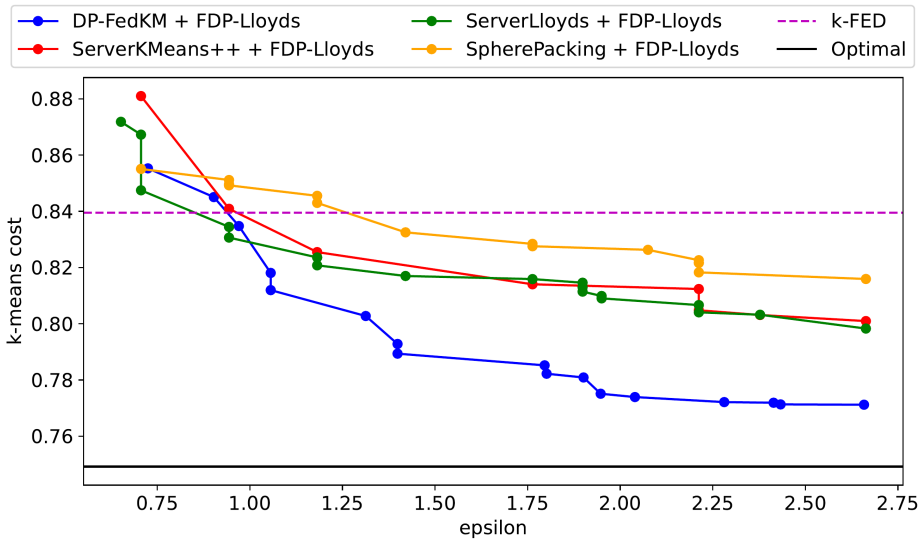

Figure D.11: Results with client-level privacy on the stackoverflow dataset with 2720 clients with topic tags plotting and cookies.

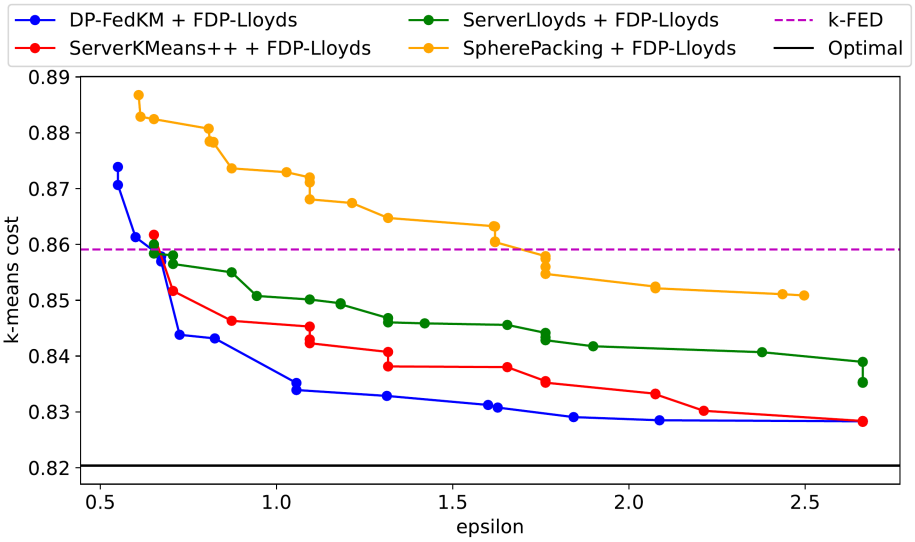

Figure D.12: Results with client-level privacy on the stackoverflow dataset with 10394 clients with topic tags machine-learning and math.