Fast, Private and Fair Federated Learning

by

Saber Malekmohammadi

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2025

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner: Shahab Asoodeh
Assistant Professor
Department of Computing and Software, McMaster University

Supervisor: Yaoliang Yu
Associate Professor
School of Computer Science, University of Waterloo

Internal Members: Gautam Kamath
Assistant Professor
School of Computer Science, University of Waterloo

Hong Zhang
Assistant Professor
School of Computer Science, University of Waterloo

Internal-External Member: Chengguo Weng
Professor
Department of Statistics and Actuarial Science, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis includes a presentation of my research works, summarized from three papers, including one conference paper, one book chapter and one paper under review. I am the first author of all the three papers, where I mainly contributed ideas, implemented the proposed methods, and wrote the papers. My co-authors are my supervisor and my collaborators, who provided me with invaluable help in refining ideas and polishing my writings.

- Malekmohammadi et al. 2021a,b, 2023: “A Unifying Framework for Federated Learning". In Springer Book on Federated and Transfer Learning, which corresponds to chapter 2.
- Malekmohammadi et al. 2024: “Noise-Aware Algorithm for Heterogeneous Differentially Private Federated Learning". In International Conference on Machine Learning (ICML), which corresponds to chapter 3.
- Malekmohammadi et al. 2025: “Differentially Private Clustered Federated Learning". On ArXiv and under the review of NeurIPS 2025, which corresponds to chapter 4.

Abstract

A remarkable progress in machine learning (ML) technologies has happened during the past decade. Federated learning (FL) is a decentralized ML setting, where some clients (data holders) collaborate with each other to train a model under the orchestration of a central server. Clients, for instance, can be some mobile devices (in cross-device FL) or some organizations (in cross-silo FL). In FL, the training data of clients remain decentralized, mitigating many systemic privacy risks and costs existing in centralized ML. This thesis investigates three main sub-problems of FL, including: 1. Convergence analysis of FL algorithms 2. Formal data privacy guarantees for clients through differentially private FL (DPFL) 3. Performance parity in DPFL systems under structured data heterogeneity. We provide further details about them in the following.

Over the past years, the FL community witnessed a plethora of new algorithms. However, there is a lack of thorough comparison of these algorithms and the theory behind them. Our fragmented understanding of the theory behind the existing algorithms is the reason behind the lack of such a formal comparison. This is the main focus of Chapter 2, where we show that many of the existing FL algorithms can be understood from the lens of operator splitting theory. This unification allows us to compare different algorithms easily, to refine previous convergence results and to propose new algorithmic variants. Our new theoretical findings reveal the remarkable role played by the step size in FL. Furthermore, the unification allows us to propose a simple and economic way to accelerate the convergence of different algorithms. Fast convergence is vital in FL, as communication between the server and clients is often considered as an overhead.

Despite the fact that FL provides clients with some level of informal data privacy by operating on their decentralized data, the orchestrating server or other malicious third parties can still attack the data privacy of participating clients. Consequently, FL has been augmented with differential privacy (DP) in order to provide rigorous data privacy guarantees to clients. In DPFL, which is the focus of Chapter 3, there is often heterogeneity in the privacy requirements of clients due to their varying privacy policies. This heterogeneity as well as the heterogeneity in batch and/or dataset sizes of clients lead to a variation in the DP noise level across clients' model updates. Hence, straightforward aggregation strategies, e.g. assigning clients' aggregation weights proportional to their privacy parameters, will lead to a low utility for the system. We propose a DPFL algorithm which efficiently estimates the true noise level in clients' uploaded model updates and uses an aggregation strategy to reduce the noise level in their aggregation. Our proposed method improves utility and convergence speed, while being safe to the clients that may maliciously send falsified privacy parameters to the server to attack the system's utility.

Finally, in Chapter 4, we investigate the intersection of FL, DP and performance fairness, under structured data heterogeneity across clients. This type of data heterogeneity is often addressed through clustering clients (a.k.a clustered FL). However, the existing clustered FL methods remain sensitive and prone to errors, further exacerbated by the DP noise in DPFL systems. In this chapter, we propose a robust algorithm for differentially private clustered FL. To this end and justified by our theoretical results, we propose to use large batch sizes in the first round and cluster clients based on both their model updates as well as their training loss values. Furthermore, for

clustering clients' model updates at the end of the first round, our proposed approach addresses the server's uncertainties by employing Gaussian mixture models (GMM) to reduce the impact of DP and stochastic noise and avoid potential clustering errors. This idea is efficient especially in privacy-sensitive scenarios with more DP noise and leads to a high accuracy for clustering clients.

Acknowledgements

Hereby, I express my gratitude to my PhD supervisor, Yaoliang, who provided me with valuable research advice. I also thank him for his support, patience and guidance.

I also would like to thank the examining committee; Prof. Sh. Asoodeh, Prof. Ch. Weng, Prof. G. Kamath and Prof. H. Zhang for their valuable time.

The Cheriton School of Computer Science staff, especially Paula Roser and Nadine Zinger (the previous and current CS PhD advisors) were always helpful and supportive. I thank them for their help.

Finally, I would like to thank my family, especially my mother, for their constant support.

Dedication

To my family.

Table of Contents

Examining Committee ii

Author's Declaration iii

Statement of Contributions iv

Abstract v

Acknowledgements vii

Dedication viii

List of Figures xiv

List of Tables xxi

1 Introduction 1

1.1 Federated learning 2

1.1.1 FL with one global model 2

1.1.2 Personalized FL 2

1.1.3 Clustered FL 3

1.1.4 Performance fairness in FL 3

1.2 Differential privacy (DP) 4

1.2.1 -DP 4

1.2.2 Approximate (;) DP 5

1.2.3 Rényi DP 5

1.2.4 Zero concentrated DP (zCDP) . . . 6
1.2.5 Exponential mechanism for private selection . . . 6
1.2.6 Differentially private SGD (DPSGD) . . . 7
1.3 Outline and contributions . . . 7

2 An Operator Splitting View of Federated Learning 9
2.1 Preface . . . 9
2.2 Introduction . . . 9
2.3 Background . . . 11
2.4 Federated learning as operator splitting . . . 12
2.4.1 FedAvg as forward-backward splitting . . . 12
2.4.2 FedProx as backward-backward splitting . . . 15
2.4.3 FedSplit as Peaceman-Rachford splitting . . . 18
2.4.4 FedPi as Douglas-Rachford splitting . . . 18
2.4.5 FedRP as Re ection-Projection splitting . . . 20
2.4.6 Effect of data heterogeneity . . . 21
2.5 Uni cation, implementation and acceleration . . . 21
2.6 Summary . . . 24

3 Noise-Aware Algorithm for Heterogeneous Differentially Private Federated Learning 25
3.1 Preface . . . 25
3.2 Introduction . . . 25
3.3 Related work . . . 26
3.4 Notations . . . 28
3.5 Robust-HDP algorithm for heterogeneous DPFL . . . 28
3.5.1 Noise level in clients' DP batch gradients . . . 30
3.5.2 Noise level in clients' DP model updates . . . 30
3.5.3 Optimum aggregation strategy . . . 31
3.5.4 Robust-HDP algorithm . . . 32
3.5.5 Scalability of Robust-HDP with the number of model parameters . . . 33
3.5.6 Reliability of Robust-HDP . . . 34

3.5.7 Privacy analysis of Robust-HDP . . . 34
3.5.8 The optimization side of Robust-HDP . . . 35
3.6 Experiments . . . 36
3.6.1 Experimental setup . . . 36
3.6.2 Experimental results . . . 37
3.7 Summary . . . 41

4 A Provably Robust Algorithm for Differentially Private Clustered Federated Learning 42
4.1 Preface . . . 42
4.2 Introduction . . . 42
4.3 Related work . . . 44
4.4 De nitions, notations and assumptions . . . 45
4.5 Methodology and proposed algorithm . . . 47
4.5.1 R-DPCFL algorithm . . . 48
4.5.2 Reducing GMM uncertainty via using full batch sizes in the rst round and small batch sizes in the subsequent rounds . . . 50
4.5.3 Effect of batch sizes $\{s_i\}_{i=1}^n$ on the separation between clusters . . . 52
4.5.4 Convergence rate of EM for learning GMM . . . 52
4.5.5 Applicability of R-DPCFL . . . 53
4.6 Experiments . . . 53
4.6.1 Baseline selection . . . 54
4.6.2 Results . . . 54
4.7 Summary . . . 57

5 Conclusion 58

References 68

A Appendix of Chapter 2 69
A Experimental Setup . . . 69
A.1 Experimental setup: least squares and logistic regression . . . 69
A.2 Experimental setup: MNIST datasets . . . 70

A.3 Experimental setup: CIFAR-10 dataset 70
B Additional experimental results 72
B.1 Effect of client sampling 72
B.2 Effect of using mini-batch gradient descent locally 74
B.3 Results with error bars for Figure 2.8, Right 74
C Proofs 76

B Appendix of Chapter 3 81
A Experimental setup 81
A.1 Datasets and models 81
A.2 DP training parameters 82
A.3 Algorithms to compare and tuning hyperparameters 82
B Derivations 84
C Used lemmas 86
D Proofs 87
D.1 Proof of Theorem 9 87
D.2 Proof of Theorem 10 87
E Detailed results 97
E.1 Test accuracy comparison 97
E.2 Ablation study on privacy level and number of clients 97
E.3 Precision of Robust-HDP 98
F Additional Experiments 102
F.1 Uniform batch sizes $\{b_i = b\}$, heterogeneous privacy parameters $\{\varepsilon_i\}$ and heterogeneous dataset sizes $\{n_i\}$ 102
F.2 Heterogeneous batch sizes $\{b_i\}$, uniform privacy parameters $\{\varepsilon_i = \}$ and heterogeneous dataset sizes $\{n_i\}$ 104
F.3 Uniform batch sizes $\{b_i = b\}$, uniform privacy parameters $\{\varepsilon_i = \}$ and heterogeneous dataset sizes $\{n_i\}$ 104
F.4 Conclusion: when to use Robust-HDP? 107
G Gradient accumulation 108
H Limitations and future works 108
H.1 Robust-HDP with moderately heterogeneous data distribution 108
H.2 DPFL with highly heterogeneous data split across clients (future work) . 110

C Appendix of Chapter 4 113
A Privacy accounting and budgeting . . . 113
B Experimental setup . . . 114
B.1 Datasets . . . 114
B.2 Models and optimization . . . 114
B.3 MR-MTL formulation . . . 115
B.4 Tuning hyperparameters of baseline algorithms . . . 115
B.5 Implementation of private local clustering for IFCA and R-DPCFL . . . 116
B.6 DP privacy parameters . . . 117
B.7 Gaussian mixture model . . . 117
C Setting hyper-parameters of R-DPCFL . . . 118
C.1 Batch size $b_i^{(t)} > 1$ . . . 118
C.2 The strategy switching time E_c . . . 118
C.3 The number of clusters M . . . 120
D Complete experimental results . . . 122
E Proofs . . . 123
E.1 Proof of Lemma 4 . . . 123
E.2 Proof of Lemma 5 . . . 126
E.3 Proof of Theorem 11 . . . 130
E.4 Formal privacy guarantees of R-DPCFL . . . 130
F The relation between Lemma 4 and the law of large numbers . . . 131
G Further related works . . . 131

List of Figures

2.1 When $f = \iota_C$ is the indicator function of a (closed) set C, $P_f(w)$ is the Euclidean projection of w onto the set C. Similarly, $R_f(v)$ is the Euclidean reflection of v w.r.t. the set C. 12

2.2 Convergence of FedAvg when changing K or C. Top: least squares; Bottom: logistic regression. 14

2.3 Running FedAvg on CIFAR-10 with different local epochs K. The number of clients is 10, local learning rate $= 0.1$. Data points are sampled and distributed to clients non-uniformly. 14

2.4 Optimality gap $f(x_{\text{FedAvg}}) - f^*$ or training loss $f(x_{\text{FedAvg}})$ of (approximate) fixed-point solutions of FedAvg for different learning rates and local epochs K. Different colored lines are for different numbers of local epochs, and dashed lines for different product values $C(K-1)$. Left: least squares (closed-form solution); Middle: logistic regression (6000 communication rounds); Right: nonconvex CNN on the MNIST dataset (200 communication rounds). 15

2.5 Effect of step size and ergodic averaging on FedProx. Left: least squares; Middle: logistic regression; Right: CNN on MNIST. The dashed and solid lines with the same color show the results obtained with and without the ergodic averaging step in Theorem 3, respectively. For exponentially decaying η_t, we use period T equal to 500 for both least squares and logistic regression experiments, and 10 for CNN experiment. 18

2.6 Effect of step size and averaging on FedRP. Left: least squares; Right: logistic regression. The dashed and solid lines with the same color show the results obtained with and without the ergodic averaging step in Theorem 3, respectively. For exponentially decaying η_t, we use period T equal to 500 for both least squares and logistic regression. 20

2.7 The effect of data heterogeneity on the performance of different splitting methods. The top row shows the results for the least squares, and the bottom row shows the results for nonconvex CNN model. Top-Left: small data heterogeneity with $H = 119 \times 10^3$; Top-Middle: moderately data heterogeneity with $H = 7.61 \times 10^6$; Top-Right: large data heterogeneity with $H = 190.3 \times 10^6$. Bottom-Left: i.i.d. data distribution; Bottom-Middle: non-i.i.d. data distribution with maximum 6 classes per client; Bottom-Right: non-i.i.d. data distribution with maximum 2 classes per client. 22

2.8 Effect of Anderson acceleration. Left: least squares with $n = 2$; Middle: logistic regression $= 2$; Right: nonconvex CNN with $= 10$. Dashed lines are the accelerated results. 24

3.1 Security model in heterogeneous DPFL, where client i has local train data $\mathcal{D}_i$ and record-level privacy parameters (ϵ_i, δ_i), and does not trust any external parties. . . . 27

3.2 Plot of z v.s. q obtained from moments accountant (Abadi et al. 2016) in a centralized setting with $E = 200$. As observed, z increases sub-linearly with q (or equivalently with b). 31

3.3 Left: 3D plot of noise variance σ_i^2 (Equation (3.5) with $K_i = 1$; $N_i = 2400$; $_I = 0.01$; $c = 3$; $p = 28939$) based on q_i and the privacy budget ϵ_i. Right: the noise variances $\{\sigma_i^2\}_{i=1}^n$ in a DPFL system with $n = 20$ clients, where $\{(\epsilon_i, q_i)\}_{i=1}^n$ are randomly selected for each client. It clearly shows an approximately sparse pattern(14 of the clients have much smaller noise variance than the other 6). Each bar plot corresponds to a point in the left figure. 32

3.4 Comparison of average test accuracy between studied algorithms. See Tables B.7 to B.10 in the Appendix for detailed results. 38

3.5 Convergence speed comparison on MNIST and Dist6. Minimum algorithm diverged in 1 out of 3 trials. 38

3.6 Performance comparison on MNIST. Left: effect of clients desired privacy on utility (detailed results in Table B.11 in the Appendix). Middle: effect of number of existing clients (privacy parameters of clients are sampled from Dist6) on utility (detailed results in Table B.12). Right: Robustness of Robust-HDP when a random client (client 12 with a moderate value of 0.95) sends falsified version of its to the server for aggregation (privacy parameters of other clients are sampled from Dist5). WeiAvg and PFA are much vulnerable to this falsification. 39

3.7 Left: Precision of Robust-HDP (red) compared to oracle optimum strategy (blue) for CIFAR10 and Dist2, when using the approximation method in Section 3.5.5 with $Q = 1$ and $p^0 = 2 \times 10^5$. Right: Effect of the parameter p^0, used in the approximation method explained in Section 3.5.5, on the precision of the weights returned by Robust-HDP for CIFAR10 (with $p = 11 \times 10^6$). 40

4.1 Loss-based clustering algorithms miscluster in the initial rounds, due to model initialization. Also, even with the assumption of perfect clustering of clients in the first rounds, clustering algorithms based on gradients (model updates) leads to clustering errors in the last rounds, due to the gradients approaching to zero. . . . 46

4.2 Left: Considered threat model in this work, where client i has local train data $\mathcal{D}_i$ and "sample-level" DP privacy parameters (ε, δ), and does not trust any external party. Right: Three main stages of the proposed R-DPCFL algorithm. 48

4.3 Plot of $\mathrm{Var}(\tilde{\Delta}_i^1(b_i^1) \mid \Delta_i^{init})$ (left) and $\mathrm{Var}(\tilde{\Delta}_i^e(b_i^e) \mid \Delta_i^{e,0})$ $(e > 1)$ (right) v.s. both b_i^1 and $b_i^{>1}$. There are two clear takeaways: 1) for all $e \in \{1, \ldots, E\}$, $\mathrm{Var}(\tilde{\Delta}_i^e(b_i^e) \mid \Delta_i^{e,0})$ decreases with b_i^e steeply (from Lemma 4). 2) The effect of $b_i^{>1}$ on $\mathrm{Var}(\tilde{\Delta}_i^1(b_i^1) \mid \Delta_i^{init})$ (left figure) is considerable. The reason is that $b_i^{>1}$ is used in $E-1$ rounds and affects the noise scale $z_i(\varepsilon, \delta, b_i^1, b_i^{>1}, N_i, K, E)$ used by DP-SGD (see Figure 4.4 for the plot of $z_i(\varepsilon, \delta, b_i^1, b_i^{>1}, N_i, K, E)$ v.s. b_i^1 and $b_i^{>1}$. The results are obtained on CIFAR10 from Rényi-DP-Accountant (Mironov et al. 2019) in a setting with $N_i = 6600$, $\varepsilon = 5$, $\delta = 10^{-4}$, $c = 3$, $K = 1$, $E = 200$, $p = \{11, 181, 642, 1\} \times 5 \times 10^{-4}$. 50

4.4 Plot of $z_i(\varepsilon, \delta, b_i^1, b_i^{>1}, N_i, K, E)$ v.s. b_i^1 and $b_i^{>1}$ obtained from Rényi-DP-Accountant (Mironov et al. 2019) in a setting with $N_i = 6600$, $\varepsilon = 5$, $\delta = 10^{-4}$, $K = 1$, $E = 200$. It is clearly observed that the effect of $b_i^{>1}$ is much more than the effect of b_i^1. The reason is that $b_i^{>1}$ is used in $E-1$ rounds, while b_i^1 is used only in the first round. So the value of $b_i^{>1}$ affects z_i the most. 51

4.5 PCA visualization of updates $\{\tilde{\Delta}_i^1\}_{i=1}^n$ on 2D space. Left: $\pi_i = 10$, $b_i^e = 32$ for all i and e. Right: $\pi_i = 10$, $b_i^1 = b^1 = N = 6600$, i.e. full batch sizes (assuming $N_i = N = 6600$ for all clients), and $b_i^{>1} = 32$ for all i. $\mu_m(b^1)$ shows the center of cluster m, when all clients use batch size b^1 during the first round. The model updates are obtained from clients running DP-SGD for $K = 1$ epochs locally on CIFAR10 with covariate shift (rotation) across clusters, and under the same values as in Figure 4.3. 51

4.6 Top: Average test accuracy for different total privacy budgets. Results are from four different runs. 10% means performing local clustering by clients only in 10% of the total number of rounds; i.e. rounds $\{1, \ldots, \lfloor \frac{E}{10} \rfloor\}$ and $\{E_c + 1, \ldots, E_c + \lfloor \frac{E}{10} \rfloor\}$ for R-DPCFL and rounds $\{1, \ldots, \lfloor \frac{E}{10} \rfloor\}$ for IFCA (see Appendix B.5). Figure D.3 in the Appendix includes the Global baseline too. Bottom: Number of times (out of 4 runs) that R-DPCFL and IFCA successfully detect the underlying cluster structure of all existing clients. 55

4.7 Top: Average test accuracy across clients belonging to the minority cluster for different total privacy budgets, averaged over four different runs (random seeds). Bottom: Number of times (out of 4 runs) that R-DPCFL and IFCA successfully detect the minority cluster. 56

4.8 MSS score v.s. for two different local dataset sizes. A small local dataset size can be compensated for by using smaller batch sizes $\{b_i\}_{i=1}^n$ to get a larger MSS score. 56

B.1 Effect of client sampling. Top-Left: least squares with $p_i = 0.5$ (participation probability); Top-Right: logistic regression with $p_i = 0.5$; Bottom-Left: least squares with $p_i = 0.7$; Bottom-Right: logistic regression with $p_i = 0.7$. 73

B.2 The effect of using minibatch gradient descent. Local datasets are divided into 5 minibatches. Top-Left: least squares with $B = 1000$. Top-Right: logistic regression with $B = 200$. Bottom: nonconvex CNN on MNIST with $B = 500$. . 75

B.3 Detailed results for Anderson acceleration for nonconvex optimization problem of CNN training with $= 10$. Dashed lines are the accelerated results. 75

E.1 Comparison of the eigen values of matrices M (left) and L (right) on MNIST dataset. The concentration of eigen values in the right figures around 0 shows that the matrix L returned by Robust PCA, is indeed low rank, while M is not, due to the noise existing in clients model updates. The results for these experiments were reported in Table B.7 (Robust-HDP, Dist3 and Dist9). 100

E.2 Comparison of weight assignments for Dist8 and MNIST with the data split in Table B.2. The assigned weights by baseline algorithms show that their weight assignment strategies are not based on the noise variance in clients model updates, hence suboptimal. The results for this experiment were reported in Table B.7 (Robust-HDP, Dist8). 101

F.3 The behavior of function $1=H(N; b; ; ; K; E)$ as a function of N (dataset size) for an instance value of $= 3$ and batch size $b = 32$. Oscillations in the curves are due to finding z empirically. 106

H.4 Comparison of weight assignments by different algorithms on MNIST, Dist1 and 60 clients with heterogeneous data distribution (maximum 8 labels per client) and uniform batch size b (= 128). The first 40 (last 20) clients, which have the smallest (largest) privacy parameters, get assigned smaller (larger) weights by Robust-HDP than by PFA and WeiAvg. The results for this experiment were reported in Table 3.5 (Robust-HDP, Dist1). 111

H.5 Comparison of weight assignments by different algorithms on MNIST, Dist1 and 60 clients with hetrogeneous data distribution (maximum 8 labels per client) and heterogeneous batch sizes. The weight assignments by Robust-HDP are close to the optimal weight assignment strategy, despite the heterogeneity in the data split. The results for this experiment were reported in Table B.7 (Robust-HDP, Dist1). 112

C.1 The effect of increasing the batch size after the first round, i.e., $b_i^t > 1$, on the model updates $\{\tilde{g}_i^1\}_{i=1}^n$ at the end of the first round. All clients have used full batch sizes in the first round, i.e., $\forall i : b_i^1 = N_i$. The level of noise in $\{\tilde{g}_i^1\}_{i=1}^n$ affects the quality and confidence of the client clustering that the server performs at the end of the first round. As can be observed, for a fixed $\epsilon = 10$, the model updates scatter further in space as $b_i^{t>1}$ increases and different clusters get less separated. This leads to a decrement in the confidence level of the MSS score of the resulting GMM, as $b_i^{t>1}$ increases (mentioned on the top of each plot). All the results are obtained on CIFAR10 with covariate shift (rotation) across clusters. . . . 119

C.2 The minimum pairwise separation score (MSS) or confidence of the GMM learned on $\{\tilde{g}_i^1\}_{i=1}^n$ peaks at the true cluster number, which is equal to 4 in all the plots above. Each figure is for a different value of ϵ (mentioned on top of each figure), and are obtained on CIFAR10 with covariate shift (rotation) across clusters, and 5 different random data splits (5 seeds). All the results are obtained with full batch sizes in the first round and $b_i^{t>1} = 32$ for all i. We can use this observation as a method to find the true number of clusters (M) when it is not given. For larger ϵ, this method work perfectly and even when ϵ is too small, e.g., $\epsilon = 3$, this method works well and predicts the true number of clusters correctly most of the times: 3 out of the 5 curves in the bottom right plot have a peak at $M = 4$ (the true cluster number). and the other 2 curves predicts 5 as the true number, which is the closest and the best alternative for the true value $M = 4$. 121

D.3 Average test accuracy across clients for different total privacy budgets (results are obtained from 4 different random seeds). 10% means performing loss-based clustering by clients only in 10% of the total rounds (E). 122

D.4 Average test accuracy across clients belonging to the minority cluster for different total privacy budgets (results are obtained from 4 different random seeds). 10% means performing loss-based clustering by clients only in 10% of the total rounds (E). 122

D.5 The MSS score of the learned GMM is indicative of whether the true underlying clusters will be detected or not: an MSS score above 2 always leads to correct detection of clusters. Each point is the result of one independent experiment. . . . 123

D.6 The number of EM iterations needed for learning the GMM decreases as b_i^1 increases. Especially, at full batch size ($\forall i : b_i^1 = N_i$), very few iterations of EM are needed. The results are obtained on CIFAR10 with covariate shift across $M = 4$ clusters. According to Lemma 5, using a large enough batch size in the first round makes the underlying clusters in $\{\tilde{g}_i^1\}_{i=1}^n$ more distinguishable. Consequently, according to Theorem 11, convergence rate of EM algorithm for learning the GMM increases with b_i^1. 123

E.7 Left: Distance between the centers of different clusters, i.e. the distance between $\mu_m(b^{-1})$ and $\mu_{m'}(b^{-1})$, decreases k times as b^{-1} increases k times. The three curves in the plot are obtained on CIFAR10 with 4 clusters $m, m' \in \{0, 1, 2, 3\}$ obtained from covariate shift (rotation). The curves are overlapping all with slope 0.95, which is very close to 1. This shows the validity of the approximation in Equation (C.8). Right: Effect of changing batch size b to full batch size in the first round on the noise scale z. In the denominator, b^{-1} is equal to $b' > 1$. Results are obtained from Rényi-DP accountant (Mironov et al. 2019) with $N = 50000$, $K = 1$ and $E = 200$. For each value of b, we have shown the results for seven values of $b' > 1$ 129

List of Tables

2.1 A unifying framework (2.15)-(2.17) for FL. Note that (a) FedAvg replaces the proximal update P_f^t with a gradient update G_f^t; (b) ? indicates properties that remain to be studied; (c) "sampling" refers to selecting a subset of clients while "stochastic" refers to updating with stochastic gradient. 23

3.1 Features of different heterogeneous DP FL algorithms. : needed at server, : not needed. 28

3.2 Notations used in Chapter 3 29

3.3 Distributions of privacy parameter(s), from which clients' privacy parameters are sampled. 37

3.4 The average per parameter noise variance (Equation (3.5) and Equation (3.3)) normalized by used learning rate ($\frac{\sum_{i=1}^n w_i^{e2}\sigma_i^2}{p\,l^2}$) in the aggregated model update ($\sum_{i=1}^n w_i^e \tilde{} {}_i^e$) at the end of first round ($e = 1$) on FMNIST with $E = 200$. Due to the projection used in PFA, computation of its noise variance was not possible. Results for Robust-HDP are shown with std variation across three experiments. . 39

3.5 Comparison of different algorithms (on MNIST, $E = 200$) with heterogeneous data split (maximum 8 labels per client) and 60 clients in the system all using uniform batch size 128. 40

4.1 Used notations in chapter 4. 46

A.1 MNIST model 71

A.2 CIFAR-10 model 71

B.1 CNN model for classification on MNIST/FMNIST datasets 82

B.2 Details of the experiments and the used datasets in the main body of the paper. ResNet-18/34 are the residual neural networks defined in (He et al. 2015). CNN: Convolutional Neural Network defined in Table B.1. 82

B.3 The learning rates used for training with each algorithm on MNIST dataset 84

B.4 The learning rates used for training with each algorithm on FMNIST dataset . . . 84
B.5 The learning rates used for training with each algorithm on CIFAR10 dataset . . . 84
B.6 The learning rates used for training with each algorithm on CIFAR100 dataset. . . . 85
B.7 Comparison of different algorithms (on MNIST, $E = 200$). FedAvg achieves 98.6%. . . . 97
B.8 Comparison of different algorithms (on FMNIST, $E = 200$). FedAvg achieves 90.28% . . . 97
B.9 Comparison of different algorithms (on CIFAR10, $E = 200$). FedAvg achieves 73.55% . . . 98
B.10 Comparison of different algorithms (on CIFAR100, $E = 200$). FedAvg achieves 61.80% . . . 98
B.11 Detailed results for ablation study on the privacy level of clients in Figure 3.6, left. 98
B.12 Detailed results for ablation study on number of clients in Figure 3.6, middle. . . . 99
B.13 Comparison of different algorithms on CIFAR10 with uniform batch sizes = 50, heterogeneous privacy parameters ε_i and heterogeneous dataset sizes. FedAvg ($E = 200$) achieves 77.58%. We have dropped the "minimum" algorithm due to its very low performance. . . . 103
B.14 Comparison of different algorithms on CIFAR10 with heterogeneous batch sizes from {50, 100}, uniform privacy parameters $\varepsilon_i = 8$ and heterogeneous dataset sizes. FedAvg ($E = 200$) achieves 73.55%. "minimum " algorithm is equivalent to DPFedAvg in this case. . . . 105
B.15 Comparison of different algorithms on CIFAR10 with uniform batch sizes = 32, uniform privacy parameters $\varepsilon_i = 8$ and heterogeneous dataset sizes. FedAvg ($E = 200$) achieves 73.55%. "minimum " algorithm is equivalent to DPFedAvg in this case. . . . 107
B.16 Comparison of different algorithms on CIFAR10 with uniform batch sizes = 32, uniform privacy parameters $\varepsilon_i = 8$ and heterogeneous dataset sizes in terms of the std of clients' test accuracies and test accuracy of the client with the smallest train set (in parentheses). "minimum" algorithm is equivalent to DPFedAvg in this case. . . . 107
B.17 Comparison of different algorithms (on MNIST, $E = 200$) with heterogeneous data split (maximum 8 labels per client) and 60 clients in the system with heterogeneous batch sizes. . . . 110

C.1 CNN model for classification on MNIST/FMNIST datasets . . . 114
C.2 Details of the used datasets in the main body of the paper. ResNet-18 is the residual neural networks defined in He et al. 2015. CNN: Convolutional Neural Network defined in Table B.1. . . . 115

Chapter 1

Introduction

The fast progress in computational resources has enabled a rapid advancement in development of large deep learning (DL) models for various learning tasks. Central to this development is the need to data to train the models. There are various scenarios that the required data is naturally distributed among some data holders, e.g., among the clients in a communication network or the hospitals in a country. This scattered nature of data often comes from the privacy concerns of the holders, which does not let them to share their data with others or a central server. For instance, the clients in a communication network do not like to share the images stored in their devices with others. Also, it could also be expensive to transfer the data stored in different locations to a central server, e.g., the set of patients' medical records stored in different hospitals of a country.

Federated Learning (FL) (McMahan et al. 2017a) has been proposed as a decentralized machine learning (ML) framework for training ML/DL models under the orchestration of a central server on the data scattered between some devices/organizations, which we refer to as clients. The clients do not share their private data with neither other clients nor the central server. This is a bene cial feature of FL, which provides some level of informal data privacy to the participating clients. Nevertheless, there are some challenges for the deployment of FL frameworks in real-world applications. Among them are the expensive communication overhead between the server and the clients (McMahan et al. 2017a) as well as the potential data heterogeneity across the clients, which leads to disparate performance of the learned model on the participating clients' data (Li et al. 2020a,d; Zhang et al. 2023). Furthermore, in the presence of sensitive data in clients' datasets, they may need a rigorous data privacy guarantee in order to participate in FL (Geiping et al. 2020; Hitaj et al. 2017; Rigaki and García 2020; Wang et al. 2019; Zhu et al. 2019). The aforementioned challenges are vital to be addressed to ensure that FL systems can be reliably and effectively deployed.

The primary aim of this thesis is to address the above-mentioned challenges, hoping that it will make the promising FL framework more applicable for real-world scenarios with distributed data. In the following sections of this chapter, we will present an introduction to various FL algorithms, performance parity in FL systems and also differential privacy. We will nally summarize the contributions of this thesis.

1.1 Federated learning

We start with a more detailed introduction to FL, its objective and some common FL algorithms.

1.1.1 FL with one global model

As mentioned earlier, an FL system consists of a central server, and some participating clients. Let n denote the number of participating clients, where client i has the loss function $f_i(\theta)$, parameterized by θ, on its local dataset $\mathcal{D}_i$. Then, the problem of FL is about collectively and efficiently optimizing the individual objective functions f_i in a decentralized, privacy-preserving and low-communication way. Following McMahan et al. 2017a, many existing FL algorithms aim at minimizing the following arithmetic average performance across the clients:

$$\min_{\theta} f(\theta) = \sum_{i=1}^{n} w_i f_i(\theta); \tag{1.1}$$

where the averaging weights $\{w_i\}_{i=1}^n$ sum to 1. Different FL algorithms can be employed to solve the above problem, and the output will be one single model parameter θ for all the participating clients. As mentioned earlier, the data distribution across clients can be heterogeneous, meaning that the data distribution can vary from one client to another client. Consequently, the single model parameter θ may not result in a satisfying performance for all clients. Personalized FL algorithms have been proposed to address this challenge.

1.1.2 Personalized FL

Model personalization (Evgeniou and Pontil 2004; Li et al. 2020b) has been used as a technique to ensure a satisfying model performance for all participating clients. Instead of learning one single global model for all clients, personalized FL learns one model specifically for each client, which is adapted to its local data distribution. Ditto (Li et al. 2020b) is a personalized FL algorithm: Assuming θ^* is the solution of the problem (1.1), Ditto solves the following problem for each client i to find its personalized model parameter:

$$\min_{v_i} g_i(v_i; \theta^*) := f_i(v_i) + \frac{\lambda}{2}\|v_i - \theta^*\|^2;$$

where λ is a regularization parameter tuning the level of personalization for clients. Mean-regularized multi-task learning (MR-MTL) (Evgeniou and Pontil 2004) is another method for personalized FL, which has less computational overhead for personalization compared to Ditto. The objective function of MR-MTL can be expressed as:

$$\min_{\theta_i, i \in \{1,\dots,n\}} \sum_{i=1}^{n} g_i(\theta_i) \quad \text{with} \quad g_i(\theta_i) = f_i(\theta_i) + \frac{\lambda}{2} \| \theta_i - \bar{\theta} \|_2^2,$$

where $\bar{\theta} = \frac{1}{n} \sum_{i=1}^{n} \theta_i$ is the average model parameter across clients. With $\lambda = 0$, MR-MTL reduces to local training. A larger regularization term λ encourages local models to be closer to each other, i.e. more federation. However, MR-MTL may not recover the solution of problem (1.1) as $\lambda \to \infty$. We refer to algorithm A1 in (Liu et al. 2022) for more details about MR-MTL.

1.1.3 Clustered FL

While personalized FL has been proposed to address mild data heterogeneity, clustered FL has been proposed for scenarios where clients can be grouped into some clusters based on their data similarity, and there is a high data heterogeneity across different clusters. In this scenario, due to the high level of data heterogeneity in the system, learning one global model leads to a poor model performance for all clients. Proposed clustered FL algorithms aim at detecting the true underlying cluster structure of clients followed by learning one model for each cluster (Briggs et al. 2020; Ghosh et al. 2020; Mansour et al. 2020; Ruan and Joe-Wong 2021; Sattler et al. 2019; Werner et al. 2023). Some existing algorithms group clients based on their loss values (Dwork et al. 2012; Ghosh et al. 2020; Liu et al. 2022; Mansour et al. 2020; Ruan and Joe-Wong 2021), i.e. client i gets assigned to the cluster model resulting in the smallest train loss on its local dataset $\mathcal{D}_i$. Some other algorithms group them based on their model updates (based on e.g., their euclidean distance (Briggs et al. 2020; Werner et al. 2023) or cosine similarity (Sattler et al. 2019)). As shown by Werner et al. 2023, the algorithms are prone to clustering errors in the early rounds of FL training – due to gradient stochasticity, model initialization or the form of loss functions far from their optima – which can even propagate in the subsequent rounds.

1.1.4 Performance fairness in FL

Performance fairness across clients is desired to ensure a satisfying performance for the trained model (e.g., test accuracy) across all clients. Depending on the level of data heterogeneity in the system, performance fairness may be achievable by training one global model for all clients or by model personalization. If there is a slight data heterogeneity across clients, learning one fair global model for all is a possible task. For instance, Mohri et al. 2019 proposed Agnostic Federated Learning (AFL), which replaces the arithmetic average in problem (1.1) with the largest loss function of all clients in each round and finds the common model parameter through a minimax optimization framework. Similarly, Li et al. 2020a proposed an algorithm called TERM based on weighted average of clients' loss functions. Another fair FL (q-FFL) (Li et al. 2020d) is an algorithm inspired by α-fairness in wireless networks. Similarly, the work by Zhang et al. 2023 proposed PropFair based on Proportional Fairness, and showed that all the aforementioned fair FL algorithms can be unified into a generalized mean framework.

1.2 Differential privacy (DP)

Differential privacy (Dwork et al. 2006a,b; Dwork and Roth 2014) is a standard method for ensuring data privacy in ML, and it has also been combined with FL in order to provide rigorous privacy guarantees to the participating clients. There are different de nitions of DP, as explained in the following.

1.2.1 -DP

The rst and main de nition of DP is -DP, which is explained in the following de nition:

De nition 1 (-DP, Dwork et al. 2006b) A randomized mechanism $M : A \to R$ with domain A and range R satis es -DP if for any two adjacent inputs $D, D' \in A$, and for any subset of outputs $S \subseteq R$ it holds that:

$$Pr[M(D) \in S] \leq e^{\varepsilon} Pr[M(D') \in S].$$

The notion of adjacency of datasets is typically used to capture the contribution of a minor change in the input on the output. Most prior work on differentially private machine learning use record-level notion of adjacency (Abadi et al. 2016). In record-level adjacency, two datasets D and D' are called record-level adjacent if D can be constructed from D' by replacing a single data sample from D', i.e. they differ in only one data point and $|D| = |D'|$.

Laplace mechanism is a mechanism used for making -DP version of algorithms from their non-DP, deterministic versions. The mechanism randomizes the output of an arbitrary f query applied to a dataset D as:

$$L_{\varepsilon}(f(D)) = f(D) + L(0; \frac{\Delta_1 f}{\varepsilon});$$

where $\Delta_1 f$ is the l_1-sensitivity of the query f:

$$\Delta_1 f = \max_{D, D'} \| f(D) - f(D') \|_1$$

The randomized output of the Laplace mechanism $L_{\varepsilon}(f(D))$ satis es -DP. Different -DP mechanisms can be composed with each other. The basic composition theorem about standard differential privacy (-DP) says that if f and g are respectively ε_1-DP and ε_2-DP, then simultaneous release of f and g is a $(\varepsilon_1 + \varepsilon_2)$-DP mechanism.

1.2.2 Approximate (;) DP

An approximation of the above-mentioned de nition is the approximate (;) DP, as follows:

De nition 2 ((;)-DP (Dwork et al. 2006a)) A randomized mechanism $M : A \to R$ with domain A and range R satis es (;)-DP if for any two adjacent inputs $D, D' \in A$, which differ only by a single record (by replacement), and for any measurable subset of outputs $S \subseteq R$ it holds that

$$\Pr[M(D) \in S] \leq e^{\varepsilon} \Pr[M(D') \in S] + \delta.$$

Gaussian mechanism, which randomizes the output of a non-private computation of a dataset D as $G_\sigma f(D) = f(D) + N(0, \sigma^2)$, provides (;)-DP for a continuum of pairs ( ): it is (;)-DP for all $\delta \in (0, 1)$ and $\varepsilon > \sqrt{2 \ln(1.25/\delta)}\, \Delta_2 f$, where $\Delta_2 f = \max_{D, D'} \| f(D) - f(D') \|_2$ is the l_2-sensitivity of the query f with respect to its input. For a xed, the guaranteed privacy parameter resulting from running Gaussian mechanism depends on the quantity $\frac{\sigma}{\Delta_2 f}$, which is called noise scale. Data subsampling can enhance the privacy guarantees (s; of) a DP mechanism. We have the following property about privacy ampli cation by subsampling.

Lemma 1 (Privacy Ampli cation by Subsampling (Dwork et al. 2006a)) If $M : X^m \to Y$ satis es (;)-DP, with respect to the replacement relationship on sets of size m, $M' = M \circ S : X^n \to Y$ satis es $\log(1 + m(e^{\varepsilon} - 1)/n)$; m/n -DP (n m), where S is random sampling operator, which samples m out of n uniformly.

A useful property of the relaxed; ( )-DP is "advanced composition", which can be used for accounting the privacy guarantees of k-fold composition of (;)-DP mechanisms:

Lemma 2 (Advanced Composition Property (Dwork et al. 2010)) $\forall \varepsilon; \delta; \delta' > 0; k \in \mathbb{N}$; the family of (;)-DP mechanisms satis es $k((e^{\varepsilon} - 1) + \sqrt{2k \ln(1/\delta')}\varepsilon; k\delta + \delta')$-DP under k-fold adaptive composition.

Despite having the above property, we use the moments accountant (Abadi et al. 2016) for accounting the composition of some (;) DP mechanisms. The accountant returns a tighter privacy guarantee than the advanced composition property above.

1.2.3 Rényi DP

There is another relaxed de nition of DP, named Rényi DP (RDP), which we use for tight privacy accounting of composition of some heterogeneous private mechanisms. It is de ned as follows:

De nition 3 (Rényi Differential Privacy (RDP) (Mironov 2017)) A randomized mechanism $M : A \to R$ with domain D and range R satis es (;)-RDP with order if for any two adjacent inputs $D, D' \in A$, which differ only in a single record (by replacement), it satis es

$$D \quad M(D) \| M(D') \quad ;$$

where $D_\alpha(P \| Q)$ is the Rényi divergence between distributions P and Q:

$$D_\alpha(P \| Q) := \frac{1}{\alpha - 1} \log E_{x \sim p} \left(\frac{P(x)}{Q(x)} \right)^{\alpha - 1}.$$

For $\alpha = 1$, we have $D_1(P \| Q) := E_{x \sim p} \left[\log \frac{P(x)}{Q(x)} \right]$, which is the KL divergence between P and Q. The following lemma is about conversion of (α, ε)-RDP to the approximate (ε, δ)-DP.

Lemma 3 (Converting RDP (Bun and Steinke 2016; Canonne et al. 2020)). If a mechanism M satisifes $(\alpha, \varepsilon(\alpha))$-RDP, then for any $\delta > 0$, it satis es $(\varepsilon(\delta), \delta)$-DP, where

$$\varepsilon(\delta) = \inf_{\alpha > 1} \varepsilon(\alpha) + \frac{1}{\alpha - 1} \log \frac{1}{\delta} + \log \left(1 - \frac{1}{\alpha} \right).$$

RDP is conveniently linearly composable, as explained in the following theorem.

Theorem 1 (Linear Composition of RDP (Mironov 2017)). If mechanism M_i satis es (α, ε_i) RDP for $i = 1, \ldots, k$, then the composed mechanism $M_1 \circ \cdots \circ M_k$ satis es $(\alpha, \sum_{i=1}^{k} \varepsilon_i)$ RDP.

Therefore, if an algorithm has E steps (e.g., E batch gradient update steps) and each satis es (α, ε)-RDP, the algorithm will satisfy $(\alpha, E\varepsilon)$-RDP. It can also be used for composition of heterogeneous private mechanisms, as in the above theorem. There are different implementations of the RDP privacy accountant, e.g., the TensorFlow Privacy used in this thesis (McMahan et al. 2019).

1.2.4 Zero concentrated DP (zCDP)

Another relaxed de nition of differential privacy is zero concentrated differential privacy (zCDP). Being ε-zCDP is equivalent to being $(\alpha, \varepsilon\alpha)$-RDP simultaneously for all $\alpha > 1$ (Bun and Steinke 2016). Therefore, standard RDP accountants, e.g., the aforementioned TensorFlow Privacy RDP accountant, can be used for accounting zCDP mechanisms as well.

1.2.5 Exponential mechanism for private selection

Exponential Mechanism is a standard method for private selection from a set of candidates. The selection is based on a score, which is assigned to every candidate (Rogers and Steinke 2021). Let us assume there is a private dataset D and a score function $s : D \times [M] \to \mathbb{R}$, which evaluates a set of M candidate models on the dataset D. The goal is to select the candidate with the highest score, i.e. $\operatorname{argmax}_{m \in [M]} s(D, m)$. Exponential mechanism performs this selection privately: It sets the probability of choosing any candidate $m \in [M]$ as:

$$\Pr[m] = \frac{\exp(\frac{\varepsilon_{\text{select}}}{2} s(D; m))}{\sum_{m' \in [M]} \exp(\frac{\varepsilon_{\text{select}}}{2} s(D; m'))};$$

where Δ is the sensitivity of the scoring function s to the replacement of a data sample in D. It can be shown that the private selection performed by exponential mechanism satisfies $\frac{1}{8}\varepsilon_{\text{select}}^2$-zCDP (Bun and Steinke 2016). From the last paragraph, we know it also satisfies $(\alpha, \frac{1}{8}\varepsilon_{\text{select}}^2\alpha)$-RDP for $\alpha > 1$. One can implement exponential mechanism by noisy selection with Gumbel noise: add independent noises from Gumbel distribution with scale $\frac{2\Delta}{\varepsilon_{\text{select}}}$ to candidate scores $s(D; m)$, for $m \in [M]$, and select the candiate with the maximum noisy score. The larger the sensitivity Δ to replacement of a single sample in D, the larger the required noise scale.

1.2.6 Differentially private SGD (DPSGD)

DP was initially designed to limit the amount of information leakage about individual data when releasing an aggregated information from a database. Abadi et al. 2016 extended DP to DL by introducing DPSGD, which is widely used for private training of ML models. DPSGD applies an ℓ_2 norm clipping to every sample gradient in a batch of sample gradients and applies Gaussian mechanism to their average by adding a calibrated noise to it. The model parameters are iteratively updated with the privatized batch gradients, and the DP guarantee is retained by sequentially composing the Gaussian mechanisms with DP guarantees (Abadi et al. 2016; Mironov et al. 2019). There is always a tradeoff between privacy and utility of DP trained models. Improving the tradeoff is constantly an important objective in related works.

1.3 Outline and contributions

Each of the following chapters are centered around one of my full-length articles. The main contributions of the following chapters are summarized as follows:

In Chapter 2, we study the optimization theory behind FL and

- We connect FL with operator splitting theory.
- We shed new insights on existing algorithms, re ne their convergence analysis and suggest new algorithmic variants.
- Present a uni cation of current algorithms and propose strategies for their acceleration.

In Chapter 3, we augment FL with DP for providing rigorous data privacy guarantees to clients.

- We show the effect of privacy parameter ε and batch/dataset size on the noise level in clients' updates, which are uploaded to the server.
- We propose "Robust-HDP", a robust algorithm with a noise-aware aggregation strategy for heterogeneous DPFL, where each client has its own privacy preference.
- Assuming heterogeneous dataset sizes, heterogeneous batch sizes, non-uniform and varying aggregation weights and partial participation of clients simultaneously, we prove convergence of our proposed algorithm under mild assumptions.
- In various privacy heterogeneity scenarios across clients, we show that Robust-HDP improves utility and convergence speed while respecting clients' privacy.

Finally, in Chapter 4, we study DPFL with structured data heterogeneity across clients.

- We propose a DP clustered FL algorithm (R-DPCFL), which combines information from both clients' model updates and their loss values. The algorithm is robust and achieves high-quality clustering of clients, even in the presence of DP noise in the system.
- We theoretically prove that increasing clients' batch sizes in the first round (and decreasing them in the subsequent rounds) reduces the amount of noise in clients' model updates at the end of the first round and enhances the server's ability to cluster the clients based on them.
- We show that utilizing sufficiently large client batch sizes in the first round (and sufficiently small batch sizes in the next rounds) enables super-linear convergence rate for learning a GMM on clients' model updates. This enables us to soft cluster clients using GMM with a low computational overhead.
- We extensively evaluate our proposed algorithm DP clustered FL across diverse datasets and scenarios, and demonstrate its effectiveness in detecting the underlying cluster structure of clients, which leads to an overall utility improvement for the system.

Chapter 2

An Operator Splitting View of Federated Learning

2.1 Preface

Over the past few years, the federated learning (FL) community has witnessed a proliferation of new FL algorithms. However, our understating of the theory of FL is still fragmented, and a thorough, formal comparison of these algorithms remains elusive. Motivated by this gap, we show that many of the existing FL algorithms can be understood from an operator splitting point of view. This unification allows us to compare different algorithms with ease, to refine previous convergence results and to uncover new algorithmic variants. In particular, our analysis reveals the vital role played by the step size in FL algorithms. The unification also leads to a streamlined and economic way to accelerate FL algorithms, without incurring any communication overhead. We perform numerical experiments on both convex and nonconvex models to validate our findings.

2.2 Introduction

The accompanying (moderate) computational power of modern smart devices such as phones, watches, home appliances, cars, etc., and the enormous data accumulated from their interconnected ecosystem have fostered new opportunities and challenges to train/tailor modern deep models. Federated learning (FL), as a result, has recently emerged as a massively distributed framework that enables training a shared or personalized model without infringing clients' privacy. Tremendous progress has been made since the seminal work of McMahan et al. (2017b), including algorithmic innovations (Huo et al. 2020; Li et al. 2020c; Pathak and Wainwright 2020; Reddi et al. 2020; Wang et al. 2020; Yurochkin et al. 2019), convergence analysis (Gorbunov et al. 2021; Khaled et al. 2020a,b; Li et al. 2020e; Malinovskiy et al. 2020), personalization (Deng et al. 2020; Diao et al. 2021; Dinh et al. 2020; Mansour et al. 2020; Zhang et al. 2021), privacy protection (Augenstein et al. 2020; Nasr et al. 2019), model robustness (Bagdasaryan et al. 2020; Bhagoji et al. 2019;

Reisizadeh et al. 2020; Sun et al. 2019), fairness (Hu et al. 2022; Malekmohammadi and Yu 2025; Mohri et al. 2019; Zhang et al. 2023), standardization (Caldas et al. 2019; He et al. 2020), applications (Smith et al. 2017), just to name a few. We refer to the excellent surveys of Kairouz et al. 2021; Li et al. 2019; Yang et al. 2019 and the references therein for the current state of affairs in FL.

The main goal of this chapter is to take a closer examination of some of the most popular FL algorithms, including FedAvg (McMahan et al. 2017b), FedProx (Li et al. 2020c), and FedSplit (Pathak and Wainwright 2020), by connecting them with the well-established theory of operator splitting in optimization. In particular, we show that FedAvg corresponds to forward-backward splitting and we demonstrate a trade-off in its step size and number of local epochs, while a similar phenomenon has also been observed in (Charles and Konečný 2021; Li et al. 2020e; Malinovskiy et al. 2020; Pathak and Wainwright 2020). FedProx, on the other hand, belongs to backward-backward splitting, or equivalently, forward-backward splitting on a related but regularized problem, which has been somewhat overlooked in the literature. Interestingly, our results reveal that the recent personalized model in (Dinh et al. 2020) is exactly the problem that FedProx aims to solve. Moreover, we show that when the step size in FedProx diminishes (sublinearly fast), then it actually solves the original problem, which is contrary to the observation in (Pathak and Wainwright 2020) where the step size is fixed and confirms again the importance of step size in FL. For FedSplit, corresponding to Peaceman-Rachford splitting (Lions and Mercier 1979; Peaceman and Rachford 1955), we show that its convergence (in theory) heavily hinges on the strong convexity of the objective and hence might be less stable for nonconvex problems.

Inspecting FL through the lens of operator splitting theory also allows us to immediately uncover new FL algorithms, by adapting existing splitting algorithms. Indeed, we show that Douglas-Rachford splitting (Douglas and Rachford 1956; Lions and Mercier 1979) (more precisely the method of partial inverse of Spingarn 1985) yields a (slightly) slower but more stable variant of FedSplit, while at the same time shares a close connection to FedProx: the latter basically freezes the update of a dual variable in the former. We also propose FedDR by combining the reflector in FedSplit and the projection (averaing) in FedAvg and FedProx, and extend an algorithm of Bauschke and Kruk (2004). We improve the convergence analysis of FedDR and empirically demonstrate its competitiveness with other FL algorithms. We believe these results are just the tip of the ice-berg and much more progress can now be expected: the marriage between FL and operator splitting theory simply opens a range of new possibilities to be explored.

Our holistic view of FL algorithms suggests a very natural unification, building on which we show that the aforementioned algorithms reduce to different parameter settings in a grand scheme. We believe this is an important step towards standardizing FL from an algorithmic point of view, in addition to standard datasets, models and evaluation protocols as already articulated in (Caldas et al. 2019; He et al. 2020). Our unification also allows one to compare and implement different FL algorithms with ease, and more importantly to accelerate them in a streamlined and economic fashion. Indeed, for the first time we show that Anderson acceleration (Anderson 1965; Fu et al. 2020), originally proposed for nonlinear fixed-point iteration, can be adapted to accelerate existing FL algorithms and we provide practical implementations that do not incur any communication overhead. We perform experiments on both convex and nonconvex models to validate our findings,

and compare the above-mentioned FL algorithms on an equal footing.

We proceed in §2.3 with some background introduction. Our main contributions are presented in §2.4, where we connect FL with operator splitting, shed new insights on existing algorithms, suggest new algorithmic variants and refine convergence analysis, and in §2.5, where we present a unification of current algorithms and propose strategies for further acceleration. We conclude in §2.6 with some future directions. All proofs are deferred to Section C in the appendix.

2.3 Background

In this section, we recall the federated learning (FL) framework introduced in Chapter 1. We consider n clients (edge devices), where the i-th client aims at minimizing a function $f_i : \mathbb{R}^p \to \mathbb{R}; i = 1, \dots, n$, defined on a shared model parameter $x \in \mathbb{R}^p$. Typically, each client function f_i depends on the respective client's local (private) dataset $\mathcal{D}_i$. The main goal in FL is to collectively and efficiently optimize individual objectives $\{f_i\}$ in a decentralized, privacy-preserving and low-communication way. Following McMahan et al. (2017b), many existing FL algorithms fall into the natural formulation that optimizes the (arithmetic) average performance:

$$\min_{x \in \mathbb{R}^p} f(x), \quad \text{where} \quad f(x) := \sum_{i=1}^n w_i f_i(x). \tag{2.1}$$

The weights w_i here are nonnegative and w.l.o.g. sum to 1. Below, we assume they are specified beforehand and fixed throughout.

To facilitate our discussion, we recall the Moreau envelope and proximal map of a function[1] $f : \mathbb{R}^p \to \mathbb{R} \cup \{\infty\}$, defined respectively as:

$$M_f(x) = \min_z \; f(z) + \tfrac{1}{2\gamma}\|z - x\|_2^2, \qquad P_f(x) = \operatorname*{argmin}_z \; f(z) + \tfrac{1}{2\gamma}\|z - x\|_2^2,$$

where $\gamma > 0$ is a parameter acting similarly as the step size. We also define the reflector

$$R_f(x) = 2P_f(x) - x.$$

As shown in Figure 2.1, when $f = \delta_C$ is the indicator function of a (closed) set $C \subseteq \mathbb{R}^p$, $P_f(x) = P_C(x)$ is the usual (Euclidean) projection onto the set C, while $R_f(x) = R_C(x)$ is the reflection of x w.r.t. C.

We are now ready to reformulate our main problem of interest (2.1) in a product space:

$$\min_{z \in H} f(z) = \langle \mathbf{1}, f(z) \rangle, \quad \text{where} \quad f(z) := f_1(z_1), \dots, f_n(z_n), \tag{2.2}$$

$\mathbf{1}$ is the vector of all 1s,

$$H := \{z = (z_1, \dots, z_n) \in \mathbb{R}^{pn} : z_1 = \cdots = z_n\}$$

[1] We allow the function f to take $+\infty$ when the input is out of our domain of interest.

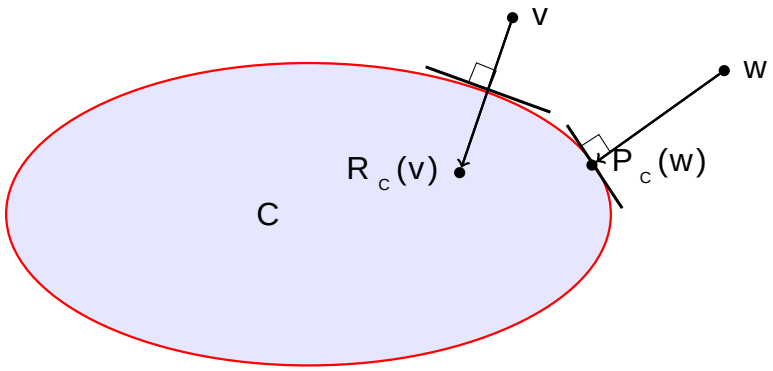

Figure 2.1: When $f = \delta_C$ is the indicator function of a (closed) set C, $P_f(w)$ is the Euclidean projection of w onto the set C. Similarly, $R_f(v)$ is the Euclidean re ection of v w.r.t. the set C.

and we equip with the inner product

$$\langle \; ; z\rangle = \langle \; ; z\rangle := \sum_i \quad {}_i \; {}_i^{>} z_i :$$

Note that the complement $H^? = \{ \; : \sum_i \; {}_i \; {}_i = 0\}$, and

$$P_H(\;) = (\; ; : : : ; \;); \quad \text{where} \quad := \sum_i \; {}_i \; {}_i ; \text{ and } R_H(\;) = (2 \quad {}_1; : : : ; 2 \quad {}_n):$$

We de ne the (forward) gradient update map w.r.t. a (sub)differentiable function f:

$$G_f := \mathrm{Id} \quad @f; \quad 7! \quad @f(\;):$$

When f is differentiable and convex, we note that $R_f = G_f \quad P_f$ and $r \; M_f = \frac{\mathrm{Id} \quad P_f}{}$.

2.4 Federated learning as operator splitting

Following Pathak and Wainwright (2020), in this section we interpret existing FL algorithms (FedAvg , FedProx , FedSplit) from the operator splitting point of view. We reveal the importance of the step size, obtain new convergence guarantees, and uncover some new and accelerated algorithmic variants.

2.4.1 FedAvg as forward-backward splitting

The FedAvg algorithm of McMahan et al. (2017b) is essentially k-step version of the forward-backward splitting approach of Bruck (1977):

$$_{t+1} \quad P_H G_{f;k}^{\;t} \quad _t; \quad \text{where} \quad G_{f;k}^{\;t} := \underbrace{G_f^{\;t} \quad G_f^{\;t} \quad G_f^{\;t}}_{k \text{ times}};$$ (2.3)

where we perform k steps of (forward) gradient updates w.r.t. f_i followed by 1 (backward) proximal update w.r.t. H. To improve efficiency, McMahan et al. (2017b) replace G_i^t with $\hat{G}_i^t$ where the (sub)gradient is approximated on a minibatch of training data and P_H with $\hat{P}_H$ where we only average over a chosen subset $S_t \subseteq [n] := \{1, \ldots, n\}$ of clients. Note that the forward steps $G_{i,k}^t$ is performed in parallel at the client side while the backward step $\hat{P}_H$ is performed at the server side.

The number of local steps k turns out to be a key factor: Setting $k = 1$ reduces to the usual (stochastic) forward-backward algorithm and enjoys the well-known convergence guarantees. On the other hand, setting $k = \infty$ (and assuming convergence on each local function f_i) amounts to (repeatedly) averaging the chosen minimizers of f_i's, and eventually converges to the average of minimizers of all n client functions. In general, the performance of the fixed point solution of FedAvg appears to have a dependency on the number of local steps k and the step size η. Let us illustrate this with quadratic functions $f_i(x) = \frac{1}{2}\|A_i x - b_i\|_2^2$ and non-adaptive step size η, where we obtain the fixed-point $x_{\text{FedAvg}}(k)$ of FedAvg in closed-form (Pathak and Wainwright 2020):

$$x_{\text{FedAvg}}(k) = \Big(\sum_{i=1}^n \frac{1}{k}\sum_{j=0}^{k-1} A_i^\top A_i (I - \eta A_i^\top A_i)^j\Big)^{-1} \Big(\sum_{i=1}^n \frac{1}{k}\sum_{j=0}^{k-1} (I - \eta A_i^\top A_i)^j A_i^\top b_i\Big).$$

For small η, we may apply Taylor expansion and ignore the higher order terms:

$$\frac{1}{k}\sum_{j=0}^{k-1} (I - \eta A_i^\top A_i)^j \approx \frac{1}{k}\sum_{j=0}^{k-1} (I - \eta j A_i^\top A_i) = I - \frac{(k-1)}{2}\eta A_i^\top A_i. \tag{2.4}$$

Thus, we observe that the fixed point $x_{\text{FedAvg}}(k)$ of FedAvg depends on $\eta(k-1)$, up to higher order terms. In particular, when $k = 1$, the fixed point does not depend on η (as long as it is small enough to guarantee convergence), but for $k > 1$, the solution that FedAvg converges to does depend on η. Moreover, the final performance is almost completely determined by $\eta(k-1)$ in this quadratic setting when η is small.

Figure 2.2 illustrates a trade-off where larger local epochs k and learning rate η leads to faster convergence (in terms of communication rounds) during the early stages, at the cost of a worse final solution. Similar results are also reported by e.g., Charles and Konečný 2021; Li et al. 2020e; Malinovskiy et al. 2020; Pathak and Wainwright 2020. As observed, when η is fixed, larger local epochs k leads to faster convergence (in terms of communication rounds) during the early stages, at the cost of a worse final solution. When local epochs k is fixed, same trade-off holds true for increasing η.

In Figure 2.3, contrary to what we observe in the previous convex experiments where fixed learning rate with less local epochs leads to a better final solution at the expense of slower convergence, we do not see similar trade-offs in this nonconvex setting, (at least) given the computation we can afford. This is probably because the solution we get is still in early stage due to the combined complexity of CNN and CIFAR-10 dataset, compared to the relatively simple least squares and logistic regression we had above.

Figure 2.2: Convergence of FedAvg when changing K or η. Top: least squares; Bottom: logistic regression.

Figure 2.3: Running FedAvg on CIFAR-10 with different local epochs K. The number of clients is 10, local learning rate $\eta = 0.1$. Data points are sampled and distributed to clients non-uniformly.

In Figure 2.4, we run FedAvg in Equation (2.3) on least squares, logistic regression and a nonconvex CNN on the MNIST dataset (LeCun et al. 2010). The experiments are run with pre-determined numbers of communication rounds and different configurations of local epochs K and learning rates (Complete details of our experimental setup are given in Section A of the appendix[2]). We examine the dependencies of FedAvg's final performance regarding K and

[2]All nonconvex experimental results in this chapter are averaged over 4 runs with different random seeds.

Figure 2.4: Optimality gap $\{f(\theta_{\text{FedAvg}}) - f^\star\}$ or training loss $\{f(\theta_{\text{FedAvg}})\}$ of (approximate) fixed-point solutions of FedAvg for different learning rates and local epochs K. Different colored lines are for different numbers of local epochs, and dashed lines for different product values $(\eta K - 1)$. Left: least squares (closed-form solution); Middle: logistic regression (600 communication rounds); Right: nonconvex CNN on the MNIST dataset (200 communication rounds).

. We can conclude that, in general, smaller local epochs and learning rates gives better final solutions (assuming sufficient communication rounds to ensure convergence). Moreover, for least squares and logistic regression, the product $(\eta K - 1)$ we derived in Equation (2.4) almost completely determines the final performance, whether η and K are within a proper range. For the nonconvex CNN (Figure 2.4, right), especially with limited communications, the approximation in Equation (2.4) is too crude to be indicative.

2.4.2 FedProx as backward-backward splitting

The recent FedProx algorithm (Li et al. 2020c) replaces the gradient update in FedAvg with a proximal update:

$$\theta_{t+1} = P_H \, P_f^{\eta} \, \theta_t; \tag{2.5}$$

where as before we may use a minibatch to approximate P_f or select a subset of clients to approximate P_H. Written in this way, it is clear that FedProx is an instantiation of the backward-backward splitting algorithm of Lions (1978) and Passty (1979). In fact, this algorithm traces back to the early works of e.g. Auslender (1969), Cimmino (1938), and Lions and Temam (1966), sometimes under the name of the Barycenter method. It was also rediscovered by Yu (2013) and Yu et al. (2015) in the ML community under a somewhat different motivation. Pathak and Wainwright (2020) pointed out that FedProx does not solve the original problem (2.1). While technically correct, their conclusion did not take many other subtleties into account, which we explain next.

Following Bauschke et al. (2005), we first note that, with a constant step size η, FedProx is actually equivalent as FedAvg but applied to a "regularized" problem:

$$\min_{\theta \in H} f(\theta); \quad \text{where} \quad f(\theta) := \langle 1; M_f(\theta) \rangle; \quad \text{and } M_f(\theta) = M_{f_1}(\theta); \ldots; M_{f_n}(\theta) : \tag{2.6}$$

Interestingly, Dinh et al. (2020) proposed exactly Equation (2.6) for the purpose of personalization, which we now realize is automatically achieved if we apply FedProx to the original formulation (2.1). Indeed, $\nabla M_f(x) = [x - P_f(x)]/\lambda$ hence $G_{\tilde{f}}(x) = x - \text{prox}_{\tilde{f}}(x) = P_f(x) = (P_{f_1}(x), \ldots, P_{f_n}(x))$. This observation turns out to be crucial in understanding FedProx.

Indeed, a significant challenge in FL is client heterogeneity (a.k.a. non-iid distribution of data), where the individual client functions f_i may be very different (due to distinct client-specific data). But, we note that[3] as $\lambda \to 0$, $M_f \to f$ (pointwise or uniformly iff f is Lipschitz) while $M_f \to \min f$ as $\lambda \to \infty$. In other words, a larger λ in (2.6) "smoothens" heterogeneity in the sense that the functions M_{f_i} tend to have similar minimizers (while the minimum values may still differ). We are thus motivated to understand the convergence behavior of FedProx, for small and large λ, corresponding to the original problem (2.1) and the smoothened problem, respectively. In fact, we can even adjust λ dynamically. Below, for simplicity, we assume full gradient (i.e. large batch size) although extensions to stochastic gradient are possible.

Theorem 2. Assuming each client participates indefinitely, the step size λ_t is bounded from below (i.e. $\liminf_t \lambda_t > 0$), the client functions $\{f_i\}$ are convex, and homogeneous in the sense that they have a common minimizer, i.e. $\mathcal{X}^* := \cap_i \operatorname{argmin}_i f_i(x_i) \neq \emptyset$, then the iterates of FedProx converge to a point in $\mathcal{X}^*$, which is a solution of (2.1).

The homogeneity assumption might seem strong since it challenges the necessity of FL in the first place. However, we point out that (a) as a simplified limiting case it does provide insight on when client functions have close minimizers (i.e. homogeneous); (b) by using a large step size λ, FedProx effectively homogenizes the clients and its convergence thus follows;[4] (c) modern deep architectures are typically over-parameterized so that achieving null training loss on each client is not uncommon. However, FL is still relevant even in this case since it selects a model that works for every client and hence provides some regularizing effect; (d) even when the homogeneity assumption fails, P_f^t still converges to a point that is in some sense closest to $\mathcal{X}^*$ (Bauschke et al. 2005).

The next result removes the homogeneity assumption by simply letting step size λ_t diminish:

Theorem 3 ((Lions and Mercier 1979; Passty 1979)). Assuming each client participates in every round, the step size λ_t satisfies $\sum_t \lambda_t = \infty$ and $\sum_t \lambda_t^2 < \infty$, and the functions $\{f_i\}$ are convex, then the averaged iterates $z_t := \frac{\sum_{s=1}^t \lambda_s x_s}{\sum_{s=1}^t \lambda_s}$ of FedProx converge to a solution of the original problem (2.1).

In Section C in the appendix, we give examples to show the tightness of the step size condition in Theorem 3. We emphasize that z_t converges to a solution of the original problem (2.1), not the regularized problem (2.6). The subtlety is that we must let the step size λ_t approach 0 reasonably slowly, a possibility that was not discussed in Pathak and Wainwright (2020) where they always fixed the step size λ_t to a constant λ. Theorem 3 also makes intuitive sense, as $\lambda_t \to 0$ slowly,

[3] These results are classic and well-known, see e.g. Rockafellar and Wets (1998).

[4] The consequence of this must of course be further investigated; see Dinh et al. (2020) and Li et al. (2020c).

we are effectively tracking the solution of the regularized problem (2.6) which itself tends to the original problem (2.1): recall that $M_f \to f$ as $\mu \to 0$.

Even the ergodic averaging step in Theorem 3 can be omitted in some cases:

Theorem 4 ((Passty 1979)). Under the same assumptions as in Theorem 3, if f is strongly convex or the solution set of (2.1) has nonempty interior, then the vanilla iterates x^k of FedProx also converge.

We remark that convergence is in fact linear for the second case. Nevertheless, the above two conditions are perhaps not easy to satisfy or verify in most applications. Thus, in practice, we recommend the ergodic averaging in Theorem 3 since it does not create additional overhead (if implemented incrementally) and in our experiments it does not slow down the algorithm noticeably.

There is in fact a quantitative relationship between the original problem (2.2) and the regularized problem (2.6), even in the absence of convexity:

Theorem 5 ((Yu et al. 2015)). Suppose each f_i is M_i-Lipschitz continuous (and possibly nonconvex), then

$$\forall \mu > 0, \quad |f(x) - f(x^*)| \leq \mu \sum_{i=1}^{N} \frac{M_i^2}{2}.$$

Thus, for a small step size, FedProx, aiming to minimize the regularized function f_μ, is not quantitatively different from FedAvg that aims at minimizing f. This again reveals the fundamental importance of step size in FL. We remark that, following the ideas in Yu et al. (2015), we may remove the need of ergodic averaging for FedProx if the functions f_i are "de nable" (which most, if not all, functions used in practice are). We omit the technical complication here since the potential gain does not appear to be practically signi cant.

In Figure 2.5, we show the effect of step size on the convergence of FedProx, and compare the results with that of FedAvg on both convex and nonconvex models. We run FedProx with both xed and diminishing step sizes. In the experiments with diminishing step sizes, we have set the initial value of μ (i.e. μ_0) to larger values - compared to the constant values - to ensure that μ does not get very small after the rst few rounds. From the convex experiments (Figure 2.5, left and middle), one can see that FedProx with a xed learning rate converges fast (in a few hundred rounds) to a suboptimal solution. In contrast, FedProx with diminishing μ converges slower, but to better quality solutions. It is interesting to note that only when the conditions of Theorem 3 are satis ed (i.e. μ diminishes neither too fast nor too slow), FedProx converges to a solution of the original problem (2.1), e.g. see the results for $\mu_t = 1/\sqrt{t}$ which satis es both conditions in Theorem 3. Surprisingly, for the nonconvex setting (Figure 2.5, right), the best results are achieved with larger learning rates. A similar observation about FedAvg in nonconvex settings was reported in (McMahan et al. 2017b, Figure 5 & 6).[5] Moreover, from the results on both convex and nonconvex models, one can see that ergodic averaging does not affect the convergence rate of FedProx noticeably.

[5] Note that the results of FedAvg and FedProx for $\mu = 100$ and $\mu = 100 \log(t)$ overlap with each other.

Figure 2.5: Effect of step size and ergodic averaging on FedProx. Left: least squares; Middle: logistic regression; Right: CNN on MNIST. The dashed and solid lines with the same color show the results obtained with and without the ergodic averaging step in Theorem 3, respectively. For exponentially decaying η_t, we use period T equal to 500 for both least squares and logistic regression experiments, and 10 for CNN experiment.

2.4.3 FedSplit as Peaceman-Rachford splitting

Pathak and Wainwright (2020) introduced the FedSplit algorithm recently:

$$z^{t+1} = R_H R_f z^t \quad (2.7)$$

which is essentially an instantiation of the Peaceman-Rachford splitting algorithm (Lions and Mercier 1979; Peaceman and Rachford 1955). As shown by Lions and Mercier (1979), FedSplit converges to a solution of (2.1) if f is strictly convex, and the convergence rate is linear if f is strongly convex and smooth (and s is small). Pathak and Wainwright (2020) also studied the convergence behaviour of FedSplit when the reflector R_f is computed approximately. However, we note that convergence behaviour of FedSplit is not known or widely studied for nonconvex problems. In particular, we have the following surprising result:

Theorem 6. If the reflector R_f is a (strict) contraction, then f must be strongly convex.

The converse is true if f is also smooth and s is small (Gabay 1983; Lions and Mercier 1979). Therefore, for non-strongly convex or nonconvex problems, we cannot expect FedSplit to converge linearly (if it converges at all).

2.4.4 FedPi as Douglas-Rachford splitting

A popular alternative to the Peaceman-Rachford splitting is the Douglas-Rachford splitting (Douglas and Rachford 1956; Lions and Mercier 1979), which, to our best knowledge, has not been applied to the FL setting. The resulting update, which we call FedPi, can be written succinctly as:

$$z^{t+1} = \frac{z^t + R_H R_f z^t}{2}; \quad (2.8)$$

i.e. we simply average the current iterate and that of FedSplit evenly. Strictly speaking, the above algorithm is a special case of the Douglas-Rachford splitting and was rediscovered by Spingarn

(1983) under the name of partial inverse (hence our name FedPi). The moderate averaging step in (2.8) makes FedPi much more stable:

Theorem 7 ((Lions and Mercier 1979; Spingarn 1983)). Assuming each client participates in every round, the step size γ_t is constant, and the functions $\{f_i\}$ are convex, then the vanilla iterates z_t of FedPi converge to a solution of the original problem (2.1).

Compared to FedSplit, FedPi imposes less stringent condition on f_i. However, when f_i is indeed strongly convex and smooth, as already noted by Lions and Mercier (1979), FedPi will be slower than FedSplit by a factor close to $\sqrt{2}$ (assuming the constant step size is set appropriately for both). More importantly, it may be easier to analyze FedPi on nonconvex functions, as recently demonstrated in (Rockafellar 2019; Themelis and Patrinos 2020).

Interestingly, FedPi also has close ties to FedProx. Indeed, this is best seen by expanding the concise formula in (2.8) and introducing a "dual variable" on the server side[6]

$$z_{t+1} = P_f(x_t + u_t) \tag{2.9}$$

$$x_{t+1} = P_H(z_{t+1} - u_t) \tag{2.10}$$

$$u_{t+1} = u_t + x_{t+1} - z_{t+1}. \tag{2.11}$$

From the last two updates (2.10) and (2.11) it is clear that u_{t+1} is always in $\mathcal{H}^{\perp}$. Thus, after performing a change of variable $v_t := x_t + u_t$ and exploiting the linearity of P_H, we obtain exactly FedPi:

$$v_{t+1} = u_t + 2x_{t+1} - z_{t+1} = v_t - P_H v_t + 2P_H P_f v_t - P_f v_t = \frac{v_t + R_H R_f v_t}{2}.$$

Comparing (2.5) and (2.9)-(2.10) it is clear that FedProx corresponds to fixing the dual variable u to the constant 0 in FedPi. We remark that step (2.9) is done at the clients' side while steps (2.10) and (2.11) are implemented at the server side. There is no communication overhead either, as the server need only communicate the sum $x_t + u_t$ to the respective clients while the clients need only communicate their $z_{t,i}$ to the server. The dual variable u is kept entirely at the server's expense.

Let us point out another subtle difference that may prove useful in FL. FedAvg and FedProx are inherently "synchronous" algorithms, in the sense that all participating clients start from a common, averaged model at the beginning of each communication round. In contrast, the local models v_t in FedPi may be different from each other, where we "correct" the common, average model x_t with client-specific dual variables u_t. This opens the possibility to personalization by designating the dual variable u in client-specific ways.

Lastly, we remark that in FedProx we need the step size γ_t to diminish in order to converge to a solution of the original problem (2.1) whereas FedPi achieves the same with a constant step size γ, although at the cost of doubling the memory cost at the server side.

[6] The acute readers may have recognized here the alternating direction method of multipliers (ADMM). Indeed, the equivalence of ADMM, Douglas-Rachford and partial inverse (under linear subspaces) has long been known (e.g. Eckstein and Bertsekas 1992; Gabay 1983).

Figure 2.6: Effect of step size and averaging on FedRP. Left: least squares; Right: logistic regression. The dashed and solid lines with the same color show the results obtained with and without the ergodic averaging step in Theorem 3, respectively. For exponentially decaying η_t, we use period T equal to 500 for both least squares and logistic regression.

2.4.5 FedRP as Reflection-Projection splitting

Examining the updates in (2.3), (2.5) and (2.7), we are naturally led to the following further variants (that have not been tried in FL to the best of our knowledge):

$$\text{FedRP}: \quad \begin{aligned} x^{t+1} &= R_H G_f^t x^t && (2.12)\\ x^{t+1} &= R_H P_f^t x^t && (2.13)\\ x^{t+1} &= P_H R_f^t x^t. && (2.14) \end{aligned}$$

Interestingly, the last variant in (2.14), which we call FedRP, has been studied by Bauschke and Kruk (2004) under the assumption that $f = \delta_K$ is an indicator function of an obtuse[7] convex cone K. We prove the following result for FedRP:

Theorem 8. Let each client participate in every round and the functions f_i be convex. If the step size η_t is constant, then any fixed point of FedRP is a solution of the regularized problem (2.6). If the reflector R_f is idempotent (i.e. $R_f R_f = R_f$) and the clients are homogeneous, then the vanilla iterates x^t of FedRP converge.

It follows easily from a result in Bauschke and Kruk (2004) that a convex cone is obtuse iff its reflector is idempotent, and hence Theorem 8 strictly extends their result. We remark that Theorem 8 does not apply to the variants (2.12) and (2.13) since the reflector R_H is not idempotent (recall H is defined in (2.2)). Of course, we can prove (linear) convergence of both variants (2.12) and (2.13) if f is strongly convex. We omit the formal statement and proof since strong convexity does not appear to be easily satisfiable in practice.

In Figure 2.6, we show the effect of step size on the convergence of FedRP on least squares and logistic regression. We run FedRP with both fixed and diminishing step sizes. In

[7] Recall that a convex cone K is obtuse iff its dual cone $K^* := \{y : \langle y, x \rangle \geq 0 \ \forall x \in K\}$ is contained in itself.

the experiments with diminishing step sizes, we set the initial value (α_0) to larger values–compared to the constant values–to ensure that does not get very small after the first few rounds. From the figure, one can see that FedRP with a fixed learning rate converges faster in early stage but to a sub-optimal solution. In contrast, FedRP with diminishing converges slower early, but to better final solutions. It is interesting to note that the convergence behaviour of FedRP is similar to that of FedProx and when the conditions of Theorem 3 are satisfied, FedRP converges to a solution of the original problem (2.1), e.g. see the results for $\alpha_t = 1/(t+1)$ which satisfies both conditions in Theorem 3. Moreover, one can see that ergodic averaging hardly affects the convergence of FedRP.

2.4.6 Effect of data heterogeneity

In Figure 2.7, we compare the performance of different splitting algorithms and how they respond to different degrees of data heterogeneity. We use least squares and a CNN model on MNIST for the convex and noconvex experiments, respectively. The details of the experimental setup are described in Section A.1 in the appendix.

In order to measure heterogeneity, we adopt the data heterogeneity measure of Khaled et al. (2020a), and quantify the amount of heterogeneity in clients' data for the least squares and logistic regression problems by

$$H := \frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(x^\star) - \nabla f(x^\star)\|_2^2,$$

where $x^\star$ is a minimizer of the original problem (2.1). When clients' data is homogeneous, all the local functions f_i have the same minimizer of $x^\star$ and $H = 0$. In general, the more heterogeneous the clients' data is, the larger H becomes.

As observed in Figure 2.7, for the convex setting, as expected, FedSplit, FedPi, and FedAvg with $k = 1$ achieve the smallest optimality gaps. The performance of all the algorithms deteriorates as clients' data become more heterogeneous (see Section A.1 in the appendix). For the nonconvex setting, the best results can be achieved by FedRP and FedAvg with a big k. It is noteworthy that in the nonconvex setting, the performance of FedAvg with $k = 1$ is significantly worse than that with $k = 100$.

2.5 Unification, implementation and acceleration

The operator splitting view of FL not only allows us to compare and understand the many existing FL algorithms but also opens the door for unifying and accelerating them. In this section we first introduce a grand scheme that unifies all aforementioned FL algorithms. Then, we provide two implementation variants that can accommodate different hardware and network constraints. Lastly, we explain how to adapt Anderson acceleration (Anderson 1965) to the unified FL algorithm, with almost no computation or communication overhead.

Figure 2.7: The effect of data heterogeneity on the performance of different splitting methods. The top row shows the results for the least squares, and the bottom row shows the results for nonconvex CNN model. Top-Left: small data heterogeneity with $H \approx 119 \cdot 10^3$; Top-Middle: moderately data heterogeneity with $H \approx 7.61 \cdot 10^6$; Top-Right: large data heterogeneity with $H \approx 190.3 \cdot 10^6$. Bottom-Left: i.i.d. data distribution; Bottom-Middle: non-i.i.d. data distribution with maximum 6 classes per client; Bottom-Right: non-i.i.d. data distribution with maximum 2 classes per client.

Uni cation. We introduce the following grand scheme:

$$z_{t+1} = (1 - \quad_t)u_t + \quad_t P_f{}^{t}(u_t) \tag{2.15}$$

$$\quad_{t+1} = (1 - \quad_t)z_{t+1} + \quad_t P_H(z_{t+1}) \tag{2.16}$$

$$u_{t+1} = (1 - \quad_t)u_t + \quad_t \quad_{t+1}: \tag{2.17}$$

Table 2.1 con rms that the FL algorithms discussed in Section 2.4 are all special cases of this unifying scheme, which not only provides new (adaptive) variants but also clearly reveals the similarities and differences between seemingly different algorithms. It may thus be possible to transfer progress on one algorithm to the others and vice versa. We studied the effect of , and and found that mostly affects the convergence speed: the closer is to 1, the faster the convergence is, while and mostly determine the nal optimality gap: the closer they are both to 2 (as in FedSplit and FedPi), the considerably smaller the nal optimality gap is. However, setting only one of them close to 2 only has a minor effect on optimality gap or convergence speed.

Implementation. Interestingly, we also have two implementation possibilities:

- decentralized: In this variant, all the three updates (2.15)-(2.17) are performed locally at each client. The server only acts as a bridge for synchronization by receiving z_{t+1} from the i-th client and returning the (same) averaged model $P_H(z_{t+1})$ to all clients. In a network where

Table 2.1: A unifying framework (2.15)-(2.17) for FL. Note that (a) FedAvg replaces the proximal update $\mathrm{Prox}_{f_i}^{t}$ with a gradient update $G_{f_i}^{t}$; (b) ? indicates properties that remain to be studied; (c) "sampling" refers to selecting a subset of clients while "stochastic" refers to updating with stochastic gradient.

Algorithm				α_t	β_t	$\sum_t \beta_t = \infty$	nonconvex	sampling	stochastic
FedAvg	1	1	1	Eq. 2.1	Eq. 2.1		X	X	X
FedProx	1	1	1	Eq. 2.6	Eq. 2.1		X	X	X
FedSplit	2	2	1	Eq. 2.1	–		?	?	?
FedPi	2	2	$\frac{1}{2}$	Eq. 2.1	–		X	?	?
FedRP	2	1	1	Eq. 2.6	Eq. 2.1		?	?	?

clients communicate directly with each other and compute $\mathrm{Prox}_{f_i}(z_{t+1})$ locally, we may then dispense the server and become fully decentralized.

- centralized: In this variant, the i-th client only stores $u_{t,i}$ and is responsible for computing and communicating $\mathrm{Prox}_{f_i}^{t}(u_{t,i})$ to the server. In return, the server performs the rest of (2.15)-(2.17) and then sends (different) $u_{t+1,i}$ back to the i-th client.

The two variants are of course mathematically equivalent in their basic form. However, as we see below, they lead to different acceleration techniques and we may prefer one over the other depending on the hardware and network constraints.

Acceleration. Let us abstract the grand scheme (2.15)-(2.17) as the map $u_{t+1} = Tu_t$, where T is nonexpansive if f is convex and $\alpha_t, \beta_t \in [0, 2]$, $\sum_t \beta_t \in [0, 1]$. Following Fu et al. (2020), we may then apply the Anderson type-II acceleration to furthur improve convergence. Let $U = [u_{t-m}, \dots, u_t]$ be given along with $TU = [Tu_{t-m}, \dots, Tu_t]$. We solve the following simple least-squares problem:

$$\gamma = \operatorname*{argmin}_{\gamma^\top 1 = 1} \{\|(TU - U)\gamma\|_2^2\} = \frac{G^{-1}1}{1^\top G^{-1}1}, \quad \text{where} \quad G = (TU - U)^\top (TU - U).$$

(For simplicity we do not require γ to be nonnegative.) Then, we update $u_{t+1} = T(U\gamma)$. Clearly, when $m = 0$, (trivially) $\gamma = 1$ and we reduce to $u_{t+1} = Tu_t$. With a larger memory size, we may significantly improve convergence. Importantly, all heavy lifting (computation and storage) is done at the server side and we do not increase communication at all. We note that the same acceleration can also be applied on each client for computing Prox_{f_i}, if the clients can afford the memory cost.

We have seen how different variants (e.g. FedAvg, FedProx, FedSplit, FedPi, FedRP) compare to each other in §2.4. We performed further experiments to illustrate their behaviour under Anderson acceleration. (See Section A for details on our experimental setup.) As can be observed in Figure 2.8, Anderson acceleration helps algorithms converge considerably faster,

Figure 2.8: Effect of Anderson acceleration. Left: least squares with $m = 2$; Middle: logistic regression $m = 2$; Right: nonconvex CNN with $m = 10$. Dashed lines are the accelerated results.

all without incurring any overhead. For the convex models (least squares and logistic regression), our implementation of Anderson-acceleration speeds up all of the algorithms, especially FedAvg, FedProx and FedRP. However, for the nonconvex CNN model, it is beneficial only for FedAvg, FedProx and FedPi, while applying it to FedRP and FedSplit makes them unstable. It is noteworthy that we already know FedProx and FedPi are more stable than FedRP and FedSplit, respectively, and hence it makes intuitive sense that acceleration improves the two more stable algorithms. Another important point we wish to point out is that our acceleration method does not affect the quality of the algorithms' final solutions, but rather it just accelerates their convergence.

2.6 Summary

We have connected FL with the established theory of operator splitting from optimization, revealed new insights on existing algorithms and suggested new algorithmic variants and analysis. Our unified view makes it easy to understand, compare, implement and accelerate different algorithms in a streamlined and standardized fashion. Our experiments demonstrate some interesting differences in the convex and nonconvex settings, and in the early and late communication rounds. While our initial experiments confirmed the potential of Anderson-acceleration for FL, further work is required to formalize its effect in theory (such as (Zhang et al. 2020)) and to understand its relation with other momentum methods in FL. Studying the effect of stochasticity and extending our analysis to nonconvex functions is another direction for future studies.

Chapter 3

Noise-Aware Algorithm for Heterogeneous Differentially Private Federated Learning

3.1 Preface

High utility and rigorous data privacy are of the main goals of a federated learning (FL) system, which learns a model from the data distributed among some clients. The latter has been tried to achieve by using differential privacy in FL (DPFL). There is often heterogeneity in clients' privacy requirements, and most existing DPFL works either assume uniform privacy requirements for clients or are not applicable when server is not fully trusted (our setting). Furthermore, there is often heterogeneity in batch and/or dataset sizes of clients, which as shown, results in extra variation in the DP noise level across clients' model updates. With these sources of heterogeneity, straightforward aggregation strategies, e.g., assigning clients' aggregation weights proportional to their privacy parameters, will lead to a lower utility. We propose Robust-HDP, which efficiently estimates the true noise level in clients' model update and reduces the noise-level in the aggregated model updates considerably. Robust-HDP improves utility and convergence speed, while being safe to the clients that may maliciously send falsified privacy parameters to the server. Extensive experimental results on multiple datasets and our theoretical analysis confirm the effectiveness of the proposed Robust-HDP.

3.2 Introduction

In the presence of sensitive information in the train data, FL algorithms must be able to provide rigorous data privacy guarantees against a potentially curious server or any third party (Geiping et al. 2020; Hitaj et al. 2017; Rigaki and García 2020; Wang et al. 2019; Zhu et al. 2019). Differential Privacy (Dwork 2011; Dwork et al. 2006a,b; Dwork and Roth 2014) has been used in DPFL systems to achieve such formal privacy guarantees. When there is a trusted server in the system, DP is provided by the server adding controlled noise to the aggregation of clients' updates

(Geyer et al. 2017; McMahan et al. 2018). When there is no trusted server, local perturbations, where each client randomizes its updates locally before sending them to the server (LDP), is also a solution (Zhao et al. 2021), while being limited in the sense that achieving privacy while preserving model utility is challenging, due to clients' independent local noise additions. Some solutions have been proposed for this, e.g., using a trusted shuffler system (Girgis et al. 2021; Liu et al. 2021b), which may be difficult to establish if the server itself is not trusted.

Clients often have heterogeneous privacy preferences coming from their varying privacy policies. Furthermore, dataset size usually varies a lot across clients. Additionally, depending on their computational budgets, some clients may use relatively smaller batch sizes locally for running DPSGD algorithm (Abadi et al. 2016). As we will show, a small privacy parameter (ε) and/or a small batch size lead to a fast increment of the noise level in a client's model update. Existing heterogeneous DPFL works mostly either depend on a trusted server (Chathoth et al. 2022; Zhou et al. 2022), or suffer from suboptimal and vulnerable aggregation strategies on an untrusted server (based on clients' privacy parameters (Liu et al. 2021a)). We consider a heterogeneous DPFL systems with an untrusted server and propose an efficient aggregation strategy for the server, which is aware of the noise level in each client's model update: we propose to employ Robust PCA (RPCA) algorithm (Candès et al. 2009) by the curious server to estimate the amount of noise in clients' model updates, which we show depends strongly on multiple factors (e.g., their privacy parameter and their batch size ratio), and assign their aggregation weights accordingly. The use of this efficient strategy on the server, which is independent of clients sending any privacy parameters to the server or not, improves model utility and convergence speed while being robust to potential falsifying clients. The highlights of our contributions are the followings:

- We show the effect of privacy parameter and batch/dataset size on the noise level in clients' updates (§3.5.2).
- We propose Robust-HDP, a robust algorithm with a noise-aware aggregation strategy for heterogeneous DPFL (§3.5.4).
- As the first work assuming heterogeneous dataset sizes, heterogeneous batch sizes, non-uniform and varying aggregation weights and partial participation of clients simultaneously, we prove convergence of our proposed algorithm under mild assumptions (§3.5.8).
- In various privacy heterogeneity scenarios across clients, we show that Robust-HDP improves utility and convergence speed compared to existing aggregation strategies (§3.6).

3.3 Related work

As explained in Chapter 1, there are various definitions of DP. In this chapter, we adopt the approximate (ε, δ) DP definition (Definition 2), and use DPSGD algorithm for achieving it. Some prior works (Gur-Ari et al. 2018) found that stochastic gradients stay in a low-dimensional space during training with Stochastic Gradient Descent (SGD). Inspired by this, (Zhou et al.

Figure 3.1: Security model in heterogeneous DPFL, where client i has local train data D_i and record-level privacy parameter $(\varepsilon_i, \delta_i)$, and does not trust any external parties.

2021) proposed projection-based variant of the DPSGD (Abadi et al. 2016) algorithm (projected DPSGD), which improves utility by removing the unnecessary noise from noisy batch gradients by projecting them on a linear subspace obtained from a public dataset. Personalized DP (PDP), which specifies a separate privacy parameter for each data sample in a dataset, was previously used for centralized settings by Alaggan et al. 2017; Huang et al. 2020; Jorgensen et al. 2015; Kotsogiannis et al. 2020; Yu et al. 2023, followed by some recent works by Boenisch et al. 2023; Heo et al. 2023. Another similar work by Niu et al. 2020 proposed "Utility Aware Exponential Mechanism" (UPEM) to pursue higher utility while achieving PDP. In the same direction of improving utility, Shi et al. (2021) proposed "Selective DP" for improving utility by leveraging the fact that private information in natural language is sparse.

Heterogeneous DPFL. Assuming the existence of a trusted server, Chathoth et al. (2022) proposed cohort-level privacy with privacy and data heterogeneity across cohorts using DP (Definition 1). Also, a work by Zhou et al. 2022 adapted the non-uniform sampling idea of Jorgensen et al. 2015 to the FL settings with a trusted server to get client-level DPFL (i.e. two FL systems differing by only one client's whole data) against membership inference attacks (Rigaki and García 2020; Wang et al. 2019). In contrast, we consider untrusted but curious servers.

In the sense of Definition 2 of (ε, δ)-DP, the output of a randomized algorithm $\mathcal{M}$ is all the information that the untrusted server observes. We consider heterogeneous DPFL (Figure 3.1), where each client i has its own privacy parameters $(\varepsilon_i, \delta_i)$, and sends data-dependent computation results $\mathcal{M}(D_i)$ (model updates) to the server. Also, in the context of Definition 2, the notion of neighboring datasets that we consider in this chapter is record-level: a pair of federated datasets $d = \{D_1, \ldots, D_n\}$ and $d' = \{D'_1, \ldots, D'_n\}$ differing in replacement of one data point in one client's dataset (e.g., D_i differs with D'_i in one data point, and $D_j = D'_j$ $(j \neq i)$).

Liu et al. (2021a) adapted a projection-based approach, similar to that of projected DPSGD (Zhou et al. 2021), to the heterogeneous DPFL setting to propose PPFA and improve utility. Although assuming an untrusted server, their proposed algorithm relies on the assumption that the server knows the clients' "true" privacy parameters $\{(\varepsilon_i, \delta_i)\}$ and uses them to cluster clients to "public" (those with larger privacy parameters) and "private". As such, as we show, PPFA is extremely vulnerable to when clients share a falsified value of their privacy parameters (often

Table 3.1: Features of different heterogeneous DP-FL algorithms. : needed at server, X: not needed.

algorithm	aggregation strategy	$\{\sigma_i\}_{i=1}^n$	clients clustering	PCA on clients updates
WeiAvg (Liu et al. 2021a, Alg. 5)	w_i / σ_i			X
PFA (Liu et al. 2021a)	w_i / σ_i			
DPFedAvg (Noble et al. 2021)	w_i / N_i	X	X	X
minimum (Liu et al. 2021a)	w_i / N_i		X	X
Robust-HDP (Alg. 1)	$w_i / \frac{1}{\sigma_i^2}$	X	X	X

larger than their true values) with the server. Also, the authors used aggregation strategy w_i/σ_i on server for PFA and another algorithm called WeiAvg (see Table 3.1). As we will show, even if the server knows clients' true privacy parameters $\{\sigma_i\}_{i=1}^n$, this information is not a perfect indication of the "true noise level" in their model updates, especially with heterogeneous privacy parameters and batch/dataset sizes.

The current state of the art in heterogeneous DP-FL calls for a robust algorithm that takes all the mentioned potential sources of heterogeneity across clients into account and achieves high utility and data privacy simultaneously.

3.4 Notations

We consider an FL setting with n clients. Let $x \in \mathcal{X} \subseteq \mathbb{R}^d$ and $y \in \mathcal{Y} = \{1, \dots, C\}$ denote an input data point and its target label. Client i holds dataset $\mathcal{D}_i = \{(x_{ij}, y_{ij})\}_{j=1}^{N_i}$ with N_i samples from distribution $P_i(x, y)$. Let $h : \mathcal{X} \to \mathbb{R}^C$ be the predictor function, which is parameterized by $\theta \in \mathbb{R}^p$ (p is the number of model parameters) shared among all clients. Also, let $\ell : \mathbb{R}^C \times \mathcal{Y} \to \mathbb{R}_+$ be the loss function used (cross entropy loss). Following McMahan et al. 2017b, many existing FL algorithms fall into the natural formulation that minimizes the (arithmetic) average loss $f(\theta) := \sum_{i=1}^n \lambda_i f_i(\theta)$, where $f_i(\theta) = \frac{1}{N_i} \sum_{(x,y) \in \mathcal{D}_i} [\ell(h(x; \theta), y)]$, with minimum value f_i^*. The weights $\lambda = (\lambda_1, \dots, \lambda_n)$ are nonnegative and sum to 1. At gradient update t, client i uses a data batch B_i^t with size $b_i = |B_i^t|$. Let $q_i = \frac{b_i}{N_i}$ be the batch size ratio of client i. There are E global communication rounds indexed by t, and in each of them, client i runs K_i local epochs. We use boldface letters to denote vectors. Table 3.2 summarizes the used notations.

3.5 Robust-HDP algorithm for heterogeneous DP-FL

In this section, we will devise a new heterogeneous DP-FL algorithm, and we first explain the intuitions behind it. At the t-th gradient update step on a current model, client i computes the

Table 3.2: Notations used in Chapter 3

n	number of clients, which are indexed by i
x_{ij}, y_{ij}	j-th data point of client i and its label
D_i, N_i	local train set of client i and its size
B_i^t	the train data batch used by client i at the t-th gradient update
b_i	batch size of client i: $\lvert B_i^t \rvert = b_i$
q_i	batch size ratio of client i: $q_i = \frac{b_i}{N_i}$
ε_i, δ_i	client i's desired DP privacy parameters
E	total number of communication rounds in the system, indexed by e
S^e	set of participating clients in round e
θ^e	global model parameter, which has size p, at the beginning of global round e
K_i	number of local train epochs performed by client i during each global round e
E_i	number of batch gradient updates of client i during each global round e: $E_i = K_i \cdot \lceil \frac{N_i}{b_i} \rceil$
h	predictor function, e.g., CNN model, with parameter θ
ℓ	used loss function, e.g., cross entropy loss
$\sigma_{i,g}^2$	variance of the noisy stochastic batch gradient $\tilde{g}_i(\theta^t)$ of client i
σ_i^2	conditional variance of the noisy model update $\tilde{\Delta}_i^e$ of client i: $\mathrm{Var}(\tilde{\Delta}_i^e \mid \theta^e)$

following noisy batch gradient by using DPSGD (Abadi et al. 2016):

$$\tilde{g}_i(\theta) = \frac{1}{b_i} \sum_{j \in B_i^t} g_{ij}(\theta) + \mathcal{N}(0, \sigma_{i,\mathrm{DP}}^2 I_p), \tag{3.1}$$

where $g_{ij}(\theta) = \mathrm{clip}_c(\nabla \ell(h(x_{ij}; \theta), y_{ij}))$, and c is a clipping threshold. For a given vector v, $\mathrm{clip}_c(v) = \min\{\|v\|, c\} \frac{v}{\|v\|}$. Also, $\sigma_{i,\mathrm{DP}} = c \cdot z(\varepsilon_i, \delta_i, q_i, K_i, E)$. knowing E (global communication rounds), client i can compute $z(\varepsilon_i, \delta_i, q_i, K_i, E)$ locally, which is the noise scale that it should use locally for DPSGD in order to achieve $(\varepsilon_i, \delta_i)$ DP at the end of E global rounds. This can be done by client i using a privacy accountant, e.g., the moments accountant (Abadi et al. 2016). Therefore, depending on its privacy preference $(\varepsilon_i, \delta_i)$, each client i computes its required noise scale $z(\varepsilon_i, \delta_i, q_i, K_i, E)$, runs DPSGD locally for K_i epochs and sends the resulting noisy model updates to the server at the end of each round. Now, an important question is that what is an efficient aggregation strategy for the server to aggregate the clients' noisy model updates? Intuitively, the server has to pay more attention to the less noisy updates. The challenge is that the server knows neither the noise added by each client nor its amount. To answer the above question, we first analyze the behavior of the noise level in clients' batch gradients in Section 3.5.1, which is used in Section 3.5.2, for a similar analysis of clients' uploaded model updates. The result of this analysis is an idea we propose for the server to estimate the noise amount in each client's model update, which leads to an efficient aggregation strategy in Section 3.5.4.

3.5.1 Noise level in clients' DP batch gradients

We consider two cases, which are easier to analyze. Our analysis gives us an understanding of the parameters affecting the noise level in clients' DP batch gradients. Depending on the value of clipping threshold c at the t-th gradient update step, we consider two general indicative cases:

1. $c < \|g_{ij}(\theta)\|, \forall j \in B_i^t$: In this case, all sample gradients in the batch are clipped and from Equation (3.1), we get to the following expectation for the clipped batch gradient:

$$E[\tilde{g}_i(\theta)] = \frac{1}{b_i}\sum_{j \in B_i^t} E[\bar{g}_{ij}(\theta)] = \frac{1}{b_i}\sum_{j \in B_i^t} G_i(\theta) = G_i(\theta); \tag{3.2}$$

where the expectation is with respect to the stochasticity of gradients and we have assumed that $E[\bar{g}_{ij}(\theta)]$ is the same for all j and is denoted by $G_i(\theta)$. For an arbitrary random vector $v = (v_1, \dots, v_p)^\top \in \mathbb{R}^{p \times 1}$, we define $\mathrm{Var}(v) := \sum_{j=1}^{p} E[(v_j - E[v_j])^2]$, i.e. variance of v is the sum of the variances of its elements. Then, the variance of the noisy stochastic gradient in Equation (3.1), which is also random, can be computed as (see Appendix B for derivation):

$$\sigma^2_{i;\tilde{g}} := \mathrm{Var}(\tilde{g}_i(\theta)) = \frac{c^2 - \|G_i(\theta)\|^2}{b_i} + \frac{pc^2 z^2(\sigma_i, \mu_i; q_i; K_i; E)}{b_i^2} - \frac{pc^2 z^2(\sigma_i, \mu_i; q_i; K_i; E)}{b_i^2}; \tag{3.3}$$

where, the estimation is valid because $b_i \gg 1$. For instance, $p \approx 2 \times 10^7$ for ResNet-34 for CIFAR100, and $c = 3$, which results in $pz^2(\sigma_i, \mu_i; q_i; K_i; E)/b_i \gg 1$.

2. $c \geq \|g_{ij}(\theta)\|, \forall j \in B_i^t$: In this case, we have a noisy version of the batch gradient $\tilde{g}_i(\theta) = \frac{1}{b_i}\sum_{j \in B_i^t} \bar{g}_{ij}(\theta)$, which is unbiased with variance bounded by $\sigma^2_{i;g}$ (see Assumption 1). Hence:

$$E[\tilde{g}_i(\theta)] = E[g_i(\theta)] = \nabla f_i(\theta);$$

$$\sigma^2_{i;\tilde{g}} = \mathrm{Var}(\tilde{g}_i(\theta)) = \mathrm{Var}(g_i(\theta)) + \frac{p\,\sigma^2_{i;\mathrm{DP}}}{b_i^2} \leq \sigma^2_{i;g} + \frac{pc^2 z^2(\sigma_i, \mu_i; q_i; K_i; E)}{b_i^2}. \tag{3.4}$$

z is a sub-linearly increasing function of q_i and b_i (see Theorem 9 and Figure 3.2). It is also clear that z is a decreasing function of σ_i and μ_i. Hence, $\sigma^2_{i;\tilde{g}}$ is a decreasing function of b_i (batch size), N_i (dataset size) and ϵ, and also an increasing function of q_i (batch size ratio).

3.5.2 Noise level in clients' DP model updates

Having found the parameters affecting $\sigma^2_{i;\tilde{g}}$, we now investigate the parameters affecting the noise level in clients' model updates. During each communication round, each participating client performs $E_i = K_i \cdot \max\{1, \frac{N_i}{b_i}\} = K_i \cdot \max\{1, \frac{1}{q_i}\}$ batch gradient updates locally with step size η_i:

$$\tilde{\theta}_i = \theta_{i;E_i} - \theta_{i;0},$$

Figure 3.2: Plot of z v.s. q obtained from moments accountant (Abadi et al. 2016) in a centralized setting with $E = 200$. As observed, z increases sub-linearly with q (or equivalently with b).

$$\theta^e_{i;k} = \theta^e_{i;k-1} - \eta g_i(\theta^e_{i;k-1}); \; k = 1; : : : ; E_i;$$

where $\theta^e_{i;0} = \theta^e$. In each update, it adds a Gaussian noise from $\mathcal{N}(0; \frac{c^2 z^2(\epsilon_i; \delta_i; q_i; K_i; E)}{b_i^2} I_p)$ to its batch gradients independently, according to Equation (3.1). Hence:

$$\sigma_i^2 := \mathrm{Var}(\tilde{\theta}^e_i | \theta^e) = K_i \left(\frac{1}{q_i}\right)^2 \sigma^2_{i;g}; \tag{3.5}$$

where $\sigma^2_{i;g}$ was computed in Equation (3.3) and Equation (3.4) for two general indicative cases. This means that σ_i^2 heavily depends on b_i (e.g., when clipping is effective, b_i appears with power 3 in denominator. Recall $q_i = \frac{N_i}{b_i}$). Hence, σ_i^2 decreases quickly when b_i increases. Similarly, σ_i^2 is a non-linearly decreasing function of q_i (see Figure 3.3, left). However, note that N_i and q_i appear twice in Equation (3.5) with opposing effects. This makes the variation of σ_i^2 with N_i and q_i small (explained in details in Section F.3 in the Appendix). An important message of these important understandings is that q_i is not the only parameter of client i that determines σ_i^2.

3.5.3 Optimum aggregation strategy

Assuming the set of participating clients S^e in round e, we have to solve the following problem to minimize the total noise after the aggregation at the end of this round:

$$\min_{w_i \geq 0} \mathrm{Var}\left(\sum_{i \in S^e} w_i \tilde{\theta}^e_i \,|\, \theta^e\right) = \sum_{i \in S^e} w_i^2 \sigma_i^2;$$
$$\text{s.t.} \sum_{i \in S^e} w_i = 1; \tag{3.6}$$

Figure 3.3: Left: 3D plot of noise variance σ_i^2 (Equation (3.5)) with $K_i = 1, N_i = 2400, \epsilon_i = 0.01, c = 3, p = 28939$) based on δ_i and the privacy budget ϵ_i. Right: the noise variances $\{\sigma_i^2\}_{i=1}^n$ in a DPFL system with $n = 20$ clients, where $\{(\epsilon_i, \delta_i)\}_{i=1}^n$ are randomly selected for each client. It clearly shows an approximately sparse pattern (14 of the clients have much smaller noise variance than the other 6). Each bar plot corresponds to a point in the left figure.

which has a unique solution $w_i \propto \frac{1}{\sigma_i^2}$. Hence, the optimum aggregation strategy weights clients directly based on $\{\sigma_i^2\}_{i=1}^n$, which as shown, not only depends on $\{\epsilon_i\}_{i=1}^n$ non-linearly, but it also depends on $\{\delta_i\}_{i=1}^n$ and $\{N_i\}_{i=1}^n$. This point makes the aggregation strategy $w_i \propto \epsilon_i$ of PFA and WeiAvg algorithms (Liu et al. 2021a) suboptimal, let alone its vulnerability to a client sharing a falsified $\epsilon_i' > \epsilon_i$ with the server to either attack the system, to get a larger aggregation weight, or to get a larger payment from a server which incentivizes participation by payment to clients (Donahue and Kleinberg 2021; Fallah et al. 2023; Kang et al. 2023; Karimireddy et al. 2022) (as a larger ϵ_i usually means a more exploitable data from client i). The same vulnerability discussion applies to the clustering of clients based on their shared privacy parameter (used in PFA). Having these shortcomings of the existing algorithms as a motivation, how can we implement the optimum aggregation strategy when the untrusted server does not have any idea of the clients' noise addition mechanisms and $\{\sigma_i^2\}_{i=1}^n$? We next propose our idea for estimating $\{\sigma_i^2\}_{i=1}^n$ and $\{w_i\}_{i=1}^n$.

3.5.4 Robust-HDP algorithm

Assuming n clients and full participation for simplicity, at the end of each global round, the server gets the matrix $M := [\tilde{e}_1 | \cdots | \tilde{e}_n]$. Assuming an i.i.d or moderately heterogeneous data split, and based on the findings in (Gur-Ari et al. 2018; Zhou et al. 2021), we would expect M to have a low rank if there was no DP stochastic noise in $\{\tilde{e}_i\}_{i=1}^n$. So we can think about writing M as the summation of an underlying low-rank matrix L and a noise matrix S:

$$M = [\tilde{e}_1 | \cdots | \tilde{e}_n] = L + S.$$

If the matrix S is sparse, not only can such a decomposition problem be solved using PCA, it can be solved by a very convenient convex optimization program called Principal Component

Algorithm 1: Robust-HDP

Input: Initial parameter θ^0, batch sizes $\{b_1, \dots, b_n\}$, dataset sizes $\{N_1, \dots, N_n\}$, noise scales $\{z_1, \dots, z_n\}$, gradient norm bound c, local epochs $\{K_1, \dots, K_n\}$, global rounds E, number of model parameters p, privacy accountant PA.

Output: θ^E; $\{\theta_1^E, \dots, \theta_n^E\}$

Initialize θ^0 randomly

for $e \in [E]$ **do**

  sample a set of clients $S^e \subseteq \{1, \dots, n\}$

  for each client $i \in S^e$ in parallel **do**

    $\tilde{\theta}_i^e \leftarrow \text{DPSGD}(\theta^e, b_i, N_i, K_i, z_i, c)$

    $\epsilon_i^e \leftarrow \text{PA}(\frac{b_i}{N_i}, z_i, K_i, e)$

  $M = [\tilde{\theta}_1^e | \dots | \tilde{\theta}_{|S^e|}^e] \in \mathbb{R}^{p \times |S^e|}$

  $L, S = \text{RPCA}(M)$

  for $i \in S^e$ **do**

    $w_i^e \leftarrow \frac{1/\|S_{:,i}\|_2^2}{\sum_{j \in S^e} 1/\|S_{:,j}\|_2^2}$

  $\theta^{e+1} \leftarrow \theta^e + \sum_{i \in S^e} w_i^e \tilde{\theta}_i^e$

Pursuit (Algorithm 6 in the appendix) without imposing much computational overhead to the server (Candès et al. 2009). Surprisingly, the entries in S can have arbitrarily large magnitudes. Theoretically, this is guaranteed to work even if $\text{rank}(L) \in O(n/(\log p)^2)$, i.e. the rank of L grows almost linearly in n (see Theorem 1.1 in (Candès et al. 2009)). Hence, we expect to be able to do such a decomposition as long as we have a moderately heterogeneous data distribution across a large enough number of clients (also, see Section G in the Appendix for detailed discussion on data heterogeneity and further experiments). Hence, L will be a low rank matrix, estimating the "true" values of clients' updates and S will capture the noises in clients model updates $\{\tilde{\theta}_i^e\}_{i=1}^n$ induced by two sources: DP additive Gaussian noise and batch gradients' stochastic noise. Therefore, we can use $\hat{\sigma}_i^2 := \|S_{:,i}\|_2^2$ ($S_{:,i}$ is the i-th column of S, corresponding to client i) as an estimate of σ_i^2 (Equation (3.5)). Indeed, we observed such approximately sparse pattern in S in Figure 3.3 (right), where each barplot corresponds to the ℓ_2-norm of one column of S. Thus, according to problem 3.6, we assign the aggregation weights as $w_i^e = \frac{1/\hat{\sigma}_i^2}{\sum_{j \in S^e} 1/\hat{\sigma}_j^2}$, where $\hat{\sigma}_i^2 = \|S_{:,i}\|^2$ (see Algorithm 1). Interestingly, this estimation is independent of clients' sharp parameter values, which makes our Robust-HDP optimal, robust and vastly applicable.

3.5.5 Scalability of Robust-HDP with the number of model parameters p

The computation time (precision) of RPCA algorithm increases (decreases) when the number of model parameters p grows. As such, in order to make the Robust-HDP scalable for large models,

we perform the noise estimation of Robust-HDP on sub-matrices of M with fewer rows:

$$\begin{aligned} M_1 &= M[0 : p^0 - 1, :] = L_1 + S_1 \\ M_2 &= M[p^0 : 2p^0 - 1, :] = L_2 + S_2 \\ &\vdots \\ M_Q &= M[p - p^0 : p - 1, :] = L_Q + S_Q, \end{aligned}$$

where $Q = \left\lfloor \frac{p}{p^0} \right\rfloor$. Then, we get a set of noise variance estimates $\{\hat{\sigma}_i^2\}_{i=1}^n$ from each $S_j, j \in \{1, \dots, Q\}$. Finally, we use the sets' element-wise average for weight assignment. For instance, for CIFAR10 and CIFAR100, we perform RPCA on sub-matrcies of M with $p^0 = 200,000$ rows, and average their noise variance estimates. Our experimental results show that this approach, even with $Q = 1$ (i.e. using just M_1), still results in assigning aggregation weights close to the optimum weights $\{w_i\}$. This idea makes Robust-HDP scalable to large models with large p.

3.5.6 Reliability of Robust-HDP

In order for Robust-HDP to assign the optimum aggregation weights $\{w_i\}$, it suffices for it to estimate the set $\{\sigma_i^2\}$ up to a multiplicative factor. Assuming participants S^e in round e, let $s_{i,j}$ in matrix S represent the true value of noise in the ith element of $\tilde{\Delta}_j^e$ ($j \in S^e$). Then, assume that S^0 is the matrix computed by Robust-HDP at the server with bounded elements $s_{i,j}^0 \in U$, where $E[s_{i,j}^0] = r s_{i,j}$, for some constant $r > 0$, and $E[|s_{i,j}^0 - r s_{i,j}|^2] \leq \sigma_j^2$ (i.e. on average, Robust-HDP is able to estimate the true noise values $s_{i,j}$ up to a multiplicative factor by using RPCA). Then, from Hoeffding's inequality, we have:

$$\Pr\left(|\hat{\sigma}_j^2 - (r^2 \sigma_j^2 + \sigma_j^2)| > \epsilon\right) \leq 2e^{-\frac{2p\epsilon^2}{U^2}},$$

meaning that estimating the entries of S up to a multiplicative factor with a small variance is enough for Robust-HDP to estimate $\{\sigma_i^2\}$ up to a multiplicative factor r^2 with high probability. This probability increases with the number of model parameters p exponentially: the p noise elements of $S_{:,i}$ are i.i.d, and larger p means having more samples from the same distribution to estimate its variance (see also Theorem 1.1 in (Candès et al. 2009)). Also, $\frac{1}{r^2 \sigma_j^2 + \sigma_j^2} \leq \frac{1}{\hat{\sigma}_j^2} \leq \frac{1}{r^2 \sigma_j^2}$. Hence, as $\sigma_j^2 \ll 1$ (it is the noise variance in the model update $\tilde{\Delta}_j^e$. See the values in Figure 3.3, right), a small deviation σ_j^2 from $r^2 \sigma_j^2$ still results in aggregation weights close to $\{w_i\}$.

3.5.7 Privacy analysis of Robust-HDP

We have the following theorem about DP guarantees of our proposed Robust-HDP algorithm.

Theorem 9. For each client i, there exist constants c_1 and c_2 such that given its number of steps $E \geq E_i$, for any $\epsilon < c_1 q^2 E \geq E_i$, the output model of Robust-HDP provides (ϵ_i, δ_i) DP to client i

for any $_i > 0$ if $z_i > c_2 \frac{q_i \sqrt{E\, E_i \log \frac{1}{i}}}{i}$, where z_i is the noise scale used by the client for DPSGD. Also, assuming clients' datasets are disjoint, the algorithm also satisfies $(_{max}; _{max})$-DP, where $(_{max}; _{max}) = \max(f _i g_{i=1}^n); \max(f _i g_{i=1}^n)$.

Therefore, the model returned by Robust-HDP guarantees $(_i; _i)$-DP for client i, satisfying heterogeneous DPFL.

3.5.8 The optimization side of Robust-HDP

We assume that $f( ) = \sum_{i \in [n]} _i f_i( )$, where $_i = \frac{N_i}{\sum_i N_i}$, has minimum value f and minimizer . We make the following assumptions about the clients' loss functions and then, we will provide the convergence analysis of the Robust-HDP algorithm.

Assumption 1 (Lipschitz continuity, -smoothness and bounded gradient variance). The $f f_i g_{i=1}^n$ are L_0-Lipschitz continuous and -smooth: $8 ; {}^0 2 R^p; i : kf_i( ) f_i({}^0)k L_0 k {}^0k$ and $kr f_i( ) r f_i({}^0)k k {}^0k$. Also, the stochastic gradient $g_i( )$ is an unbiased estimate of $r f_i( )$ with bounded variance: $8 2 R^p : E_{B_i^t}[g_i( )] = r f_i( ); E_{B_i^t} kg_i( ) r f_i( )k^2 {}^2_{i;g}$. We also assume that for every $i; j 2 [n]; f_i f_j$ is -Lipschitz continuous and $kr f_i( ) r f_j( )k :$

Assumption 2 (bounded sample gradients). There exists a threshold C such that for all $i; j$:

$$kg_{ij}( )k_2 := kr `(h(x_{ij};); y_{ij})k_2 C$$

Note that this condition always holds if $`$ is Lipschitz continuous or if is bounded.

Theorem 10 (Robust-HDP). Assume that Assumptions 1 and 2 hold, and for every learning rate $_l$ satisfies: $_l \frac{1}{6 E_i}$ and $_l \frac{1}{12 \sqrt{\frac{(1+\sum_{i=1}^n E_i)}{\sum_{i=1}^n E_i^4}}}$. Then, we have:

$$\min_{0 e E 1} E[kr f({}^e)k^2] \frac{12}{11 E_l^{min} 7} \frac{f({}^0) f}{E _l} + + {}_p ;$$

where $E_l^{min} = \min_i E_i$, i.e. the minimum number of local DPSGD steps per round across clients. Also, $_p$ and are two constants controlling the quality of the nal model parameter returned by Robust-HDP explained below. Also, assuming a large E, $\min_{0 e E 1} E[kr f({}^e)k^2]$ happens close to the convergence time, i.e. at the last communication rounds.

Discussion. Our convergence guarantees are quite general: we allow for partial participation, heterogeneous number of local steps fE_ig, non-uniform batch sizes fb_ig, varying and nonuniform aggregation weights fw_i^eg. When $f f_i g$ are convex, Robsut-HDP solution converges to a neighborhood of the optimal solution. The term decreases when data split across clients is

more i.i.d, and variance of mini-batch gradients $\{\sigma_i^2\}$ decrease (e.g., when clients are less privacy sensitive). Similarly, Γ_p decreases when clients participate more often, and the set of local steps $\{E_i\}$ is more uniform (e.g., clients have similar dataset sizes and batch sizes). Also, smaller local steps $\{E_i\}$, which can be achieved by having smaller local epochs $\{K_i\}$ and larger batch sizes $\{b_i\}$, result in reduction of both Γ_p and Γ, and higher quality solutions (Malekmohammadi et al. 2023). Compared to the results in previous DP-FL works, we have the most general results with more realistic assumptions. For instance, (Liu et al. 2021a) (WA-Avg and DP-FA) assumes uniform number of local SGD updates for all clients, or (Noble et al. 2021) (DP-FedAvg) assumes uniform aggregation weights and uniform number of local updates. These assumptions may not be practical in real systems. In a more general view, when we have no DP guarantees, we recover the results for the simple FedAvg algorithm (Zhang et al. 2023). When we additionally have $\zeta = 0$ (i.e. FedAvg on i.i.d data), our results are the same as those of SGD (Ghadimi and Lan 2013):

$$\min_{0 \le e \le E-1} \mathbb{E}[\|\nabla f(x^e)\|^2] \le \frac{12}{11 E \eta_{\min}} \frac{f(x^0) - f^*}{E \eta} + O(\eta);$$

which shows convergence rate $\frac{1}{\sqrt{E}}$ with $\eta = O(\frac{1}{\sqrt{E}})$.

3.6 Experiments

In this section, we evaluate the proposed Fed-Robust-HDP algorithm experimentally. See Appendix A for details of experimental setup and hyperparameter tuning used for evaluation of algorithms.

3.6.1 Experimental setup

Datasets, models and baseline algorithms: We evaluate our proposed method on four benchamrk datasets: MNIST (LeCun et al. 2010), FMNIST (Xiao et al. 2017) and CIFAR10/100 (Krizhevsky 2009) using CNN-based models. We compare four baseline algorithms with 1. WA-Avg (Liu et al. 2021a) 2. DP-FA (Liu et al. 2021a) 3. DP-FedAvg (Noble et al. 2021) 4. non-private minimum.

Privacy preference and batch size heterogeneity: We consider an FL setting with 20 clients as explained in Appendix A.1, which results in homogeneous $\{N_i\}_{i=1}^n$. We also assume full participation and one local epoch for each client ($K_i = 1$ for all i). Batch size heterogeneity leads to heterogeneity in the number of local steps $\{E_i\}_{i=1}^n$. We sample $\{\sigma_i\}_{i=1}^n$ from a set of distributions, as shown in Table 3.3. We also sample batch sizes $\{b_i\}_{i=1}^n$ uniformly from $\{16, 32, 64, 128\}$. Therefore, we consider heterogeneous $\{E_i\}_{i=1}^n$, heterogeneous $\{b_i\}_{i=1}^n$ and uniform $\{N_i\}_{i=1}^n$ in this section. We have also considered various other heterogeneity scenarios for clients and more experimental results are reported in Appendix F and Appendix G.

Distribution	Parameter setting
Dist1	Gaussian distribution $\mathcal{N}(2.0, 1.0)$
Dist2	mixture of $\mathcal{N}(0.2, 0.01)$, $\mathcal{N}(1.0, 0.1)$ and $\mathcal{N}(5.0, 1.0)$ with weights $(0.2, 0.6, 0.2)$
Dist3	Uniform distribution $\mathcal{U}[0.2, 5]$
Dist4	mixture of $\mathcal{N}(0.2, 0.01)$, $\mathcal{N}(0.5, 0.1)$ and $\mathcal{N}(2.0, 1.0)$ with weights $(0.2, 0.6, 0.2)$
Dist5	Uniform distribution $\mathcal{U}[0.2, 2]$
Dist6	mixture of $\mathcal{N}(0.2, 0.01)$, $\mathcal{N}(0.5, 0.1)$ and $\mathcal{N}(1.0, 0.1)$ with weights $(0.3, 0.5, 0.2)$
Dist7	Uniform distribution $\mathcal{U}[0.2, 1]$
Dist8	mixture of $\mathcal{N}(0.2, 0.01)$ and $\mathcal{N}(0.5, 0.1)$ with weights $(0.6, 0.4)$
Dist9	Uniform distribution $\mathcal{U}[0.2, 0.5]$

Table 3.3: Distributions of privacy parameters (ϵ's), from which clients' privacy parameters are sampled.

3.6.2 Experimental results

In this section, we investigate five main research questions about Robust-HDP, as follows.

RQ1: How do various heterogeneous DPFL algorithms affect the system utility? In Fig. 3.4, we have done a comparison in terms of the average test accuracy across clients. We observe that Robust-HDP outperforms the baselines. It achieves higher system utility by using an efficient aggregation strategy, where it assigns smaller weights to the model updates that are indeed more noisy and minimizes the noise level in the clients' aggregated model updates. The aggregation strategy of FedPFA and WeiAvg is sub-optimal, as it can not take the batch size heterogeneity and privacy parameter heterogeneity into account simultaneously.

RQ2: How does Robust-HDP improve convergence speed during training? We have also compared different algorithms based on their convergence speed in Figure 3.5. While the baseline algorithms suffer from high levels of noise in the aggregated model update $\sum_{i \in S^t} w_i^e \tilde{\Delta}_i^e$ (see Table 3.4), Robust-HDP enjoys its efficient noise minimization, which performs very close to the optimum aggregation strategy, and not only results in faster convergence but also improves utility. In contrast, based on our experiments, the baseline algorithms have to use smaller learning rates to avoid divergence of their training optimization. Note that fast convergence of DPFL algorithms is indeed important, as the privacy budgets of participating clients does not let the server to run the federated training for more rounds.

RQ3: Is Robust-HDP indeed Robust? In Figure 3.6, we compare Robust-HDP with others based on clients' desired privacy level and number of clients. As clients become more privacy sensitive, they send more noisy updates to the server, making convergence to better solutions harder. Robust-HDP shows the highest robustness to the larger noise in clients' updates and achieves the highest utility, especially in more privacy sensitive scenarios, e.g., Dist8. Also, we observe that it achieves the highest system utility when the number of clients in the system increases. Furthermore, it is completely safe in scenarios that some clients report a falsified privacy parameter to the server (Figure 3.6, right).

Figure 3.4: Comparison of average test accuracy between studied algorithms. See Tables B.7 to B.10 in the Appendix for detailed results.

Figure 3.5: Convergence speed comparison on MNIST and Dist6. Minimal algorithm diverged in 1 out of 3 trials.

RQ4: How accurate Robust-HDP is in estimating $\{w_i\}$? Figure 3.7 left, compares the weight assignment of Robust-HDP with the optimum assignment (computed from Equation (3.6)) for CIFAR10 dataset and Dist2. As the model used for CIFAR10 is relatively large (p with 1.1×10^6), we have used the approximation method in Section 3.5.5 (with $\alpha = 1$ and $p^0 = 2 \times 10^5$). Figure 3.7 left, has sorted clients based on their privacy parameter in ascending order. WeiAvg and PFA assign smaller weights to more privacy sensitive clients, while Robust-HDP assigns smaller weights to the clients with less noisy model updates.

We have also studied the effect of parameter p^0 in Section 3.5.5 on the precision of the aggregation weights returned by Robust-HDP. In Figure 3.7 right, and for CIFAR10, we have shown the increasing precision of the weights returned by Robust-HDP when p^0 grows. The larger p^0 gets, the more samples we have for estimating the noise variance in clients' model updates, hence more precise weight assignments. As explained in Section 3.5.5, when p is already large, we also avoid using too large values for p^0, as the main point of Section 3.5.5 was to pass a matrix with smaller number of rows to RPCA to avoid its low precision and high computation time when the number of rows (p) in the original input matrix M is large.

RQ5: What is the effect of data heterogeneity on the performance of Robust-HDP?

So far, we assumed an i.i.d data distribution across clients. What if the data distribution is moderately/highly heterogeneous? Assuming full participation of clients in rounds, in order

Table 3.4: The average per parameter noise variance (Equation (3.5) and Equation (3.3)) normalized by used learning rate ($\frac{\sum_{i=1}^{n} w_i^{e2} \tilde{\sigma}_i^2}{p\,\eta^2}$) in the aggregated model update ($\sum_{i=1}^{n} w_i^e \tilde{\Delta}_i^e$) at the end of first round ($e = 1$) on FMNIST with $E = 200$. Due to the projection used in PFA, computation of its noise variance was not possible. Results for Robust-HDP are shown with std variation across three experiments.

alg \ dist	Dist1	Dist2	Dist3	Dist4	Dist5	Dist6	Dist7	Dist8	Dist9
WeiAvg	1.02	1.89	0.92	3.22	4.58	28.29	9.85	48.15	34.91
DPFedAvg	1.27	16.94	16.28	26.87	25.64	70.71	18.50	85.70	43.20
minimum	4.68	103.91	103.91	127.18	103.91	1868.45	74.41	241.37	87.15
Robust-HDP	0.27 1.9e-5	0.47 5.7e-5	0.07 2.9e-6	0.64 1.0e-4	0.39 9.3e-6	7.62 1.3e-3	2.25 5.5e-5	13.86 9.9e-4	5.95 2.7e-4
Oracle (Equation (3.6))	0.27	0.47	0.07	0.64	0.39	7.60	2.25	13.81	5.93

Figure 3.6: Performance comparison on MNIST. Left: effect of clients desired privacy on utility (detailed results in Table B.11 in the Appendix). Middle: effect of number of existing clients (privacy parameters of clients are sampled from Dist6) on utility (detailed results in Table B.12). Right: Robustness of Robust-HDP when a random client (client 12 with a moderate value of 0.95) sends falsified version of its to the server for aggregation (privacy parameters of other clients are sampled from Dist5). WeiAvg and PFA are much vulnerable to this falsification.

to have a useful RPCA decomposition $M = [\tilde{\Delta}_1^e | \dots | \tilde{\Delta}_n^e] = L + S$ at the end of the round, two conditions should be met (Candès et al. 2009): 1. There should be an underlying low-rank matrix L in M 2. The difference between the matrix L and M, i.e. the noise matrix S, should be (approximately) sparse.

Whether the first condition is met or not mainly depends on how much heterogeneous the data split across clients is. Note that $\text{rank}(L)$ should be low, and not necessarily close to 1. If we assume that the second condition is met, it was shown in Theorem 1.1 in (Candès et al. 2009) that the decomposition is guaranteed to work even if $\text{rank}(L) \leq O(n/(\log p)^2)$, i.e. the rank of L grows almost linearly in n. Therefore, given that the noise matrix S is sparse and there are large enough number of clients, we expect Robust-HDP to be successful in at least the decomposition task and the following noise estimation, even if the data split across clients is moderately heterogeneous.

Figure 3.7: Left: Precision of Robust-HDP (red) compared to oracle optimum strategy (blue) for CIFAR10 and Dist2, when using the approximation method in Section 3.5.5 with $Q = 1$ and $p^0 = 2 \quad 10^5$. Right: Effect of the parameter p^0, used in the approximation method explained in Section 3.5.5, on the precision of the weights returned by Robust-HDP for CIFAR10 (with $p \quad 11 \quad 10^6$).

Table 3.5: Comparison of different algorithms (on MNIST, $T = 200$) with heterogeneous data split (maximum 8 labels per client) and 60 clients in the system all using uniform batch size 128.

alg \ distr	Dist1	Dist2	Dist3	Dist4	Dist5	Dist6	Dist7	Dist8	Dist9
WeiAvg (Liu et al. 2021a)	81.14	81.21	84.44	71.64	81.45	72.27	80.55	71.28	72.07
PFA (Liu et al. 2021a)	81.29	77.40	80.65	74.45	81.89	64.48	81.40	73.39	72.97
DPFedAvg (Noble et al. 2021)	82.61	74.32	83.12	70.8	81.34	66.51	76.10	65.16	73.03
Robust-HDP	84.84	82.78	80.78	78.91	81.66	72.30	79.32	70.68	73.82

Whether the second condition is met or not, mainly depends on how much variation exists in the amount of noise in clients' model updates, i.e. how (approximately) sparse the set $\{\sigma_1^2, \ldots, \sigma_n^2\}$ is. As shown in Equations 3.3, 3.4 and 3.5, this mainly depends on clients' privacy parameters $(f(\epsilon_i, \delta_i)\}_{i=1}^n)$, and batch sizes $\{b_i\}_{i=1}^n)$, and is independent of whether the data split is i.i.d or not. The more the variation in clients' privacy parameters/batch sizes (similar to what we saw in Figure 3.3), the better we can consider S as an approximately sparse matrix, which validates our RPCA decomposition.

In order to evaluate Robust-HDP when the data split is moderately heterogeneous, we run experiments on MNIST with 60 clients in total (compared to 200 clients before) and uniform batch size $b = 128$, and we split data such that each client holds data samples of at maximum 8 classes. The results obtained are reported in Table 3.5. As observed, Robust-HDP still outperforms the baselines in most of the cases. However, compared to the previous results, which were obtained for i.i.d data split, its superiority to the baseline algorithms has decreased. Detailed discussion of these results and scenarios with highly heterogeneous data splits are included in Appendix G.

3.7 Summary

In heterogeneous DPFL systems, heterogeneity in privacy preference, batch/dataset size results in large variations across the noise levels in clients' model updates, which existing algorithms can not fully take into account. To address this heterogeneity, we proposed a robust heterogeneous DPFL algorithm that performs noise-aware aggregation on an untrusted server, and is independent of clients' privacy parameter values shared with the server. The proposed aggregation strategy is robust, vastly applicable, scalable, and improves utility and convergence speed. It is also optimal in the sense that it minimizes the amount of noise in the aggregated model updates on the server.

Chapter 4

A Provably Robust Algorithm for Differentially Private Clustered Federated Learning

4.1 Preface

Federated learning (FL), which is a decentralized machine learning (ML) approach, often incorporates differential privacy (DP) to provide rigorous data privacy guarantees. Previous works attempted to address high structured data heterogeneity in vanilla FL settings through clustering clients (a.k.a clustered FL), but these methods remain sensitive and prone to errors, further exacerbated by the DP noise. This vulnerability makes the previous methods inappropriate for differentially private FL (DPFL) settings with structured data heterogeneity. To address this gap, we propose an algorithm for differentially private clustered FL, which is robust to the DP noise in the system and identifies the underlying clients' clusters correctly. To this end, we propose to cluster clients based on both their model updates and training loss values. Furthermore, for clustering clients' model updates at the end of the first round, our proposed approach addresses the server's uncertainties by employing large batch sizes as well as Gaussian Mixture Models (GMMs) to reduce the impact of DP and stochastic noise and avoid potential clustering errors. This idea is efficient especially in privacy-sensitive scenarios with more DP noise, where the clustering task is more prone to errors. We provide theoretical analysis to justify our approach and evaluate it across diverse data distributions and privacy budgets. Our experimental results show its effectiveness in addressing large structured data heterogeneity in DPFL.

4.2 Introduction

Federated learning (FL) (McMahan et al. 2017a) is a collaborative ML paradigm, which allows multiple clients to train a shared global model without sharing their data. However, in order for

FL algorithms to ensure rigorous privacy guarantees against data privacy attacks (Geiping et al. 2020; Hitaj et al. 2017; Rigaki and García 2020; Wang et al. 2019; Zhu et al. 2019), they are reinforced with DP (Dwork 2011; Dwork et al. 2006a,b; Dwork and Roth 2014). This is done in the presence of a trusted server (Geyer et al. 2017; McMahan et al. 2018) as well as its absence (Duchi et al. 2013, 2018; Zhao et al. 2021). In the latter case and for sample-level DP, each client runs DPSGD (Abadi et al. 2016) locally and shares its noisy model updates with the server at the end of each round.

A key challenge in FL settings is ensuring an acceptable performance across clients under heterogeneous data distributions. Several existing works focus on accuracy parity across clients with a single common model by agnostic FL (Mohri et al. 2019) and client reweighting (Li et al. 2020a,d; Zhang et al. 2023). However, a single global model often fails to adapt to the data heterogeneity across clients (Dwork et al. 2012), especially when high data heterogeneity exists. Furthermore, when using a single model and augmenting FL with DP, different subgroups of clients are unevenly affected - even with loose privacy guarantees (Bagdasaryan and Shmatikov 2019; Farrand et al. 2020; Fioretto et al. 2022). In fact, subgroups with minority clients experience a larger drop in model utility, due to the inequitable gradient clipping in DPSGD (Abadi et al. 2016; Bagdasaryan and Shmatikov 2019; Esipova et al. 2022; Xu et al. 2021). Accordingly, some works proposed to use model personalization by multi-task learning (Li et al. 2020b; Marfoq et al. 2021; Smith et al. 2017; Wu et al. 2023), transfer learning (Li and Wang 2019; Liu et al. 2020) and clustered FL (Briggs et al. 2020; Ghosh et al. 2020; Mansour et al. 2020; Ruan and Joe-Wong 2021; Sattler et al. 2019; Werner et al. 2023). The latter has been proposed for vanilla FL and is suitable when “structured data heterogeneity" exists across clusters of clients (as in this work): subsets of clients can be naturally grouped together based on their data distributions and one model is learned for each group (cluster). However, as discussed in (Werner et al. 2023), the existing non-private clustered FL approaches are vulnerable to errors in clustering due to their sensitivity to: 1. model initialization 2. randomness in clients' model updates due to stochastic noise. The DP noise existing in DPFL systems' training mechanism exacerbates this vulnerability.

To address the aforementioned gap, we propose a differentially private clustered FL algorithm which uses both clients' model updates and loss values for clustering clients, making it more robust to DP/stochastic noise (Algorithm 2): 1) Justified by our theoretical analysis (Lemma 4 and 5) and in order to cluster clients correctly, our proposed algorithm uses a full batch size in the first FL round and a small batch size in the subsequent rounds, to reduce the noise in clients' model updates at the end of the first round. 2) Then, the server soft clusters clients based on these less noisy model updates using a Gaussian Mixture Model (GMM). Depending on the “confidence" of the learned GMM, the server keeps using it to soft cluster clients during the next few rounds (Section 4.5.5). 3) Finally, the server switches the clustering strategy to local clustering of clients based on their loss values in the remaining rounds. These altogether make our DP clustered FL algorithm effective and robust. The highlights of our contributions are as follows:

- We propose a DP clustered FL algorithm (R-DPCFL), which combines information from both clients' model updates and their loss values. The algorithm is robust and achieves high-quality clustering of clients, even in the presence of DP noise in the system (Algorithm 2).

- We theoretically prove that increasing clients' batch sizes in the first round (and decreasing them in the subsequent rounds) improves the server's ability to cluster clients based on their model updates at the end of the first round with high accuracy (Lemma 5).
- We show that utilizing sufficiently large client batch sizes in the first round (and sufficiently small batch sizes in the next rounds) enables super-linear convergence rate for learning a GMM on clients' model updates at the end of the first round. This leads to soft clustering of clients using a GMM with a low computational overhead (Theorem 11).
- We extensively evaluate across diverse datasets and scenarios, and demonstrate the effectiveness of our robust DP clustered FL algorithm in detecting the underlying cluster structure of clients, which leads to an overall utility improvement for the system (§4.6).

4.3 Related work

Model personalization is a technique for improving utility under "moderate" data heterogeneity (Li et al. 2020b; Liu et al. 2022), which usually leverages extra computations, e.g., extra local iterations (Li et al. 2020b). On the other hand, clustered FL has been proposed for personalized FL with structured data heterogeneity, where clients can be naturally partitioned into clusters: clients in the same cluster have similar data distributions, while there is a significant heterogeneity across different clusters. The ultimate goal in clustered FL is to find the underlying cluster structure of clients so that after learning one model for each cluster the overall performance for all clients will be maximized. This is not achievable by learning one global model for all clients, due to the existing data heterogeneity. Existing clustered FL algorithms group clients based on their loss values (Dwork et al. 2012; Ghosh et al. 2020; Liu et al. 2022; Mansour et al. 2020; Ruan and Joe-Wong 2021) or their model updates (based on e.g., their euclidean distance (Briggs et al. 2020; Werner et al. 2023) or cosine similarity (Sattler et al. 2019)). As shown by Werner et al. 2023, the algorithms are prone to clustering errors in the early rounds of training –due to gradient stochasticity, model initialization or the form of loss functions far from their optima– which can even propagate in the subsequent rounds. This vulnerability gets exacerbated by DP noise. Our proposed DP clustered FL algorithm is applicable to any setting characterized by a number of clients, where each client holds many data samples and needs sample-level privacy protection. Cross-silo FL systems can be considered as an instance. The closest study to this setting was recently done by Liu et al. 2022 studying the interplay between privacy and data heterogeneity in silo-specific sample-level DP. They show that when clients have large dataset sizes and "moderate" data heterogeneity: 1. participation of all clients is encouraged over local training, as the model averaging on the server side yields to mitigation of DP noise 2. under the same total privacy budget, model personalization - through mean regularized multi-task learning (MR-MTL) - leads to a better performance compared to training a single global model or local training by clients (see Section B.3 in the Appendix about MR-MTL formulation). Complementing the work, we show that in MR-MTL, local training and even loss-based client clustering are not efficient for DPFL with "structured" data heterogeneity across clusters of clients.

4.4 De nitions, notations and assumptions

We keep using the approximate (;) DP (De nition 2) as the de nition adopted by every client. We consider a DPFL system (see Figure 4.2, left), where there are clients running DPSGD with the same "sample-level" privacy parameters (;): the set of information (including model updates and cluster selections) sent by client to the server satis es (;)-DP for all adjacent datasets and of the client differing in one sample (by replacement).

Let x 2 X R^d and y 2 Y = f1; : : : ; Cg denote an input data point and its target label. Client holds dataset with samples from distribution P (x; y) = P (yjx)P (x). Let h : X ! R^C be the predictor function, which is parameterized by 2 R^p. Also, let ` : R^C Y ! R+ be the used loss function (cross-entropy loss). Client in the system has empirical train loss f ( ) = 1/ (x;y)2D [`(h(x;); y)], with minimum value f . There are communication rounds indexed by and local epochs with learning rate during each round. There are clusters of clients indexed by , and the server holds cluster models { }^M_{m=1} for them at the beginning of round . Clients and belonging to the same cluster have the same data distributions, while there is a high data heterogeneity across clusters. (i) denotes the unknown true cluster of client and R^e(i) denotes the cluster assigned to it at the beginning of round . Let us assume the batch size used by client in the rst round = 1 is b^1, which may be different from the batch size b^{>1} that it uses in the rest of the rounds > 1. At the -th gradient update during the round , client uses batch B^{e;t} with size b^e, and computes the following DP noisy batch gradient:

$$\tilde{g}_i^{e;t}(\theta) = \frac{1}{b_i^e}\Big(\sum_{j \in B_i^{e;t}} g_{ij}(\theta) + \mathcal{N}(0, \sigma^2_{i;DP} I_p)\Big), \tag{4.1}$$

where g_ij( ) = clip (r `(h(x_ij;); y_ij); c), and c is a clipping threshold to clip sample gradients: for a given vector v, clip (v; c) = min{kvk; c} v/kvk. Also, N is the Gaussian noise distribution with variance ^2_{i;DP}, where _{i;DP} = c z_i(; ; b_i^1; b_i^{>1}; N_i; K; E), and z_i is the noise scale needed for achieving (;) DP by client , which can be determined with a privacy accountant, e.g., Rényi-DP accountant (Mironov et al. 2019) used in this chapter. The accountant is capable of accounting composition of heterogeneous DP mechanisms (Mironov 2017). The privacy parameter is xed to 10^{-4} in this section and for every client it holds that b_i^e ≤ N_i. Similar to Chapter 3, for an arbitrary random v = (v_1; : : : ; v_p)^> 2 R^p, Var (v) := Σ^p_{j=1} E[(v_j − E[v_j])^2]. Table 4.1 summarizes the used notations. Finally, we have the following assumption:

Assumption 3. The stochastic gradient g̃_i^{e;t}( ) = 1/b_i^e Σ_{j∈B_i^{e;t}} g_ij( ) is an unbiased estimate of r f_i( ) with a bounded variance 8 2 R^p : Var (g̃_i^{e;t}( )) ≤ ^2_{i;g}(b_i^e). The tight bound ^2_{i;g}(b_i^e) is a constant depending only on the used batch size: the larger the batch size b_i^e, the smaller ^2_{i;g}(b_i^e).

Table 4.1: Used notations in chapter 4.

n	number of clients, which are indexed by i
x_{ij}, y_{ij}	j-th data point of client i and its label
D_i, N_i	local train set of client i and its size
$D_{i,aug}$	augmented local train set of client i
$B_i^{e,t}$	the train data batch used by client i in round e and at the t-th gradient update
b_i^e	batch size of client i in round e: $\lvert B_i^{e,t} \rvert = b_i^e$
b_i^1	batch size of client i during the first round $e = 1$
$b_i^{>1}$	set of batch sizes of client i during the rounds $e > 1$
ε, δ	desired DP privacy parameters
E	total number of communication rounds in the DP-FL system, indexed by e
m^e	model parameter for cluster m, at the beginning of global round e
K	number of local train epochs performed by clients during each global round
η_l	the common learning rate used for DP-SGD
h	predictor function, e.g., CNN model, with parameter θ
ℓ	cross entropy loss
$s(i)$	the true cluster of client i
$R^e(i)$	the cluster assigned to client i in round e
$\theta_i^{e,0}$	the model parameter passed to client i at the beginning of round e to start its local training
$\tilde{\theta}_i^e(b_i^e)$	the noisy update of client i at round e, starting from $\theta_i^{e,0}$, with batch size b_i^e
$(\sigma_i^e(b_i^e))^2$	conditional variance of the update $\tilde{\theta}_i^e$ of client i: $\mathrm{Var}(\tilde{\theta}_i^e(b_i^e) \mid \theta_i^{e,0})$
$\mu_m(b^1)$	the center of the m-th cluster (when all use batch size b^1 in the first round)
$\Sigma_m(b^1)$	the covariance matrix of the m-th cluster (when all clients use batch size b^1 in the first round)
π_m	the prior probability of the m-th cluster

Figure 4.1: Loss-based clustering algorithms miscluster in the initial rounds, due to model initialization. Also, even with the assumption of perfect clustering of clients in the first rounds, clustering algorithms based on gradients (model updates) leads to clustering errors in the last rounds, due to the gradients approaching to zero.

4.5 Methodology and proposed algorithm

As discussed in (Werner et al. 2023), clustered FL algorithms which cluster clients based on their loss values (Ghosh et al. 2020; Mansour et al. 2020; Ruan and Joe-Wong 2021), i.e. assign client i to cluster $R^e(i) = \text{argmin}_m f_i(\theta_m^e)$ at the beginning of round e, are prone to clustering errors in the first few rounds, mainly due to random initialization of cluster models $\{\theta_m^0\}_{m=1}^M$. On the other hand, clustering clients based on their model updates (gradients) (Briggs et al. 2020; Sattler et al. 2019; Werner et al. 2023) makes sense only when the updates are obtained on the same model initialization. Additionally, even if we assume these algorithms can initially cluster clients perfectly in each round e, the clients' model updates (gradients) will approach zero as the clusters' model parameters converge. Hence, clients from different clusters may appear to belong to the same cluster, which results in clustering mistakes.

We now provide an example to elaborate on why clustering clients based on their losses (model updates) is prone to errors in the first (last) rounds. For example, consider Figure 4.1, where there are $M = 2$ clusters (red and blue) and $n = 4$ clients. The clients in the red cluster have loss functions $f_1(\theta) = 4(\theta + 6)^2$ and $f_2(\theta) = 4(\theta + 5)^2$ with optimum cluster parameter $\theta_1^* = -5.5$. Also, the the clients in the blue cluster have loss functions $f_3(\theta) = 4(\theta - 5)^2$ and $f_4(\theta) = 4(\theta - 6)^2$ with optimum cluster parameter $\theta_2^* = 5.5$. Clustering algorithms, which cluster clients based on their loss values on clusters' models, are vulnerable to model initialization. For example, in Figure 4.1, if we initialize the clusters' parameters with $\theta_1^0 = -11$ and $\theta_2^0 = 0$ (shown in the figure), all four clients will initially select cluster 2, since they have smaller losses on its parameter. At $\theta_2^0 = 0$, the average of clients' gradients (model updates) is zero, so all clients will remain stuck at θ_2^0 and will always select cluster 2.

On the other hand, clustering clients based on their model updates (gradients) (Briggs et al. 2020; Sattler et al. 2019; Werner et al. 2023) have clearly issues. One of these issues appears after some rounds of training. For instance, even if we assume these algorithms can initially cluster clients "perfectly" in each round e, the clients' model updates (gradients) will approach zero as the clusters' models converge to their optimum parameters. Hence, clients from different clusters may appear to belong to the same cluster, which results in clustering mistakes. For example, as shown in Figure 4.1, let us assume after T rounds of "correct" clustering of clients, the clusters' parameters get to $\theta_1^T = -4.5$ and $\theta_2^T = 5.5$ (shown in the figure). At these parameters, clients 1 and 2 (which have been "correctly" assigned to cluster 1 so far) will have gradients $f_1^0(\theta_1^T) = 12$ and $f_2^0(\theta_1^T) = 4$. Similarly, clients 3 and 4 (which have been "correctly" assigned to cluster 2 so far) will have $f_3^0(\theta_2^T) = 4$ and $f_4^0(\theta_2^T) = -4$. We see that θ_2^0 is closer to f_3^0 and f_4^0 than to f_1^0, and in the next round it will be wrongly assigned to cluster 2. This happens while the clients are clearly distinguishable based on their losses, as some progress in training has been made after T rounds $f_2(\theta_1^T) = 1$, while $f_2(\theta_2^T) = 21^2$, which clearly means that client 2 correctly belongs to cluster 1. Therefore, after making some progress in training the clusters' models, it makes more sense to use a loss-based clustering strategy than using a strategy based on clients' gradients (model updates).

Motivated by this vulnerability, which will get exacerbated by DP noise, we next propose a DP clustered FL algorithm which starts with clustering clients based on their model updates for the

Figure 4.2: Left: Considered threat model in this work, where client i has local train data $\mathcal{D}_i$ and "sample-level" DP privacy parameters (ε, δ), and does not trust any external party. Right: Three main stages of the proposed R-DPCFL algorithm.

first several rounds and then switches its strategy to cluster clients based on their loss values. We augment this idea with some other non-obvious techniques to enhance the clustering accuracy.

4.5.1 R-DPCFL algorithm

R-DPCFL algorithm has three main steps (see Figure 4.2, right and Algorithm 2), as explained in the following:

1. In the first round, clients train the initial model θ^{init} locally. They use full batch sizes in this round to make their model updates $\{\tilde{g}_i^1\}_{i=1}^n$ less noisy[1]. Then, the server soft clusters them by learning a GMM on their model updates. The number of clusters M is either given or can be found by maximizing the confidence of the learned GMM (§4.5.5).

2. During the subsequent rounds $t \in \{2, \ldots, E_c\}$, the server uses the learned GMM to soft-cluster clients: client i uses a small batch size $b_i^t \geq 1$ and contributes to the training of each cluster (m) model proportional to the probability of its assignment to that cluster $\gamma_{i,m}(\cdot)$. The number of rounds E_c is set based on "confidence level" of the learned GMM (§4.5.5).

3. After the first E_c rounds, some progress has been made in the training of the cluster models $\{\theta_m^{E_c}\}_{m=1}^M$. Now, clients' train loss values on cluster models are meaningful and is the right time to hard cluster clients based on their loss values during the remaining rounds to build more personalized models per cluster: $\mathrm{CI}^e(i) = \operatorname{argmin}_m f_i(\theta_m^e)$.

In §4.5.2 and §4.5.3, we provide theoretical justifications for the proposed method and why using a full batch size in the first round and a small batch size in the subsequent rounds helps.

[1] Note that even when clients have a limited memory budget, they can still perform DP-SGD with full batch size and no computational overhead by using gradient accumulation technique (see Appendix G of Chapter 3)

Algorithm 2: R-DPCFL

Input: Initial parameter θ^{init}, dataset sizes $\{N_1, \ldots, N_n\}$, batch sizes $\{b_1^{>1}, \ldots, b_n^{>1}\}$, clip bound c, local epochs K, global rounds E, number of clusters M (optional)

Output: cluster models $\{\theta_m^E\}_{m=1}^M$

```
for each client i ∈ {1, ..., n} do
    b_i^1 ← N_i ;                                              // full batch size
    z_i ← RDP(·, ·, b_i^1, b_i^{>1}, N_i, K, E)
for e ∈ {1, ..., E} do
    if e = 1 then
        for each client i ∈ {1, ..., n} in parallel do
            θ̃_i^1 ← DPSGD(θ^init, b_i^1, N_i, K, z_i, c)
        on server:
        if M is unknown then
            M = argmax_{M'} MSS(GMM(M', θ̃_1^1, ..., θ̃_n^1, M'))   ;   // (Sec. 4.5.5)
        {θ_1^1, ..., θ_M^1, MPQ} = GMM(M, θ̃_1^1, ..., θ̃_n^1, M) ;     // 1st stage: GMM
        set E_c ← (MPQ) ;                                      // (Sec. 4.5.5)
        Initialize cluster models uniformly: θ_1^2 = ··· = θ_M^2 = θ^init
        continue ;                                             // go to round e = 2
    else if e ∈ {2, ..., E_c} then
        for each client i ∈ {1, ..., n} do
            R^e(i) ← m with probability π_i[m] ;               // 2nd stage: soft clustering
    else
        on server: broadcast cluster models {θ_m^e}_{m=1}^M to all clients
        for each client i ∈ {1, ..., n} do
            R^e(i) = argmin_m f_i(θ_m^e) ;                     // 3rd stage: private clustering
    for each client i ∈ {1, ..., n} in parallel do
        θ̃_i^e ← DPSGD(θ^e_{R^e(i)}, b_i^{>1}, N_i, K, z_i, c) ;   // batch size b_i^{>1}
    on server:
    for each client i ∈ {1, ..., n} do
        w_i^e ← 1 / Σ_{j=1}^n 1_{R^e(j)=R^e(i)}
    for m ∈ {1, ..., M} do
        θ_m^{e+1} ← θ_m^e + Σ_{i∈{1,...,n}} 1_{R^e(i)=m} w_i^e θ̃_i^e
```

Figure 4.3: Plot of $\mathrm{Var}(\tilde{g}_i^{-1}(b_i^1) \mid \theta_i^{init})$ (left) and $\mathrm{Var}(\tilde{g}_i^{e}(b_i^e) \mid \theta_i^{e,0})$ $(e > 1)$ (right) v.s. both b_i^1 and $b_i^{>1}$. There are two clear takeaways: 1) for all $e \in \{1, \ldots, E\}$, $\mathrm{Var}(\tilde{g}_i^{e}(b_i^e) \mid \theta_i^{e,0})$ decreases with b_i^e steeply (from Lemma 4). 2) The effect of $b_i^{>1}$ on $\mathrm{Var}(\tilde{g}_i^{-1}(b_i^1) \mid \theta_i^{init})$ (left figure) is considerable. The reason is that it is used in $E - 1$ rounds and affects the noise scale $z_i(\epsilon, \delta, b_i^1, b_i^{>1}, N_i, K, E)$ used by DP-SGD. See Figure 4.4 for the plot of $z_i(\epsilon, \delta, b_i^1, b_i^{>1}, N_i, K, E)$ v.s. b_i^1 and $b_i^{>1}$. The results are obtained on CIFAR10 from Rényi DP-Accountant (Mironov et al. 2019) in a setting with $N_i = 6600$, $\epsilon = 5$, $\delta = 10^{-4}$, $c = 3$, $K = 1$, $E = 200$, $p = 11,181,642$, $\sigma_l = 5 \cdot 10^{-4}$.

4.5.2 Reducing GMM uncertainty via using full batch sizes in the first round and small batch sizes in the subsequent rounds

The DP noise in $\{\tilde{g}_i^{-1}\}_{i=1}^n$ makes it harder for the server to cluster clients by learning a GMM on the model updates. The following lemma quantifies this noise amount by extending the results in Section 3.5.2 of Chapter 3 to when different batch sizes are used in the first and the next rounds:

Lemma 4. Let us assume $\theta_i^{e,0}$ is the model parameter passed to client i at the beginning of round e. After K local epochs with step size η, the client generates the noisy DP model update $\tilde{g}_i^{e}(b_i^e)$ at the end of the round. The amount of noise in the resulting model update can be quantified as:

$$(\sigma_i^{e}(b_i^e))^2 := \mathrm{Var}(\tilde{g}_i^{e}(b_i^e) \mid \theta_i^{e,0}) \quad K \quad N_i \quad \sigma_l^2 \; \frac{p c^2 z_i^2(\epsilon, \delta, b_i^1, b_i^{>1}, N_i, K, E)}{(b_i^e)^3}.$$

The first conclusion from the lemma is that the noise level in $\tilde{g}_i^{e}$ rapidly declines as b_i^e increases: See Figure 4.3 and the effect of batch size b_i^1 on $\mathrm{Var}(\tilde{g}_i^{-1} \mid \theta_i^{init})$ (on the left) and the effect of batch size $b_i^{>1}$ on $\mathrm{Var}(\tilde{g}_i^{e} \mid \theta_i^{e,0})$ $(e > 1)$ (on the right). Let us consider $e = 1$ especially: If all clients use full batch sizes in the first round (i.e. $b_i^1 = N_i$ for every client i), it becomes much easier for the server to cluster them at the end of the first round by learning a GMM on $\{\tilde{g}_i^{-1}\}_{i=1}^n$, as their updates become more separable. An illustration of this is shown in Figure 4.5. In the