PRIVACY-PRESERVING DATA NORMALIZATION TECHNIQUES WITH MULTIPARTY HOMOMORPHIC ENCRYPTION FOR FEDERATED LEARNING

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

COMPUTER ENGINEERING

By

Melih Coşğun

August 2025

Privacy-Preserving Data Normalization Techniques with Multiparty Homomorphic Encryption for Federated Learning
By Melih Coşğun
August 2025

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Sinem Sav(Advisor)

Can Alkan

Ali Aydın Selçuk

Approved for the Graduate School of Engineering and Science:

Orhan Arıkan
Director of the Graduate School

ABSTRACT

PRIVACY-PRESERVING DATA NORMALIZATION TECHNIQUES WITH MULTIPARTY HOMOMORPHIC ENCRYPTION FOR FEDERATED LEARNING

Melih Coşğun
M.S. in Computer Engineering
Advisor: Sinem Sav
August 2025

Data normalization is a crucial preprocessing step for enhancing model performance and training stability. In federated learning (FL), where data remains distributed across multiple parties during collaborative model training, normalization presents unique challenges due to the decentralized and often heterogeneous nature of the data. Traditional methods rely on either independent client-side processing, i.e., *local normalization*, or normalizing the entire dataset before distributing it to parties, i.e., *pooled normalization*. Local normalization can be problematic when data distributions across parties are non-IID, while the pooled normalization approach conflicts with the decentralized nature of FL. In this thesis, we explore the adaptation of widely used normalization techniques to FL and define the term *federated normalization*. Federated normalization simulates pooled normalization by enabling the collaborative exchange of normalization parameters among parties. Thus, it achieves performance on par with pooled normalization without compromising data locality. However, sharing normalization parameters such as the median introduces potential privacy risks, which we further mitigate through a robust privacy-preserving solution. Our contributions include: (i) We systematically evaluate the impact of various federated and local normalization techniques in non-IID FL scenarios, (ii) We propose a novel homomorphically encrypted k-th ranked element (and median) calculation tailored for the federated setting, enabling secure and efficient federated normalization, (iii) We propose privacy-preserving implementations of widely used normalization techniques for FL, leveraging multiparty fully homomorphic encryption (MHE).

Keywords: Federated learning, Data Normalization, Data Privacy.

ÖZET

FEDERE ÖĞRENME İÇİN GİZLİLİĞİ ÇOK TARAFLI HOMOMORFİK ŞİFRELEME İLE KORUYAN VERİ NORMALLEŞTİRME TEKNİKLERİ

Melih Coşğun
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Danışmanı: Sinem Sav
Ağustos 2025

Veri normalleştirme, model performansını ve eğitim kararlılığını artırmak için çok önemli bir ön işleme adımıdır. İşbirliğine dayalı model eğitimi sırasında verilerin birden fazla tarafa dağıtıldığı federe öğrenmede (FL), verilerin merkezi olmayan ve genellikle heterojen yapısı nedeniyle normalleştirme benzersiz zorluklar ortaya çıkarır. Geleneksel yöntemler ya bağımsız istemci tarafı işlemeye, yani *yerel normalleştirme*, ya da tüm veri kümesini taraflara dağıtmadan önce normalleştirmeye, yani *havuzlanmış normalleştirme*, dayanır. Yerel normalleştirme heterojen veri dağılımlarında sorunluyken, havuzlanmış normalleştirme FL'nin merkeziyetsiz yapısıyla çelişir. Bu tezde, yaygın olarak kullanılan normalleştirme tekniklerinin FL'ye uyarlanmasını araştırıyor ve *federe normalleştirme* terimini tanımlıyoruz. Federe normalleştirme, taraflar arasında parametre paylaşımıyla havuzlanmış normalleştirmeyi simüle eder. Böylece, veri yerelliğinden ödün vermeden havuzlanmış normalleştirme ile eşit performans elde eder. Bununla birlikte, medyan gibi normalleştirme parametrelerinin paylaşılması potansiyel gizlilik risklerini beraberinde getirir ve biz de bu riskleri gizliliği koruyan sağlam bir çözümle azaltıyoruz. Katkılarımız şunları içermektedir: (i) Bağımsız ve özdeş dağılımlı olmayan FL senaryolarında çeşitli federe ve yerel normalleştirme tekniklerinin etkisini sistematik olarak değerlendiriyoruz, (ii) Güvenli ve verimli federe normalleştirme sağlayan, federe ortam için uyarlanmış yeni bir homomorfik olarak şifrelenmiş k-inci sıralı eleman (ve medyan) hesaplaması öneriyoruz, (iii) Çok partili tam homomorfik şifrelemeden (MHE) yararlanarak FL için yaygın olarak kullanılan normalleştirme tekniklerinin gizliliği koruyan uygulamalarını öneriyoruz.

Anahtar sözcükler: Federe Öğrenme, Veri Normalleştirme, Veri Mahremiyeti.

Acknowledgement

First and foremost, I would like to express my deepest gratitude to my advisor, Assistant Professor Sinem Sav, for her invaluable guidance, support, and encouragement throughout the course of this thesis. Her expertise and insightful feedback have been instrumental in shaping the direction and quality of this work.

I am grateful to Associate Professor Can Alkan and Professor Ali Aydın Selçuk for serving on my thesis committee and providing valuable feedback. I am particularly thankful to Professor Ali Aydın Selçuk for his additional guidance and perspective on both academic matters and life in general. His wisdom and mentorship have left a lasting impact on me.

I am deeply thankful to my friend Mert Gençtürk, who accompanied me on this long journey to bring the project to life. Without his dedication and collaboration, this work would not have been possible.

I am especially grateful to my dear friends Mehmet Can Şakiroğlu, Murat Şahin, Ömer Kaan Gürbüz, and Kaan Efe Keleş for their continuous support and motivation during this process. Their insightful feedback, countless hours of brainstorming—both on the project and on many other topics—and their friendship made this journey more enjoyable and intellectually enriching.

Finally, I would like to acknowledge the financial support provided by the Scientific and Technological Research Council of Turkey (TÜBİTAK) under the 3501 Career Development Program (Grant #124E091), which made this research possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning (ML) has become an integral part of numerous applications, ranging from healthcare and finance to entertainment and smart devices. However, the increasing prevalence of sensitive user data in training ML models necessitates stringent privacy protections. Federated learning (FL) [1, 2] has emerged as a promising paradigm for addressing this challenge by enabling decentralized model training. In FL, data remains on local devices, and only model updates are shared with a central server, reducing the risk of exposing raw data and thereby enhancing user privacy. Despite its advantages, FL remains vulnerable to several privacy attacks [3, 4, 5, 6], prompting the emergence of *privacy-preserving federated learning (PPFL)* research [7].

A critical aspect of building effective ML models is data normalization, which involves scaling or transforming input features to improve convergence and performance. Normalization reduces the influence of feature-scale disparities, enabling models to learn more effectively. Widely used normalization techniques,

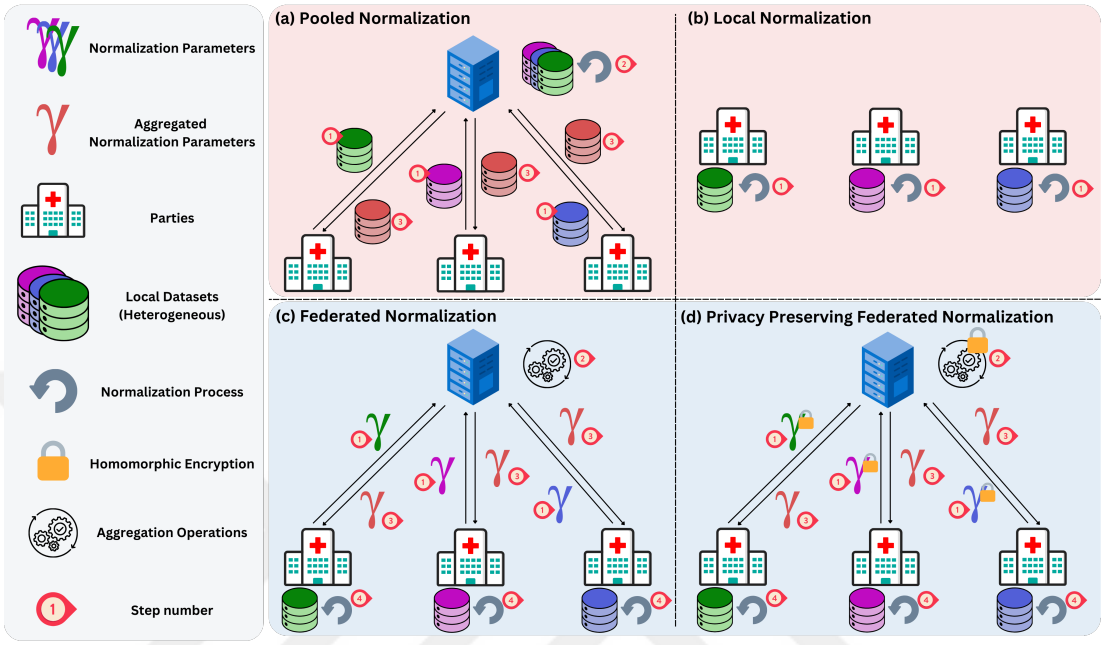

Figure 1.1: Overview of data normalization approaches in federated learning. The top (red) section shows traditional methods: (a) *Pooled Normalization*, where raw datasets are centrally normalized, and (b) *Local Normalization*, where each client normalizes data independently. The bottom (blue) section introduces our federated methods: (c) *Federated Normalization*, achieving comparable performance to pooled normalization by aggregating normalization parameters while keeping data local, and (d) *Privacy-Preserving Federated (PPF) Normalization*, further enhancing privacy of (c) through homomorphic encryption to **securely** aggregate parameters and minimize information leakage.

such as MinMax scaling, z-score normalization, and robust scaling, have consistently proven their utility in centralized machine learning workflows. However, adapting these methods to FL presents unique challenges. Thus, especially *PPFL* approaches that employ differential privacy [8, 9, 10, 11, 12], homomorphic encryption [13, 14, 15, 16, 17, 18, 19], or secure multiparty computation [20, 21, 22, 23, 24, 25] require privacy-preserving data normalization prior to training.

In FL, the heterogeneity of client data and the lack of direct access to raw data make normalization a non-trivial task. Traditional approaches normalize datasets in two ways. The first approach, which we refer to as the *pooled setting*, as also referred to in [26], involves normalizing the entire dataset before distributing it

to clients (see Figure 1.1-a). This method applies only to experimental settings, not real FL scenarios, as it assumes centralized data access, contradicting FL's decentralized nature. The second approach, which we refer to as the *local setting*, involves each client normalizing their own data independently on their local device (see Figure 1.1-b). While this method aligns with FL principles and ensures data privacy, it encounters challenges in real-world scenarios where federated datasets are typically non-independent and identically distributed (non-IID). This non-IID nature leads to inconsistencies, reducing the effectiveness of local normalization compared to the pooled approach.

We define *federated normalization* as normalizing federated data to achieve performance similar to pooled normalization by exchanging specific information among clients without sharing raw data. (see Figure 1.1-c). While this approach provides some privacy by keeping raw data local, the exchanged information can still potentially leak dataset insights, depending on the normalization technique. For example, federated MinMax scaling requires computing global feature-wise minima and maxima, which involves clients sharing their local extremes. However, this can risk revealing sensitive information, depending on the dataset and application context.

1.1 Problem Definition.

Existing literature lacks a thorough comparison between federated and local normalization approaches in FL. Key questions remain: What is the practical impact of federated versus local normalization on model performance, and how does data heterogeneity affect their effectiveness? Finally, is it possible to implement federated normalization techniques in real-world settings while preserving client privacy?

To the best of our knowledge, the only prior work addressing a similar problem is [26], which proposes a protocol for secure Yeo-Johnson (YJ) transformation in FL. While this work offers valuable insights, it is focused only on federated

YJ Transformation and does not address the broader range of normalization techniques. Moreover, their evaluation is limited to regression tasks and does not fully address real-world heterogeneous FL scenarios. This leaves an opportunity for further investigation into the key questions we aim to address. Similar to [26], our work focuses on cross-silo FL, where a limited number of parties (typically tens to hundreds) collaboratively train a model. We use the terms 'clients' and 'parties' interchangeably throughout this thesis.

1.2 Contributions.

In this thesis, we make the following contributions to address the defined problems.

1. For the first time, we present a comprehensive comparison of local and federated normalization techniques in non-IID FL scenarios. Our analysis spans diverse datasets and varying client counts, evaluating both local and federated settings. Using standard evaluation metrics, we demonstrate the critical role of federated normalization in enhancing FL performance.

2. We introduce novel privacy-preserving implementations of various data normalization techniques, including z-score normalization, robust scaling, and MinMax scaling, tailored for FL. These implementations aim to reduce communication overhead while preserving client privacy. By leveraging multiparty fully homomorphic encryption (MHE), we ensure that client data remains confidential throughout the normalization process. For an illustration of the core idea behind privacy-preserving federated (PPF) normalization, see Figure 1.1-d.

3. PPF robust scaling, one of our proposed privacy-preserving implementations, requires securely computing the *k-th ranked element* among multiple parties. To address this, we propose a novel encrypted collaborative k-th ranked element (and median) calculation by extending the algorithm

in [27] to support floating-point numbers. Leveraging MHE, our approach ensures data confidentiality while enabling computations directly on encrypted floating-point data with potential applications *beyond normalization*.

By addressing the intersection of data preprocessing and privacy-preserving machine learning, this thesis offers valuable insights and practical guidance to advance FL methodologies.

Chapter 2

Related Work

We summarize related work on local data normalization, activation layer normalization and PPF data normalization. We also discuss prior approaches for securely computing the k-th ranked element, relevant to our contributions in Section 5.

2.1 Local Data Normalization.

Current research applies data normalization either in a pooled or local manner. Zhang et al. [28] note in their FL healthcare survey that normalization is typically performed before data distribution (pooled) or locally on each client. They further suggest that local normalization may propagate biases in feature values into the pre-processed data [28]. Example FL works which rely on local or pooled data normalization include [29, 30, 31, 32, 33, 34]. To examine the impact of local normalization, we compare it against both no normalization and federated normalization scenarios.

2.2 Activation Layer Normalization.

Several studies have investigated activation layer normalization techniques in FL [35, 36, 37]. Du et al. [35] introduce external covariate shift in FL and show that layer normalization outperforms batch normalization under non-IID conditions. Subsequent studies further explore batch normalization's limitations in FL and compare it with alternative normalization techniques [36, 37]. These normalization methods operate within neural network layers during training, in contrast to our focus on data normalization as a preprocessing step. Nonetheless, we include prominent activation normalization techniques—batch normalization, group normalization, instance normalization, and layer normalization—in our experiments. This allows for a comparative analysis of activation and data normalization methods across diverse FL scenarios, providing a comprehensive evaluation of their impact on model performance.

2.3 Secure Computation of k-th Ranked Element.

The robust scaling normalization technique transforms data using the median, 25th percentile, and 75th percentile values. Thus, implementing robust scaling in a PPFL environment requires the secure and federated computation of k-th ranked elements (see Section 5). Aggarwal et al. [27] compute the k-th ranked element in a multiparty setting via an iterative binary search, starting from a global range and querying data holders for counts above or below a candidate value. However, the method is limited to integers and lacks a secure implementation. Goelz et al. [38] address this by providing a secure multiparty implementation and benchmark, though it remains restricted to integer values and is not tailored for ML tasks. To support FL, we adapt this algorithm for floating-point inputs and incorporate MHE into its design. We introduce, for the first time, a federated and encrypted k-th ranked element calculation, designed to enable privacy-preserving federated normalization. To enhance security and minimize

data leakage, we incorporate MHE into our design, which differs from previous work. Our implementation optimizes communication rounds by parallel processing of input features and is specifically designed for robust scaling in FL, ensuring both privacy and efficiency.

2.4 Privacy-Preserving Federated (PPF) Data Normalization.

The only related work by Marchand et al. [26] propose a secure implementation of the YJ transformation using secure multiparty computation within FL. Although YJ is an established normalization technique, its application is limited to specific tasks. In contrast, our work presents privacy-preserving implementations of additional widely used normalization methods, extending their applicability to a broader range of tasks and data types. In addition, the experiments in [26] are limited to regression tasks with numeric datasets due to the specific nature of the YJ transformation. Their evaluation primarily considers IID and quantity-imbalanced non-IID data distributions, which do not fully represent the broader spectrum of non-IID scenarios crucial for real-world FL applications. Furthermore, the datasets used in their experiments are relatively small (up to 1,797 samples), limiting their relevance for practical scenarios. As emphasized by [39], incorporating diverse non-IID data distributions is essential to accurately reflect real-world FL challenges.

Our work addresses these limitations by investigating a wider range of non-IID data settings, incorporating both numeric and image datasets, and evaluating performance on regression and classification tasks. We systematically compare the proposed federated normalization techniques, including federated and local YJ transformations, across diverse non-IID scenarios using various neural network models. This comprehensive approach enhances the applicability of our findings, bridging important gaps and aligning more closely with practical FL deployments.

Chapter 3

Background

In this section, we summarize the normalization techniques, federated learning with non-iid data and, multiparty fully homomorphic encryption.

3.1 Normalization Techniques

Normalization is a crucial preprocessing step in machine learning that standardizes feature scales, ensuring that models train effectively. For a dataset that contains N number of features and each feature indexed as j, the most common normalization techniques can be defined as:

3.1.1 Z-Score Normalization:

This method scales features to have a mean of zero and a standard deviation of one. It is particularly effective when the data follows a Gaussian distribution. Formally, for a feature value x_j, the normalized value is given by: $x'_j = \frac{x_j - \mu_j}{\sigma_j}$, where μ_j and σ_j are the mean and standard deviation of the j-th feature, respectively.

3.1.2 MinMax Scaling:

This technique transforms data to lie within a fixed range, typically [0, 1], by applying the formula: $x'_j = \frac{x_j - \min(x_j)}{\max(x_j) - \min(x_j)}$, where $min(x_j)$ and $max(x_j)$ represents the minimum and maximum values of the feature j. MinMax scaling is widely used when feature magnitudes vary significantly.

3.1.3 Robust Scaling:

Designed to mitigate the impact of outliers, robust scaling scales features using the interquartile range (IQR) instead of the mean and standard deviation. The transformation is given by: $x'_j = \frac{x_j - \text{median}(x_j)}{\text{IQR}(x_j)}$.

3.1.4 Yeo-Johnson Transformation:

The Yeo-Johnson transformation [40], an extension of the Box-Cox transformation [41], handles both positive and negative values. It stabilizes variance and helps data approximate a normal distribution. The transformation is defined as follows:

$$y = \begin{cases} \frac{(x+1)^{\lambda}-1}{\lambda} & \text{if } x \geq 0, \\ -\frac{(-x+1)^{2-\lambda}-1}{2-\lambda} & \text{if } x < 0, \end{cases}$$

where λ is a parameter that can be optimized to minimize the deviation from normality.

3.1.5 Activation Layer Normalization:

Unlike the previously mentioned methods, activation layer normalization techniques operate within neural network layers during training rather than in preprocessing. Prominent examples include layer normalization [42], batch normalization [43], instance normalization [44], and group normalization [45]. While

sharing a common goal, these methods differ in their data processing approaches. They incorporate learnable parameters that integrate seamlessly into FL algorithms, similar to standard neural network components.

3.2 Federated Learning with Non-IID Data

Federated learning (FL) is a decentralized machine learning paradigm that enables model training across multiple clients without requiring data to leave their devices. Instead of sharing raw data, each client i computes local models W_i^k for k-th FL iteration and sends them to a central server. The central server then aggregates the local models to update the global model, e.g., $W^{k+1} \leftarrow \sum_{i=1}^{N} \frac{n_i}{n} W_i^k$ for FedAvg algorithm [2]. FL offers several advantages, including enhanced privacy, lower communication overhead, and the ability to utilize data that would otherwise be inaccessible due to privacy or regulatory constraints.

In an *IID (independent and identically distributed)* FL scenario, all clients' data is assumed to be drawn from the same underlying distribution. This assumption simplifies the learning process, as local updates accurately reflect the global distribution, facilitating effective aggregation and faster convergence. However, in real-world scenarios, the IID assumption rarely applies, requiring FL to address the challenges of *non-IID data* [39], where the data distribution across clients is inherently heterogeneous. This heterogeneity introduces several challenges that are critical to the success of FL.

Following Li et al. [39], who proposed data partitioning to simulate common non-IID scenarios, we explore key non-IID cases to evaluate normalization techniques across diverse FL settings with P parties:

- **Distribution-based Label Imbalance:** Simulates non-uniform label distributions across clients—for example, some may predominantly hold samples from certain classes, while others have a more uniform distribution. To simulate this, a Dirichlet distribution (Dir_P) is employed, in which each

client i is assigned a proportion p_i of the samples for each label, with $p_i \sim \text{Dir}_P(\beta_\ell)$. The concentration parameter β_ℓ determines the level of imbalance: smaller values result in pronounced imbalance, while larger values yield more uniform distributions [39].

- **Noise-based Feature Imbalance:** The feature distributions differ across clients, often caused by variations such as environmental factors or sensor noise. To simulate this, the dataset is first randomly and equally partitioned among clients. Then, Gaussian noise $n_i \sim \mathcal{N}(0, \beta_f \cdot i/P)$ with varying intensity levels is added to the features of each client C_i's local dataset. The user-defined parameter β_f controls the level of noise added. This technique ensures that while the overall feature space remains consistent, the specific details vary significantly across clients [39].

- **Quantity Imbalance:** This case represents scenarios where FL clients possess datasets of varying sizes. This is simulated by allocating data samples to clients based on a Dirichlet distribution, where the concentration parameter dictates the extent of imbalance in the data quantities [39].

These cases help us assess how non-IID data affects our proposed federated normalization and test the robustness of normalization methods in real-world scenarios.

3.3 Multiparty Fully Homomorphic Encryption

Fully homomorphic encryption (FHE) [46] enables computations directly on encrypted data without decryption. It supports both addition and multiplication on ciphertexts, producing correct results upon decryption—as if the operations were performed on the original plaintexts. FHE schemes such as **CKKS (Cheon-Kim-Kim-Song)** [47] are specifically designed to efficiently support such computations. CKKS encodes plaintexts into polynomials packed into "slots," allowing parallel processing of multiple values. Designed for approximate arithmetic on

complex numbers, it supports operations on floating-point arrays with minimal noise—ideal for machine learning tasks that tolerate slight approximation errors.

In this work, we use the MHE variant of CKKS [48], which distributes the encryption key across multiple parties. No single party can decrypt the data alone; decryption requires partial shares from all participants. This collaborative process ensures data remains protected unless all parties agree to decrypt, strengthening privacy and preventing unauthorized access. MHE is especially relevant to FL, enabling secure computation across distributed data while preserving data privacy. The core CKKS functionalities we rely on are summarized below:

1. **Key Generation** ($\mathsf{KeyGen}(\lambda)$): Each party i generates a secret key sk_i and participates in a secure key exchange to establish a collective public key pk depending on the security parameter λ.
2. **Encryption** ($\mathsf{Encrypt}(m, pk)$): Message m is encrypted using pk. We denote the encrypted message as $\langle m \rangle$.
3. **Computations:** Homomorphic operations such as multiplication or addition are performed on the encrypted data. The resulting ciphertexts remain valid for decryption under the MHE scheme. However, repeated homomorphic operations can cause ciphertext noise to grow, potentially leading to decryption errors, which is eliminated with collective bootstrapping.
4. **Collective Bootstrapping** ($\mathsf{CBootstrap}(\langle m \rangle, \{sk_i\})$): To reduce noise accumulation and enable deeper computations, the protocol uses collective bootstrapping [48, 49], where all parties jointly refresh ciphertexts.
5. **Collective Decryption** ($\mathsf{CDecrypt}(\langle m \rangle, \{sk_i\})$): Each party generates a partial decryption share using sk_i, which is then securely aggregated to reconstruct the computed result in plaintext.

We omit the details of certain functionalities, such as relinearization and rescaling, provided by the MHE-CKKS scheme, which facilitate efficient management of ciphertext dimensions and scaling factors during computations.

Chapter 4

Evaluation of Normalization Techniques for Federated Learning

To address the questions outlined in Section 1, we design a comprehensive experimental setup to evaluate the performance of local normalization, activation layer normalization, and our proposed federated normalization techniques. This section introduces federated normalization, outlines the experimental setup and its rationale, and concludes with an analysis and discussion of the results, highlighting key takeaways.

4.1 Federated Normalization

Before presenting the experimental evaluation, we begin by introducing the concept of federated normalization and outlining our proposed methods. In federated normalization, clients share certain information about their datasets, which is then aggregated to compute global normalization parameters (see Figure 1.1-c). These global parameters can be used to normalize data locally, ensuring that the

Dataset	Type	Count	Task
HCV	Tabular	615 (13)	Multi-class (5)
Parkinson's	Tabular	5,875 (16)	Regression
BCW	Tabular	569 (30)	Binary
CIFAR-10	Image	60,000 (3,072)	Multi-class (10)
MNIST	Image	70,000 (784)	Multi-class (10)

Table 4.1: Dataset Summary: Type, Sample Count (Features), and Task (Classes).

data never leaves the client devices while enabling normalization as if the data were centrally aggregated.

Since the process is straightforward, we use MinMax as a representative example and present all algorithms, along with their PPF variants, in detail in Section 5. In the MinMax scaling scenario, each client shares the minimum and maximum values of its local dataset for each feature. The central server aggregates these values to determine the global minimum and maximum. These global statistics are then used to normalize each client's local data, resulting in outcomes equivalent to normalization performed on a centrally pooled dataset.

We propose and evaluate federated versions of MinMax scaling, robust scaling, and z-score normalization. Additionally, we include a federated version of the YJ transformation and activation layer normalizations in our experiments to provide a more comprehensive analysis.

4.2 Experimental Setup

We run experiments on five datasets across diverse non-IID scenarios. Key parameters include dataset choice, non-IID types and imbalance levels, number of clients, and normalization techniques. For all experiments, we use the FedAvg [2] strategy, with one epoch of local training and evaluation during training is performed on a server-side validation set. Note that the validation dataset can also

be distributed for federated loss computation without affecting results. We use early stopping based on validation loss, adjusting the patience parameter. Each experiment is run three times with different random seeds, and we report the average. Experiments are implemented using the Flower FL library [50] or our custom simulation, depending on memory constraints. All experiments are run on NVIDIA GTX 1080 Ti GPUs and AMD Opteron(TM) 6212 CPU with 256GB RAM.

4.2.1 Datasets

We conduct our experiments two image and three tabular datasets. Four of these are used for classification tasks, while one is used for regression. We select these datasets to ensure diversity in size, features, tasks, and model architectures. In particular, we use CIFAR10 [51], MNIST [52], Breast Cancer Wisconsin [53], Hepatitis C [54] and Parkinson's Monitoring [55]. We reserve 20% of the datasets for the unseen test set and split the remaining data into 90% for training and 10% for validation. Table 4.1 summarizes the dataset specifications.

4.2.2 Model Architecture

We employ neural network-based architectures to incorporate activation layer normalization techniques in our experiments. For image datasets, we adopt convolutional neural networks (CNNs), while for tabular datasets, we use fully connected neural networks integrating variable activation layer normalization. Based on the normalization type, we use layer, group, batch, or instance normalization for the layers. We omit activation layer normalization from our networks during data normalization experiments. Model architecture details are provided in Appendix B.

Imbalance Type	Low Imbalance	High Imbalance
Quantity	$\beta = 0.5$ $(0.7)^*$	$\beta = 5$
Label	$\beta = 0.5$	$\beta = 5$
Feature	$\sigma = 0.3$ $(0.01)^*$	$\sigma = 0.7$ $(0.1)^*$

* Parameters in parentheses indicate values used for image data.

Table 4.2: Summary of imbalance parameters.

4.2.3 Non-IID settings

Following the dataset imbalance methods described in [39], we apply six non-IID settings and one IID setting per dataset. Non-IID types include feature, label, and quantity imbalance, each with high and low imbalance configurations (see Table 4.2). For image datasets, we use different feature imbalance settings than for tabular data, as strong feature imbalance on images degrades quality and leads to model divergence.

4.2.4 Number of clients (P)

We rely on cross-silo FL setting with 10, 20, and 50 clients for image datasets and 10, 20, 30 clients for tabular datasets-due to data size limitations.

4.2.5 Normalization Techniques

For all datasets, we evaluate both federated and local versions of z-score normalization, MinMax scaling, and robust scaling across all datasets. For tabular datasets, we also include local and federated YJ transformations. Additionally, we explore activation layer normalization methods, including layer, batch, group, and instance normalization. To efficiently simulate federated normalization, we normalize all training, validation, and test sets using parameters computed from the training set. For local normalization, each client computes normalization

parameters from its local training data and applies them to its own training, validation, and test sets. After the normalization, validation and test sets are merged and moved to the server to speed up evaluation.

4.2.6 Evaluation Metrics.

For all datasets, we track the training loss curves for both decentralized and centralized setups, the validation loss curve, and the final test loss. For classification tasks, we additionally track training and validation accuracy curves, final test accuracy, F1 score, precision, and recall, using cross-entropy as the loss function. For regression tasks, performance is evaluated using mean absolute error (MAE) and R^2 on both validation and test sets, with MAE as the loss.

4.3 Experimental Results

4.3.1 Federated Normalization Outperforms Local Normalization

Figure 4.1 displays the average performance of each normalization technique across all datasets. Federated normalization methods consistently outperform their local counterparts, supporting our key claim that aggregating normalization statistics globally leads to better generalization and robustness in FL—particularly under heterogeneous data distributions. Table 4.3 summarizes the number of runs in which federated normalization achieved an average performance improvement between %1 and %8. Overall, federated normalization outperformed local normalization in 232 runs, while local normalization performed better in 59 runs. The comparison metric is test F1 score for all datasets, except Parkinson's, where we use the test R^2 score (range $(-\infty, 1]$). For Parkinson's, only runs with $R^2 > 0$ were included to ensure meaningful improvement calculations. Results were counted as ties if the improvement was below 0.5% for all

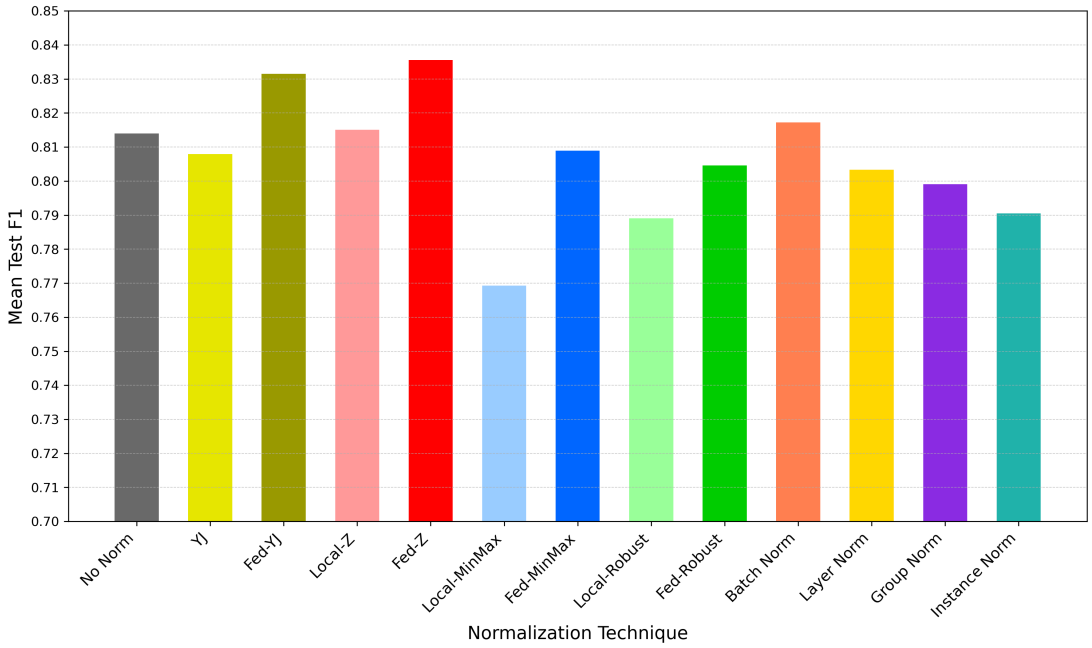

Figure 4.1: Test F1 scores across different normalization techniques. Results are averaged over all datasets, client numbers, and imbalance types. Note that YJ transformations were not applied to image datasets; thus, average YJ F1 scores are calculated over tabular datasets. Only classification datasets are included in the computations due to representation with F1 score metric. Labels prefixed with 'Fed' denote the federated versions of normalization methods, while 'No Norm' represents the baseline without normalization.

datasets.

Except for CIFAR10, results show notable gains in non-IID settings. CIFAR10's smaller gap likely stems from its higher complexity, which lessens the effect of applied feature imbalance. Overall, local normalization, while effective centrally, struggles in FL due to client-specific biases. A key observation is that, on average, models perform better even *without any normalization* than *with local normalization* (see Figure 4.1). This aligns with the findings of Zhang et al. [28], who argue that local normalization can amplify client-specific biases in FL. These patterns are even more evident when examining specific imbalance types, as discussed in the next subsection.

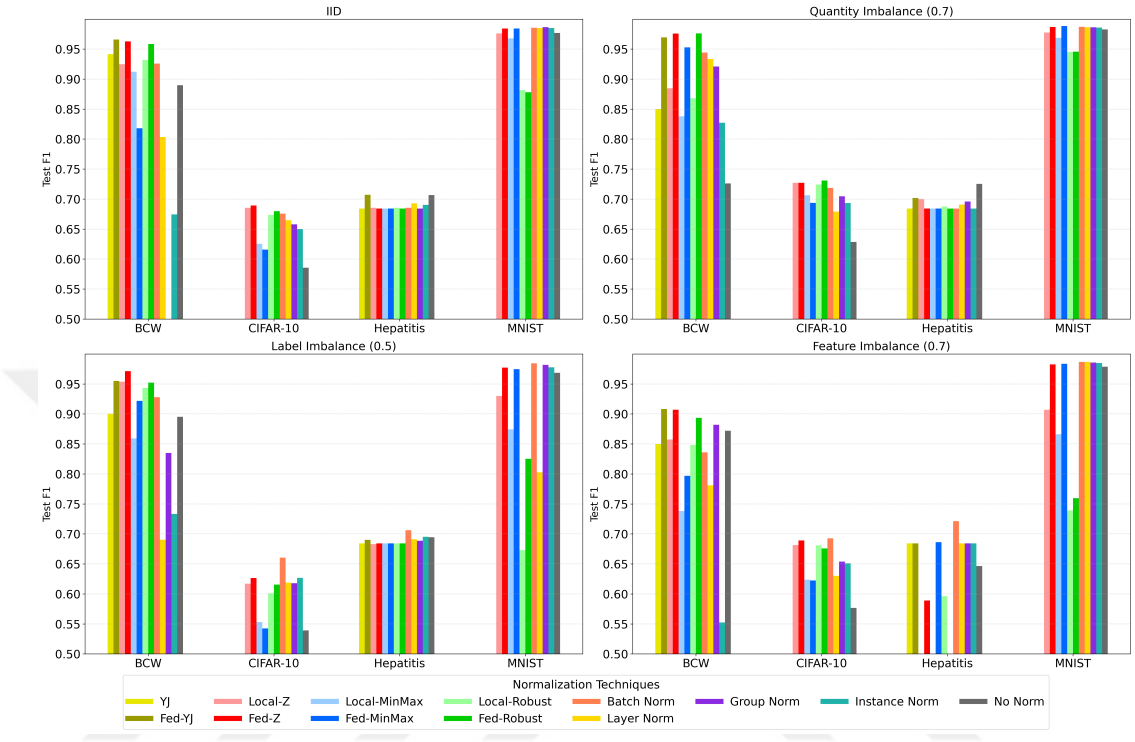

Figure 4.2: Test F1 scores of different imbalance settings for 30 clients (50 for image datasets). Highly imbalanced (more heterogeneous) parameters are selected for each imbalance type. Only classification datasets (BCW, CIFAR-10, Hepatitis, MNIST) are included due to representation with F1 score metric. The x-axis shows the dataset names.

From Figure 4.1 and Figure 4.2, we also observe that activation layer normalizations—i.e., batch, layer, instance, and group normalization—perform comparably to various federated normalization techniques. However, it is important to note that activation layer normal ization techniques are not universally applicable to all PPFL methods, particularly those that rely on FHE [36, 16] and require certain reformulations [56]. In privacy-critical federated applications, such techniques must be applied cautiously or augmented with privacy-preserving mechanisms—an effort that is often complex and not always straightforward. Thus, we focus on the federated normalization techniques in later observations.

Dataset	Federated	Local	Ties	Mean Fed. Impr.
BCW	71	4	9	0.0414
CIFAR10	16	15	32	0.0113
Hepatisis	33	21	30	0.0567
MNIST	45	2	16	0.0817
Parkinson's	67	17	0	0.0645*
Total	232	59	87	

* Mean Fed. Impr. for Parkinson's dataset calculated on test R^2 score, while others calculated on test F1 score.

Table 4.3: Comparison of Federated vs. Local Normalization Across Datasets. The "Federated" and "Local" columns report the number of runs each method outperformed the other, while "Mean Fed. Impr." reports the average gain when federated normalization is superior.

4.3.2 Performance Under Imbalanced Distributions

Figure 4.2 presents the performance of all normalization techniques across different imbalanced settings—IID, quantity, label, and feature—with 30 clients (see Appendix A, Figure A.1 for other variations of these settings). Under IID distributions, most techniques achieve comparable results, as expected. However, under imbalanced data distributions, the performance gap between local and federated normalization techniques becomes substantial.

Federated techniques retain high F1 scores across all imbalance scenarios. In contrast, local normalization methods degrade significantly under non-IID settings, especially when clients exhibit non-overlapping feature ranges or label distributions. This trend is further reinforced by results on the MNIST dataset shown in Figure 4.3 and also for Hepatitis, Parkinson's and BCW datasets in Appendix A, Figure A.2, A.3 and A.4. Although all normalization methods perform well under IID settings, federated normalization techniques continue to outperform local methods across all types of imbalance. Particularly under *label* and *feature* imbalance, federated methods maintain strong performance while local methods degrade sharply. In quantity imbalance, performance resembles IID settings, as clients with large data shares can dominate training, mimicking

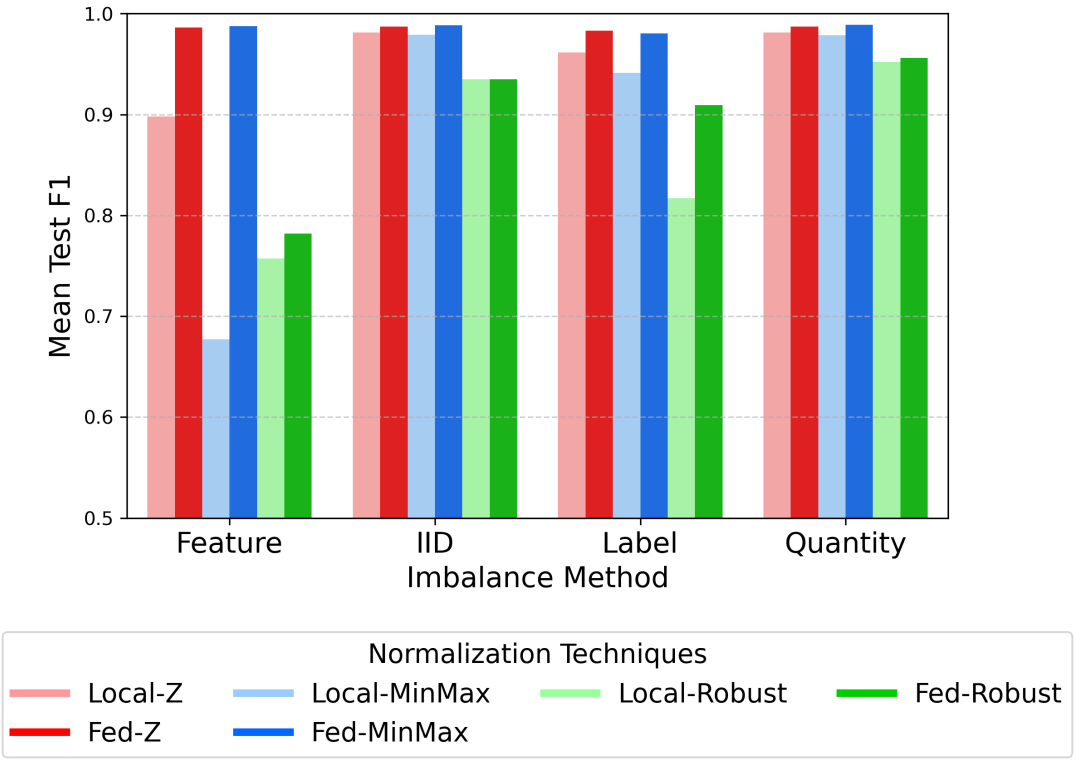

Figure 4.3: Test F1 results of MNIST dataset averaged over client numbers to observe the impact of different heterogeneous scenarios. We observe that under the feature imbalanced distribution, federated normalization types performs much better than local normalizations.

centralized learning.

In regression tasks, the gap between federated and local normalization is more pronounced. As shown in Figure 4.4, under feature imbalance, local normalization often fails to converge, whereas federated normalization consistently yields strong performance.

4.3.3 Impact of Client Count Under Feature Imbalance

To examine the impact of the number of clients, we provide Figure 4.5, focusing on the Hepatitis dataset, comparing performance across IID and feature-imbalanced settings (see Appendix A, Figure A.5 and A.6 for MNIST and BCW datasets). While performance gap between federated and local normalizations remain stable

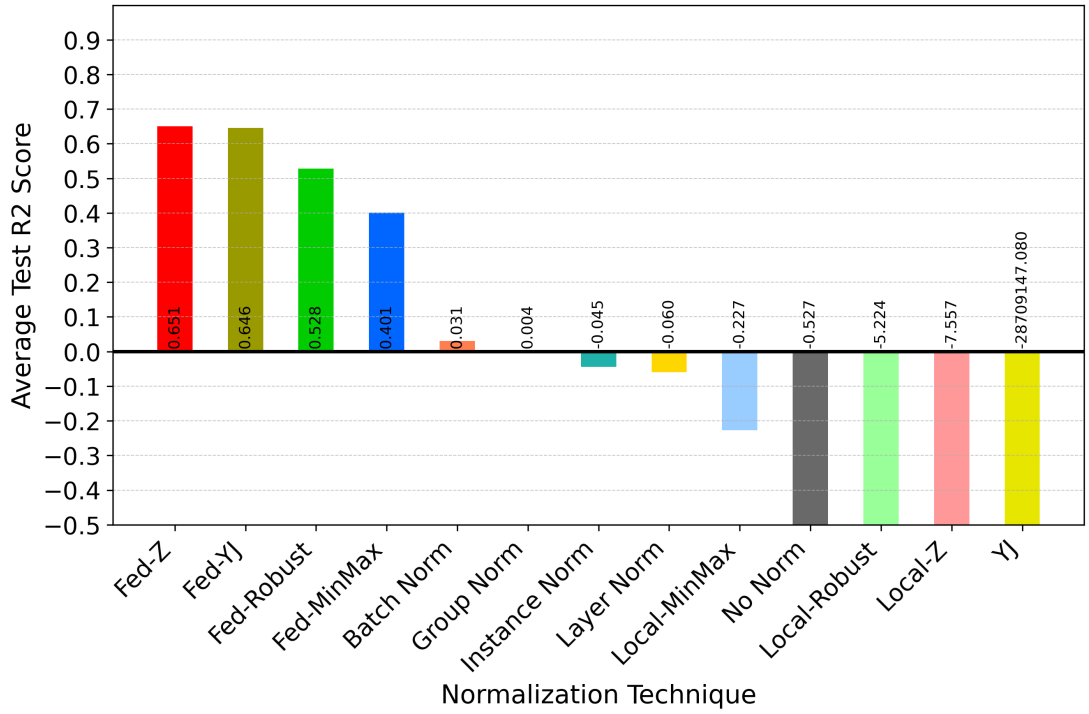

Figure 4.4: Test R^2 scores of Parkinson's Monitoring dataset experiments under feature imbalance, averaged over normalization techniques used. Results are sorted according to R^2 scores.

under IID across varying client counts, feature imbalance reveals a mild decline in local normalization performance as client numbers grow. This suggests that smaller client datasets benefit more from global normalization parameters.

4.4 Key Takeaways

- **Federated normalization techniques outperform local ones** on average, especially under feature and label imbalance, aligning with our hypothesis. Local normalization, in contrast, fails to align the feature distributions across clients, leading to less performance and instability.
- On average, models **without normalization outperform those using local normalization**, suggesting that local normalization may amplify client-specific biases.

- **Feature imbalance** introduces significant performance challenges; only robust federated methods remain stable.
- **Quantity imbalance** does not significantly affect performance and yields results similar to IID scenarios.
- **Activation layer normalizations** perform comparably to federated normalization but are challenging to integrate into PPFL frameworks due to privacy constraints.

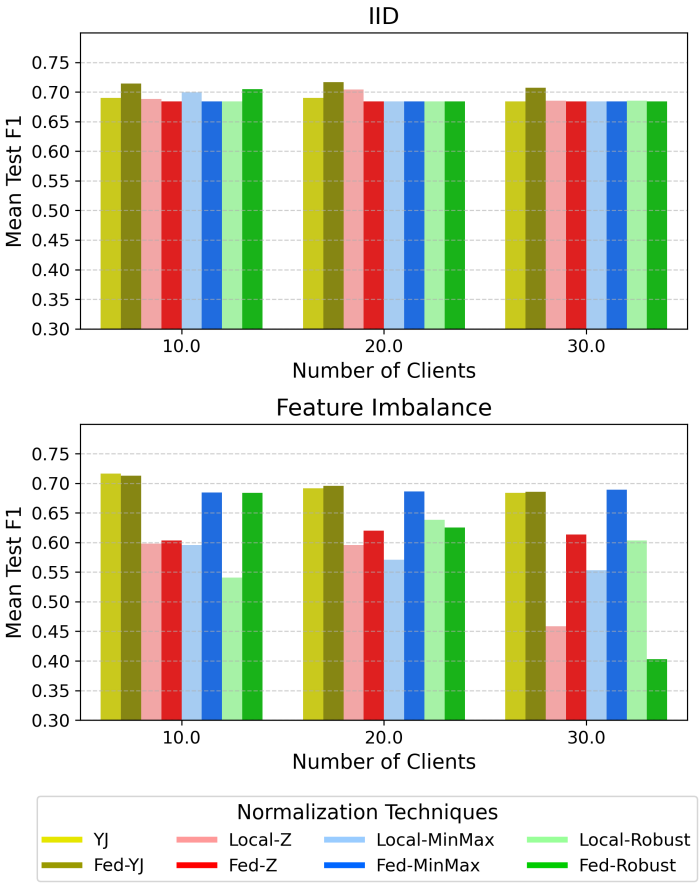

Figure 4.5: Test F1 scores for Hepatitis dataset for feature imbalance & IID comparison. In feature-imbalanced scenarios, increasing the number of clients amplifies the performance gap between local and federated normalization. Under IID settings, the results remain closely aligned regardless of the normalization method.

Chapter 5

Privacy Preserving Federated Normalization

Our study in Section 4.3 demonstrates that federated normalization outperforms other normalization techniques. However, sharing local parameters with the aggregation server, such as the minimum or maximum values of a client's dataset, poses a privacy risk, particularly in heterogeneous FL environments. To mitigate this leakage, we propose novel algorithms that enable federated normalization under MHE. Our primary goal is to enable the execution of previously mentioned federated normalization techniques *under encryption, ensuring that sensitive values remain encrypted throughout the process.*

We present privacy-preserving federated (PPF) protocols for z-score normalization, MinMax scaling, and robust scaling with MHE-CKKS. The computations for these protocols are executed on the FL aggregation server. Below, we describe the protocols for PPF normalization techniques.

Notations. In the algorithms presented in this section, $\bigoplus_{p=1}^{P}$ denotes the homomorphic summation of elements, which is the homomorphic counterpart of $\sum_{p=1}^{P}$. Similarly, $\odot$ represents the homomorphic multiplication. We use $\langle X \rangle$ to denote the encrypted version of a plaintext X. Other homomorphic operations

are self-explanatory based on their names. We define the core operations (e.g., encryption) in the Background section 3.3. The $\mathsf{Inv}(\langle m \rangle)$ operation represents the homomorphic inverse on encrypted message $\langle m \rangle$, which we rely on implementation in the Lattigo package [57]. This implementation in Lattigo relies on the Goldschmidt division algorithm [58, 59]. $\mathsf{Min}(\langle m_1 \rangle, \langle m_2 \rangle)$, $\mathsf{Max}(\langle m_1 \rangle, \langle m_2 \rangle)$ stands for homomorphic minimum and maximum comparison operations, which we rely on the minimax approximation on the Lattigo package [57]. The prefix C in operation names stands for Collective; for example, $\mathsf{CBootstrap}(\langle m_1 \rangle, \{sk_p\})$ refers to the Collective Bootstrapping operation. PPF Robust Scaling protocol uses PPF MinMax Scaling and PPF k-th Element Calculation protocols, which we present in Algorithm 2 and Algorithm 4 as $\mathsf{PPFMinMax()}$ and $\mathsf{PPF\text{-}Kth()}$, respectively.

5.1 PPF Z-Score Normalization

Federated z-score normalization requires computing global mean (μ_g) and variance (σ_g^2) per feature as if the data were pooled, without centralizing it. Our protocol reveals only the final outputs: the collaboratively computed mean (μ_g) and variance (σ_g^2). We summarize the protocol in Algorithm 1 and provide detailed steps below.

Parties begin by generating secret keys and jointly establishing a public encryption key (Line 1, Algorithm 1). Each party then computes the local summation S_p, and the number of samples n_p for each feature in their dataset (Lines 2–7), forming arrays of size N, where N is the total number of features. These arrays are encrypted using the collective public key—denoted as $\langle S_p \rangle$ (local sums) and $\langle n_p \rangle$ (local sample counts)—, with each value occupying a slot in the ciphertext, utilizing N slots per ciphertext (Line 8). The encrypted values are then sent to the server (Line 9), which aggregates them to compute $\langle S_g \rangle$ (global sums) and $\langle n_g \rangle$ (global sample counts), representing the feature-wise encrypted summation and sample count as if the data were centrally pooled (Lines 11–13).

Algorithm 1 PPF Z-Score Normalization Protocol

Initialization: $\{sk_p\}, pk \leftarrow \mathsf{KeyGen}(\lambda)$
Compute Local Summations (Client Side):
for each party p **in** $\{1, \ldots, P\}$ **do**
  for each feature j **in** $\{1, \ldots, N\}$ **do**
    $n_p[j] \leftarrow$ number of samples for feature j
    $S_p[j] \leftarrow \sum_{i=1}^{n_p[j]} D_p[j][i]$
  end for
  $\langle S_p \rangle, \langle n_p \rangle \leftarrow \mathsf{Encrypt}(S_p, pk), \mathsf{Encrypt}(n_p, pk)$
  Send $\langle S_p \rangle$ and $\langle n_p \rangle$ to the server
end for
Global Summations (Server side):
$\langle S_g \rangle \leftarrow \bigoplus_{p=1}^{P} \langle S_p \rangle$
$\langle n_g \rangle \leftarrow \bigoplus_{p=1}^{P} \langle n_p \rangle$
Compute Global Means (Server side):
$\langle n_g^{-1} \rangle \leftarrow \mathsf{Inv}(\langle n_g \rangle)$
$\langle n_g^{-1} \rangle \leftarrow \mathsf{CBootstrap}(\langle n_g^{-1} \rangle, \{sk_p\})$
$\langle \mu_g \rangle \leftarrow \langle S_g \rangle \odot \langle n_g^{-1} \rangle$
$\mu_g \leftarrow \mathsf{CDecrypt}(\langle \mu_g \rangle, \{sk_p\})$
Compute Local Summation of Squared Differences (Client Side):
for each party p **in** $\{1, \ldots, P\}$ **do**
  for each feature j **in** $\{1, \ldots, N\}$ **do**
    $S_{p,\mathrm{sq}}[j] \leftarrow \sum_{i=1}^{n_p[j]} (D_p[j][i] - \mu_g)^2$
  end for
  $\langle S_{p,\mathrm{sq}} \rangle \leftarrow \mathsf{Encrypt}(S_{p,\mathsf{sq}}, pk)$
  Send $S^*_{p,\mathrm{sq}}$ to the server
end for
Compute Global Variances (Server side):
$\langle S_{g,\mathrm{sq}} \rangle \leftarrow \bigoplus_{p=1}^{P} \langle S_{p,\mathrm{sq}} \rangle$
$\langle \sigma_g^2 \rangle \leftarrow \langle S_{g,\mathrm{sq}} \rangle \odot \langle n_g^{-1} \rangle$
Decryption and Local Normalization (Client Side):
$\sigma_g^2 \leftarrow \mathsf{CDecrypt}(\langle \sigma_g^2 \rangle, \{sk_p\})$
Parties apply z-score normalization locally using μ_g, σ_g^2.

Then, the server computes the inverse of $\langle n_g \rangle$ in the encrypted domain and parties apply collective bootstrapping to restore the level of the ciphertext (Lines 14–16), yielding $\langle n_g^{-1} \rangle$. The calculated inverse and $\langle S_g \rangle$ are then homomorphically multiplied to compute $\langle \mu_g \rangle$ (Line 17). Finally, $\langle \mu_g \rangle$ is collectively decrypted, making the global mean (μ_g) available to all parties (Line 18). Upon decryption, our protocol proceeds to calculate the global variances. Each party subtracts μ_g from their local feature values and squares the results (Lines 20–23). The sums of these squared differences for each feature are computed locally, resulting in arrays $S_{p,\text{sq}}$. These arrays are encrypted, denoted as $\langle S_{p,\text{sq}} \rangle$, and sent to the server (Lines 24–25). The server aggregates the encrypted values to compute $\langle S_{g,\text{sq}} \rangle$ (global squared difference sums) (Line 28). Subsequently, the server homomorphically multiplies $\langle S_{g,\text{sq}} \rangle$ with $\langle n_g^{-1} \rangle$ to compute $\langle \sigma_g^2 \rangle$ (Line 29). $\langle \sigma_g^2 \rangle$ is then collectively decrypted to obtain σ_g^2 (Line 31). Finally, each client receives μ_g and σ_g^2, which are then used locally to apply z-score normalization to their data (Line 32).

5.2 PPF MinMax Scaling

Federated MinMax scaling requires global feature-wise minima and maxima, as if the data were pooled. Our protocol reveals only the global minimum ($X_{\min,g}$) and maximum ($X_{\max,g}$) for each feature. To preserve privacy, the protocol uses homomorphic comparison functions, which operate on inputs within the range $[-1, 1]$. These functions compute element-wise minima or maxima across ciphertexts without revealing which party contributed the values.

This protocol also begins by key generation (Line 1 of Algorithm 2). Each party then computes local feature-wise minima and maxima ($X_{\min,p}$, $X_{\max,p}$) across their dataset (Lines 3–7), producing two arrays of length N, where N is the number of features. These values are encrypted using pk to obtain ciphertexts $\langle X_{\min,p} \rangle$ and $\langle X_{\max,p} \rangle$, where each feature's value is stored in a separate slot of a ciphertext (Lines 8–9). The encrypted values are then sent to the server (Line 10).

Algorithm 2 PPF MinMax Scaling Protocol (PPFMinMax())

1: **Initialization:** $\{sk_p\}, pk \leftarrow \mathsf{KeyGen}(\lambda)$
2: **Compute Local MinMax (Client Side):**
3: **for** each party p **in** $\{1, \ldots, P\}$ **do**
4: **for** each feature j **in** $\{1, \ldots, N\}$ **do**
5: $X_{\min,p}[j] \leftarrow \min(D_p[j])$
6: $X_{\max,p}[j] \leftarrow \max(D_p[j])$
7: **end for**
8: $\langle X_{\min,p} \rangle \leftarrow \mathsf{Encrypt}(X_{\min,p}, pk)$
9: $\langle X_{\max,p} \rangle \leftarrow \mathsf{Encrypt}(X_{\max,p}, pk)$
10: Send $\langle X_{\min,p} \rangle$ and $\langle X_{\max,p} \rangle$ to the server
11: **end for**
12: **Normalization Factor Selection:**
13: Parties agree on a normalization vector V_{abs}
14: **Global MinMax Computation (Server Side):**
15: Normalize received ciphertexts: $\langle X_{\min,p} \rangle \leftarrow \langle X_{\min,p} \rangle \odot V_{abs}$, $\langle X_{\max,p} \rangle \leftarrow \langle X_{\max,p} \rangle \odot V_{abs}$
16: Initialize $\langle X_{\min,g} \rangle$ and $\langle X_{\max,g} \rangle$ using the normalized values of the first party
17: **for** each remaining party p **in** $\{2, \ldots, P\}$ **do**
18: $\langle X_{\min,g} \rangle \leftarrow \mathsf{Min}(\langle X_{\min,g} \rangle, \langle X_{\min,p} \rangle)$
19: $\langle X_{\max,g} \rangle \leftarrow \mathsf{Max}(\langle X_{\max,g} \rangle, \langle X_{\max,p} \rangle)$
20: $\langle X_{\min,g} \rangle \leftarrow \mathsf{CBootstrap}(\langle X_{\min,g} \rangle, \{sk_p\})$
21: $\langle X_{\max,g} \rangle \leftarrow \mathsf{CBootstrap}(\langle X_{\max,g} \rangle, \{sk_p\})$
22: **end for**
23: Restore original scale: $\langle X_{\min,g} \rangle \leftarrow \langle X_{\min,g} \rangle \odot V_{abs}^{-1}$, $\langle X_{\max,g} \rangle \leftarrow \langle X_{\max,g} \rangle \odot V_{abs}^{-1}$
24: **Decryption and Local Normalization (Client Side):**
25: $X_{\min,g} \leftarrow \mathsf{CDecrypt}(\langle X_{\min,g} \rangle, \{sk_p\})$
26: $X_{\max,g} \leftarrow \mathsf{CDecrypt}(\langle X_{\max,g} \rangle, \{sk_p\})$
27: Parties apply MinMax scaling locally using $X_{\min,g}$,$X_{\max,g}$

Before computation, the parties collaboratively agree on a normalization vector V_{abs}, which contains estimated feature-wise absolute maximum values. (Line 13). This vector scales the minimum and maximum values into the range $[-1, 1]$. Assuming an absolute maximum is reasonable for most ML datasets, since feature

ranges—such as 'age' or 'weight'—are typically well-defined. This value serves only for temporary scaling during normalization and does not need to be exact, as the original scale is restored afterward.

Upon receiving the encrypted minimum and maximum values from all parties, the server scales them using V_{abs} to ensure they fall within the required range by utilizing homomorphic multiplication (Line 15). The server initializes global minimum ($\langle X_{\min,g} \rangle$) and global maximum ($\langle X_{\max,g} \rangle$) ciphertexts using the normalized values from the first party (Line 16). It then iteratively compares $\langle X_{\min,g} \rangle$ with each party's encrypted minimum values using homomorphic comparison and updates $\langle X_{\min,g} \rangle$. After each comparison, collective bootstrapping is used to refresh ciphertext levels. The same process is used to compute $\langle X_{\max,g} \rangle$ through homomorphic maximum comparison (Lines 17–22). Once the $\langle X_{\min,g} \rangle$ and $\langle X_{\max,g} \rangle$ values are determined, the server rescales them by multiplying with the inverse of the normalization vector, V_{abs}^{-1}, to restore the original range (Line 23). The final encrypted results are sent back to the parties, who decrypt them collaboratively to obtain $X_{\min,g}$ and $X_{\max,g}$ (Lines 25–26). Each party then applies MinMax scaling to its local dataset (Line 27).

5.3 PPF Robust Scaling

Applying PPF Robust Scaling requires computing the median, 25th percentile, and 75th percentile values of each feature as if the datasets of all parties were pooled. The core algorithm is a general-purpose method for computing the k-th ranked elements of features in FL while preserving privacy.

The PPF Robust Scaling protocol also begins with key generation (Line 1 of Algorithm 3). Next, each party computes the number of samples (n_p) for every feature in its local dataset (Lines 3–6) and encrypts these counts ($\langle n_p \rangle$) before sending them to the server (Lines 7–8). On the server side, these encrypted counts are aggregated (Lines 10–11) to obtain global sample counts ($\langle n_g \rangle$). The parties then collectively decrypt $\langle n_g \rangle$ to obtain the global sample counts n_g (Lines 12).

Using the global sample counts, the indices corresponding to the median, 25th percentile, and 75th percentile —stored as arrays K_{Q1}, K_{Q2}, and K_{Q3} along with boolean arrays $K_{b,Q1}$, $K_{b,Q2}$, and $K_{b,Q3}$ respectively— are calculated for each feature (Lines 13–17). Boolean arrays ($K_{b,Q*}$) indicate whether the percentile for feature j is at an exact index or requires interpolation (e.g., median of a length-10 array). The protocol then computes global feature-wise minima and maxima using PPF MinMax Scaling presented in Algorithm 2 (Lines 18-19).

Then, for each percentile (denoted as $Q*$ in {Q1, Q2, Q3}), the server calls the PPF k-th Element Calculation protocol (Algorithm 4). This algorithm performs a binary search over a floating-point range to find the value meeting the rank condition. The search range is bounded by global min-max values, and precision is controlled by a threshold ϵ to ensure termination. Once the target percentiles are identified, each party applies robust scaling locally using these values (Lines 25–26).

Below, we introduce our PPF k-th Ranked Element Calculation algorithm which we build upon the algorithm proposed in [27]. Our novel approach computes the k-th ranked element using homomorphic encryption based on the CKKS scheme, enabling privacy preservation while supporting floating-point inputs for the first time in this context. This contribution represents a significant advancement not only for federated normalization but also broadly for PPF protocols.

Algorithm 3 PPF Robust Scaling Protocol

Initialization: $\{sk_p\}, pk \leftarrow \mathsf{KeyGen}(\lambda)$
Compute Local Sample Counts (Client Side):
for each party p **in** $\{1, \ldots, P\}$ **do**
  for each feature j **in** $\{1, \ldots, N\}$ **do**
    $n_p[j] \leftarrow$ number of samples for feature j
  end for
  $\langle n_p \rangle \leftarrow \mathsf{Encrypt}(n_p, pk)$
  Send $\langle n_p \rangle$ to the server
end for
Global Sample Counts (Server Side):
$\langle n_g \rangle \leftarrow \bigoplus_{p=1}^{P} \langle n_p \rangle$
$n_g \leftarrow \mathsf{CDecrypt}(\langle n_g \rangle, \{sk_p\})$
Determine Percentile Indices (Server Side):
for each feature j in $\{1, \ldots, N\}$ **do**
  Compute the 25th, 50th, and 75th percentiles of $n_g[j]$ as $K_{Q1}[j]$, $K_{Q2}[j]$, $K_{Q3}[j]$
  For each $K_{Q*}[j]$, set $K_{b,Q*}[j] \leftarrow$ **true** if given index is exact; else set to **false**
end for
Compute Global Min-Max:
$X_{\min,g}, X_{\max,g} \leftarrow \mathsf{PPFMinMax}(D_1, D_2, \ldots, D_P)$
Compute Percentiles (Server Side):
for each percentile $Q*$ **in** {Q1, Q2, Q3} **do**
  Set $K \leftarrow K_{Q*}$, $K_b \leftarrow K_{b,Q*}$
  $Q*_g \leftarrow \mathsf{PPF\text{-}Kth}(X_{\min,g}, X_{\max,g}, K_q, K_{b,q}, n_g, \epsilon)$
end for
Local Normalization (Client Side):
Parties locally apply robust scaling using $Q1_g$, $Q2_g$, $Q3_g$

PPF k-th Ranked Element Calculation: Our proposed protocol begins by initializing an empty boolean array B of size N to track whether the k-th element for each feature has been found (Lines 1–2 of Algorithm 4). Then, for each feature, the lower ($\min_j$) and upper ($\max_j$) bounds of the search range are set using the global minimum and maximum values (Line 3).

Algorithm 4 PPF k-th Element Calculation (PPF-Kth())

Initialization:
Initialize $B[j] \leftarrow$ False $\forall j$, $Q_g[j] \leftarrow 0$ $\forall j$
$\min_j \leftarrow X_{\min,g}[j]$, $\max_j \leftarrow X_{\max,g}[j]$ $\forall j$
repeat
  Compute Midpoints (Server Side):
  for each feature j **in** $\{1, \ldots, N\}$ **do**
    $M[j] \leftarrow (\min_j + \max_j)/2$
  end for
  Send M to the parties.
  Count Smaller and Greater Values (Client Side):
  for each party p **in** $\{1, \ldots, P\}$ **do**
    for each feature j **in** $\{1, \ldots, N\}$ **do**
      $Lcount_p[j] \leftarrow \sum_{i=1}^{|D_p[j]|} \mathbf{1}(D_p[j][i] \leq M[j])$
      $Gcount_p[j] \leftarrow \sum_{i=1}^{|D_p[j]|} \mathbf{1}(D_p[j][i] > M[j])$
    end for
    $\langle Lcount_p \rangle \leftarrow \mathsf{Encrypt}(Lcount_p, pk)$
    $\langle Gcount_p \rangle \leftarrow \mathsf{Encrypt}(Gcount_p, pk)$
    Send $\langle Lcount_p \rangle$, $\langle Gcount_p \rangle$ to server.
  end for
  Aggregate Counts (Server Side):
  $\langle Lcount_g \rangle \leftarrow \bigoplus_{p=1}^{P} \langle Lcount_p \rangle$
  $\langle Gcount_g \rangle \leftarrow \bigoplus_{p=1}^{P} \langle Gcount_p \rangle$
  $Lcount_g \leftarrow \mathsf{CDecrypt}(\langle Lcount_g \rangle, \{sk_p\})$
  $Gcount_g \leftarrow \mathsf{CDecrypt}(\langle Gcount_g \rangle, \{sk_p\})$
  Update Bounds and Results (Server Side):
  for each feature j **in** $\{1, \ldots, N\}$ **do**
    if not $B[j]$ **then**
      if $K_b[j]$ **then**
        if $Lcount_g[j] <= K[j] - 1$ **and** $Gcount_g[j] <= n_g[j] - K[j]$ **then**
          $Q_g[j] \leftarrow M[j]$, $B[j] \leftarrow$ True
          continue
        end if
      else
        if $Lcount_g[j] <= K[j]$ **and** $Gcount_g[j] <= n_g[j] - K[j]$ **then**
          $Q_g[j] \leftarrow M[j]$, $B[j] \leftarrow$ True
          continue
        end if
      end if
      Adjust the search range:
      if $Lcount_g[j] \geq K[j]$ **then**
        $\max_j \leftarrow M[j]$
      else
        $\min_j \leftarrow M[j]$
      end if
      Check if reached precision limit:
      if $\max_j - \min_j \leq \epsilon$ **then**
        $Q_g[j] \leftarrow (\min_j + \max_j)/2$, $B[j] \leftarrow$ True
        continue
      end if
    end if
  end for
until $\forall j$, $B[j] =$ True

In each iteration (starting at Line 4), the server computes the midpoint $M[j]$ of the current search interval for each feature j (Lines 5–8). These midpoints are sent to the parties (Line 9), which locally count the number of data values smaller ($Lcount_p$) than or greater ($Gcount_p$) than the midpoint (Lines 10–15). Each party encrypts the resulting counts as $\langle Lcount_p \rangle$ and $\langle Gcount_p \rangle$ and sends to the server (Lines 16–18).

On the server side, the encrypted counts from all parties are aggregated and then collectively decrypted to obtain the global counts (Lines 20–24). Next, the server checks whether the current midpoint satisfies the rank condition by comparing the counts to the target index $K[j]$ (Lines 28–38). If the condition is met, the k-th element is recorded (Lines 30, 35) and the corresponding boolean flag is set to *true*.

If the condition is not met, the algorithm adjusts the search range (Lines 39–44): If $Lcount_g[j] \geq K[j]$, $\max_j$ is updated to $M[j]$; otherwise, $\min_j$ is updated to $M[j]$. Additionally, if the updated search range is smaller than the precision threshold ϵ, the algorithm finalizes the value as the average of the current $\min_j$ and $\max_j$ (Lines 45–49). This iterative process continues until all features have their k-th element computed (Line 52).

Chapter 6

Evaluation of PPF Normalization

We evaluate the proposed PPF normalization protocols through theoretical and empirical analyses, detailing complexity via dominant operations and benchmarking performance across various settings.

6.1 Theoretical Analysis

In this section, we present the theoretical worst-case time and communication complexity analysis of the proposed protocols. For clarity and ease of interpretation, we express time complexity in terms of the time required for each dominant operation. Specifically, we denote the execution time of each fundamental operation by $T_{operation}$ —for example, $T_{encrypt}$ represents the encryption time. Later, we provide microbenchmark measurements of these operations to facilitate interpretation and extrapolation in the Empirical Analysis section 6.2 under settings with varying number of parties.

Note that the PPF Robust Scaling protocol incorporates the PPF MinMax Scaling and the PPF k-th Ranked Element Calculation protocols. We denote the time complexities of these protocols as T_γ and T_β, respectively, and use these to represent the overall time complexity of the PPF Robust Scaling protocol. The

Protocol	Time Complexity	Communication Complexity
PPF Z-Score Normalization	$3PT_{sum} + T_{cbootstrap} + T_{inv} + 2T_{mul}$	$3P\|c\| + 3S_{cdecrypt} + K_{inv}S_{cbootstrap}$
PPF MinMax Scaling (γ)	$4PT_{mul} + PT_{min} + PT_{max} + 2PT_{cbootstrap}$	$2P\|c\| + 2S_{cdecrypt} + 2PK_{mm}S_{cbootstrap}$
PPF Robust Scaling	$PT_{sum} + T_{\gamma} + 3T_{\beta}$	$P\|c\| + S_{cdecrypt} + S_{\gamma} + 3S_{\beta}$
PPF k-th Ranked Element Calculation (β)	$\mathcal{X} \times (2PT_{sum} + 2T_{cdecrypt} + 2T_{encrypt})$	$\mathcal{X} \times (2P\|c\| + 2S_{cdecrypt} + P\|p\|)$

Table 6.1: Theoretical Time and Communication Complexities of PPF Protocols.

parameter $\mathcal{X}$ appearing in both the time and communication complexity analysis for the PPF k-th Ranked Element Calculation protocol in Table 6.1 is defines as:

$$\mathcal{X} = \log\left(\frac{\arg\max_j(max_j - min_j)}{\epsilon}\right),$$

where $\arg\max_j(max_j - min_j)$ denotes the search range for the feature j that has the maximum search range for the k-th element calculation protocol. The maximum search range among features determines the worst-case time and communication complexity, as the protocol must wait until computations for all features are completed in parallel.

For the theoretical time complexity representations, we consider only the dominant terms, i.e., the time-consuming repetitive homomorphic operations. One-time operations, such as key generation, encryption, or client-side plaintext operations, are omitted as their impact is negligible.

We present the communication complexities in Table 6.1 in terms of total bits transferred. In these representations, P denotes the number of parties, while $|c|$ and $|p|$ denote the size of a ciphertext and a plaintext, respectively. Additional variables, denoted by the letter S, are used to simplify the presentation. Specifically, S_{γ} and S_{β} denote the communication complexities of the PPF MinMax Scaling and the PPF k-th Ranked Element Calculation protocols.

Furthermore, $S_{cdecrypt}$ and $S_{cbootstrap}$ represent the communication complexities of the collective decryption and collective bootstrapping operations, respectively, both with a predefined worst-case complexity of $S_{cdecrypt} = S_{cbootstrap} = (P-1)|c|$.

The collective bootstrapping operation is the dominant factor in the overall communication complexities, as it is executed several times and requires the participation of all parties. Since the homomorphic min, max, and inverse operations incorporate the collective bootstrapping operation, we define the terms $\mathcal{K}_{inv}$ and $\mathcal{K}_{mm}$ as multipliers for $S_{cbootstrap}$ in Table 6.1.

The PPF Z-Score and PPF MinMax protocols rely on homomorphic inverse, minimum, and maximum operations, and the communication costs associated with these operations depend on $\mathcal{K}_{inv}$ and $\mathcal{K}_{mm}$. The values of $\mathcal{K}_{inv}$ and $\mathcal{K}_{mm}$ are determined by the chosen security parameters and, for $\mathcal{K}_{inv}$, also by the acceptable input value range of homomorphic inverse operation. For example, in a 10-party setting with a security parameter of $\log N = 15$, we have $\mathcal{K}_{mm} = 1$ for the homomorphic minimum and maximum operations, and $\mathcal{K}_{inv} = 13$ for the homomorphic inverse operation where maximum absolute input value is $|2^{20}|$ (i.e., it can take inputs in the range -2^{20} to 2^{20}). Our analysis shows that PPF MinMax scaling is the most time- and communication-intensive, due to bootstrapping operations that scale with the number of parties. This, in turn, increases the overall cost of PPF Robust Scaling.

6.2 Empirical Analysis

6.2.1 Implementation Details

We evaluate the empirical runtime on a system with a 12$^{\text{th}}$ Gen Intel® Core™ i5-12400F processor (2.50 GHz base frequency, 6 physical cores, 12 logical threads) and 32 GB of DDR4 RAM operating at 3600 MT/s. Unless otherwise specified, the security parameters for the CKKS scheme are set to $\lambda = 128$-bit security with a ring size of $\log N = 15$, and the maximum absolute input value for the homomorphic inverse operation is set to $|2^{30}|$.

6.2.2 Microbenchmarks

Table 6.2 presents benchmark results for micro-level homomorphic operations with 10 clients. We also present the same benchmarks for 20, 30, and 50 party settings in Appendix A, Tables A.1, A.2, and A.3. Tables categorize operations by protocol, with common steps such as encryption and collective decryption listed at the top.

Operation Name	**Time (ms)**
Common Operations	
Secret Key Generation (T_{skgen})	9.67
Collective Public Key Generation (T_{pkgen})	138.71
Collective Relinearization Key Generation ($T_{relkgen}$)	5507.03
Encryption ($T_{encrypt}$)	59.28
Collective Decryption ($T_{decrypt}$)	536.63
PPF Z-Score Normalization	
Homomorphic Multiplication (T_{mul})	173.77
Homomorphic Summation (T_{sum})	1.73
Homomorphic Inverse (T_{inv})	30135.78
Collective Bootstrapping ($T_{bootstrap}$)	825.34
PPF MinMax Scaling	
Collective Galois Keys generation (T_{gkgen})	1908.93
Homomorphic Multiplication (T_{mul})	173.77
Homomorphic Inverse (T_{inv})	30135.78
Homomorphic Minimum Comparison (T_{min})	12255.31
Homomorphic Maximum Comparison (T_{max})	12790.85
Collective Bootstrapping ($T_{bootstrap}$)	825.34
PPF Robust Scaling	
Homomorphic Summation (T_{sum})	1.73

Table 6.2: Microbenchmarks with 10 clients.

Some operations—namely, T_{skgen}, $T_{encrypt}$, T_{sum}, and T_{mul}—are independent of the number of parties, as they do not require any collaboration. Therefore,

Protocol	Time (sec)
PPF Z-Score Normalization	44
PPF MinMax Scaling	253
PPF Robust Scaling	368

Table 6.3: Total Runtimes of PPF Normalizations for 10 Parties. Precision value (ϵ) for PPF Robust Scaling is taken as $1e^{-4}$.

the empirical runtimes of these operations remain constant across different party settings. In contrast, other operations involve inter-party collaboration and scale linearly with the number of parties (see Appendix A). Reported runtimes exclude communication overhead, which varies with bandwidth, party location, and network latency.

From all microbenchmark tables, we observe that certain operations dominate the computational cost for specific protocols—for instance, T_{inv}, T_{min}, and T_{max}. These operations are inherently dependent on $T_{bootstrap}$, as discussed in Section 6.1. Additionally, the empirical benchmark values provided in microbenchmark tables, in combination with the theoretical analysis in Table 6.1, can be used to estimate the runtime for new use cases under different settings. The following section demonstrates an example experimental total runtime for a 10-party setup.

6.2.3 Total Runtime

The computational runtimes of the proposed protocols for a 10-party setting are presented in Table 6.3. It is important to note that these runtimes do not account for communication overhead or client-side plaintext operations—such as calculating the mean of a local dataset—as these are negligible compared to computationally intensive ciphertext operations. Since the protocols are executed only once during the data preprocessing stage for PPML and require only a few minutes, as indicated in Table 6.3, they are practical and readily applicable to real-world FL scenarios.

We further investigate the impact of the precision value (ϵ) on the PPF k-th

Norm. Type	Parameters	Large Valued	Small Valued
PPF Z-Score	Mean	1.80×10^{-5}	2.32×10^{-6}
	Variance	1.80×10^{-5}	2.35×10^{-5}
PPF MinMax	Min	2.92×10^{-9}	1.78×10^{-4}
	Max	1.27×10^{-8}	4.64×10^{-4}
PPF Robust	Median ($\epsilon=10^{-6}$)	1.45×10^{-9}	3.69×10^{-7}
	Median ($\epsilon=10^{-3}$)	9.13×10^{-7}	3.07×10^{-4}

Table 6.4: Precision loss of PPF normalization techniques. **Large Valued** indicates datasets with high-magnitude floats, while **Small Valued** refers to values near zero.

element algorithm (see Appendix A, Figure A.7). We observe a linear increase in runtime as ϵ decreases exponentially, which supports the theoretical analysis in Table 6.1. Additionally, we observe a linear relationship between the number of clients and the runtime, indicating that the protocol scales well and remains practical for large deployments, including cross-device FL.

6.2.4 Precision

While CKKS encryption allows us to securely compute normalization parameters, it introduces a slight precision loss due to approximate encoding and noise accumulation inherent in fixed-point arithmetic on encrypted data. Table 6.4 summarizes the precision loss for each privacy-preserving normalization technique compared to computations on plaintext data. We evaluate using a 10-party scenario and consider two cases: one involving small floating-point values (close to zero), and another involving large floating-point values. We observe that as the values move away from zero (i.e., become larger), precision loss decreases. For example, for PPF MinMax Scaling, in a small-valued (all values close to 0) dataset, we can calculate the minimum value of the dataset with only a 1.73×10^{-4} relative error, while for a larger-valued dataset, the relative error even decreases to 2.915×10^{-9}. For the median calculation in PPF Robust Scaling, precision depends on the ϵ value defined by the parties. We observe a direct correlation between the defined ϵ value and the precision loss, as shown in Table 6.4.

Chapter 7

Conclusion

In this thesis, we addressed the critical challenge of data normalization in federated learning, particularly under non-IID data settings. We proposed and evaluated federated normalization techniques that simulate pooled normalization, demonstrating their superiority over traditional local normalization methods. To further enhance privacy, we introduced novel protocols leveraging MHE for z-score normalization, MinMax scaling, and robust scaling. Notably, we also introduced the PPF k-th Element Calculation protocol, a general-purpose, privacy-preserving primitive that enables robust scaling and has potential applications beyond normalization. Collectively, our contributions advance the field of privacy-preserving federated learning by providing effective, secure, and practical normalization strategies suitable for real-world deployments.

Bibliography

[1] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *CoRR*, vol. abs:1610.02527, 2016.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*, pp. 1273–1282, PMLR, 2017.

[3] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the GAN: Information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, (New York, NY, USA), p. 603–618, Association for Computing Machinery, 2017.

[4] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 2512–2520, 2019.

[5] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE Symposium on Security and Privacy (SP)*, pp. 691–706, 2019.

[6] Z. Luan, W. Li, M. Liu, and B. Chen, "Robust federated learning: Maximum correntropy aggregation against byzantine attacks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 1, pp. 62–75, 2025.

[7] L. Lyu, H. Yu, X. Ma, C. Chen, L. Sun, J. Zhao, Q. Yang, and P. S. Yu, "Privacy and robustness in federated learning: Attacks and defenses," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 35, no. 7, pp. 8726–8746, 2024.

[8] S. Truex, L. Liu, K.-H. Chow, M. E. Gursoy, and W. Wei, "Ldp-fed: Federated learning with local differential privacy," in *Proceedings of the third ACM International Workshop on Edge Systems, Analytics and Networking*, pp. 61–66, 2020.

[9] X. Wu, Y. Zhang, M. Shi, P. Li, R. Li, and N. N. Xiong, "An adaptive federated learning scheme with differential privacy preserving," *Future Generation Computer Systems*, vol. 127, pp. 362–372, 2022.

[10] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," *CoRR*, vol. abs/1710.06963, 2018.

[11] K. Wei, J. Li, M. Ding, C. Ma, H. H. Yang, F. Farokhi, S. Jin, T. Q. S. Quek, and H. V. Poor, "Federated learning with differential privacy: Algorithms and performance analysis," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3454–3469, 2020.

[12] N. Wu, F. Farokhi, D. Smith, and M. A. Kaafar, "The value of collaboration in convex machine learning with differential privacy," *CoRR*, vol. abs/1906.09679, 2019.

[13] G. Xu, X. Han, S. Xu, T. Zhang, H. Li, X. Huang, and R. H. Deng, "Hercules: Boosting the performance of privacy-preserving federated learning," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 5, pp. 4418–4433, 2023.

[14] H. Tian, C. Zeng, Z. Ren, D. Chai, J. Zhang, K. Chen, and Q. Yang, "Sphinx: Enabling privacy-preserving online learning over the cloud," in *2022 IEEE Symposium on Security and Privacy*, pp. 2487–2501, 2022.

[15] D. Froelicher, J. R. Troncoso-Pastoriza, A. Pyrgelis, S. Sav, J. S. Sousa, J.-P. Bossuat, and J.-P. Hubaux, "Scalable privacy-preserving distributed learning," *PETS*, 2021.

[16] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux, "Poseidon: Privacy-preserving federated neural network learning," in *NDSS*, 2021.

[17] S. Sav, A. Diaa, A. Pyrgelis, J.-P. Bossuat, and J.-P. Hubaux, "Privacy-preserving federated recurrent neural networks," *PoPETs*, no. 4, pp. 500–521, 2021.

[18] B. Bozdemir, B. A. Özdemir, and M. Önen, "Prida: Privacy-preserving data aggregation with multiple data customers," in *ICT Systems Security and Privacy Protection* (N. Pitropakis, S. Katsikas, S. Furnell, and K. Markantonakis, eds.), pp. 46–60, Springer Nature Switzerland, 2024.

[19] X. Yang, Z. Liu, X. Tang, R. Lu, and B. Liu, "An efficient and multi-private key secure aggregation scheme for federated learning," *IEEE Transactions on Services Computing*, vol. 17, no. 5, pp. 1998–2011, 2024.

[20] W. Zheng, R. A. Popa, J. E. Gonzalez, and I. Stoica, "Helen: Maliciously secure coopetitive learning for linear models," in *IEEE S&P*, 2019.

[21] Y. Li, Y. Zhou, A. Jolfaei, D. Yu, G. Xu, and X. Zheng, "Privacy-preserving federated learning framework based on chained secure multiparty computing," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6178–6186, 2021.

[22] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "FALCON: Honest-majority maliciously secure framework for private deep learning," *PETS*, 2020.

[23] R. Kanagavelu, Z. Li, J. Samsudin, Y. Yang, F. Yang, R. S. Mong Goh, M. Cheah, P. Wiwatphonthana, K. Akkarajitsakul, and S. Wang, "Two-phase multi-party computation enabled privacy-preserving federated learning," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pp. 410–419, 2020.

[24] L. Chen, D. Xiao, Z. Yu, and M. Zhang, "Secure and efficient federated learning via novel multi-party computation and compressed sensing," *Information Sciences*, vol. 667, p. 120481, 2024.

[25] N. Jawalkar, K. Gupta, A. Basu, N. Chandran, D. Gupta, and R. Sharma, "Orca: Fss-based secure training and inference with gpus," in *2024 IEEE Symposium on Security and Privacy*, pp. 597–616, 2024.

[26] T. Marchand, B. Muzellec, C. Beguier, J. O. d. Terrail, and M. Andreux, "Securefedyj: a safe feature gaussianization protocol for federated learning," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, (Red Hook, NY, USA), Curran Associates Inc., 2024.

[27] G. Aggarwal, N. Mishra, and B. Pinkas, "Secure computation of the median (and other elements of specified ranks)," *J. Cryptology*, vol. 23, pp. 373–401, 2010.

[28] F. Zhang, D. Kreuter, Y. Chen, S. Dittmer, S. Tull, T. Shadbahr, M. Schut, F. Asselbergs, S. Kar, S. Sivapalaratnam, S. Williams, M. Koh, Y. Henskens, B. de Wit, U. D'Alessandro, B. Bah, O. Secka, P. Nachev, R. Gupta, S. Trompeter, N. Boeckx, C. van Laer, G. A. Awandare, K. Sarpong, L. Amenga-Etego, M. Leers, M. Huijskens, S. McDermott, W. H. Ouwehand, J. Rudd, C.-B. Schonlieb, N. Gleadall, M. Roberts, J. Preller, J. H. Rudd, J. A. Aston, C.-B. Schonlieb, N. Gleadall, and M. Roberts, "Recent methodological advances in federated learning for healthcare," *Patterns*, vol. 5, no. 6, p. 101006, 2024.

[29] R. Kerkouche, G. Ács, C. Castelluccia, and P. Genevès, "Privacy-preserving and bandwidth-efficient federated learning: an application to in-hospital mortality prediction," in *Proceedings of the Conference on Health, Inference, and Learning*, CHIL '21, (New York, NY, USA), p. 25–35, Association for Computing Machinery, 2021.

[30] B. Camajori Tedeschini, S. Savazzi, R. Stoklasa, L. Barbieri, I. Stathopoulos, M. Nicoli, and L. Serio, "Decentralized federated learning for healthcare networks: A case study on tumor segmentation," *IEEE Access*, vol. 10, pp. 8693–8708, 2022.

[31] S. Han, H. Ding, S. Zhao, S. Ren, Z. Wang, J. Lin, and S. Zhou, "Practical and robust federated learning with highly scalable regression training," *IEEE*

Transactions on Neural Networks and Learning Systems, vol. 35, no. 10, pp. 13801–13815, 2024.

[32] Z. Lian, Q. Yang, W. Wang, Q. Zeng, M. Alazab, H. Zhao, and C. Su, "Deepfel: Decentralized, efficient and privacy-enhanced federated edge learning for healthcare cyber physical systems," *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 5, pp. 3558–3569, 2022.

[33] J. H. Yoo, H. M. Son, H. Jeong, E.-H. Jang, A. Y. Kim, H. Y. Yu, H. J. Jeon, and T.-M. Chung, "Personalized federated learning with clustering: Non-iid heart rate variability data application," in *2021 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1046–1051, 2021.

[34] S. M. Shah and V. K. N. Lau, "Model compression for communication efficient federated learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 9, pp. 5937–5951, 2023.

[35] Z. Du, J. Sun, A. Li, P.-Y. Chen, J. Zhang, H. H. Li, and Y. Chen, "Rethinking normalization methods in federated learning," in *Proceedings of the 3rd International Workshop on Distributed Machine Learning*, DistributedML '22, (New York, NY, USA), p. 16–22, Association for Computing Machinery, 2022.

[36] Y. Wang, Q. Shi, and T.-H. Chang, "Why batch normalization damage federated learning on non-iid data?," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 1, pp. 1692–1706, 2025.

[37] B. Casella, R. Esposito, A. Sciarappa, C. Cavazzoni, and M. Aldinucci, "Experimenting with normalization layers in federated learning on non-iid scenarios," *CoRR*, vol. abs/2303.10630, 2023.

[38] C. Goelz, S. Vieluf, and H. Ballhausen, "A secure median implementation for the federated secure computing architecture," *Applied Sciences*, 2024.

[39] Q. Li, Y. Diao, Q. Chen, and B. He, "Federated learning on non-iid data silos: An experimental study," pp. 965–978, 05 2022.

[40] I.-K. Yeo and R. A. Johnson, "A new family of power transformations to improve normality or symmetry," *Biometrika*, vol. 87, no. 4, pp. 954–959, 2000.

[41] G. E. P. Box and D. R. Cox, "An analysis of transformations," *Journal of the royal statistical society series b-methodological*, vol. 26, pp. 211–243, 1964.

[42] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016.

[43] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 448–456, PMLR, 07–09 Jul 2015.

[44] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *CoRR*, vol. abs/1607.08022, 2016.

[45] Y. Wu and K. He, "Group normalization," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 3–19, 2018.

[46] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, (New York, NY, USA), p. 169–178, Association for Computing Machinery, 2009.

[47] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *ASIACRYPT*, 2017.

[48] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux, "Multiparty homomorphic encryption from ring-learning-with-errors," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 4, pp. 291–311, 2021.

[49] J.-P. Bossuat, C. Mouchet, J. Troncoso-Pastoriza, and J.-P. Hubaux, "Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys," in *EUROCRYPT*, pp. 587–617, 2021.

[50] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, H. L. Kwing, T. Parcollet, P. P. d. Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework," *CoRR*, vol. abs/2007.14390, 2020.

[51] A. Krizhevsky, "Learning multiple layers of features from tiny images," *Technical Report, University of Toronto*, 2012.

[52] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[53] W. H. Wolberg, W. N. Street, and O. L. Mangasarian, "Breast cancer wisconsin (diagnostic) data set," 1995.

[54] G. Ayana, P. Smets, S. Destercke, and J. Ma, "Hcv data set," 2019.

[55] A. Tsanas, M. A. Little, P. E. McSharry, J. Spielman, and L. O. Ramig, "Parkinson's telemonitoring data set," 2009.

[56] A. Ibarrondo and M. Önen, "Fhe-compatible batch normalization for privacy preserving deep learning," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology: ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings 13*, pp. 389–404, Springer, 2018.

[57] "Lattigo v6." Online: https://github.com/tuneinsight/lattigo, Aug. 2024. EPFL-LDS, Tune Insight SA.

[58] J. H. Cheon, W. Kim, and J. H. Park, "Efficient homomorphic evaluation on large intervals." Cryptology ePrint Archive, Paper 2022/280, 2022.

[59] R. Goldschmidt, "Applications of division by convergence," 08 2005.

Appendix A

Supplementary Figures and Tables

In this section, we provide the supplementary figures and tables for our experiments.

Figure A.1 presents the test F1 scores of federated and local normalizations for less heterogeneous scenarios on 30 clients (50 clients for image datasets). Figure A.2, A.3, and A.4 present the performance of federated and local normalizations for Hepatitis, Parkinson's, and BCW datasets under different heterogeneous FL scenarios, respectively. Figure A.5 and Figure A.6 present the impact of the number of clients on federated and local normalizations for MNIST, and BCW datasets. These findings are discussed in Section 4.3 in the main manuscript.

Additionally, Tables A.1, A.2, and A.3 present the PPF microbenchmarks with 20, 30, and 50 client settings, respectively. Figure A.7 illustrates the empirical impact of the precision value (ϵ) on the PPF k-th element algorithm. These tables and figures facilitate the findings and discussion in Section 6.2 of the main manuscript.

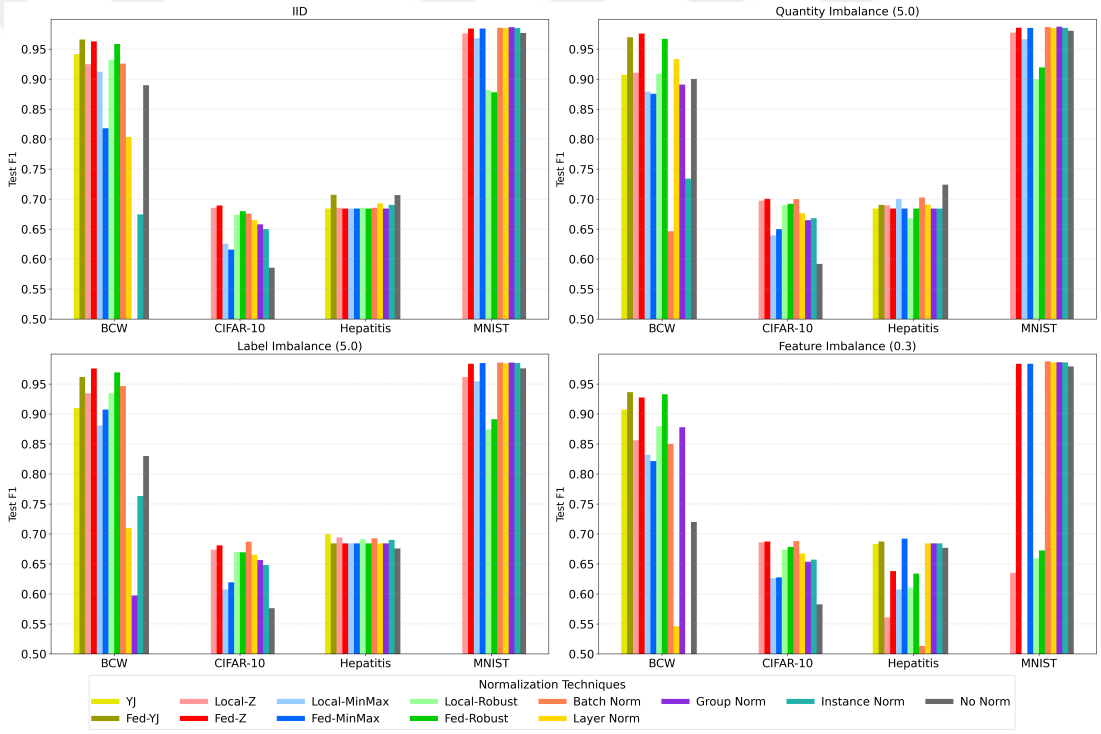

Figure A.1: Test F1 scores of different imbalance settings for 30 clients (50 for image datasets). Less imbalanced (less heterogeneous) parameters are selected for each imbalance type. Only classification datasets are included due to representation with F1 score metric.

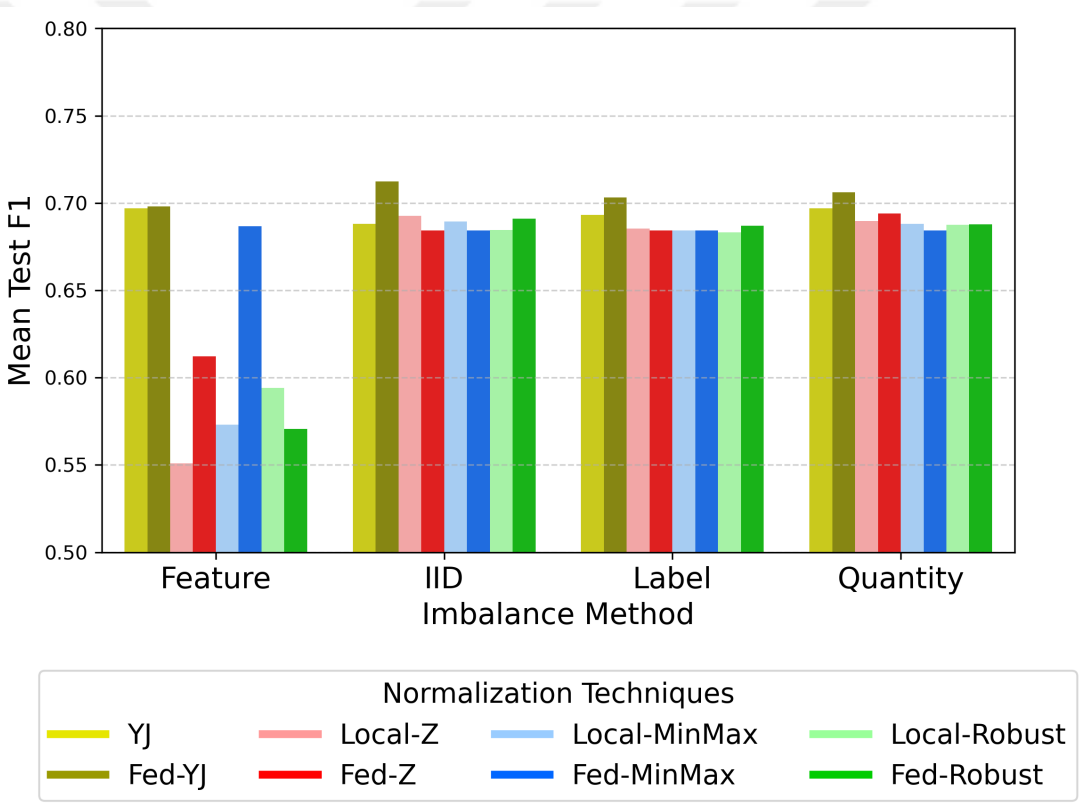

Figure A.2: Test F1 results of Hepatitis dataset averaged over client numbers to observe the impact of different heterogeneous scenarios.

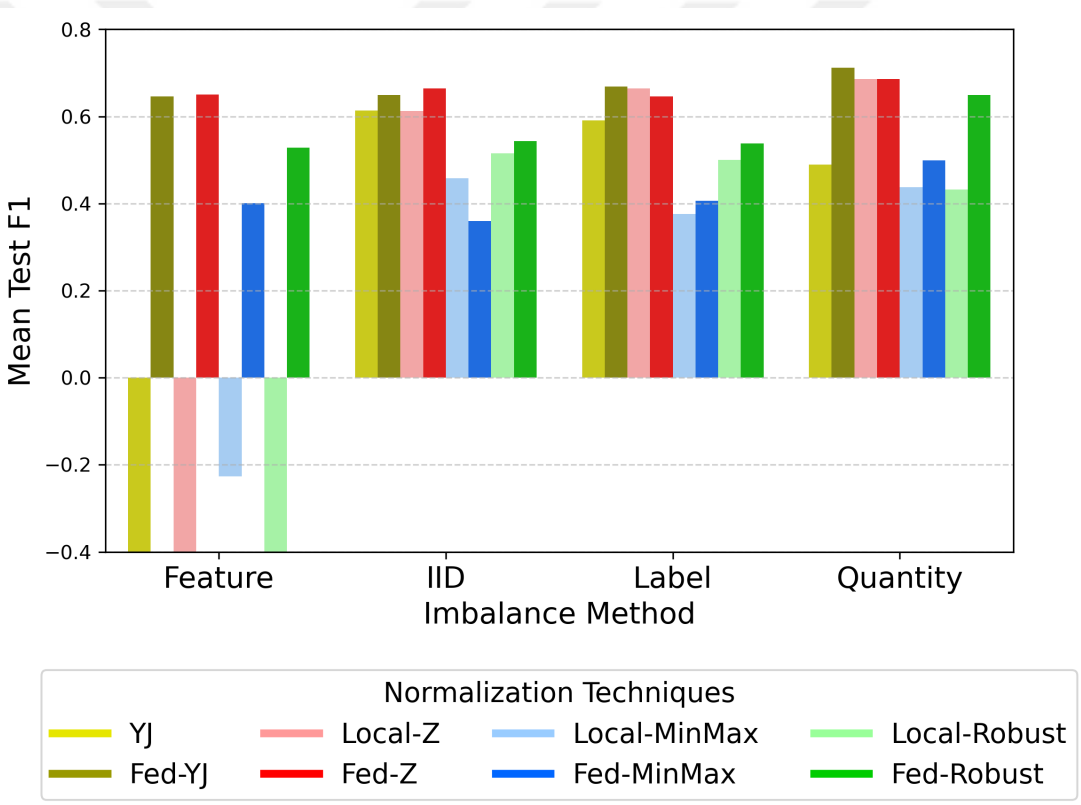

Figure A.3: Test R^2 results of Parkinson's dataset averaged over client numbers to observe the impact of different heterogeneous scenarios.

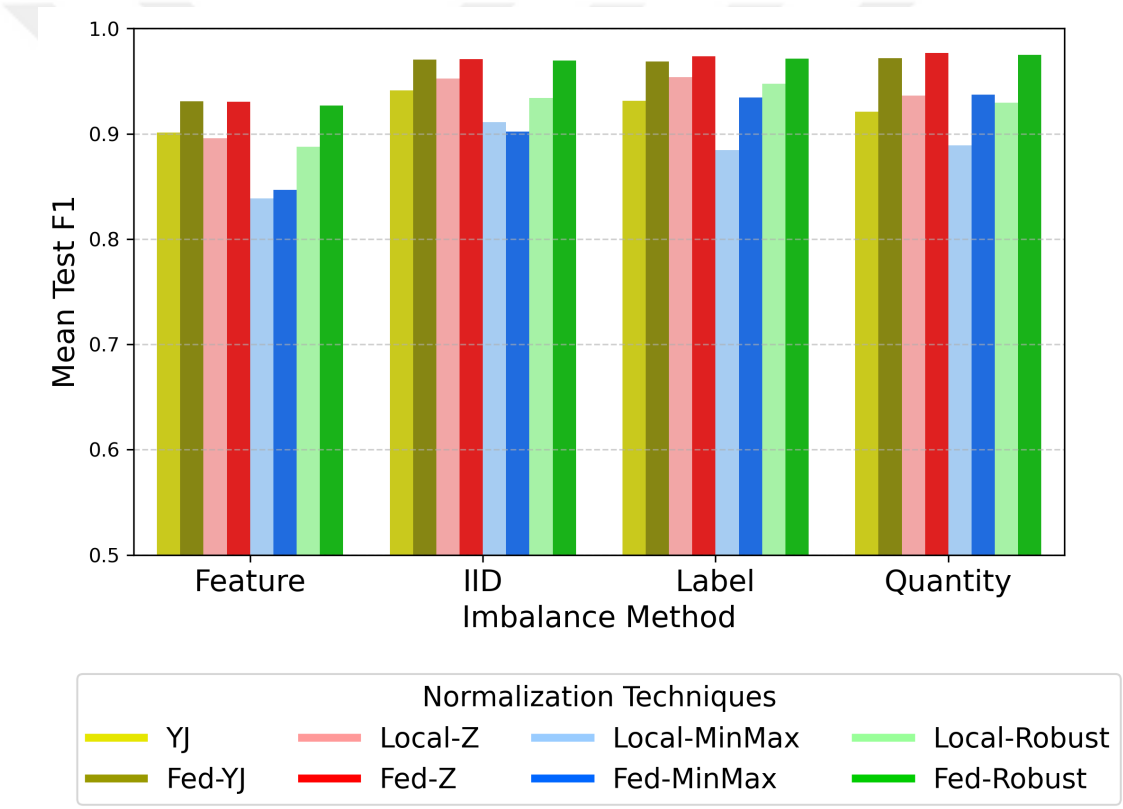

Figure A.4: Test F1 results of BCW dataset averaged over client numbers to observe the impact of different heterogeneous scenarios.

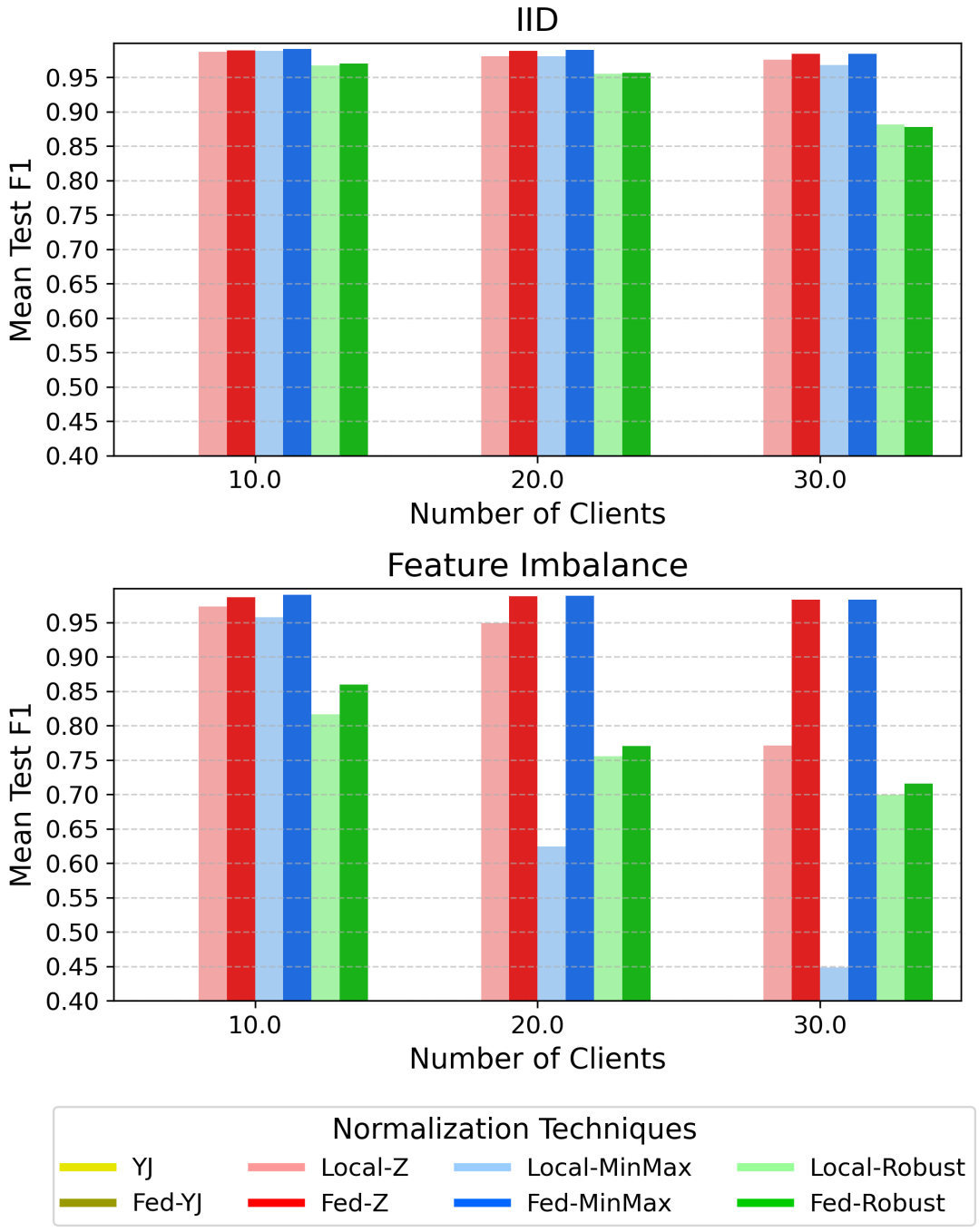

Figure A.5: Test F1 scores for MNIST dataset for feature imbalance & IID comparison for observing the impact of number of clients.

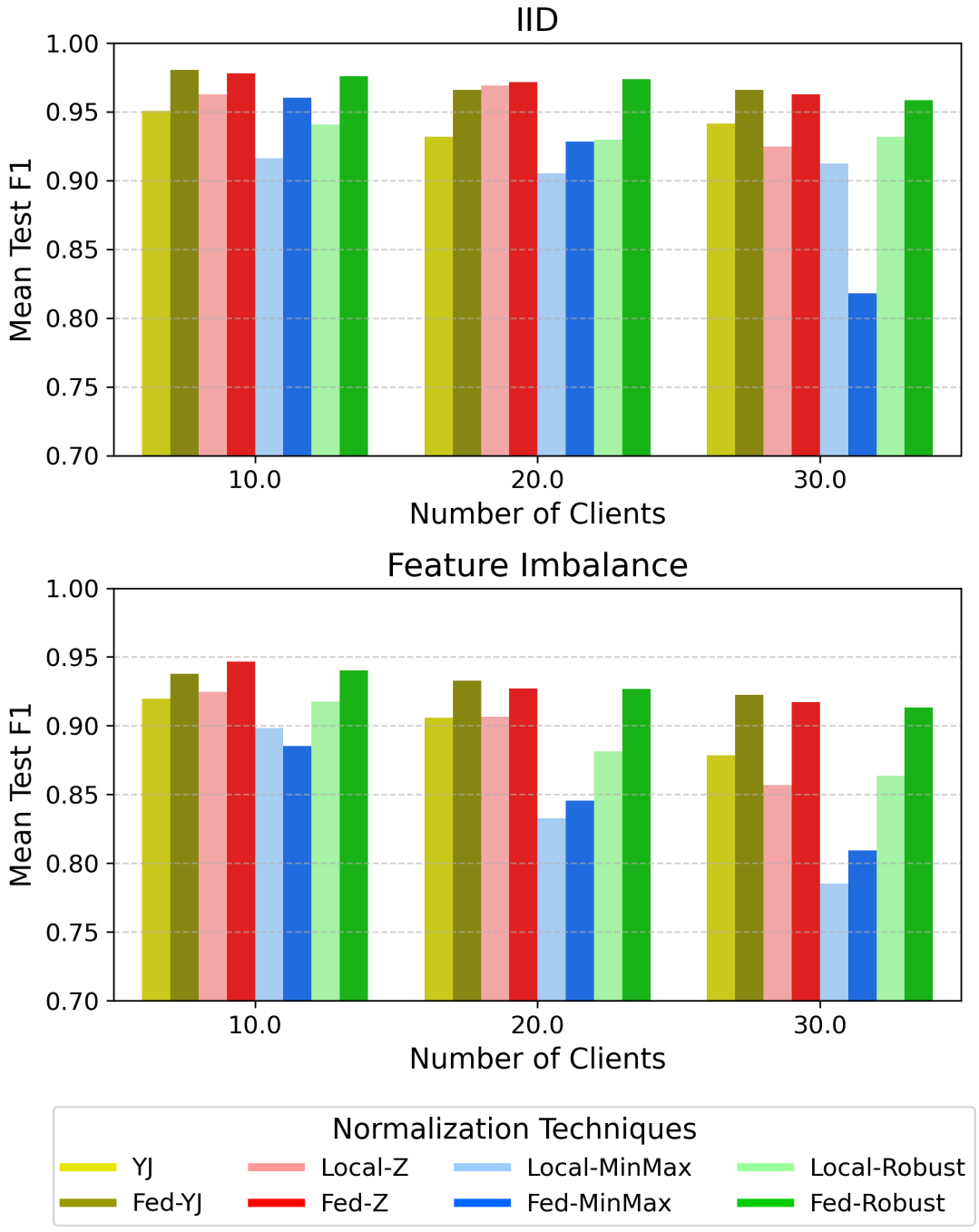

Figure A.6: Test F1 scores for BCW dataset for feature imbalance & IID comparison for observing the impact of number of clients.

Operation Name	Time (ms)
Common Operations	
Secret Key Generation (T_{skgen})	9.67
Collective Public Key Generation (T_{pkgen})	233.04
Collective Relinearization Key Generation ($T_{relkgen}$)	10052.85
Encryption ($T_{encrypt}$)	59.28
Collective Decryption ($T_{decrypt}$)	949.49
PPF Z-Score Normalization	
Homomorphic Multiplication (T_{mul})	173.77
Homomorphic Summation (T_{sum})	1.73
Homomorphic Inverse (T_{inv})	43269.14
Collective Bootstrapping ($T_{bootstrap}$)	1423.71
PPF MinMax Scaling	
Collective Galois Keys generation (T_{gkgen})	3407.75
Homomorphic Multiplication (T_{mul})	173.77
Homomorphic Inverse (T_{inv})	43269.14
Homomorphic Minimum Comparison (T_{min})	15377.25
Homomorphic Maximum Comparison (T_{max})	15486.16
Collective Bootstrapping ($T_{bootstrap}$)	1423.71
PPF Robust Scaling	
Homomorphic Summation (T_{sum})	1.73

Table A.1: Microbenchmarks with 20 clients.

Operation Name	Time (ms)
Common Operations	
Secret Key Generation (T_{skgen})	9.67
Collective Public Key Generation (T_{pkgen})	342.94
Collective Relinearization Key Generation ($T_{relkgen}$)	13703.95
Encryption ($T_{encrypt}$)	59.28
Collective Decryption ($T_{decrypt}$)	1357.74
PPF Z-Score Normalization	
Homomorphic Multiplication (T_{mul})	173.77
Homomorphic Summation (T_{sum})	1.73
Homomorphic Inverse (T_{inv})	56877.70
Collective Bootstrapping ($T_{bootstrap}$)	2051.37
PPF MinMax Scaling	
Collective Galois Keys generation (T_{gkgen})	5244.26
Homomorphic Multiplication (T_{mul})	173.77
Homomorphic Inverse (T_{inv})	56877.70
Homomorphic Minimum Comparison (T_{min})	18146.61
Homomorphic Maximum Comparison (T_{max})	18914.46
Collective Bootstrapping ($T_{bootstrap}$)	2051.37
PPF Robust Scaling	
Homomorphic Summation (T_{sum})	1.73

Table A.2: Microbenchmarks with 30 clients.

Operation Name	Time (ms)
Common Operations	
Secret Key Generation (T_{skgen})	9.67
Collective Public Key Generation (T_{pkgen})	567.72
Collective Relinearization Key Generation ($T_{relkgen}$)	22486.37
Encryption ($T_{encrypt}$)	59.28
Collective Decryption ($T_{decrypt}$)	2224.26
PPF Z-Score Normalization	
Homomorphic Multiplication (T_{mul})	173.77
Homomorphic Summation (T_{sum})	1.73
Homomorphic Inverse (T_{inv})	86618.77
Collective Bootstrapping ($T_{bootstrap}$)	3263.24
PPF MinMax Scaling	
Collective Galois Keys generation (T_{gkgen})	8133.07
Homomorphic Multiplication (T_{mul})	173.77
Homomorphic Inverse (T_{inv})	86618.77
Homomorphic Minimum Comparison (T_{min})	23936.17
Homomorphic Maximum Comparison (T_{max})	22475.76
Collective Bootstrapping ($T_{bootstrap}$)	3263.24
PPF Robust Scaling	
Homomorphic Summation (T_{sum})	1.73

Table A.3: Microbenchmarks with 50 clients.

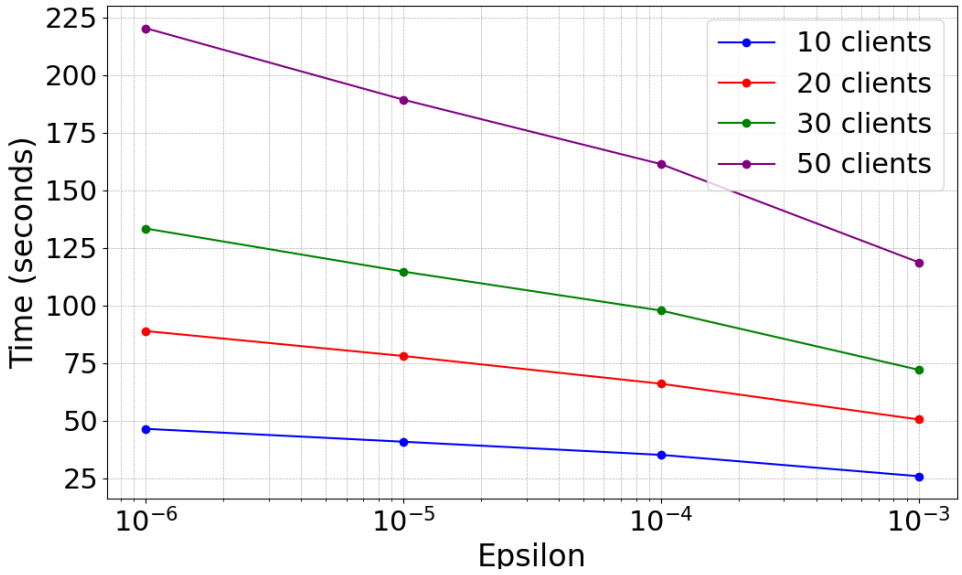

Figure A.7: Effect of the precision (ϵ) on the PPF k-th element algorithm. We observe that runtime increases linearly, while ϵ decreases exponentially.

Appendix B

Detailed Model Information

Table B.1 describes the models used in the experiments in Section 4.2 of the main manuscript for each dataset.

Dataset	Model Architecture
Parkinson's	**Type:** Feedforward NN **Layers:** Dense(128, ReLU, input: DATASET_INPUT_SHAPE) → Norm → Dropout(0.2) → Dense(64, ReLU) → Dropout(0.2) → Dense(32, ReLU) → Dropout(0.2) → Dense(16, ReLU) → Output Dense(1)
MNIST	**Type:** CNN **Layers:** Conv2D(6, 5×5, ReLU, input: DATASET_INPUT_SHAPE) → Norm → MaxPool2D(2×2) → Conv2D(16, 5×5, ReLU) → Norm → MaxPool2D(2×2) → Flatten → Dense(120, ReLU) → Dense(84, ReLU) → Output Dense(10, softmax)
CIFAR10	**Type:** Deep CNN **Blocks:** Conv2D(64, $(3,3)$, ReLU, same, L2) → Norm → Pool → Conv2D(128, $(3,3)$, ReLU, same, L2) → Norm → Pool → 2×Conv2D(256, $(3,3)$, ReLU, same, L2) → Norm → Pool → 2×Conv2D(512, $(3,3)$, ReLU, same, L2) → Norm → Pool **FC:** Flatten → Dense(1024, ReLU, L2) → Dropout(0.5) → Dense(1024, ReLU, L2) → Dropout(0.5) → Output Dense(10, softmax)
CIFAR10-50	**Type:** Reduced CNN **Blocks:** Conv2D(32, $(3,3)$, ReLU, same) → Norm → Pool → Conv2D(64, $(3,3)$, ReLU, same) → Norm → Pool → Conv2D(64, $(3,3)$, ReLU, same) → Norm → Pool **FC:** Flatten → Dense(128, ReLU) → Dropout(0.5) → Output Dense(10, softmax) *(Reduced model for 50-client hardware limitations)*
Hepatitis	**Type:** Feedforward NN **Layers:** Dense(16, ReLU, input: 12) → Norm → Dense(8, ReLU) → Dense(8, ReLU) → Output Dense(5, softmax)
BCW	**Type:** Feedforward NN **Layers:** Dense(12, ReLU, input: 30) → Norm → Dense(8, ReLU) → Output Dense(1, sigmoid)

Table B.1: Model Architectures for Each Dataset. Norm = Normalization variants (BatchNorm/InstanceNorm/GroupNorm/LayerNorm). For CIFAR10 with 50 clients (CIFAR10-50), a reduced model was used due to hardware limitations.