Feature Selection in the Federated Machine Learning setting

Ioannis Christofilogiannis

Thesis submitted in fulfillment of the requirements for the

Diploma of Electrical and Computer Engineering

Technical University of Crete
School of Electrical and Computer Engineering
University Campus, Akrotiri, Chania, GR-73100, Greece

Thesis Supervisor: Associate Professor *Sotiris Ioannidis*

Technical University of Crete
School of Electrical and Computer Engineering

Feature Selection in the Federated Machine Learning setting

Thesis submitted by
Ioannis Christofilogiannis
in fulfillment of the requirements for the
Diploma of Electrical and Computer Engineering

THESIS APPROVAL

Author: _____
Ioannis Christofilogiannis

Committee approvals: _____
Sotiris Ioannidis
Associate Professor, Thesis Supervisor, Committee Member

_____
Samoladas Vasilis
Professor, Committee Member

_____
Spyropoulos Thrasyvoulos
Professor, Committee Member

Chania, May 2025

Abstract

This thesis presents two main contributions to advance research in Federated Learning (FL): Feature Election and the FLEx framework. Feature Election is a novel federated Feature Selection algorithm that enables conventional Feature Selection (FS) methods to operate in horizontal federated settings without altering their core logic. The algorithm leverages client-generated vote vectors with preference scores while preserving data privacy, using a freedom degree parameter to control selection granularity. The second contribution, FLEx (Federated Learning Exchange), is a comprehensive framework that combines C++'s network performance with Python's machine learning capabilities through Cython integration, secured by both symmetric and asymmetric encryption. This framework compares favorably with competing solutions based on evaluation metrics from a recent survey. Experimental validation across five datasets using three Machine Learning (ML) model types demonstrates that these contributions significantly reduce communication overhead with model parameter size reductions across all experiments (up to 93.4%), while maintaining or improving model performance and reducing noise, overfitting and computational cost. Integration of the Feature Election algorithm with the Flower framework achieved model size reductions up to 67.7%, while feature augmentation experiments confirmed robustness in high-dimensional spaces. Feature Election in FLEx establishes a new paradigm for network-efficient FL in bandwidth-constrained scenarios where data privacy is paramount.

Keywords: Federated Machine Learning, Federated Feature Selection, Feature Election

Περίληψη

Αυτή η διπλωματική εργασία παρουσιάζει δύο κύριες συνεισφορές για την προώθηση της έρευνας στην Ομοσπονδιακή Μάθηση (FL): Feature Election και το σύστημα FLEx. Το Feature Election είναι ένας νέος ομοσπονδιακός (federated) αλγόριθμος επιλογής χαρακτηριστικών που επιτρέπει στις παραδοσιακές μεθόδους επιλογής χαρακτηριστικών (FS) να λειτουργούν σε οριζόντια ομοσπονδιακά περιβάλλοντα χωρίς να αλλάζει η βασική τους λογική. Ο αλγόριθμος αξιοποιεί διανύσματα ψήφων που δημιουργούνται από τους πελάτες με βαθμολογίες προτίμησης, διατηρώντας παράλληλα το απόρρητο των δεδομένων, χρησιμοποιώντας μια παράμετρο βαθμού ελευθερίας που ελέγχει το πλήθος των χαρακτηριστικών που επιλέγονται. Η δεύτερη συνεισφορά, το FLEx (Federated Learning Exchange), είναι ένα ολοκληρωμένο σύστημα που συνδυάζει την απόδοση δικτύου της C++ με τις δυνατότητες μηχανικής μάθησης της Python με ενσωμάτωση μέσω της Cython, προστατευμένο από συμμετρική και από ασύμμετρη κρυπτογράφηση. Αυτό το σύστημα συγκρίνεται ευνοϊκά με ανταγωνιστικές λύσεις βάσει μετρικών αξιολόγησης από μια πρόσφατη έρευνα. Η πειραματική επικύρωση σε πέντε σύνολα δεδομένων χρησιμοποιώντας τρεις τύπους μοντέλων ML αποδεικνύει ότι αυτές οι συνεισφορές μειώνουν σημαντικά το κόστος της επικοινωνίας με μειώσεις μεγέθους παραμέτρων μοντέλου σε όλα τα πειράματα (έως 93,4%), διατηρώντας ή βελτιώνοντας την απόδοση του μοντέλου και μειώνοντας τον θόρυβο, την υπερπροσαρμογή και το υπολογιστικό κόστος. Η ενσωμάτωση του αλγορίθμου Feature Election με το σύστημα Flower πέτυχε μειώσεις μεγέθους μοντέλου έως 67,7%, ενώ τα πειράματα επαύξησης χαρακτηριστικών επιβεβαίωσαν την ανθεκτικότητα σε χώρους υψηλών διαστάσεων. Μαζί, το Feature Election και το FLEx καθιερώνουν ένα νέο παράδειγμα για δικτυακά αποδοτική ομοσπονδιακή μάθηση σε σενάρια περιορισμένου εύρους ζώνης με ευαίσθητα δεδομένα.

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my thesis supervisor, Associate Professor Sotiris Ioannidis, for his invaluable guidance, expertise, and support throughout this research journey. His feedback was instrumental in shaping this work.

I am also deeply grateful to Professor Vasilis Samoladas and Professor Thrasyvoulos Spyropoulos for being on my thesis committee.

Special thanks go to Alexander Shevtsov and Ioannis Lamprou for their dedicated assistance, constructive feedback, and willingness to discuss research challenges. I would also like to extend my appreciation to the entire AI research group for introducing me to cutting-edge research and creating an environment of continuous learning and growth.

To my friends who provided much-needed breaks from research and constant encouragement, thank you for being there and reminding me of life beyond my thesis.

My deepest appreciation goes to my family for their patience, and support throughout this journey. This work would not have been possible without their understanding and sacrifices.

Contents

List of Figures

Chapter 1

Introduction

During the past decade, ML has experienced a significant surge in popularity, largely due to its success in various domains and its adaptability to different types of problems [3]. Two critical factors contributing to the high adaptability of ML models are: (1) their ability to identify and eliminate noise from tabular datasets through Feature Selection techniques, and (2) their capability to adapt hyperparameters to the specific demands of each task and dataset (known as hyperparameter optimization or fine-tuning). These capabilities significantly enhance model performance while reducing computational complexity.

Feature Selection offers several crucial benefits in ML applications, as highlighted in the comprehensive review by Theng and Bhoyar [4]. By eliminating irrelevant and redundant features, this technique effectively reduces the dimensionality of the feature space without significantly impacting decision-making quality. This leads to reduced computational time, improved prediction power, better generalization capabilities, and enhanced model interpretability. As the authors emphasize, feature selection is "indispensable in many tasks that require a significant subset of features for knowledge discovery" [4]. Properly selected features can minimize the risk of overfitting, which is particularly valuable when working with high-dimensional datasets where the number of features greatly exceeds the number of observations (samples).

Traditional centralized approaches to ML face significant challenges related to data privacy, regulatory compliance, and computational efficiency, particularly in sensitive domains such as healthcare, finance, and autonomous systems. FL, introduced by Google in 2016 [5], has emerged as a transformative paradigm that enables multiple devices to collaboratively train ML models without sharing their private data, effectively addressing these privacy concerns while introducing new technical challenges.

In healthcare applications, FL facilitates collaborative training of machine learning

models across multiple institutions without sharing sensitive patient data, enabling critical applications such as predicting clinical outcomes in patients with COVID-19 [6]. Similar privacy-preserving benefits have been demonstrated in financial fraud detection, where institutions can collaboratively enhance their detection capabilities without compromising customer confidentiality. The Internet of Vehicles (IoV) represents another promising application domain, where FL enables vehicles to collectively learn models for traffic prediction and intelligent routing while preserving sensitive location and vehicle information.

The distributed nature of FL introduces significant challenges in managing high-dimensional feature spaces efficiently across participating nodes. FS faces unique challenges in federated environments. The inability to directly share data between parties necessitates novel approaches for coordinating Feature Selection across distributed participants while maintaining privacy constraints and ensuring model convergence. Moreover, the communication overhead in FL systems becomes particularly critical when dealing with high-dimensional feature spaces, making efficient FS even more essential.

The statistical heterogeneity inherent in FL environments presents additional complexities. The non-independent and identically distributed (non-IID) nature of data across clients can lead to significant performance degradation, with accuracy reductions of up to 55% in cases where each client only has access to one class [7]. Weight divergence, where model weights trained on different clients deviate substantially from the optimal weights, further compounds these challenges. These issues are particularly pronounced in real-world applications where data distributions vary significantly across participating nodes, necessitating robust solutions that can maintain model performance while preserving privacy and reducing communication overhead.

In cross-silo FL scenarios, where organizations collaborate with consistent participation throughout the training process, the challenges of Feature Selection become even more pronounced. The need to maintain data privacy while enabling effective model training across organizational boundaries requires sophisticated approaches to feature selection that can operate without direct access to raw data. This scenario is common in healthcare settings, where multiple hospitals could collaborate to develop predictive models for rare diseases while maintaining strict patient confidentiality.

This thesis addresses these fundamental challenges by adopting Feature Selection methods into Federated Learning environments, with particular emphasis on maintaining data privacy and reducing communication overhead. Our research explores how FS can be effectively implemented in federated settings to simultaneously improve model performance and reduce communication costs, while ensuring robust privacy guarantees through secure parameter transmission.

1.1 Contributions

This thesis introduces Feature Election (FE), a novel federated FS algorithm that enables privacy-preserving, collaborative Feature Selection across distributed data silos. Inspired by voting systems, FE seamlessly integrates with existing Feature Selection techniques and federated learning infrastructures.

The FLEx framework uses this algorithm with existing FS techniques, such as Lasso [8] , Sequential Attention [9], and PyImpetus [10], to operate seamlessly within a federated environment without altering their core selection logic. The algorithm works by allowing clients to generate vote vectors with their selected feature sets and preference scores, enabling privacy-preserving, collaborative FS. Through a sophisticated voting mechanism, the algorithm supports both uniform and fair weighting schemes, where client contributions can be weighted based on their local dataset sizes to ensure equitable representation in the FS process.

The Feature Election (FE) algorithm introduces the concept of freedom rate, a parameter that controls the number of features from the union of client voting vectors. This parameter allows for fine-tuning of the FS process, enabling administrators to adjust the trade-off between model complexity and performance. When the freedom degree is set to zero, the algorithm selects only features that are unanimously chosen by all clients, while a freedom degree of one includes all features selected by any client. This flexibility allows the algorithm to adapt to various application requirements and data distributions.

To validate and implement our proposed solutions, we developed the FLEx (Federated Learning Exchange) framework, a comprehensive system implemented using C++ and Python. This implementation leverages C++'s superior network performance and Python's extensive machine learning capabilities through Cython integration. The framework incorporates a secure communication protocol that utilizes both symmetric (AES) and asymmetric encryption (RSA) for parameter transmission, ensuring data privacy throughout the training process. The system is deployed using Docker and coordinated by Docker compose, enabling scalable and reproducible experimentation in a simulated distributed environment.

The FLEx framework architecture is designed for optimal performance and flexibility. It implements efficient client-server communication with optimized parameter aggregation using the FedAvg algorithm, which weights client contributions based on their local dataset sizes. The framework supports multiple feature selection methods and integrates with Optuna [11] for hyperparameter optimization, utilizing Asynchronous Successive Halving (ASHA) for efficient parameter tuning. This comprehensive approach addresses critical challenges in FL, including the management of non-IID data distributions and the optimization of communication efficiency.

Through extensive experimentation with multiple real-world datasets, including

the Mushroom [12], Adult Income [13], Breast Cancer Wisconsin [14], and Heart Disease datasets [15], we demonstrate the effectiveness of our approach across different FL scenarios. Our experimental results reveal substantial improvements in both model performance and communication efficiency. Notably, our approach achieves a reduction in parameter size of up to 93.4% with Gaussian Naive Bayes models and 88.8% with Stochastic Gradient Descent models and up to 84.9% with Multi Layer Perceptron models as shown in Table7.3, while maintaining or improving model accuracy. These improvements in communication efficiency make our approach particularly valuable for real-world deployments where bandwidth constraints are significant.

This research can continue in various ways in the future. An automated mechanism for tuning the Feature Election freedom degree parameter automatically without relying on user input can be developed. The FLEx framework can evolve to support different FL paradigms such as Vertical Federated Learning (VFL) and Federated Transfer Learning (FTL). The source code of FLEx and the Feature Election algorithm will be publicly available after the publication of a research paper. This enables users to customize FLEx to their liking or implement the Feature Election algorithm into their systems.

The key contributions of this thesis are as follows:

- We introduce the Feature Election algorithm that allows the use of centralized FS algorithms in Horizontal Federated Learning (HFL) environments. We use a novel parameter, freedom degree, to flexibly control federated FS granularity.
- We developed FLEx, a feature rich FL framework with secure parameter sharing using hybrid encryption. FLEx compares favorably to the competition based on the criteria of recent surveys [16] and is built using C++ and Python
- We applied S.Attention algorithm [9] and PyImpetus [10] on a HFL environment using Feature Election.
- We applied Lasso [8] on a horizontal federated setting and compared it's performance to different more complex FS methods.
- We showcase great parameter and model size reduction by using Feature Election, which makes network communication much more efficient.
- We applied the Feature Election algorithm to the industry leading FL framework, Flower [17] and showcased some favorable results for federated FS, with great model size reductions.
- We combined strong work from different domains, like Optuna [11] for FS method tuning, PyImpetus [10] for FS, Cython [18] for coordinating a complex C++ and Python system

1.2 Organization of Thesis

The remainder of this thesis is organized as follows:

- **Chapter 2** provides a comprehensive theoretical background on FL and FS categories, establishing the foundation for this research. This chapter introduces the key concepts of FL paradigms, including horizontal, vertical, and transfer learning approaches, along with explanations of different types of FS methods that form the basis of our work.
- **Chapter 3** explores the applications and limitations of FL, examining both current use cases and challenges. We discuss real-world implementations across healthcare, finance, transportation, and other domains, while addressing fundamental challenges including statistical heterogeneity, communication overhead, and privacy concerns that motivate our research.
- **Chapter 4** reviews related work in federated FS and existing frameworks. This chapter critically evaluates current approaches to FS in federated environments and assesses the capabilities of leading FL frameworks, identifying gaps that our work aims to address.
- **Chapter 5** presents Feature Election in detail, including a step-by-step sample run. We introduce the algorithm's voting mechanism, freedom degree parameter, and different modes, explaining how it enables conventional FS methods to operate effectively within federated environments.
- **Chapter 6** details the FLEx framework architecture, optimization strategies, and security measures. This chapter covers the development of our FL system, its distributed architecture, communication protocols, parameter aggregation techniques, and integrated security features that ensure privacy preservation during training.
- **Chapter 7** presents our experimental results, including detailed performance analyses across different datasets, models, and FS methods. We evaluate our approach using multiple real-world datasets and machine learning models, demonstrating improvements in both model performance and communication efficiency compared to baseline approaches and existing frameworks.

Chapter 2

Background

In order to better understand the problem definition and the proposed solutions, it is necessary to first understand the fields of Federated Learning and Feature Selection. The intersection of these domains presents challenges and opportunities that form the foundation of this work. The first section makes an introduction to FL and showcases its different categories, while the second section gives an overview on the different categories of FS methods. The third section surveys multiple FS methods and the benefits they provide. Some of these methods are used in the Experiments chapter 7. In the final section, the models used in the experiments are described briefly.

2.1 Federated Learning

FL, first introduced by Google in 2016 [5], enables multiple devices to collaboratively train machine learning models without sharing their private data. It is a paradigm shift in ML where models are trained across multiple decentralized devices or servers holding local data samples, without exchanging them. This learning paradigm has found applications in critical domains such as healthcare and finance where sharing private user information poses risks.

Federated Learning can be categorized based on different perspectives, including data distribution and system architecture, as described by Mammen et al. [19].

From a data distribution perspective, HFL is applied when datasets share the same feature space but differ in sample instances. In this approach, multiple parties with similar feature spaces but different data instances collaborate to train a shared model without exposing their raw data. This is commonly seen in scenarios where data is horizontally partitioned, such as when hospitals in different regions hold medical records of different patients but share the same features like lab results and diagnoses [20, 21]. A notable example is Google's keyboard implementation, where mobile phones have different training data with the same features.

The primary challenge in HFL is managing the diversity and heterogeneity of client data, commonly referred to as non-IID data. This diversity often leads to issues such as model bias and slower convergence rates. Techniques like model aggregation, personalization, and weighted averaging have been developed to address these challenges effectively [21].

In contrast, VFL addresses scenarios where multiple parties have datasets with different features but the same sample instances. In VFL, data alignment is a crucial initial step, ensuring that each participating party identifies the common entities across their datasets. The training process involves secure computation protocols to protect data privacy while enabling joint model training. A typical example would be a bank and a local retail company holding distinct attributes about common customers [22].

Recent advancements in VFL have addressed challenges related to effectiveness, security, and applicability. Researchers have developed methods to enhance model performance despite limited overlap in data samples among parties. Additionally, robust security measures like Homomorphic Encryption (HE) and Secure Multi-Party Computation (SMPC) have been proposed to safeguard against potential privacy breaches during the learning process [22].

FTL extends these approaches to handle scenarios where datasets differ in both feature and sample space. FTL is designed for cases where participating parties have different feature spaces or non-overlapping datasets but need to collaborate to train a shared model. For example, in healthcare, a hospital with patient health records can collaborate with a genomic research institute with DNA data to develop a model for predicting disease risks without sharing sensitive information [23]. FTL leverages pre-trained models to transfer knowledge between domains, enabling efficient training even when data distributions differ significantly.

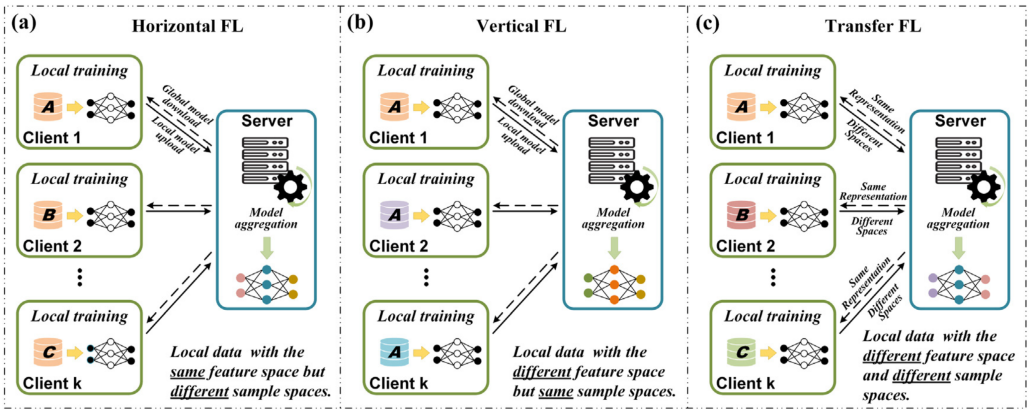

Figure 2.1: Different FL data distribution paradigms, from a recent survey of aggregation techniques [1].

From a system architecture perspective, Federated Learning implementations can

be categorized as either Cross-Device or Cross-Silo. Cross-Device Federated Learning (Cross-Device Federated Learning (CD-FL)) is designed for scenarios involving a large number of edge devices, such as smartphones, IoT devices, and wearable sensors. Each device holds a unique subset of training data and participates in the collaborative model training process. A well-known example is again the Google Keyboard, where millions of users' devices collaboratively train predictive text models while preserving user privacy [20].

The key challenge in CD-FL is the heterogeneity of devices in terms of computational power, network conditions, and battery life. Efficient client selection mechanisms play a crucial role in ensuring that only suitable devices participate in each training round. Additionally, compression techniques are often used to minimize communication overhead, as the scale of devices can lead to significant bandwidth constraints [24].

Cross-Silo Federated Learning (Cross-Silo Federated Learning (CS-FL)), on the other hand, is tailored for scenarios involving a smaller number of participants, typically organizations or institutions with reliable and consistent participation throughout the training process. Examples include collaborations between hospitals or banks, where each party contributes high-quality data from a specific domain. Unlike CD-FL, CS-FL can accommodate both horizontal and vertical data distributions, making it highly versatile [2].

In CS-FL, the reliability of participating parties allows for more efficient coordination and consistent communication. Secure protocols such as Homomorphic Encryption or Differential Privacy are often employed to ensure data privacy during training. For instance, in a healthcare setting, multiple hospitals might collaborate to develop a predictive model for rare diseases while maintaining patient confidentiality [20].

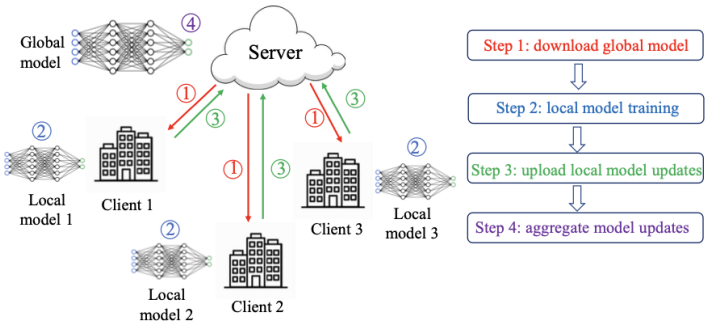

Figure 2.2: Cross-silo FL as shown in the work by Huang et al. [2]

2.2 Feature Selection Categories

Feature selection techniques can be broadly categorized into three main approaches: filter methods, wrapper methods, and embedded methods. Each category is different in terms of how they evaluate and select features, as described in the recent survey by Theng and Bhoyar [4]:

Filter methods determine feature relevance by estimating relationships between features and the dependent variable using statistical metrics, without involving any learning algorithms. These methods utilize measures such as correlation, consistency, information theory, or distance metrics to evaluate features. Filter methods are computationally efficient and fast, independent of the learning algorithm, and scalable to high-dimensional datasets, making them suitable for initial feature reduction in large-scale problems.

Wrapper methods adopt a black-box approach where they use a specific classifier's performance as feedback to assess feature subsets. These methods combine a search strategy with a ML algorithm that serves as an evaluation metric. Wrapper methods typically achieve higher accuracy compared to filter methods and have the ability to detect feature interactions. However, they are computationally more intensive and carry a risk of overfitting to the specific classifier used, which may limit their generalizability across different models.

Embedded methods integrate FS as part of the model training process. These methods attempt to combine the advantages of both filter and wrapper approaches. Embedded methods perform FS during model training, offering better computational efficiency than wrapper methods while maintaining higher accuracy than filter methods. They consider feature interactions while being less prone to overfitting, which makes them particularly valuable for complex prediction tasks where both performance and interpretability are important.

The choice of FS method depends on various factors including dataset characteristics, computational resources, and specific application requirements. Filter methods are often preferred for high-dimensional data due to their scalability, while wrapper methods might be more suitable when accuracy is the primary concern and computational resources are available. Embedded methods offer a balanced approach, making them increasingly popular in modern machine learning applications [4].

Attention mechanisms have recently emerged as a powerful approach for Feature Selection, particularly in deep learning contexts. These methods leverage attention weights to determine feature importance while accounting for complex interactions and dependencies [25]. Attention-based approaches apply trainable attention masks to input features and learn importance weights during model training. They consider feature interactions through multi-head attention structures and offer interpretable feature importance scores through attention weights. Recent work has

shown that attention-based methods, like Sequential attention, can be theoretically grounded and have shown promise by implementing adaptive FS processes, reducing computational overhead while maintaining strong performance [9].

2.3 Feature Selection Algorithms

This section covers FS algorithms studied and tested in ML and FL environments. Some of these methods are used in experiments conducted in the evaluation chapter. The criteria for using methods on the experiments was code availability, quality and performance. The following algorithms are grouped into categories based on their background.

Lasso, introduced by Tibshirani [8], is an embedded method and a form of regularized least squares regression that applies a continuous shrinking operation, effectively producing coefficients that can be exactly zero. In the context of variable selection it helps create a sparse model and shows flexibility across various models. It combines advantages of subset selection (producing interpretable models) and ridge regression (providing stable estimates).
The general Lasso optimization problem is:

$$(\hat{\alpha}, \hat{\beta}) = \arg\min \left\{ \sum_{i=1}^{N} \left(y_i - \alpha - \sum_j \beta_j x_{ij} \right)^2 \right\} \text{ subject to } \sum_j |\beta_j| \leq t$$

The paper notes that for all t, the solution for α is $\hat{\alpha} = \bar{y}$. When the predictors are standardized (mean 0, variance 1), we can assume without loss of generality that $y = 0$ and simplify:

$$\min_{\beta} \left\{ \sum_{i=1}^{N} \left(y_i - \sum_j \beta_j x_{ij} \right)^2 + \lambda \sum_j |\beta_j| \right\}$$

Notes: Find $\hat{\beta}$ that minimizes $\sum_{i=1}^{N}(y_i - \sum_j \beta_j x_{ij})^2$ subject to the constraint $\sum_j |\beta_j| \leq t$, where the predictors x_{ij} are standardized s.

The **Elastic Net** [26] is another embedded method that addresses key limitations of Lasso [8] in FS, particularly for high-dimensional data with correlated features. Specifically, Lasso tends to select only one variable from a group with multiple pairwise associations and is not well defined unless the bound of the L1-norm is smaller than a threshold. Given predictors $\mathbf{X}$ and response $\mathbf{y}$, the elastic net estimator $\hat{\boldsymbol{\beta}}$ minimizes:

$$L(\lambda_1, \lambda_2, \boldsymbol{\beta}) = |\mathbf{y} - \mathbf{X}\boldsymbol{\beta}|^2 + \lambda_2 |\boldsymbol{\beta}|^2 + \lambda_1 |\boldsymbol{\beta}|_1 \tag{2.1}$$

where $|\boldsymbol{\beta}|_2^2 = \sum_{j=1}^{p} \beta_j^2$ and $|\boldsymbol{\beta}|_1 = \sum_{j=1}^{p} |\beta_j|$ are the L_2 and L_1 norms respectively. This criterion combines ridge regression's L_2 penalty with lasso's L_1 penalty, controlled by parameters λ_1 and λ_2.

The elastic net exhibits three key properties that make it particularly suitable for Feature Selection:

1. It performs automatic variable selection and continuous shrinkage simultaneously through the L_1 penalty term
2. It can select more than n variables in the $p \gg n$ case, overcoming lasso's limitation of selecting at most n variables
3. It demonstrates the grouping effect: highly correlated predictors tend to be selected or removed together

The estimator can be reformulated as:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \boldsymbol{\beta}^T \left(\frac{\mathbf{X}^T\mathbf{X} + \lambda_2 \mathbf{I}}{1 + \lambda_2} \right) \boldsymbol{\beta} - 2\mathbf{y}^T\mathbf{X}\boldsymbol{\beta} + \lambda_1 |\boldsymbol{\beta}|_1 \tag{2.2}$$

This formulation reveals that the elastic net can be viewed as a stabilized version of lasso, where the sample correlation matrix $\mathbf{X}^T\mathbf{X}$ is shrunk toward the identity matrix. When $\lambda_2 \to \infty$, the method approaches simple soft thresholding of individual coefficients, while $\lambda_2 = 0$ reduces to the lasso.

The elastic net has proven particularly effective in domains like gene expression analysis where features exhibit natural grouping and the number of features far exceeds the number of samples. This makes it a foundational method in the Feature Selection literature, spawning numerous variants and extensions.

Attention mechanisms like the Transformer [25] have become pivotal in modern ML, enabling models to focus on the relevant parts of input data, thus enhancing their performance across various complex tasks like natural language processing, image recognition and Feature Selection.

Sequential Attention [9] is an adaptive attention-based approach to Feature Selection that combines an iterative attention mechanism with greedy forward selection. The key insight is using attention weights to efficiently approximate the marginal gains of all candidate features simultaneously.

The algorithm proceeds in rounds, with each round selecting one feature according to:

$$w^* \leftarrow \arg\min_{\theta, w} \ell(f(X \circ W; \theta), y) \tag{2.3}$$

where $W = 1_n \text{softmax}(w, S)^\top$ and S is the set of currently selected features. The feature with the highest attention weight is then selected:

$$i^* \leftarrow \arg\max_{i \notin S} w_i^* \tag{2.4}$$

This approach offers two key advantages:

(1) It reduces computational complexity from $O(kd)$ to $O(k)$ models by approximating all feature marginal gains in parallel through attention, and

(2) It allows for adaptive Feature Selection by iteratively updating the attention weights based on previously selected features.

Empirical results demonstrate that Sequential Attention achieves state-of-the-art performance while maintaining high computational efficiency [9]. Additionally, theoretical analysis shows connections to classical Orthogonal Matching Pursuit, providing provable guarantees for the linear case.

The **Attention-based Feature Selection (AFS)** mechanism [27] introduces a novel approach to supervised Feature Selection by reformulating the feature relevance problem into a binary classification task for each feature, allowing for learned attention weights to determine feature importance.

The architecture consists of two key components [27]. First, a dense network extracts intrinsic relationships E among input features:

$$E = \text{Tanh}(X^T W_1 + b_1) \tag{2.5}$$

where W_1 and b_1 are learnable parameters. Second, a feature-specific attention network generates selection probabilities. For each feature k, the network produces selection and non-selection values:

$$p^k = w_p^k h_L^k + b_p^k \tag{2.6}$$

$$n^k = w_n^k h_L^k + b_n^k \tag{2.7}$$

where h_L^k is the L-th hidden layer output. The final attention weight a^k for feature k uses softmax:

$$a^k = \frac{\exp(p^k)}{\exp(p^k) + \exp(n^k)} \tag{2.8}$$

The final feature weight s^k is the mean attention weight across m samples:

$$s^k = \frac{1}{m} \sum_{i=1}^{m} a_i^k \tag{2.9}$$

This approach offers key advantages over traditional methods: dedicated attention networks provide comprehensive feature relationship modeling, softmax ensures bounded weights for training stability, and the dense network acts as an initial redundancy filter [27].

Another approach to FS is to find the Markov Blanket (MB) of the target variable in the feature set, which provides a theoretically optimal subset of features for prediction. A MB of target T is the minimal set of variables containing all necessary information to predict T, formally defined as the set that makes T conditionally independent of all other variables X,

$$\text{MB}(T) = \{Par(T) \cup Ch(T) \cup Sp(T)\} \Rightarrow P(T|\text{MB}(T), X) = P(T|\text{MB}(T))$$

where parents are direct causes, children direct effects and spouses are sharing children with T, as defined in the theory of Bayesian Networks [28].

While MB based methods provide a principled approach to Feature Selection through probabilistic relationships, kernel-based methods have advantages in Feature Selection, particularly in problems involving complex nonlinear relationships by implicitly mapping data into high-dimensional spaces.

Feature Space Markov Blanket (FSMB) [29] is a filter method that combines the theoretical properties of MB and Kernel-based approaches. The algorithm operates as follows:

- Mapping the data to a higher-dimensional feature space using kernel methods, where multivariate associations become detectable as pairwise associations. Specifically, using a polynomial kernel of degree d, features are constructed as $\Phi_q = c_q \prod_{v=1}^{m} X_v^{q_v}$, where c_q represents coefficient weights.
- Identifies the Markov Blanket in this feature space ($MB_\Phi(T)$) and maps the selected features back to the original variable space. To maintain computational efficiency, feature importance scores $s_{i,j}$ are used to identify the most relevant features without explicitly constructing the entire feature space.

This approach effectively addresses the limitations of traditional MB methods, which may fail to detect variables involved in complex multivariate relationships (e.g., 2 features X1, X2 have no correlation with T, but their XOR output has) while maintaining the theoretical guarantees. The method has been shown to be particularly effective in high-dimensional datasets, consistently identifying minimal predictive feature sets while maintaining high classification accuracy, as demonstrated in both synthetic XOR problems and real-world applications like gene expression analysis.

The **Balanced Markov Blanket discovery (BAMB)** algorithm [30] is a filter algorithm designed to identify the Markov Blanket (MB) of a target variable for Feature Selection.

It concurrently learns the parents-children (PC) and spouses (SP) of the target variable in a single phase. This simultaneous learning aims to balance computational efficiency and learning accuracy by iteratively updating candidate PC and SP sets, removing false positives during each iteration. This integrated process reduces computational overhead and enhances the precision of the selected feature subset.

The algorithm achieves a computational complexity of $O(2^{|PC|} \cdot |U|)$, where $|PC|$ is the size of the parents-children set and $|U|$ is the number of features. For a target T and feature X, with conditioning set Z, BAMB employs conditional independence testing:

$$T \perp\!\!\!\perp X|Z \text{ implies conditional independence} \tag{2.10}$$

It maintains minimal separating sets ($Sep_T\{X\}$) and updates candidate sets CPC_T and $CSP_T\{X\}$ concurrently, allowing it to achieve higher accuracy than traditional constraint-based methods while maintaining computational efficiency.

Recent work by Hassan et al. [10] introduced **Predictive Permutation Feature Selection (PPFS)**, the first wrapper-based MB Feature Selection method. PPFS employs a novel conditional independence test called Predictive Permutation Independence (PPI) that leverages modern supervised learning algorithms. The PPI test operates through a null hypothesis $H_0 : \mathbb{E}(R) \geq \mathbb{E}(\tilde{R})$, where R represents the empirical risk with the original feature and $\tilde{R}$ represents the risk with a permuted version, allowing the detection of complex multivariate associations. Key innovations of PPFS include:

- A universal approach that works with both categorical and continuous features, supporting both classification and regression tasks through a unified independence testing framework
- Integration with any supervised learning algorithm through its wrapper-based design, enabling utilization of advanced methods like gradient boosting machines
- An optional MB aggregation step that combines results from K folds using a scoring function $z_i = \frac{\sum_{j=1}^{|MB_i(Y)|} \text{freq}(X_j)}{|MB_i(Y)|}$ to handle violations of the faithfulness assumption
- Theoretical guarantees of correctness under the faithfulness condition, with a growth phase ensuring capture of parents and children, while the permutation property enables detection of spouse features

Empirically, PPFS demonstrates superior performance compared to traditional MB methods, achieving approximately 7% higher F1-scores on benchmark Bayesian network datasets while maintaining computational efficiency comparable to existing approaches. On high-dimensional datasets like Arcene (10,000 features) and Dorothea (100,000 features), PPFS achieves better prediction accuracy while selecting significantly fewer features - often less than 50% of those selected by competing methods. The authors of this paper created PyImpetus which is a Python framework that can serve as a plug and play FS method for real world datasets.

2.3.1 Machine Learning models

To perform FL experiments, three different ML models were selected. These models run on the client side, are updated on each FL round using the same training data but new parameters and when FS is performed.

Gaussian Naive Bayes (GNB) is a probabilistic classification algorithm that applies Bayes' theorem with the assumption that features are conditionally independent given the class label. In the Gaussian variant, it is further assumed that the continuous features follow a normal (Gaussian) distribution. This model calculates the probability of each class based on the input features and selects the class with the highest posterior probability. GNB is particularly effective for classification tasks where the assumption of feature independence holds. The scikit-learn implementation of this algorithm was used [31].

Stochastic Gradient Descend (SGD) [32] is an optimization algorithm commonly used to train machine learning models, especially when dealing with large datasets. When applied with the log loss function, SGD effectively performs logistic regression [33]. In this context, the model predicts probabilities for binary outcomes using the logistic function, and the log loss measures the discrepancy between the predicted probabilities and the actual class labels. By minimizing this loss through iterative updates on individual or small batches of data points, SGD efficiently converges to the optimal parameters of the logistic regression model. This approach is particularly advantageous for large-scale data due to its computational efficiency. The scikit-learn implementation of this algorithm was also used [31].

Multi Layer Perceptron Classifier (MLPC) is a supervised learning algorithm that trains a non-linear function approximator for classification tasks. It implements a feed-forward artificial neural network with one or more hidden layers between the input and output layers. Each neuron in the hidden layers applies a non-linear activation function to a weighted sum of inputs, enabling the model to learn complex non-linear decision boundaries. The network is trained using backpropagation, where the gradient of the loss function is calculated with respect to the weights and subsequently used to update them iteratively. MLPC is particularly effective for classification problems with complex relationships between features. The implementation in scikit-learn offers various optimization techniques, includ-

ing stochastic gradient descent and its variants, along with regularization options to prevent overfitting [31]. Typically, FS is not used for neural network models like this, but in federated settings it gives a significant reduction of model parameter size, something that is demonstrated in the Experiments chapter 7.

Chapter 3

Applications and Limitations of Federated Learning

This chapter focuses on the reasons FL is a popular research topic, having use cases in multiple domains and the limitations that researchers are still tackling. It serves as a link between the Background and Related Work sections to give required context to the reader.

3.1 Applications

FL has been applied in various domains, allowing collaborative model training while preserving data privacy. In **healthcare**, FL facilitates collaborative training of machine learning models across multiple institutions without sharing sensitive patient data. For instance, FL has been employed to predict clinical outcomes in patients with COVID-19 by aggregating data from different healthcare providers, enhancing model accuracy while maintaining patient privacy [6]. Additionally, healthcare data usually contains more features than necessary, like the presence or absence of a bacteria [34], so FS becomes necessary.

In the financial sector, FL allows institutions to collaboratively **detect fraudulent activities** by training models on decentralized data. This collaboration enhances fraud detection capabilities without compromising the confidentiality of customer information [23]. Similarly, in biometrics, FL is transforming recognition systems by enabling collaborative model training across distributed data sources while preserving privacy. This approach addresses privacy concerns and regulatory constraints, allowing for improved model accuracy and generalization in applications such as facial and iris recognition [35].

In the transportation domain, FL aids in the development of **self-driving car technologies** by allowing vehicles to collaboratively improve their models related

to obstacle detection and navigation. This collaboration enhances the safety and reliability of autonomous driving systems without necessitating the sharing of raw data [36]. Building on this, the **Internet of Vehicles (IoV)** represents a promising branch of Internet of Things that enables large-scale applications such as Waze and Uber. In these applications, vehicles report real-time information to cloud servers which then train machine learning models for intelligent traffic management. However, this paradigm has a huge communication overhead and raises significant privacy concerns, as applications can potentially infer sensitive personal information including users' location data, traffic patterns, and vehicle information. In IoV systems, FL enables vehicles to collaboratively learn models for traffic prediction and intelligent routing without sharing individual data, thus improving data privacy and security. This approach supports the development of intelligent transportation systems that can adapt to real-time conditions [37]. Local Differential Privacy (LDP) techniques are applied to perturb the gradients before transmission, to ensure strong privacy guarantees while maintaining model utility.

In industrial applications, FL supports **smart manufacturing within Industry 4.0** by enabling companies to collaboratively train models on production data without exposing proprietary information. This approach enhances production efficiency and product quality while safeguarding sensitive operational data [38]. In **robotics**, FL allows multiple robots to learn from each other's experiences to improve navigation and task execution. For example, a federated reinforcement learning framework enables robots to share knowledge about different environments, leading to more robust and adaptable robotic systems [39].

On the consumer technology front, FL has been utilized in mobile devices to improve **keyboard prediction models**. By training on-device data, models can learn user-specific language patterns without transmitting personal data to central servers, thereby enhancing user privacy [40]. This approach extends to edge computing scenarios where data is generated at the edge of the network. Devices can collaboratively train models on local data, reducing the need for data transmission to central servers and thereby improving efficiency and privacy [41].

In **urban development**, smart city applications leverage FL to enable the development of models for predicting environmental factors like air quality by aggregating data from distributed sensors. This method supports real-time monitoring and decision-making while maintaining data privacy [42].

3.2 Limitations

While Federated Learning (FL) offers significant advantages in preserving data privacy and enabling distributed model training, it faces several fundamental challenges that impact its effectiveness. A primary challenge lies in the nature of the data itself - **the non-independent and identically distributed (IID) distribution** across clients leads to statistical heterogeneity. This disparity can cause

local models to diverge significantly from the global model, resulting in reduced overall performance and slower convergence. The impact can be severe, with accuracy reductions of up to 55% in cases where each client only has access to one class. This performance degradation can be attributed to **weight divergence** - the phenomenon where model weights trained on different clients diverge significantly from the optimal weights that would be learned on the complete dataset [7]. Various studies have explored techniques to address these challenges, such as creating a small subset of globally shared data to substantially improve model training under non-IID conditions [23, 41, 7]. Surveys have also analyzed the influence of non-IID data on both parametric and non-parametric machine learning models in FL [35].

The distributed nature of FL introduces significant **privacy and security vulnerabilities**. Despite its decentralized approach, there remains a risk of data leakage through inference attacks. Adversaries can analyze shared model updates to infer sensitive information about local datasets, compromising privacy. Research highlights that FL suffers from performance loss and privacy leakage due to inference attacks when dealing with non-IID data [23, 35]. Techniques such as Local Differential Privacy (LDP) have been proposed to mitigate these risks [37], while maintaining acceptable model performance for real-world applications. Various LDP mechanisms have been developed, including the Laplace mechanism which adds direct noise to numeric data, and the piecewise mechanism which offers improved utility for large privacy budgets. Hybrid approaches combine multiple mechanisms to optimize performance across different privacy requirements.

Beyond privacy concerns, FL is susceptible to security threats such as **poisoning attacks**, where malicious clients inject false data to corrupt the global model, and backdoor attacks, where adversaries introduce specific vulnerabilities into the model. These threats can undermine the integrity and reliability of the FL system. Research has explored robust aggregation rules and defense mechanisms to make distributed learning algorithms resilient to a minority of malicious participants, especially in non-IID settings [35, 37].

Technical challenges arise from system heterogeneity and communication needs. FL requires frequent client-server communication to exchange model updates. **Limited bandwidth and unstable connections** can impair learning efficiency, prompting solutions like reduced communication rounds [40, 37]. Varying client capabilities and network conditions can create stragglers, requiring mechanisms like FedDyn for effective model aggregation [23, 41].

Resource constraints also present implementation challenges. Running FL on devices with limited computational power strains resources and affects performance. Solutions like FL with local differential privacy help enable machine learning on resource-constrained devices while maintaining privacy [37, 40].

Chapter 4

Related Work

This section surveys recent work on federated Feature Selection and Federated Learning frameworks in two separate sections. Both of these topics are linked to the main contributions of this thesis, as both a custom FL framework and a unique approach to horizontal federated Feature Selection were developed.

4.1 Federated Feature Selection

Feature Selection in FL presents unique challenges due to data privacy requirements and the inability to directly share data between parties. Recent works have proposed various approaches to address these challenges while maintaining selection effectiveness and preserving privacy. In the context of IoT security, Shafiq et al. [43] advanced Feature Selection techniques through a novel hybrid approach combining bijective soft set technique with correlation-based metrics. Their work introduced the CorrACC metric, integrating Correlation Attribute Evaluation with accuracy metrics from machine learning classifiers. The correlation between features is calculated using:

$$Corr = \frac{kavg(corr_{fc})}{\sqrt{k + k(k-1)avg(corr_{ff})}} \tag{4.1}$$

where k represents the number of features, $corr_{fc}$ is the correlation between features and the dependent class, and $corr_{ff}$ represents the correlation among the features. Their experimental validation on the BoT-IoT dataset demonstrated significant dimensionality reduction while maintaining over 95% accuracy across multiple classifiers, addressing key challenges in IoT security through efficient Feature Selection that maintains high accuracy while reducing computational overhead. Cassará et al. [44] address the challenge of distributed FS in autonomous vehicle networks by proposing a federated FS algorithm. The architecture comprises a local Mutual Information-based FS algorithm running on autonomous vehicles and

a Bayesian aggregation function executed on an Edge Server. A key component is their mutual information metric for FS:

$$I(U; y) = H(y) - H(y|U) \tag{4.2}$$

where $H(y|U)$ is the conditional entropy measuring the information needed to describe target y given feature subset U. The system leverages Cross-Entropy optimization for FS while implementing a weighted average scheme for aggregating local probabilities:

$$\mathrm{p}_G = \sum_l \mathrm{p}_l \omega_l \tag{4.3}$$

where $\omega_l = \frac{n_l}{\sum_l n_l}$ weights each vehicle's contribution by its local dataset size n_l. The authors validate their method on both autonomous vehicle sensor data and physiological monitoring datasets, showing robustness to node failures and efficient convergence with limited communication rounds. Their experiments demonstrate significant feature reduction (up to 99 % for vehicle data) while maintaining model accuracy comparable to centralized FS methods. This work advances FL in cyber-physical systems by eliminating raw data sharing, reducing communication overhead through feature compression, and providing theoretical guarantees for distributed FS. For vertical FL settings, Castiglia et al. [45] introduce LESS-VFL, which integrates FS directly into the training process. Their key innovation is using pre-trained model parameters and proximal stochastic gradient descent to optimize FS while minimizing communication overhead. The method operates in three stages: pre-training to obtain initial parameters, server-side embedding component selection, and local FS. Their approach minimizes an optimization problem of the form:

$$\min_{\Theta} R(\Theta; \mathcal{P}_{X,y}) \text{ s.t. } \mathbf{U}_m^k \neq 0; \forall m \in [M], k \in [d_m^s],; \mathbf{V}_m^l = 0; \forall m \in [M], l \in [d_m^z] \tag{4.4}$$

where $R(\cdot)$ is a generalization risk, $\mathbf{U}_m$ and $\mathbf{V}_m$ are weights for significant and non-significant features respectively, and d_m^s, d_m^z represent the number of significant and non-significant features at party m. LESS-VFL shows particular effectiveness in high-dimensional settings, achieving feature reduction while maintaining model accuracy with substantially reduced communication costs compared to standard approaches. Fu et al. [46] proposed FEAST, a communication efficient federated FS framework designed for vertically partitioned data scenarios, where different organizations hold different features for the same set of individuals. The framework calculates feature scores using conditional mutual information, with their core scoring formula:

$$J_{FEAST}(F_i) = \frac{1}{|V|} \sum_{V_j \in V} SU(Y; F_i|V_j) \tag{4.5}$$

where $SU(Y; F_i|V_j)$ represents the symmetric uncertainty between feature F_i and label Y conditioned on statistical variable V_j. FEAST employs this measure to

select informative features with minimal redundancy, thereby enhancing both computation and communication efficiency in FL. Their approach demonstrates state-of-the-art accuracy and reduced overhead in federated FS, marking a significant advance in vertically partitioned FL scenarios. Hu et al. [47] proposed a federated FS algorithm based on Particle Swarm Optimization (PSO) that allows multiple participants to collaboratively select high-quality feature subsets under privacy constraints and has advantages such as fast convergence and easy implementation. The approach maximizes the classification accuracy J across all participants according to:

$$J(X) = \frac{\sum_{i=1}^{M}(|Dat_i| \times J(X, Dat_i))}{\sum_{i=1}^{M}|Dat_i|} \tag{4.6}$$

where $|Dat_i|$ is the size of the dataset held by participant i and $J(X, Dat_i)$ represents the classification accuracy of feature subset X on that participant's data. This approach leverages the FL paradigm by incorporating a third-party coordinator to aggregate feature selections without exposing individual data, significantly enhancing classification accuracy while maintaining privacy. In this approach, sensitive features are deleted effectively, using Privacy-aware wrappers but there are limitations such as slow running speed due to high complexity. They propose the use of a real parallel environment to improve performance and they envision frameworks like their being used in scenarios like cross-hospital learning of rare diseases and cross-bank consumer behavior analysis. In another study, Hermo et al. [48] proposed Fed-mRMR, a lossless federated adaptation of the classic Minimum Redundancy Maximum Relevance (mRMR) algorithm. The key innovation lies in their mathematical reformulation of the mutual information calculations using an occurrences matrix M, defined as:

$$M = (m_{ij}) \in \mathbb{N}^{|V|\times|V|}, m_{ij} = \vec{bf}^{-1}(i) \cdot \vec{bf}^{-1}(j) \tag{4.7}$$

where $\vec{b}_v$ represents a bitmap vector for feature value v, and f maps feature values to matrix indices. This formulation enables federated Feature Selection by sharing only aggregated statistics rather than raw data, while preserving the exact mRMR ranking. Their approach achieves lossless Feature Selection in federated settings through a merge operation that combines local statistics, making it particularly suitable for non-IID data distributions across nodes.

Building on FL's strengths in privacy-preserving healthcare, Kapila and Saleti [49] presented a novel fusion framework combining FL with Feature Selection and feature extraction techniques for disease prediction. Their methodology employs statistical methods including Chi-Square and ANOVA for Feature Selection, and Linear Discriminant Analysis (LDA) for feature extraction. A key component in their FS process was the chi-square test statistic:

$$\chi_f^2 = \Sigma\frac{(OC_i - EC_i)^2}{EC_i} \tag{4.8}$$

where f is the degree of freedom, OC represents the observed values, and EC represents the expected values. This test helped identify features with stronger correlations to the target variable. Their fusion approach demonstrated significant improvements over existing methods, with their Chi-Square with LDA and ANOVA with LDA models achieving identical performance metrics. For the Cleveland heart disease dataset, both models achieved 88.52% accuracy and 89.23% F1-score. For diabetes prediction, both models reached 92.3% accuracy and 94.36% F1-score. These results surpassed several contemporary approaches while maintaining data privacy through FL. The study highlights how the integration of Feature Selection and extraction techniques within a FL framework can enhance disease prediction accuracy while preserving the confidentiality of distributed healthcare data.

Banerjee et al.[50] proposed Fed-MOFS, a hybrid federated Feature Selection approach for horizontal FL scenarios. Unlike approaches that use a scoring function for global feature ranking, Fed-MOFS employs multiobjective optimization to prioritize features based on their higher relevance and lower redundancy. The local Feature Selection component of Fed-MOFS leverages mutual information and clustering, while the global component employs a Pareto optimization strategy using the NSGA-II algorithm. Their formulation optimizes two objective functions simultaneously:

$$\max_{\forall f_k \in F'_{server}} f_{kFCMI} \quad \text{and} \quad \min_{\forall f_k \in F'_{server}} f_{kaFFMI} \tag{4.9}$$

where f_{kFCMI} represents the averaged Feature-Class Mutual Information score of feature f_k across all clients, and f_{kaFFMI} represents the averaged Feature-Feature Mutual Information, which measures redundancy.

The authors demonstrate that Fed-MOFS achieves global model performance enhancement with up to 50% reduction in feature space and operates twice as fast as comparable federated Feature Selection methods based on wall-clock running time analysis. Empirical evaluation across multiple datasets indicates that Fed-MOFS is stable across both IID and non-IID data distributions, is scalable across multiple clients, and does not hamper the convergence of the global model. Their computational complexity analysis shows that Fed-MOFS operates at $O(d^2)$, which is lower than several state-of-the-art federated Feature Selection methods.

A key trend emerging from these works is the focus on communication efficiency alongside selection effectiveness. Recent surveys [22] indicate growing interest in developing methods that can scale to high-dimensional features while maintaining practical communication costs in federated environments.

While these approaches represent significant advances in federated FS, they differ fundamentally in their methodological approaches to handling FS across distributed clients. FEAST [46] leverages conditional mutual information to select informative features while having low redundancy, using statistical variables for

efficient communication between parties. Fed-mRMR [48] provides a lossless federated adaptation of the classic mRMR algorithm through a novel occurrences matrix approach, achieving identical feature rankings as the original method while preserving privacy. The PSO-based approach [47] introduces a credible third participant to assemble and integrate optimal feature subsets from multiple participants, using particle swarm optimization with specialized operators for federated scenarios. LESS-VFL [45] focuses on communication efficiency through a three-stage approach that leverages pre-trained model parameters and proximal stochastic gradient descent specifically for VFL settings.

However, there remains a gap in leveraging the extensive body of conventional FS algorithms within federated environments. Existing approaches either develop specialized federated variants (like Fed-mRMR), create entirely new selection mechanisms (like LESS-VFL and the PSO-based approach), or introduce novel scoring systems (like FEAST). Additionally, many are tailored to specific FL paradigms - LESS-VFL and FEAST focus on vertical FL, while the PSO-based approach and Fed-mRMR address horizontal scenarios.

This gap is particularly noteworthy given the extensive theoretical foundations and empirical validation of traditional FS algorithms. Methods like filter, wrapper, and embedded techniques have been rigorously studied for decades, with well-understood theoretical properties, proven convergence guarantees, and are effective across diverse domains. These algorithms have benefited from extensive comparative studies [4] and numerous refinements based on practical applications. Attention methods are a recent development and these algorithms show promising results for centralized FS. By developing specialized federated variants, existing federated FS approaches potentially forego this rich foundation of theoretical and empirical research.

These observations motivate the work on Feature Election, which takes a fundamentally different approach. Rather than creating specialized federated variants or entirely new selection mechanisms, it is a framework that **enables conventional centralized FS algorithms to operate effectively within federated environments through a novel strategy**. This approach bridges the gap between traditional and federated FS by providing a mechanism to aggregate FS decisions across distributed clients while preserving the functionality of established selection methods.

This methodological distinction is significant for several reasons. First, it allows organizations to take advantage of their existing FS implementations without requiring substantial modifications or specialized protocols. Second, it provides flexibility in choosing different FS methods based on specific requirements or constraints of different clients. Finally, it facilitates the easier adoption of FL in scenarios where existing FS pipelines are used.

4.2 Federated Learning Frameworks

FL has gained significant attention as a decentralized approach to machine learning, enabling multiple clients to collaboratively train models without sharing their private data. Several frameworks have been developed to facilitate the implementation of different FL applications.

A recent survey by Riedel et al. [16] attempts to evaluate existing FL frameworks focusing on features, interoperability, and user friendliness. The authors generated a benchmark with weighted criteria like support of different FL algorithms, paradigms and Docker support. They compared 18 different openly available frameworks avaiilable in platforms like GitHub.

In the following section, some of the most prominent open-source FL frameworks are highlighted:

Flower: A user-friendly FL framework by Beutel et al. [17] that stands out for its exceptional scalability and flexibility, demonstrating capability to handle up to 15M clients using just two GPUs. Its unique architecture supports heterogeneous environments through language and ML framework-agnostic design, while providing efficient resource management via its Virtual Client Engine. Comparative evaluations showed Flower outperforms most frameworks in training efficiency, making it suitable for both research and production environments. However, it has limited security mechanisms compared to frameworks like PySyft, incomplete documentation for advanced features, limited options for training speed optimization.

FedML: A comprehensive FL framework by He et al. [**?**] offering options for on-device training, distributed computing, and single-machine simulation. The framework provides standardized implementations of popular FL algorithms and enables large-scale workloads using minimal GPU resources. Key strengths include its worker/client-oriented programming interface and support for heterogeneous client environments. However, comparative analysis shows some limitations as moderate development effort is required due to less extensive documentation. The framework balances scalability and features making it suitable for large-scale FL deployments despite these trade-offs.

TensorFlow Federated (TFF): Google's open-source FL framework [51] provides two key interfaces: a high-level FL API and a lower-level Federated Core API for implementing novel FL algorithms. Built on TensorFlow, it offers strong type safety and comprehensive simulation capabilities. However, key limitations include: lack of edge device deployment support, restriction to TensorFlow models, and higher cross-platform installation complexity. It requires moderate development effort despite good documentation, making it more suitable for research and prototyping than production.

PySyft: An open source Python library developed by OpenMined [52] that facilitates secure and private machine learning by integrating with popular frameworks

such as PyTorch, Keras, and TensorFlow. It enables privacy-preserving techniques such as FL, differential privacy, and encrypted computations, allowing data scientists to perform analyses without direct access to the underlying data. Although PySyft offers robust functionalities for secure computations, it introduces additional complexity, potentially requiring increased development effort. Moreover, the added security measures may impact training speeds, leading to slower performance relative to centralized approaches. Despite these , PySyft remains a powerful tool for implementing privacy-preserving machine learning models across various platforms.

Although many open-source frameworks are available, none of them focus in FS and ways to use traditional variable selection methods in federated environments. For the requirements of this thesis the benefits of this specialized framework are many:

- **Total Control**: FLEx framework can be customized to the user's liking by modifying Python code for ML or FS modifications, C++ code for modifying the underlying network implementation. It can work with different FL specifications and settings and can scale well to different tasks.
- **Speed**: Using C++ for networks and Python for popular ML libraries provides a speedy implementation coordinated by Cython [18].
- **Feature Selection**: This framework has a direct focus in federated FS with different methods, unlike most popular work mentioned above. In fact, even the industry leaders mentioned above don't have integrated FS methods on runtime.
- **Resource Efficiency**: This custom framework is optimized to work best for the requirements of this thesis, by sending a "params" (which stands for parameters) dictionary between server and clients, containing different data in each communication round, like model parameters and keys.
- **Increased Flexibility**: Building a custom solution enables the implementation of and experimentation with different algorithms, while providing valuable software development experience.
- **Security Measures**: The security process was manually implemented to mitigate potential vulnerabilities, enhancing integrity and privacy.

Compared to the other mentioned frameworks FLEx offers an integrated horizontal federated FS approach that manages to greatly decrease network transmission size and improve or maintain performance. Even without its key feature however it is a flexible framework that has many of features that the industry and open source community requires. As evidenced in this Table 7.6 in the end of the this thesis (Chapter 7), FLEx remains highly competitive using the evaluation metrics of the survey by Riedel et al. [16].

More information about the development of the FLEx framework can be found in the Framework Chapter 5.

xdc

Chapter 5

Feature Election: A different federated Feature Selection approach

One of the goals of this work is to use traditional FS algorithms on a HFL environment. This poses significant challenges, as there is no established methodology for aggregating Feature Selection results from distributed client-side computations into a cohesive global model on the server side. A way to approach this challenge is to send the feature preferences and selected feature sets from each client to the server. These results, along with the number of samples (to weight the results from each client in a fair way), are enough for the server to make an informed decision on which features will be selected globally. The selected feature mask should be returned to the clients in order to train their models and then participate in the FL rounds. Selecting the same features is a necessity of HFL and guarantees the same model dimensions on every client.

To address this FS problem in HFL, the Federated Feature Election Algorithm is proposed. Each client generates a vote vector with their selected feature set (after performing their local FS method like Sequential Attention [9] and PyImpetus [10]), their preference score for all features and the total number of samples used to train their model. The server receives both feature masks from each client. If the algorithm mode is set to 'uniform'(unweighted) each client has the same voting power but if the mode is set to 'fair'(weighted), the number of samples of each client divided by the total number of samples of all clients, is used as the client's voting weight. The features that are selected by all clients are always added in the Global feature mask and then the corresponding preference score is set to zero. The parameter freedom degree decides the percentage of features from the features of $Union - Intersection$, meaning the ones that are selected by at least one but not all clients, are added to G. The server normalizes and then sums all

the preference vectors (with a weighted sum if using 'fair' mode) and then has a complete ranking of features by preference score from all clients, storing it in V. Then only the elements that have been selected from at least one client are kept, in the "Election result" vector. Finally, the x features that have the highest score from this vector are selected and added to the global mask G, which already contains the features from the $Intersection$. The value of x is directly calculated as the ceiling of freedom degree multiplied by the number of features in the "Election result" vector. If for example freedom degree is equal to 0.2, then 20% percent of features from the "Election result" vector will be added to G. An interesting concept is the usage of mixed Feature Selection methods. The server's min-max normalization, along with special methods on the client side that align the arrays consistently, allows clients to select features with different methods and to combine the results. For example Sequential Attention [9] can be used on one client and PyImpetus [10] on another and participate in a feature election round to receive a global mask G which takes the results of both algorithms into consideration. This idea was not explored in this work however as the goal is to use conventional FS methods in a federated environment, but the consistency and alignment of feature masks guarantees fairness in experiments.

The algorithm provides fairness in 'fair' (weighted) mode, as the sample population is equally represented using the calculated weight, meaning that clients with a larger sample size will have more influence on the results. This is directly inspired by the FedAvg algorithm [53], which provides the guarantee of fair representation. The idea of voting was partly motivated by the work of Yue et al. [54], who used voting as a protection against Byzantine attacks in FL environments. This work also reinforces the idea of using binary and ternary masks in a FL environment but for quantized model parameters instead of feature preferences. To demonstrate the effect of freedom degree on the global mask G based on the client selected feature sets, these Venn [55] diagrams can help connect the freedom degree values to the known set operations. In figure 5.1 freedom degree is set to 0 and so the Global mask: G becomes the intersection of selected feature sets of Clients 1,2 and 3. In contrast, figure 5.2 shows how when the freedom degree is set to 1, the Global mask: G becomes the union of selected feature sets of Clients 1,2 and 3. The third and most interesting case is shown in figure 5.3, where freedom degree is set to a value between 0 and 1. The Global mask: G will contain a percentage of the features with the top scores from the $Union \setminus Intersection$ set. In this approximate figure, clients 2 and 3 have a higher weight compared to client 1 others and so it is more probable that more selected features will be from it, with the internal scaling limiting scores from 0 to 1. This visual approximates a freedom degree of $0.25 \sim 0.3$.

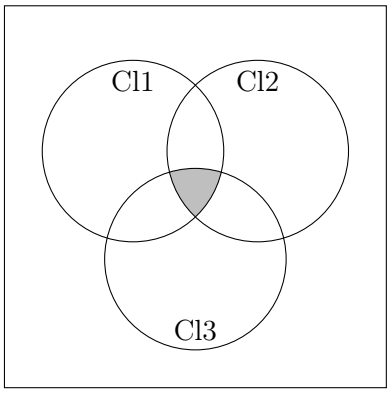

Figure 5.1: Intersection visualization

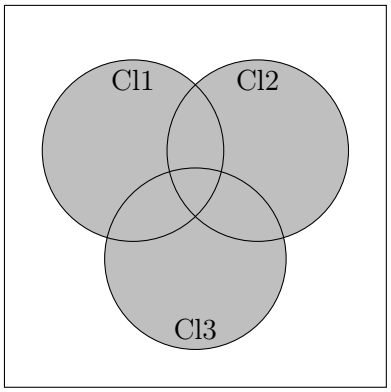

Figure 5.2: Union visualization

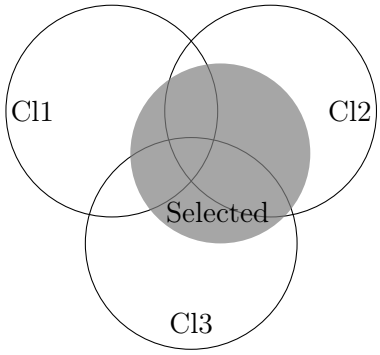

Figure 5.3: Approximate visualization of freedom degree

Algorithm 1 Feature Election Algorithm

Require: K clients with local datasets, each client i has binary mask of selected features $S_i[j]$, feature preference scores $M_i[j]$, number of samples n_i, and freedom_degree $\in [0, 1]$

Ensure: Global feature mask G

Calculate global masks based on client selection:
$Intersection \leftarrow \bigcap_{i=1}^{K} S_i$
$Union \leftarrow \bigcup_{i=1}^{K} S_i$
if freedom_degree $= 0$ **then**
  $G \leftarrow Intersection$
else if freedom_degree $= 1$ **then**
  $G \leftarrow Union$
else
  Compute client weights:
  $w_i \leftarrow \frac{n_i}{\sum_{k=1}^{K} n_k}$ for each client i
  Initialize:
  Selected $\leftarrow \emptyset$
  Normalize preference scores:
  $M_i \leftarrow \hat{M}_i$ for each client i
  Select intersection features:
  Selected $\leftarrow$ Selected $\cup \bigcap_{i=1}^{K} S_i$
  Compute aggregated votes of all clients:
  $V \leftarrow \sum_{i=1}^{K} w_i \cdot M_i$
  Only process features that belong in the Union but not in the Intersection:
  $V_E \leftarrow V \cap (Union \setminus Intersection)$
  $x \leftarrow \lceil \text{freedom\_degree} \cdot |V_E| \rceil$
  Select top x features, decided by freedom degree:
  Selected $\leftarrow$ Selected $\cup$ TopScores(V_E, x)
  $G \leftarrow$ Selected
end if
return G

Sample Algorithm run with 3 clients, LASSO for Feature Selection, freedom degree set to 0.5, mode set to 'uniform' (unweighted)

$freedom\_degree$ =0.5

Binary Mask from Client 1 $S_1[1, 0, 1, 1]$

Binary Mask from Client 2 $S_2[1, 0, 0, 1]$

Binary Mask from Client 3 $S_3[0, 0, 1, 1]$

Client Weights (Calculated)$[0.2, 0.5, 0.3]$

$Intersection$ $[0, 0, 0, 1]$

$Union$ $[1, 0, 1, 1]$

$Union - Intersection$ $[1, 0, 1, 0]$

LASSO Coefficients from Client 1 $M_1[0.530, 0.049, 0.919, 0.121]$

LASSO Coefficients from Client 2 $M_2[1.267, 0.129, 0.048, 0.231]$

LASSO Coefficients from Client 3 $M_3[0.000, 0.229, 1.244, 0.518]$

Normalized preferences, Client 1 $\hat{M}_1$ $[0.327, 0.030, 0.568, 0.075]$

Normalized preferences, Client 2 $\hat{M}_2$ $[0.756, 0.077, 0.029, 0.138]$

Normalized preferences, Client 3 $\hat{M}_3$ $[0.000, 0.115, 0.625, 0.260]$

$[0.327, 0.030, 0.568, 0.0]$

Set $Intersection$ features to 0: $[0.756, 0.077, 0.029, 0.0]$

$[0.756, 0.077, 0.029, 0.0]$

Add Intersection features to Selected

$Selected \leftarrow [X, X, X, 1]$

If mode is "fair" use weighted sum to calculate V

If mode is "uniform" ignore weights:

V =$[1.083, 0.222, 1.222, 0.0]$

$Election\ result$ =$[1.083, 0.0, 1.222, 0.0]$

x =$ceil(freedom\_degree * 2)$

Add x feature(s) with top scores to Selected

Add feature with index 2 to Selected

$Selected \leftarrow [X, X, 1, X]$

$Selected$ =$[0, 0, 1, 1]$

$G \leftarrow Selected$

Chapter 6

Implementation

Based on the methodology described in the previous section, we approached the development of the proposed framework at the communication level, as this is a critical aspect of Federated Learning, where clients are required to communicate frequently over the network. To optimize these communications, we chose to implement the framework using the C++ programming language, which allows for fine-grained control over each communication step. In order to train ML models, Python was the most logical choice for ML tasks as it is compatible with high-performance, popular and easy to set up libraries such as sci-kit learn [31]. It was also used for encryption purposes, as libraries like pyca cryptography [56] are reputable and well maintained.

To combine the benefits of these 2 programming languages and perform FL experiments we use Cython [18] which was introduced on the aptly named paper "Cython: The best of both worlds". It allows the framework to run C++ threads inside a Python environment and control them with traditional OS mechanisms like locks, barriers and mutexes, while also allowing Python to call C++ functions. This implementation is fast, adaptable and cross platform (except for Windows, which currently needs WSL to run FLEx). Another useful feature is the ability to have C++ structures like buffers mapped to Python objects using .pyx wrappers, allowing both programming languages to interact with them as shown below:

```
1 void participate() {
2      std::thread clientThread(&FL_Client::participateThread, this);
3      clientThread.detach();
4 }
```

Listing 6.1: C++ participate method

```
1 cdef class py_fl_client:
2     cdef FL_Client* _client
3
4 def participate(self):
```

```
5       self._client.participate()
```

Listing 6.2: Cython wrapper for C++ methods, _client is a C++ pointer

```
1  def main(ip_address="127.0.0.1", portnum=8080):
2       client = py_fl_client(ip_address, portnum,...)
3       client.participate()
```

Listing 6.3: Python client participation code (simplified) that calls the C++ method participate()

Following the above code shows how a C++ method is called from a Python script, after building the C++ and Cython code, which creates a Python module.

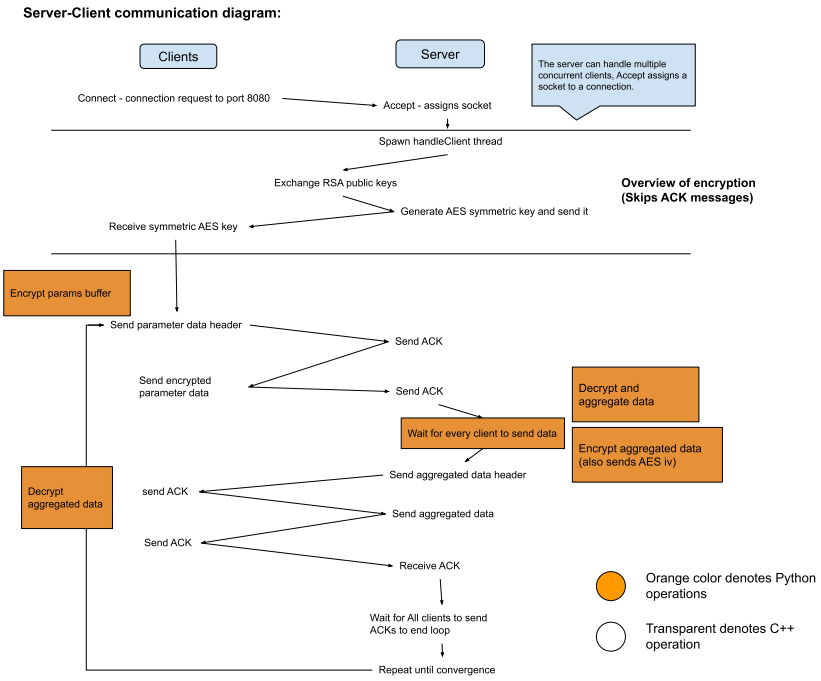

Figure 6.1: The process of server-client communication

Figure 6.1 presents the general communication process of the FLEx framework.

In the beginning key exchange happens and each client trains a model with their local dataset, also making an initial f1 score performance evaluation. In each client Python notifies the C++ script to send the parameters, which are received by the server. These parameters contain the knowledge of different dataset splits due to the nature of FL. The received parameters are aggregated in the server, using the Python class ServerAggregator. These aggregated data parameters are sent to the clients which update their model (using partial fit sci-kit learn [31] methods with

the initial dataset split but with updated model parameters). Then this process is repeated until convergence, which is decided by the server by a value it returns to the clients inside the aggregated parameters.

The system is implemented using Python 3.12 with Cython extensions for performance. Docker containerization is utilized for deployment, using a Debian Slim Bullseye [57] image built for python projects. The implementation includes Docker Compose for facilitating inter-container communication. For model implementations, Scikit-learn and TensorFlow are employed. The system features custom Cython [18] buffer management for efficient data transfer. In this architecture, the Client performs training and model updates while the Server performs parameter aggregation. Data integrity is ensured through MD5 checksums for verification.

The Docker compose configuration file:

```
services:
  server:
    build:
      context: .
      dockerfile: DockerfileServer
      args:
        BUILDKIT_INLINE_CACHE: '0'
    environment:
      - EXPERIMENT_TIME=${EXPERIMENT_TIME:-default}
      - PYTHONUNBUFFERED=1
    command: python server.py --port 8080 --clients ${NUM_CLIENTS:-3}
    --model ${MODEL:-GNB} --fe-method ${FS_METHOD:-impetus}
    --freedom ${FREEDOM_RATE:-0.5}
    networks:
      - app_network
    volumes:
      - ./datasets:/app/datasets:ro
      - ./server.py:/app/server.py:ro
      - ./data/server:/app/data
      - ./logs:/app/logs

  client:
    build:
      context: .
      dockerfile: DockerfileClient
      args:
        BUILDKIT_INLINE_CACHE: '0'
    environment:
      - EXPERIMENT_TIME=${EXPERIMENT_TIME:-default}
      - PYTHONUNBUFFERED=1
    command: python client.py --host server --port 8080
    --dataset ${DATASET:-heartdisease}
    networks:
      - app_network
    depends_on:
      - server
    volumes:
      - ./datasets:/app/datasets:ro
      - ./client.py:/app/client.py:ro
      - ./data:/app/data
      - ./logs:/app/logs

```

```
43 networks:
44   app_network:
45     driver: bridge
```

Listing 6.4: Docker compose setup

Federated Learning is by nature more secure than sharing datasets which may contain sensitive data. However, there are still challenges like data leakage or byzantine attacks. All the data shared between the server and the clients is end to end encrypted. This is important in order to protect the data from malicious outsiders. Symmetric (AES) and asymmetric (RSA) encryption was used accordingly on the Python side. We chose the Python Cryptography library [56] which is a robust, secure and simple solution.

In the beginning, the server and client exchange RSA public keys with each other. The server then generates a symmetric key for AES which is encrypted with the server's private key and sent, to be decrypted with the server's public key by the client. After that, every piece of data sent is encrypted with the shared key using AES, also generating an IV which is necessary for the decryption and sent alongside the encrypted data.

In summary, the security protocol consists of the following stages:

1. RSA key pair generation for each participant
2. Public key exchange between server and clients
3. Symmetric key distribution using asymmetric encryption
4. Encrypted parameter transmission using AES

Federated learning enables collaborative model training by aggregating locally computed updates into a global model without sharing raw data, preserving privacy while leveraging distributed datasets to improve performance across organizations and devices. This aggregation is performed with techniques such as the Federated Averaging (FedAvg) [53] algorithm, which has emerged as the standard in FL scenarios and the more advanced FedProx [58] algorithm that handles heterogeneity better.

The core principle of FedAvg [53] involves multiple rounds of training, where in each round: The server distributes the current global model to a subset of participating clients. Each client trains the model on its local dataset. Clients send their updated model parameters back to the server. The server aggregates these updates into an improved global model.

For all model types, the core aggregation follows a weighted averaging approach based on client dataset sizes. The general formula for aggregating any parameter

θ from clients is:

$$\theta_{\text{global}} = \frac{\sum_{i=1}^{n} N_i \times \theta_i}{\sum_{i=1}^{n} N_i} \tag{6.1}$$

where N_i represents the number of samples at client i, and θ_i represents the model parameter from client i. This ensures that clients with larger datasets have proportionally greater influence on the resulting global model.

Different models require specific parameter handling: For GNB models, we aggregate both means and variances separately:

$$\theta_{\text{global}} = \frac{\sum_{i=1}^{n} N_i \theta_i}{\sum_{i=1}^{n} N_i} \tag{6.2}$$

$$\text{Var}_{\text{global}} = \frac{\sum_{i=1}^{n} N_i \text{Var}_i}{\sum_{i=1}^{n} N_i} \tag{6.3}$$

For linear models like SGD and neural networks like MLPC, we apply the same aggregation principle to their respective parameters.

For SGD, we aggregate coefficients (w): weight vectors and intercepts (b): bias terms:

$$w_{\text{global}} = \frac{\sum_{i=1}^{n} N_i w_i}{\sum_{i=1}^{n} N_i} \tag{6.4}$$

$$b_{\text{global}} = \frac{\sum_{i=1}^{n} N_i b_i}{\sum_{i=1}^{n} N_i} \tag{6.5}$$

For MLPC, this extends to weights and biases across all layers:

$$\text{coefs\_}^{(l)}\text{global} = \frac{\sum_{i=1}^{n} N_i * \text{coefs\_}^{(l)} i}{\sum_{i=1}^{n} N_i} \tag{6.6}$$

$$\text{intercepts\_}^{(l)}\text{global} = \frac{\sum_{i=1}^{n} N_i * \text{intercepts\_}^{(l)}i}{\sum_{i=1}^{n} N_i} \tag{6.7}$$

To improve convergence stability in heterogeneous environments, our framework implements the FedProx algorithm [58] across all model types. FedProx extends standard FedAvg by introducing a proximal term to constrain local updates:

$$F_i(w) + \frac{\mu}{2}||w - w^t||^2 \tag{6.8}$$

where $F_i(w)$ is the local loss function, w represents the local model parameters, w^t denotes the global model parameters at iteration t, and μ is a regularization parameter. The aggregation formula for FedProx modifies the standard approach:

$$w^{t+1} = w^t + \mu \cdot \frac{\sum_{i=1}^{n} N_i(w_i - w^t)}{\sum_{i=1}^{n} N_i} \tag{6.9}$$

This proximal term applies uniformly across all model architectures in our framework. The hyperparameter μ controls the regularization strength: higher values favor stability near the previous global model, while lower values permit greater adaptation to local data distributions. FedProx handles statistical heterogeneity, system heterogeneity, and stragglers in the federated network, significantly improving convergence properties without requiring model-specific modifications to the core algorithm.

To stop the aggregation rounds when necessary, FLEx implements a performance-based convergence criterion, in our case using the average F1 score across all clients, which begins with the standard F1 calculation for each client:

$$\text{f1}_i = 2 \cdot \frac{\text{precision}_i \cdot \text{recall}_i}{\text{precision}_i + \text{recall}_i} \tag{6.10}$$

$$\text{precision}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FP}_i} \tag{6.11}$$

$$\text{recall}_i = \frac{\text{TP}_i}{\text{TP}_i + \text{FN}_i} \tag{6.12}$$

Where TP = True Positives, FP = False Positives and FN = False Negatives.

Then, the average F1 score across all N clients is computed as:

$$\overline{\text{f1}} = \frac{1}{N} \sum_{i=1}^{N} \text{f1}_i \tag{6.13}$$

where $f1_i$ is the f1 score for the i-th client, and N is the total number of clients in the experiment.

The convergence criterion is defined as follows:

$$|\overline{f1}_t - \overline{f1}_{t-1}| < \epsilon \tag{6.14}$$

where ϵ is a predefined threshold (default: 0.05). There is also the option to limit experiments to a set number of FL rounds for better consistency and to compare different methods and configurations in a fair way.

When performing aggregation rounds with default model hyper-parameters and without FS, some FL experiments have shown unstable behavior after using the aggregated parameters. This performance instability is manifesting as dramatic f1 spikes, and poor initial convergence. These issues are well-documented in FL literature [5]. In figure 6.2 the performance spikes were characterized by oscillations between relatively high (∼0.7) and very low scores (∼0.1), while initial training often struggled to achieve consistent baseline performance across clients and oscillation continued for all rounds. This unstable performance was greatly improved when using FS and partially motivates the research done in this thesis.

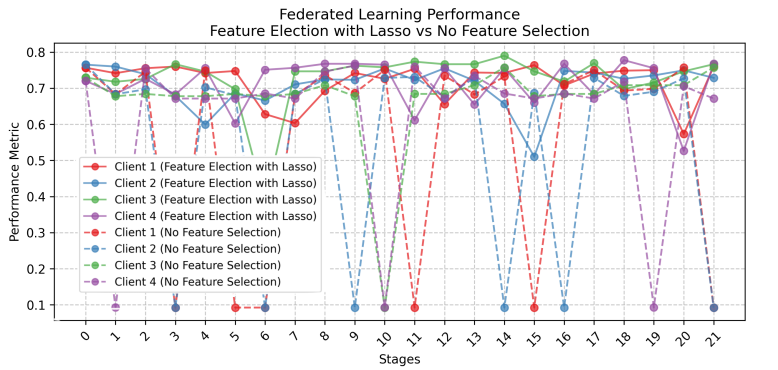

Figure 6.2: Performance spikes when not using Feature Selection.

To optimize the results of the FS methods Lasso [8] and Sequential Attention [9], hyper-parameters such as Lasso's 'alpha' or Sequential Attention's 'stop_percentage' must be carefully tuned. Finding optimal values for these hyper-parameters requires a tuning process, which is standard practice in ML applications. We perform this tuning locally on each client using the robust framework Optuna.

Optuna by Akiba et al. [59] represents a significant advancement in hyperparameter optimization frameworks through its innovative "define-by-run" API and efficient pruning mechanisms. Unlike traditional frameworks that require static definition of parameter search spaces upfront, Optuna allows dynamic construction of search

spaces during runtime, enabling more flexible and intuitive optimization workflows. The authors demonstrate that this design choice, combined with their implementation of Asynchronous Successive Halving (ASHA) for pruning, provides superior performance compared to existing methods.

Optuna's effectiveness is empirically validated through comprehensive experiments. When testing against other frameworks like Hyperopt [60] and GPyOpt on a collection of 56 test cases, Optuna's TPE+CMA-ES sampling strategy showed statistically significant improvements in most scenarios while maintaining much faster computation times. For example, while GPyOpt achieved better optimization in some cases, it took 20 times longer to complete each study.

ASHA pruning was specifically chosen because it "scales linearly with the number of workers in a distributed environment" [11]. The authors demonstrate this through experiments optimizing AlexNet on the SVHN dataset, where TPE with ASHA pruning explored 1,278.6 trials on average compared to just 35.8 trials without pruning in the same 4-hour timeframe. This dramatic efficiency gain, combined with Optuna's lightweight deployment requirements and versatile architecture, makes it well-suited as a plug-and-play optimization solution across different scales of applications - from single-machine experiments to large distributed systems.

The authors validate Optuna's real-world effectiveness by optimizing Faster-RCNN models to achieve 2nd place in the Google AI Open Images Object Detection competition and tuning High Performance Linpack parameters for TOP500 supercomputer benchmarking.

For this work Optuna with ASHA was used locally in each client to fine-tune FS method hyperparameters, like finding the best alpha for Lasso [8], in order to achieve best performance, as demonstrated here:

```
1 study = optuna.create_study(
2              pruner=optuna.pruners.SuccessiveHalvingPruner(),
3              direction='maximize'
4          )
```

Listing 6.5: Optuna study with ASHA

Chapter 7

Experiments

In this section, we take a step further and present experiments to evaluate the performance of FLEx and Feature Election. The 2 benchmarks used to measure performance were the average f1 score of all clients and the total bytes sent by the server, to measure how federated FS reduces network packet size. This reduction in parameter size happens because the size of the ML models we train scales depending on the number of features, when using default parameters and libraries like scikit learn [31]. We tested FL settings with multiple datasets and different numbers of clients, but a split of 3 clients was decided. For these experiments GNB and SGD and MLPC models were used, with three different FS methods: PyImpetus [10], Lasso [8] and Sequential Attention [9]. For all experiments the FedAvg [53] aggregation strategy was used instead of FedProx [58], as the focus is not in complex aggregation but FS performance in HFL.

7.1 Data

To train ML models we used real world data from datasets found on the web, with a large number of features so that the effect of FS would be clear. These datasets have been created by different organizations and use different encoding systems and file formats.

A Python dataset loader was developed for the gut microbiome metabolome preprocessed datasets, [34] based on the given R scripts that generated the pre-processed data. These datasets are very valuable as they contain sensitive data and a large number of features. This loader used pandas and numpy to load the specific dataset using it's name in string format. Then the data was processed by a custom pipeline that handles both numerical and categorical features. The numerical features were processed by a MinMax scaler while the categorical features were processed by a OneHot encoder.

The RNA-Seq cancer dataset [61] loader was implemented to handle gene expression

data from five different tumor types (BRCA, KIRC, COAD, LUAD, and PRAD). The dataset comprises 801 samples with 20,531 features representing gene expression levels measured by the Illumina HiSeq platform.

Unfortunately these datasets had too few samples and could not be used for FL for that reason. The results were too unreliable to present and could not adapt to federated splits properly. However, these datasets were instrumental in fine-tuning the Optuna [11] optimizers for both Sequential Attention [9] and Lasso [8], enabling a detailed analysis of the trade-off between feature count, model performance and runtime. These Optuna tuners are executed on runtime and find the best FS method hyperparameters for each dataset and model.

Two of the datasets used for the following experiments datasets were recommended on the Flower [17] database as ideal options for FL. The "Mushroom" [12] and "Adult" [13] datasets were available on the UC Irvine Machine Learning repository. One of the reasons they were selected is the high number of samples, at 8.1k and 32.6k respectively. These high numbers means that splitting the dataset into federated splits will still retain enough data to train ML models on the client side.

For the actual experiments 5 datasets with a great number of samples were used. We utilized two datasets with health data: the Breast Cancer Wisconsin (Diagnostic) dataset [14] and the Heart Disease dataset [15]. The Breast Cancer dataset comprises 569 instances with 30 real-valued features, each representing characteristics of cell nuclei derived from digitized images of fine needle aspirates of breast masses. The target variable indicates whether the tumor is malignant or benign. This dataset is well-suited for evaluating privacy-preserving FL methods due to its sensitive health information. Similarly, the Heart Disease dataset [15] is derived from the CDC's Behavioral Risk Factor Surveillance System (BRFSS), containing health information from over 400,000 adult respondents across the United States. The dataset includes approximately 40 variables related to demographic factors, lifestyle choices, and medical history selected from the original BRFSS survey. Variables cover demographic characteristics, behavioral risk factors such as smoking and physical activity, and health indicators including BMI and existing conditions. The target variable "HadHeartAttack" is binary, indicating presence or absence of heart disease, with notable class imbalance that requires appropriate handling during model development. The TUANDROMD [62] (Tezpur University Android Malware Dataset) dataset was also used. It is a comprehensive dataset for Android malware detection containing 4,465 instances and 241 attributes. It includes 214 permission-based features and 27 API-based features, with each instance classified as either malware or goodware. The dataset contains no missing values and is designed for classification tasks in computer security research. Even though it's data is not sensitive it performs well for experiments and has a far larger number of features compared to the others.

For all the above-mentioned datasets, the data was split horizontally with a Python program, following the HFL paradigm and the experiments follow a CS-FL setup

as the number of clients does not change dynamically and is set to a small number. Each dataset was split into 3 clients with a 50%, 30%, 20% split. This creates an imbalanced scenario in which the FedAvg [53] algorithm usually has great results.

To monitor the effectiveness of FLEx we built an extensive logging mechanism for the experiments. Data like selected features from each client, server bytes sent or received and model performance after each FL round is stored. To make the experiments more deterministic we set a numpy random seed of 42. A Graphical User Interface (GUI) tool based on PyQt6 was built to visualize data and plot results. It allows monitoring the performance metrics during the execution. In addition, it allows us to visualize the differences between FS method, freedom degrees, and datasets.

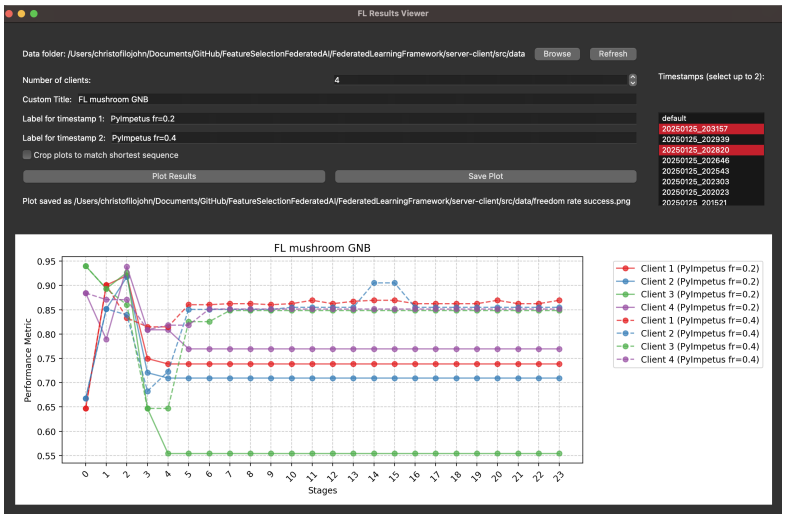

Figure 7.1: GUI tool for plotting

7.2 Evaluation

For our FL setup, we horizontally partitioned each dataset among three clients with a 50%, 30%, 20% distribution of the samples, maintaining the original feature space across all clients. This uneven split was chosen to evaluate our framework's performance in realistic scenarios where data is not uniformly distributed among participants.

It currently implements three diverse FS methods, chosen based on their research backing, code availability, and novelty. Lasso [8] functions as an embedded method, implementing regularized least squares regression with a continuous shrinking operation that can reduce coefficients to zero. This established method is available in Sklearn [31]. Sequential Attention [9] combines an iterative attention mechanism with greedy forward selection, reducing computational complexity from examining

k features among d candidates to k evaluations through parallel approximation of feature marginal gains. The recently introduced PPFS by [10] represents the first wrapper-based MB Feature Selection method, using Predictive Permutation Independence testing to evaluate features by comparing model performance with original versus permuted features. PPFS provides theoretical guarantees under faithfulness conditions and demonstrates superior empirical results using fewer features. The implementation is available in the established PyImpetus library. To optimize the hyperparameters of some of these FS methods, multiple runs with Optuna by Akiba et al. [11] are performed on runtime on each client. Then the client runs the selector with the best performing parameters to perform local FS and generates the voting vector for Feature Election based on the feature mask and preference scores.

Using the GUI tools some plots were generated to showcase the most notable out of the many experiments that were performed. These experiments span 5 datasets, 3 model types and multiple freedom degree value tests (each value from 0.1 to 1 with a step of 0.1 is tested and the experiment with the best performing freedom degree variable is kept).

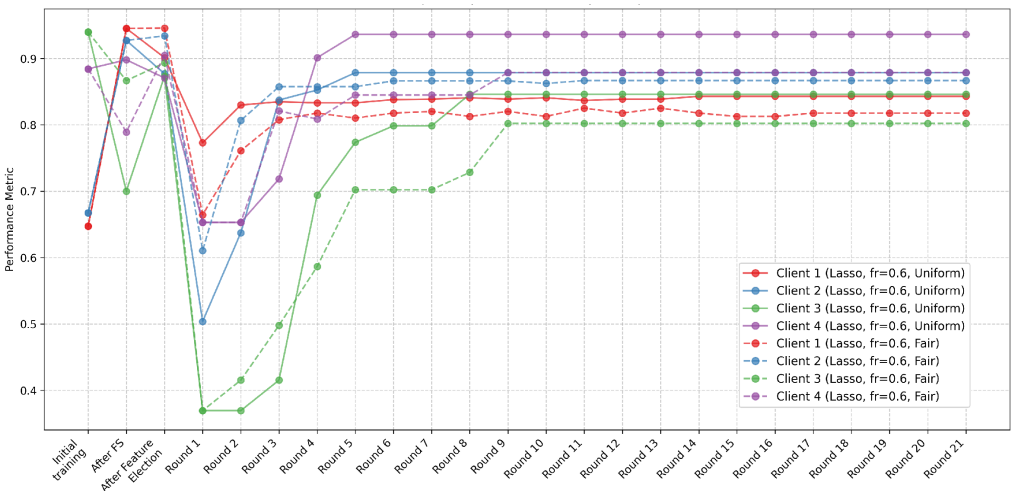

Figure 7.2: Mushroom dataset GNB model, Lasso 'uniform VS fair'

For figure 7.2 the Lasso Feature Election was performed in 'uniform'(unweighted) mode. In the above figure there is a comparison between Feature Election 'fair'(weighted) vs 'uniform'(unweighted). Because of the above mentioned split, that leaves less voting power to the clients with only 10% of the dataset, 'uniform' method produces a better score as even these clients with less number of samples can make a greater impact in the global feature mask. Lasso is a simpler FS algorithm compared to the others, so thinking that it needs more samples to reach similar performance is a logical assumption. This section proves the adaptability of the Feature Election algorithm as this different mode can help in some cases where the federated splits

are more extreme. It is worth noting that in the other experiments 'uniform' mode did not improve performance drastically enough or reduced performance compared to 'fair' and so the latter was chosen to maintain fairness.

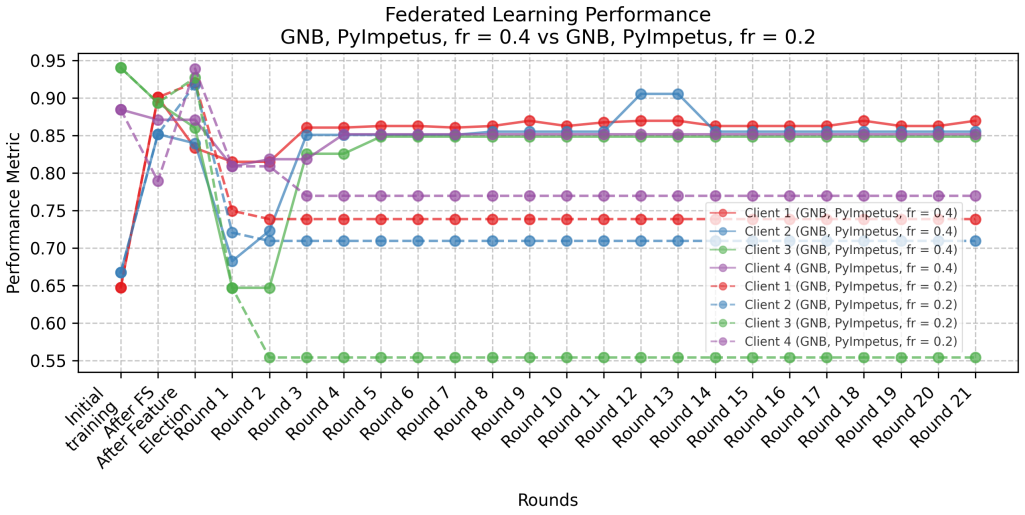

Figure 7.3: Freedom degree comparison, GNB model, mushroom dataset

In plot 7.3, we can see how different freedom degree values impact performance. A stricter freedom degree of 0.2 which is closer to the *Intersection* of selected feature sets loses in this specific experiment to a value of 0.4, that allows 20% more features that exist in *Union* but not in *Intersection* into G. This does not mean that lower freedom degree values have worse performance. In other datasets lower freedom degree values perform better. The freedom degree range associated with peak performance differs substantially across datasets and models, suggesting that optimal thresholds should be calibrated for each specific dataset.

In the following plots the progression of the average f1 performance of all clients is shown (sum of f1 performance of clients divided by the number of clients). The first 3 stages of the x axis document the federated FS process, with "Initial" being the performance with the complete feature set, "FS" being the results with local dataset FS and "FE" symbolizing the results with global feature mask from Feature Election. In the next points the progression of average f1 performance up to round 15 is documented.

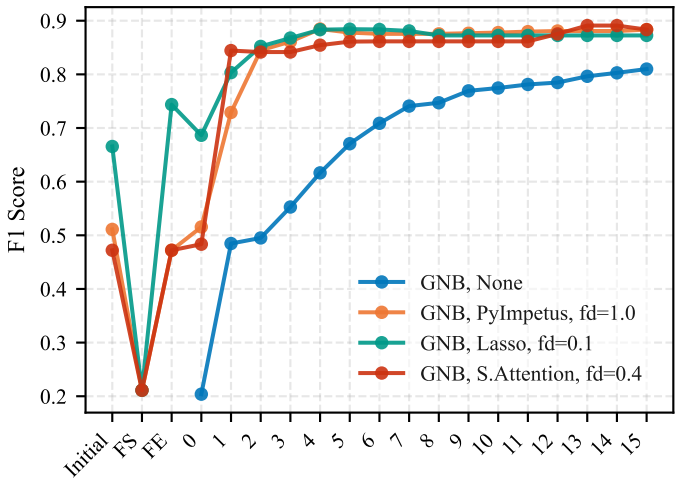

Figure 7.4: Mushroom dataset GNB

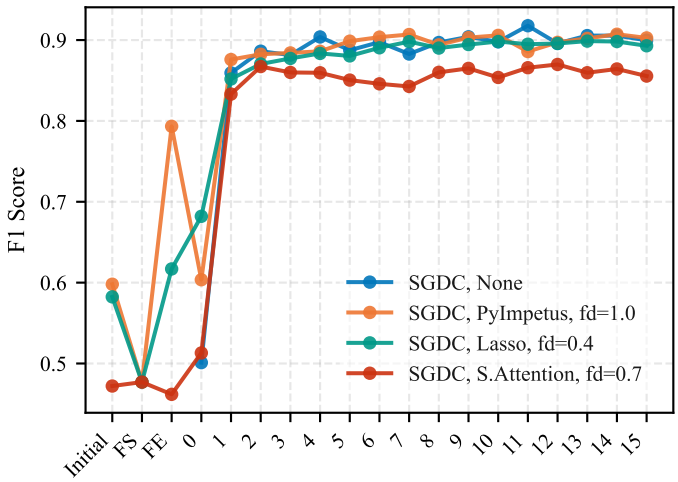

Figure 7.5: Mushroom dataset SGDC

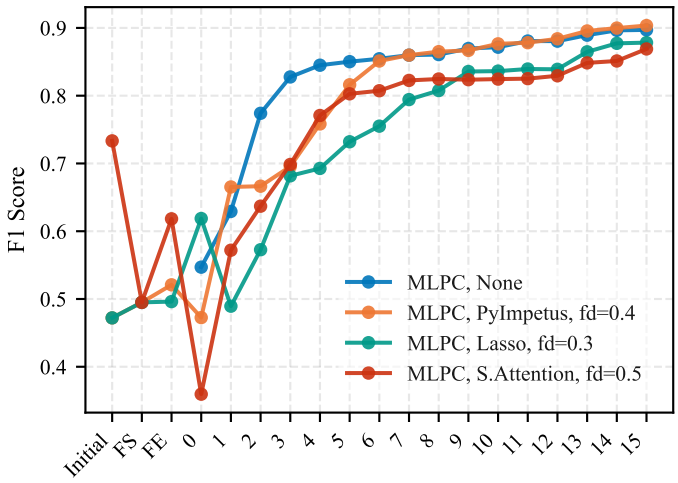

Figure 7.6: Mushroom dataset MLPC

In figures 7.4, 7.5, 7.6 we can see some examples of successful federated FS experiments. In all plots, PyImpetus manages to have better performance than the experiments without FS, with greatly reduced parameter sizes (53.9% in GNB, 35.2% in SGDC and 49.9% in MLPC models). With GNB models especially every FS method improves performance meaningfully, resulting in faster convergence and improved f1 scores. In the SGD comparison both PyImpetus and Lasso perform very well with both offering a substantial parameter reduction (35.2% and 38.2% respectively). In the final, MLPC plot the no FS converges faster to a good performance compared to FS methods. Generally MLPC models don't require FS as they have a complicated architecture that can handle multiple features. However, during the initial fit the neural network's dimensions are set, taking into account the number of dataset features. When a dataset has less features the model is initialized with a smaller layer size, leading to the reduction in parameter size shown in figure 7.21 and Table 7.3.

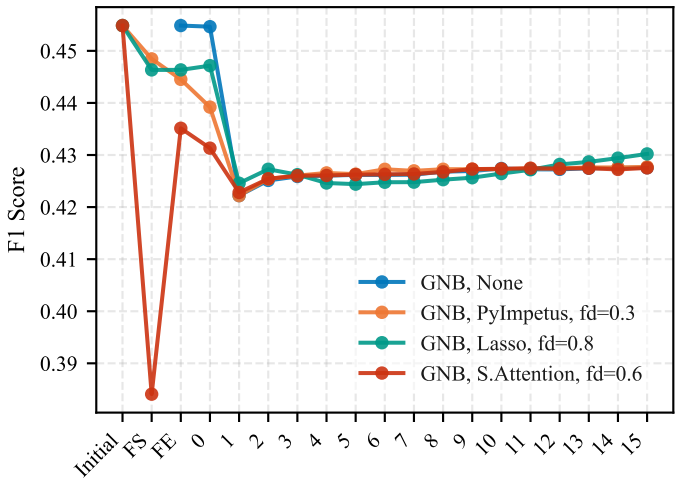

Figure 7.7: Adult income dataset GNB

In figure 7.7 the f1 scores for the GNB model are very close to each other, with every case having better performance than the SGD models. This result is interesting as GNB is a much simpler model, but the Income dataset in general has very low f1 score performance with all of our experiments, reaching a maximum of 0.47 even in scenarios without FL as shown in Table 7.4. In the GNB experiments Lasso performed the best, with a freedom degree of 0.8. Lasso, the simplest method and GNB the simplest of the models had the best prediction performance while also reducing parameter size by 42.9%.

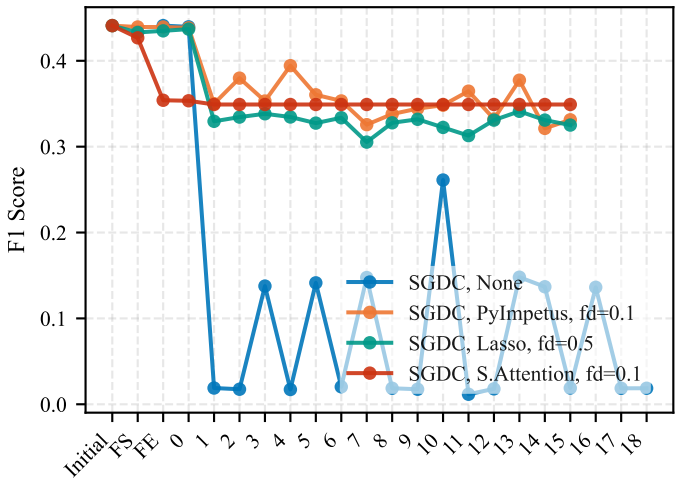

Figure 7.8: Adult income dataset SGDC

A clear win for federated FS can be seen in figure 7.8 as performance without FS is very unstable with the average f1 score going down to 0.2. In this case

S.Attention is the best performing FS method, having the highest average f1 score (0.35) and highest percentage reduction of parameter size (88.8%). In this specific experiment the freedom degree was set to 0.1. This shows that even with low freedom degree (fd) values a great parameter reduction can be achieved, if the local FS experiments resulted in large feature masks. A very similar observation can be made for figure 7.9, but this time PyImpetus is the best performing method in f1 score but without a great parameter size reduction (20%) compared to the others (40%). This plot is a clear demonstration of the stability benefits of performing Feature Selection.

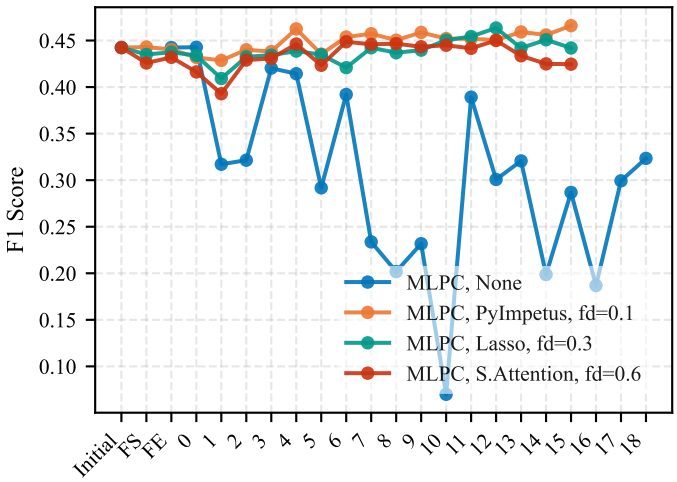

Figure 7.9: Adult income dataset MLPC

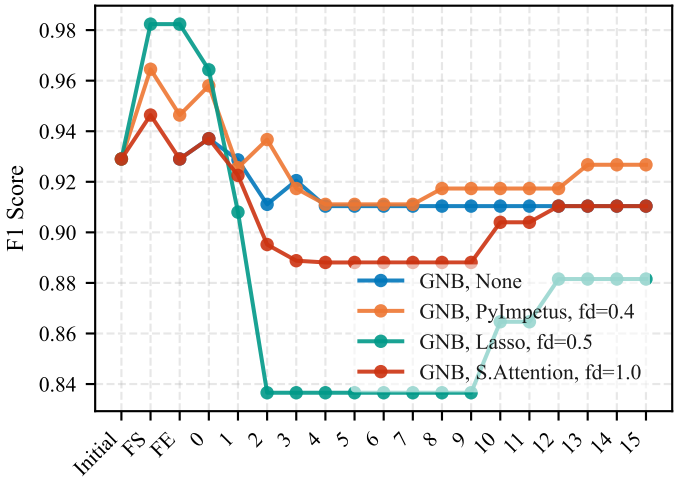

Figure 7.10: Breast cancer dataset GNB

In this figure 7.10 only PyImpetus manages to meaningfully surpass the No FS run

in f1 score. S.Attention manages the same score but with a worse progression plot and Lasso FS f1 performance drops a lot in the beginning.

Figure 7.11: Breast cancer dataset SGDC

In this figure 7.11 all FS methods are defeated by the No FS experimental run. This doesn't show in Table 7.1 as all the numbers are rounded up. It's important to note that FS is not expected to increase performance in every dataset and model configuration. Results like this are expected and the difference is minimal, considering the network traffic reduction benefits. In this specific random state (42) the Optuna optimization could not find good enough combinations for Lasso, S.Attention to improve performance.

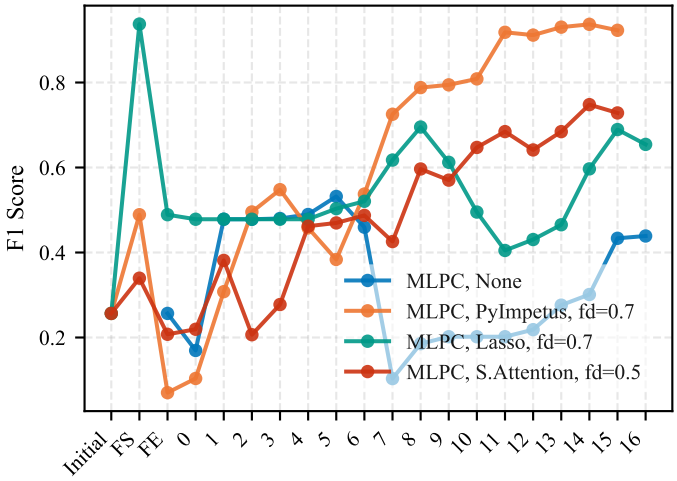

Figure 7.12: Breast cancer dataset MLPC

This result changes in figure 7.12 where all FS methods perform better than the baseline. PyImpetus and S.Attention especially are showing a mostly upward trend , meaning that more FL rounds could be beneficial. The Lasso FS run took one more round, as for these experiments the convergence threshold criteria was enabled. It did not terminate on round 15 because the difference of performance from round 14 to 15 was greater than the threshold.

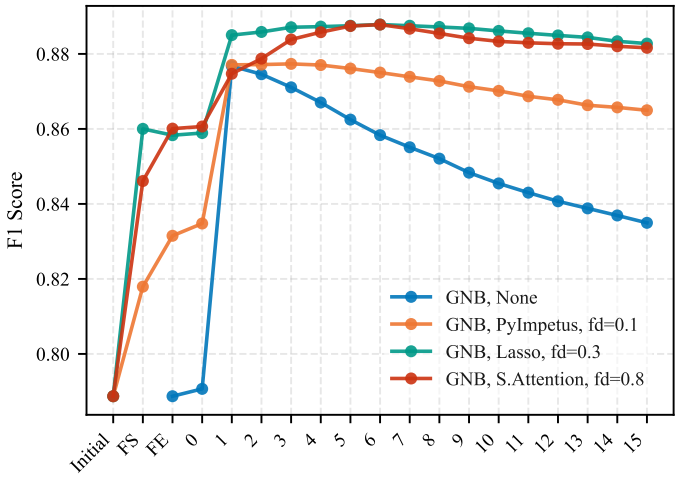

Figure 7.13: Heart disease dataset GNB

In figure 7.13 a successful FS experiment can be seen as all the FS methods outperform the No FS run by a substantial amount (0.05), with PyImpetus having the worst performance out of the 3 methods and Lasso the best, with a 41% drop in parameter size. The most impressive result is however S.Attention which manages to have the highest average f1 score (0.88) while reducing parameter size by 84.35%.

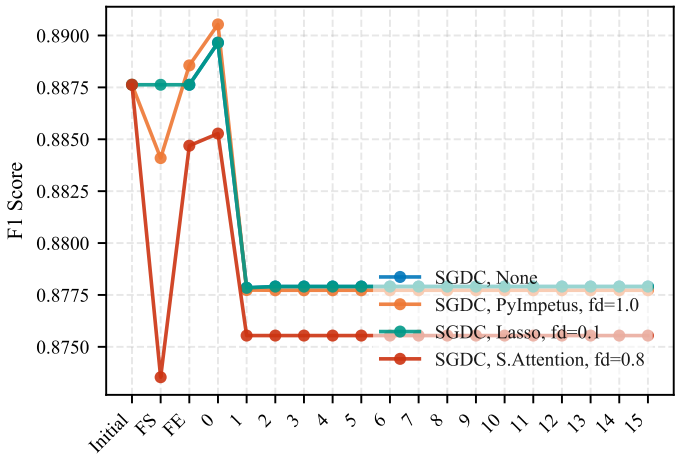

Figure 7.14: Heart disease dataset SGDC

In figure 7.14 the performace metrics are very close with all rounding up to 0.88, but Lasso has the highest score while still offering a 34.2% percentage reduction in parameter size.

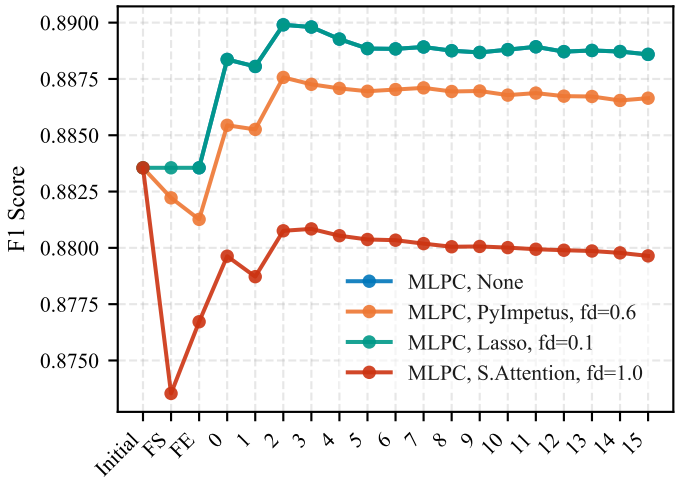

Figure 7.15: Heart disease dataset MLPC

In figure 7.15 the No FS scenario is not clearly visible as it overlaps completely with the Lasso average f1 scores. In this case Lasso performs exactly the same, as it selects all the features in all clients, and so the FS global mask contains all features, regardless of freedom degree value. This results in a very small reduction of parameter size of 4%, which is just random variation of default model parameter size. PyImpetus had the same rounded up score but with a bigger parameter reduction of 22.0%, by selecting 28 out of 37 features.

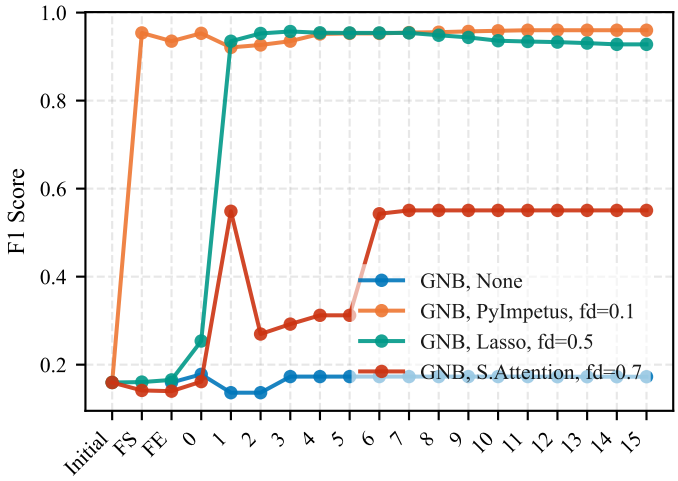

Figure 7.16: TUANDROMD dataset GNB

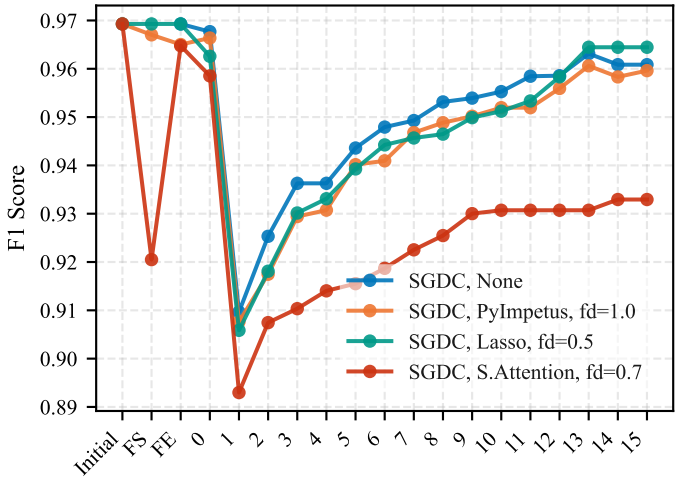

Figure 7.17: TUANDROMD dataset SGDC

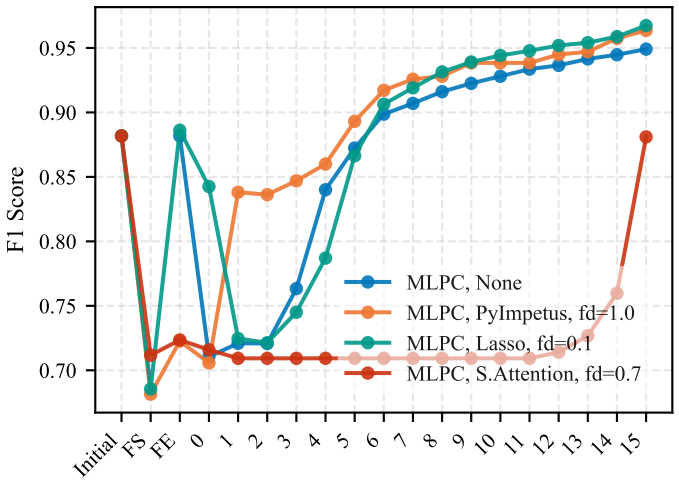

Figure 7.18: TUANDROMD dataset MLPC

In all the experiments with the TUANDROMD dataset as shown in figures 7.16, 7.17, 7.18 a bad performance by S.Attention can be observed, only beating the No FS scenario with the GNB model. This can be attributed to Attention not being a suitable method for performing FS on this dataset or it's approximations, mentioned in the Background chapter being too restrictive. Lasso wins on the SGDCC and MLPC models, while PyImpetus remains generally consistent and wins in the GNB model, showing an impressive lead in the first rounds as shown in figure 7.16 and reducing parameter size greatly on every model as presented in figure 7.22 for the TUANDROMD dataset. Lasso, as the simplest of the 3 methods, shows very good performance and proves a robust and simple solution that also requires the least computational power.

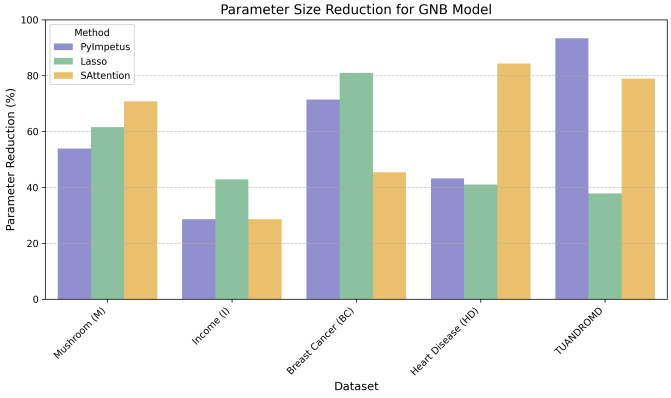

Figure 7.19: Parameter reduction on GNB models

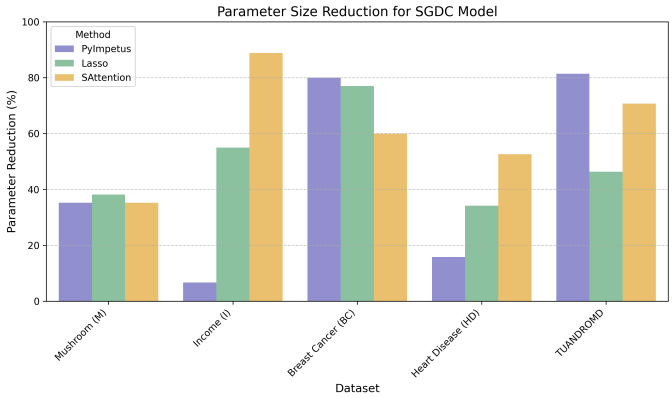

Figure 7.20: Parameter reduction on SGDC models

In figures 7.19, 7.20, 7.21 we can see that the parameter size reduction is similar, in general. Performing FS gives similar or improved performance with great reductions in parameter size.

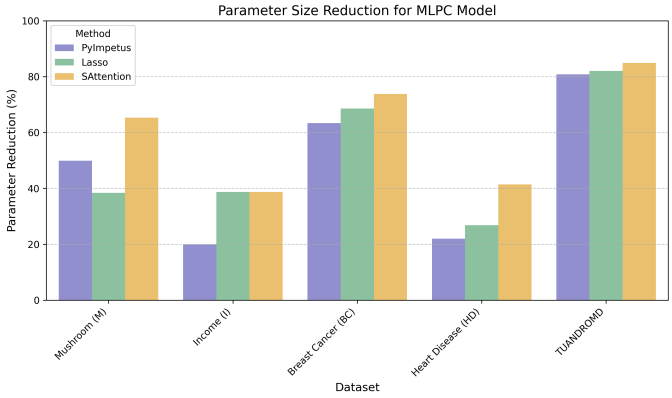

Figure 7.21: Parameter reduction on MLPC models

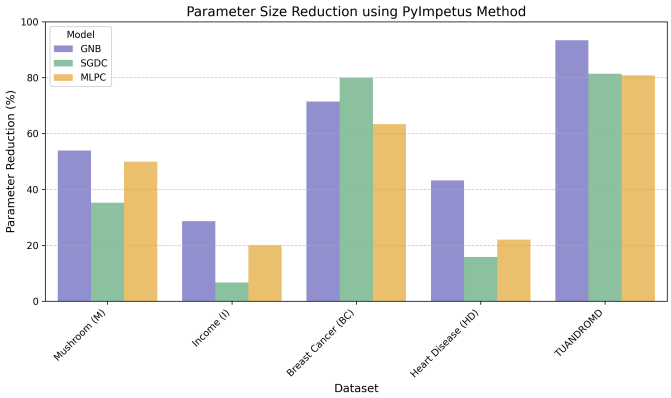

Figure 7.22: Parameter reduction with PyImpetus FS

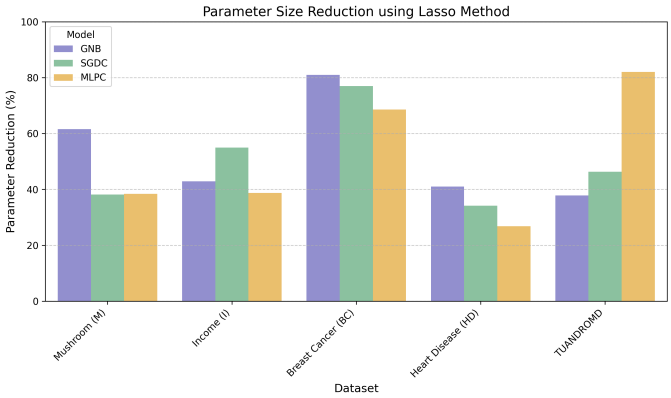

Figure 7.23: Parameter reduction with Lasso FS

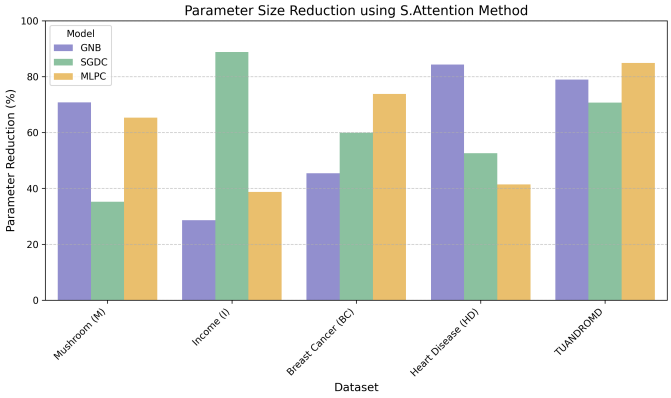

Figure 7.24: Parameter reduction with S.Attention FS

In figures 7.22, 7.23, 7.24 we can see that all methods are capable of greatly reducing parameter size. S.Attention seems on average better in this aspect compared to PyImpetus but all perform similarly. For more detailed comparisons see Table 7.3. In some cases the reduction of parameter size is vast , reaching a 93.4% difference (PyImpetus) in GNB, with SGD models reaching an 88.8% decrease (S.Attention) and MLPC models reaching 84.9% (S.Attention). This is a great result as FL should aim for lower network usage, especially if deployed on a CD-FL setting in which mobile devices are involved. It also makes the network communication part of the experiments faster, although there is an initial runtime cost for each client to perform FS. Based on these numbers, **one round of Feature Election saves data equivalent to multiple communication rounds**.

The main results of these experiments are grouped in the Tables 7.1 and 7.3. In the first table (7.1) the performance of the Feature Election experiments with and without different FS methods are shown. As mentioned before, the experiments are performed with multiple fd values from 0.1 to 1 and the best performing one is registered. Using a specific random state results in reproducible experiments for fair comparisons.The best performing federated FS results are in bold. In the second Table (7.3) the total server parameter size reduction (compared to the no FS results) are shown. Based on the experimental results, in every run FS resulted in substantial (Up to: 93.4%, Median: 55.0%, Average: 53.59%) reduction in parameter size while maintaining or improving the average performance per client with at least one FS method. These results showcase that federated FS is effective compared to the baseline and that Feature Election can make conventional algorithms such as PyImpetus [10] work in federated settings.

In Table 7.4 the performance without FL, on the same settings is displayed, along with the difference to the federated performance displayed in Table 7.1. For this experiments the training and testing tests of the 3 clients were merged again into the complete dataset, following the same train-test split of 80-20 and ML experiments were performed. Using the fist column a comparison can be made with the federated and non-federated results. The expectation is for the non federated performance to be higher, as the dataset is centralized in one machine, but there are some outlier cases, like the Mushroom dataset, GNB model. In most cases the federated methods converge to or are very close to the non federated performance. PyImpetus [10] seems to work exceptionally well in that remark, as on average its federated performance is below the non federated by only 0.035. These f1 score differences showcase the effectiveness of the FedAvg [53] algorithm, which enables very similar performance without sharing private data.

Table 7.1: Performance Comparison Across Models (F1-scores)

Model	Dataset	None	PyImpetus	Lasso	S.Attention
GNB	M	0.81	0.88	0.87	**0.88**
	I	0.43	0.43	0.43	0.43
	BC	0.93	0.92	0.85	0.93
	HD	0.83	0.86	**0.88**	**0.88**
	TU	0.17	**0.96**	0.93	0.55
SGDC	M	0.90	0.90	**0.91**	0.86
	I	0.02	**0.35**	0.32	**0.35**
	BC	**0.85**	0.80	0.76	0.80
	HD	0.88	0.88	0.88	0.88
	TU	0.96	0.96	0.96	0.93
MLPC	M	0.90	0.91	**0.92**	0.87
	I	0.31	**0.47**	0.46	0.45
	BC	0.41	**0.94**	0.68	0.70
	HD	0.89	0.89	0.89	0.88
	TU	0.95	0.96	**0.97**	0.91

Dataset codes - M: Mushroom, I: Income, BC: Breast Cancer, HD: Heart Disease, TU: TUANDROMD

Table 7.4: Non federated performance comparison (F1-scores(Difference to federated))

Model	Dataset	None	PyImpetus	Lasso	S.Attention
GNB	M	0.50 (-0.31)	0.89 (+0.01)	0.89 (+0.02)	0.64 (-0.24)
	I	0.45 (+0.02)	0.45 (+0.02)	0.45 (+0.02)	0.41 (-0.02)
	BC	0.97 (+0.04)	0.97 (+0.05)	0.97 (+0.12)	0.95 (+0.02)
	HD	0.79 (-0.04)	0.86 (+0.00)	0.86 (-0.02)	0.86 (-0.02)
	TU	0.17 (+0.00)	0.95 (-0.01)	0.95 (+0.02)	0.12 (-0.43)
SGDC	M	0.96 (+0.06)	0.93 (+0.03)	0.93 (+0.02)	0.78 (-0.08)
	I	0.44 (+0.42)	0.44 (+0.09)	0.43 (+0.11)	0.43 (+0.08)
	BC	0.99 (+0.14)	0.98 (+0.18)	0.99 (+0.23)	0.97 (+0.17)
	HD	0.89 (+0.01)	0.88 (+0.00)	0.89 (+0.01)	0.87 (-0.01)
	TU	0.98 (+0.02)	0.97 (+0.01)	0.98 (+0.02)	0.96 (+0.03)
MLPC	M	1.00 (+0.10)	1.00 (+0.09)	1.00 (+0.08)	0.96 (+0.09)
	I	0.47 (+0.16)	0.47 (+0.00)	0.46 (+0.00)	0.43 (-0.02)
	BC	0.99 (+0.58)	0.99 (+0.05)	0.98 (+0.30)	0.98 (+0.28)
	HD	0.89 (+0.00)	0.88 (-0.01)	0.89 (+0.00)	0.87 (-0.01)
	TU	0.96 (+0.01)	0.98 (+0.02)	0.99 (+0.02)	0.92 (+0.01)

Dataset codes - M: Mushroom, I: Income, BC: Breast Cancer, HD: Heart Disease, TU: TUANDROMD

Table 7.2: Feature Augmentation - Performance Comparison VS Flower and using different FS methods (F1-scores)

Model	Dataset	None	Flower	PyImpetus	Lasso	S.Attention
GNB	BCa	0.97	0.97	0.97	**0.98**	0.96
	TUa	0.14	0.28	**0.96**	**0.96**	0.89
SGDC	BCa	0.87	**0.97**	0.87	0.66	0.88
	TUa	0.96	0.95	**0.96**	**0.96**	0.86
MLPC	BCa	0.83	0.84	**0.85**	0.83	0.90
	TUa	0.96	**0.97**	0.96	0.96	0.85

Dataset codes - BCa: Breast Cancer augmented, TUa: TUANDROMD augmented

Table 7.5: Flower with Feature Selection using PyImpetus (F1-scores)

Model	Dataset	Baseline	With FS	FS method	Model Size Reduction	fd
GNB	M	0.752	0.847	PyImpetus	44.4%	0.7
	I	0.454	0.427	PyImpetus	48.3%	0.3
	BCa	0.965	0.972	PyImpetus	50.3%	0.3
	TUa	0.278	0.451	PyImpetus	67.7%	0.3
SGDC	M	0.954	0.930	Lasso	47.8%	0.3
	I	0.428	0.407	PyImpetus	46.7%	0.3
	BCa	0.966	0.963	PyImpetus	50.0%	0.3
	TUa	0.949	0.949	PyImpetus	65.8%	0.7
MLPC	M	0.982	0.945	Lasso	42.3%	0.1
	I	0.463	0.451	Lasso	7.7%	0.1
	BCa	0.836	0.932	PyImpetus	48.2%	0.3
	TUa	0.970	0.971	PyImpetus	66.4%	0.5

Dataset codes - M: Mushroom, I: Income, BCa: Breast Cancer augmented, TUa: TUANDROMDaugmented

For Table 7.2 we performed some FL experiments again, with feature augmentation. The reasoning behind this decision is that the selected datasets have an adequate number of samples but not enough features to showcase the power of FS. To solve this, all the numerical features are selected, ignoring the categorical ones and a multiplication operation between all possible pairs of numerical features are generated as $fn = fi * fj$,with $i \neq j$. These new columns are then appended to the existing ones, with the expectation of performance without FS to drop, as the model has increased dimensionality and not all new features are informative.

Table 7.3: Parameter size reduction using FS methods (%)

Model	DS	PyImpetus	Lasso	S.Attention
GNB	M	53.9%	61.6%	70.8%
	I	28.6%	42.9%	28.6%
	BC	71.4%	81.0%	45.4%
	HD	43.2%	41.0%	84.3%
	TU	93.4%	37.8%	78.9%
SGD	M	35.2%	38.2%	35.2%
	I	6.7%	55.0%	88.8%
	BC	80.0%	77.0%	59.9%
	HD	15.8%	34.2%	52.6%
	TU	81.4%	46.3%	70.7%
MLPC	M	49.9%	38.4%	65.3%
	I	20.0%	38.7%	38.7%
	BC	63.4%	68.6%	73.8%
	HD	22.0%	4.0%	41.4%
	TU	80.8%	82.0%	84.9%

DS: Dataset codes - M: Mushroom, I: Income, BC: Breast Cancer, HD: Heart Disease, TU: TUANDROMD

Performance with FS is expected to stay the same, or even improve, as the augmentation features can possibly be selected instead of some of the originals. This process greatly increases the number of features for the datasets TUANDROMD and breastcancer from 241 to 648 and from 30 to 81 respectively. Other datasets like Mushroom were ignored as they contain mostly categorical features. For the TUANDROMD dataset we can see a wide gap between No FS and PyImpetus for the GNB model and very similar performance in SGDC models. For the Breast Cancer dataset the MLPC performance without FS greatly increases with augmentation, contrary to expectations but the peak FS performance in this dataset drops to 0.9 with S.Attention, compared to 0.94 with PyImpetus without augmentations. The very low MLPC performance without augmentation is an outlier and so it's logical that by running in different conditions this outlier. To prove this hypothesis the experiment with MLPC was rerun without FS and by changing the model's random state to a different number and has an average f1 score of 0.92 without augmentations and 0.86 with. This does not invalidate the previous FS experiments as all comparisons are done with the same model random state to be fair. To make these comparisons more interesting, a custom runner using the industry leading Flower [17] framework was developed. This shows that without FS Flower performs better than our framework, but when applying the Feature Election algorithm, Flex manages to win 4 out of the 6 comparisons.

Table 7.5 presents the some results for the flower runner mentioned in the above text. The performance evaluation without FS shows this framework exceeds our own framework's f1 score performance, particularly with the SGDC and MLPC models. Following these results, we implemented the Feature Election algorithm into the flower framework. For these experiments, only the PyImpetus and Lasso FS methods were tested as they have more stable results. This allows the testing of the horizontal federated FS algorithm developed for this thesis, in the framework that scored the best in the survey [16]. The average f1 score of all clients and the model size - measured on a client before and after fitting data with the new global feature mask, after feature election. The results are favorable as in every run the reduction of model size was great (peaking at 67.7%) and the performance improved greatly in Mushroom and TUANDROMD augmented datasets with the GNB model, and Breast Cancer augmented MLPC model. In some cases like the Income dataset on MLPC models FS loses performance and in other like Breast Cancer augmented dataset SGD model is within the margin of error, but Feature Election reduces the model size greatly. These comparisons prove that Feature Election is a versatile algorithm that can be applied to multiple frameworks and reduce model sizes, which in turn will reduce network traffic, while usually maintaining or improving performance.

In most of the experiments PyImpetus showed impressive stability, by rarely losing performance compared to the No FS scenarios, while greatly reducing parameter size as demonstrated in figure 7.22. Taking into account that this method does not require any hyper parameter tuning and seeing how its federated performance with Feature Election is **incredibly close to its non federated performance** as seen in Table 7.4, this is the standout FS methods from the ones tested. The combination of Feature Election and PyImpetus [10] creates a very strong horizontal federated FS system, with the fd being the only hyperparameter to tune.

Table 7.6: Comparison of Federated Learning Frameworks

Framework	Features	Interoperability	User Friendliness	Total Score
Flower	82.50%	100.00%	80.00%	84.75%
FLARE	85.00%	100.00%	70.00%	80.50%
Fed.Scope	100.00%	75.00%	67.50%	78.75%
PySyft	75.00%	100.00%	60.00%	72.50%
FedML	82.50%	100.00%	52.50%	71.00%
FLEx	**67.50%**	**62.50%**	**75.00%**	**70.25%**
OpenFL	67.50%	37.50%	82.50%	69.00%
EasyFL	25.00%	62.50%	95.00%	67.50%
IBM FL	67.50%	100.00%	52.50%	66.50%
TFF	55.00%	37.50%	77.50%	62.75%
FedLab	37.50%	50.00%	77.50%	60.00%
FLSim	60.00%	25.00%	62.50%	54.25%
FLUTE	55.00%	27.50%	42.50%	43.25%
FATE AI	70.00%	50.00%	15.00%	38.50%
PaddleFL	75.00%	50.00%	5.00%	35.00%
FedLearner	57.50%	25.00%	5.00%	24.75%

In Table 7.6 there is a comparison of many different FL frameworks, based on the recent survey by Riedel et al. [16]. In this survey a ranking of different frameworks were made based on the 3 mentioned criteria (Features, Interoperability User Friendliness) in order to calculate a total score and rank the different frameworks. In order to compare FLEx to the others a simple python script was made following the survey's guidelines (this script will be available along with the rest of the FLEx framework's code). With this data we can compare FLEx to the 2024 versions of other frameworks and the results are very favorable, beating Google's TFF and IBM FL, but still getting beaten by Flower, FLARE and others. Some notable missing features from FLEx are Windows support, less than 0.9 performance on mnist dataset (averaging 0.89 on 5 clients), VFL support and the requirement of Medium development effort. FLEx compares very favorably to other frameworks even though it's key feature, Feature Election is not taken into account for this review, as Feature Selection was not one of the criteria.

The evaluation of FLEx follows the methodology presented in [16], which assesses federated learning frameworks across three main categories: Features (30%), Interoperability (20%), and User Friendliness (50%). Each category contains specific criteria with weighted importance.

Criterion	Assessment	Score
Security Mechanisms (35%)	Cryptographic security only	0.5
FL Algorithms (25%)	FedAvg and FedProx	1.0
ML Models (25%)	Sklearn and experimental PyTorch support	1.0
FL Paradigms (15%)	Horizontal FL only	0.0
Features Total		**67.50%**

Table 7.7: Evaluation of FLEx Features

Criterion	Assessment	Score
Edge Device Rollout (50%)	Supported with restrictions	0.5
OS Support (25%)	MacOS (not Windows)	0.5
GPU Support (15%)	Fully supported	1.0
Docker Installation (10%)	Available	1.0
Interoperability Total		**62.50%**

Table 7.8: Evaluation of FLEx Interoperability

Criterion	Assessment	Score
Development Effort (25%)	Medium effort required	0.5
Model Accuracy (25%)	89% on MNIST (50-90% range)	0.5
Documentation (20%)	Moderate quality	0.5
Training Speed (10%)	11 seconds with GPU (below 60s)	1.0
Data Preparation (10%)	Low effort	1.0
Model Evaluation (5%)	Built-in methods	1.0
Pricing (5%)	All features free	1.0
User Friendliness Total		**75.00%**

Table 7.9: Evaluation of FLEx User Friendliness

The total weighted score for FLEx is:

$$(67.50\% \times 0.30) + (62.50\% \times 0.20) + (75.00\% \times 0.50) = 70.25\% \tag{7.1}$$

This places FLEx in a competitive position among existing federated learning frameworks.

Chapter 8

Conclusions

In this thesis we proposed and evaluated Feature Election, an algorithm for federated Feature Selection that enables the use of conventional FS methods in HFL environments. We performed a series of experiments, testing our implementation on a diverse set of real-world datasets (Mushroom, Adult Income, Breast Cancer, Heart Disease, and TUANDROMD) with three different model types (GNB, SGDC, and MLPC). These experiments demonstrated that federated FS consistently produces substantial benefits.

We tested FLEx with and without Feature Election on the above mentioned configurations. The results show that our approach achieves an average parameter size reduction of 53.59%, with peak reductions of up to 93.4%, demonstrated in Table 7.3 while maintaining or improving the average performance per client with at least one Feature Selection method. Particularly noteworthy is the combination of Feature Election with PyImpetus [10], which demonstrated remarkable stability across experiments and produced results comparable to non-federated implementations and could work as a standalone horizontal federated FS solution. The reduction in communication overhead helps address one of the key challenges in federated learning: the bandwidth bottleneck [5], by using computational power to perform Feature Selection. Li et al. [41] state that minimizing data transfer between server and clients can also help mitigate potential privacy leakage through the network layer.

The integration of our Feature Election algorithm with the industry-leading Flower framework further validates versatility and effectivenes. Even when compared against established federated learning frameworks, FLEx demonstrates competitive performance in key metrics, outperforming several prominent solutions as shown in Tables 7.7, 7.8, 7.9.

These findings establish that Feature Election offers a robust approach to federated Feature Selection that significantly reduces communication costs while preserving

or enhancing model performance. This work represents an important contribution to making FL more efficient and practical for real-world applications where bandwidth constraints and data privacy are critical concerns.

8.1 Future Work

The proposed system can be further enhanced by incorporating additional optimizations and extensions. Testing on a broader range of datasets will help expand its applicability across different domains. Furthermore, an automated mechanism for tuning the Feature Election freedom degree parameter will streamline the process and could improve consistency. This is very important as running experiments multiple times and selecting the best performing results is not applicable to real world problems. More research can help in finding correlation between model, dataset, FS method to the optimal freedom degree parameter, instead of arbitrarily testing different values. Multiple Feature Election rounds with an FL round in between could also help in choosing the best freedom degree value dynamically.

As for the FLEx framework itself, more FL paradigms should be explored, like VFL and FTL and, of course, CD-FL. By integrating more advanced ML models and exploring alternative Feature Selection algorithms that provide preference scores, this framework can be further refined for even more robust and reliable results. Feature Election links the domains of federated and conventional FS in a simple way and so testing the FL performance improvements and parameter size reduction of cutting-edge FS algorithms can help continue this research. Implementing a version of the Feature Election algorithm officially into the Flower [17] or Tensorflow federated [51] frameworks would be a way to more thoroughly test the claims of this thesis and contribute to open source development. The work for implementing Feature Election in Flower is mostly done as seen in Table 7.5. The source code of FLEx and the Feature Election algorithm will be publicly available after the publication of a research paper.

An interesting idea to test the Feature Election algorithm is to use different FS methods on different clients. This is possible because even though different clients use different methods, the output - meaning the binary and preference masks are aligned. The MinMax scaling performed in the Feature Election method allows the combination of Lasso [8] and PyImpetus [10] clients, for example. The sharing of knowledge of two completely different algorithms could provide promising results, showing that mixing algorithms based on the client's processing power and capabilities is viable. In the above example, Lasso with less Optuna optimization trials is much lighter in system resources, compared to the PyImpetus wrapper method, which attempts to find complex relationships between features.

Chapter 9

List of Abbreviations

Acronyms used throughout this work:

ML Machine Learning

FL Federated Learning

FS Feature Selection

FE Feature Election

MB Markov Blanket

FSMB Feature Space Markov Blanket

BAMB Balanced Markov Blanket discovery

HFL Horizontal Federated Learning

VFL Vertical Federated Learning

FTL Federated Transfer Learning

CS-FL Cross-Silo Federated Learning

CD-FL Cross-Device Federated Learning

LDP Local Differential Privacy

IID independent and identically distributed

GNB Gaussian Naive Bayes

SGD Stochastic Gradient Descend

MLPC Multi Layer Perceptron Classifier

GUI Graphical User Interface

PPFS Predictive Permutation Feature Selection

fd freedom degree

Bibliography

[1] P. Qi, D. Chiaro, A. Guzzo, M. Ianni, G. Fortino, and F. Piccialli, "Model aggregation techniques in federated learning: A comprehensive survey," *Future Generation Computer Systems*, vol. 150, pp. 272–293, 2024.

[2] C. Huang, J. Huang, and X. Liu, "Cross-silo federated learning: Challenges and opportunities," 2022.

[3] J. Dean, "A golden decade of deep learning: Computing systems and applications," *Daedalus*, vol. 151, no. 2, pp. 99–116, 2022.

[4] D. Theng and K. Bhoyar, "Feature selection techniques for machine learning: a survey of more than two decades of research," *Knowledge and Information Systems*, vol. 66, 12 2023.

[5] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Aguera y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1273–1282, PMLR, 2017.

[6] I. Dayan, H. R. Roth, A. Zhong, A. Harouni, A. Gentili, A. Abidin, A. Liu, A. Costa, B. J. Wood, C. Tsai, *et al.*, "Federated learning for predicting clinical outcomes in patients with covid-19," *Nature Medicine*, vol. 27, no. 10, pp. 1735–1743, 2021.

[7] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *CoRR*, vol. abs/1806.00582, 2018.

[8] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[9] T. Yasuda, M. Bateni, L. Chen, M. Fahrbach, G. Fu, and V. Mirrokni, "Sequential attention for feature selection," 2023.

[10] A. Hassan, J. H. Paik, S. R. Khare, and S. A. Hassan, "A wrapper feature selection approach using markov blankets," *Pattern Recognition*, vol. 158, p. 111069, 2025.

[11] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631, 2019.

[12] "Mushroom." UCI Machine Learning Repository, 1981. DOI: https://doi.org/10.24432/C5959T.

[13] B. Becker and R. Kohavi, "Adult." UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5XW20.

[14] W. Wolberg, O. Mangasarian, N. Street, and W. Street, "Breast cancer wisconsin (diagnostic)." UCI Machine Learning Repository, 1993. DOI: https://doi.org/10.24432/C5DW2B.

[15] K. Pytlak, "Personal key indicators of heart disease," 2022.

[16] P. Riedel, L. Schick, R. von Schwerin, M. Reichert, D. Schaudt, and A. Hafner, "Comparative analysis of open-source federated learning frameworks - a literature-based survey and review," *International Journal of Machine Learning and Cybernetics*, vol. 15, pp. 5257–5278, nov 2024.

[17] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, J. Fernandez-Marques, Y. Gao, L. Sani, K. H. Li, T. Parcollet, P. P. B. de Gusmão, and N. D. Lane, "Flower: A friendly federated learning research framework," 2022.

[18] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith, "Cython: The best of both worlds," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 31–39, 2011.

[19] P. M. Mammen, "Federated learning: Opportunities and challenges," 2021.

[20] Z. Zhang, W. Li, S. Wang, and Q. Yang, "Towards method of horizontal federated learning: A survey," *IEEE Transactions on Big Data*, 2022.

[21] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, and W. Zhang, "A survey on federated learning: Challenges and applications," *International Journal of Machine Learning and Cybernetics*, vol. 14, pp. 513–535, 2023.

[22] Y. Liu, Y. Kang, T. Zou, Y. Pu, Y. He, X. Ye, Y. Ouyang, Y.-Q. Zhang, and Q. Yang, "Vertical federated learning: Concepts, advances, and challenges," *IEEE Transactions on Knowledge and Data Engineering*, 2024.

[23] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 13, no. 3, pp. 1–207, 2019.

[24] S. Ji, Y. Tan, T. Saravirta, Z. Yang, Y. Liu, L. Vasankari, S. Pan, G. Long, and A. Walid, "Emerging trends in federated learning: From model fusion to federated x learning," *International Journal of Machine Learning and Cybernetics*, vol. 15, pp. 3769–3790, 2024.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, vol. 30, pp. 5998–6008, Curran Associates, Inc., 2017.

[26] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society Series B*, vol. 67, no. 2, pp. 301–320, 2005.

[27] N. Gui, D. Ge, and Z. Hu, "AFS: an attention-based mechanism for supervised feature selection," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 3705–3713, 2019.

[28] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufmann Publishers, 1988.

[29] L. E. Brown and I. Tsamardinos, "Markov blanket-based variable selection in feature space," *Journal of Machine Learning Research*, vol. 8, pp. 1145–1164, 2007.

[30] Z. Ling, K. Yu, H. Wang, L. Liu, W. Ding, and X. Wu, "Bamb: A balanced markov blanket discovery approach to feature selection," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 5, pp. 1–25, 2019.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[32] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, vol. 22, no. 3, pp. 400–407, 1951.

[33] J. Cramer, "The origins of logistic regression," *Tinbergen Institute Working Paper*, 2002.

[34] E. Muller, Y. M. Algavi, and E. Borenstein, "The gut microbiome-metabolome dataset collection: a curated resource for integrative meta-analysis," *npj Biofilms and Microbiomes*, vol. 8, no. 1, pp. 1–7, 2022.

[35] J. Guo, Y. Zhang, S. Li, Q. Wu, J. Kittler, F. Wang, X. Wu, and J. Li, "Federated learning for biometric recognition: a survey," *Artificial Intelligence Review*, pp. 1–40, 2023.

[36] A. M. Elbir and S. Coleri, "Federated learning for vehicular networks," *IEEE Communications Letters*, vol. 24, no. 12, pp. 2795–2799, 2020.

[37] Y. Zhao, L. Zhu, J. Xu, P. Liu, and C. Zhou, "Local differential privacy based federated learning for internet of things," *IEEE Internet of Things Journal*, vol. 8, no. 11, pp. 8836–8845, 2021.

[38] R. Cioffi, M. Travaglioni, G. Piscitelli, A. Petrillo, and F. De Felice, "Artificial intelligence and machine learning applications in smart production: Progress, trends, and directions," *Sustainability*, vol. 11, no. 2, p. 492, 2019.

[39] B. Liu, L. Wang, and M. Liu, "Lifelong federated reinforcement learning: A learning architecture for navigation in cloud robotic systems," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4555–4562, 2019.

[40] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[41] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.

[42] K. T. Putra and H.-C. Chen, "Federated compressed learning edge computing framework with ensuring data privacy for pm2.5 prediction in smart city sensing applications," *Sensors*, vol. 21, no. 1, p. 222, 2021.

[43] M. Shafiq, Z. Tian, A. K. Bashir, X. Du, and M. Guizani, "Iot malicious traffic identification using wrapper-based feature selection mechanisms," *Computers & Security*, vol. 94, p. 101863, 2020.

[44] P. Cassará, A. Gotta, and L. Valerio, "Federated feature selection for cyber-physical systems of systems," *IEEE Transactions on Vehicular Technology*, vol. 71, pp. 9937–9950, September 2022.

[45] T. Castiglia, Y. Zhou, S. Wang, S. Kadhe, N. Baracaldo, and S. Patterson, "LESS-VFL: Communication-efficient feature selection for vertical federated learning," in *Proceedings of the 40th International Conference on Machine Learning* (A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, eds.), vol. 202 of *Proceedings of Machine Learning Research*, pp. 3757–3781, PMLR, 23–29 Jul 2023.

[46] R. Fu, Y. Wu, Q. Xu, and M. Zhang, "Feast: A communication-efficient federated feature selection framework for relational data," in *Proceedings of the ACM International Conference on Management of Data*, p. 107, ACM, 2023.

[47] Y. Hu, Y. Zhang, X. Gao, D. Gong, X. Song, J. Guo, and J. Wang, "A federated feature selection algorithm based on particle swarm optimization under privacy protection," *Knowledge-Based Systems*, vol. 260, p. 110122, 2023.

[48] J. Hermo, V. Bolón-Canedo, and S. Ladra, "Fed-mrmr: A lossless federated feature selection method," *Information Sciences*, vol. 669, p. 120609, 2024.

[49] R. Kapila and S. Saleti, "Federated learning-based disease prediction: A fusion approach with feature selection and extraction," *Biomedical Signal Processing and Control*, vol. 100, p. 106961, 2025.

[50] S. Banerjee, D. Bhuyan, E. Elmroth, and M. Bhuyan, "Cost-efficient feature selection for horizontal federated learning," *IEEE Transactions on Artificial Intelligence*, vol. 5, no. 12, pp. 6551–6565, 2024.

[51] TensorFlow, "Tensorflow/federated." https://github.com/tensorflow/federated, 2024. Accessed: 2025-01-10.

[52] OpenMined, "Openmined/pysyft." https://github.com/OpenMined, 2024. Accessed: 2025-01-10.

[53] H. B. McMahan, E. Moore, D. Ramage, and B. A. y Arcas, "Federated learning of deep networks using model averaging," *CoRR*, vol. abs/1602.05629, 2016.

[54] K. Yue, R. Jin, C.-W. Wong, and H. Dai, "Federated learning via plurality vote," 2022.

[55] J. Venn, "On the diagrammatic and mechanical representation of propositions and reasonings," *Philosophical Magazine and Journal of Science*, vol. 10, no. 59, pp. 1–18, 1880.

[56] T. P. C. A. (PyCA), *pyca/cryptography*, 2025. Python package.

[57] I. Docker, *Official Python Docker Image: 3.12-slim-bullseye*, 2025. Docker Image.

[58] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," 2020.

[59] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.

[60] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, and D. Cox, "Hyperopt: A python library for model selection and hyperparameter optimization," *Computational Science & Discovery*, vol. 8, p. 014008, 07 2015.

[61] S. Fiorini, "gene expression cancer RNA-Seq." UCI Machine Learning Repository, 2016. DOI: https://doi.org/10.24432/C5R88H.

[62] P. Borah and D. K. Bhattacharyya, "TUANDROMD (Tezpur University Android Malware Dataset)." UCI Machine Learning Repository, 2020. DOI: https://doi.org/10.24432/C5560H.